

Fujitsu Software Technical Computing Suite V4.0L20

Development Studio Fortran User's Guide

J2UL-2473-02ENZ0(11)
September 2023

Preface

Purpose of This Manual

This manual explains how to use the Fortran system (called "this system" in this manual) intended for the system that has the Fujitsu CPU A64FX installed (called "FX system" in this manual).

It also describes the following aspects of the system:

- Fortran programming
- Restrictions

Intended Readers

This manual was written for people who write Fortran programs and process them using this system.

This manual assumes that readers know how to use Linux(R) commands, file manipulation and shell programming.

This manual assumes an understanding of the Fortran language specification. For details of the Fortran language specification, refer to "Fortran Language Reference" on online manual, the Fortran language standard, or commercial books.

Structure of This Manual

The structure of this manual is as follows:

[Chapter 1 Overview of the Fortran System](#)

This chapter provides an overview of how to use this system to process Fortran programs.

[Chapter 2 Compiling and Linking Fortran Programs](#)

This chapter describes how to compile and link Fortran programs.

[Chapter 3 Executing Fortran Programs](#)

This chapter describes how to execute Fortran programs.

[Chapter 4 Messages and Listings](#)

This chapter describes the system messages and listings produced as a result of compiling a Fortran source program or executing a Fortran program.

[Chapter 5 Data Types, Kinds, and Representations](#)

This chapter describes the Fortran data types that can be used in this system.

[Chapter 6 Notes on Programming](#)

This chapter describes some of notes on programming on this system. Input/output features are described in "[Chapter 7 Input/Output Processing](#)".

[Chapter 7 Input/Output Processing](#)

This chapter describes files processed by Input/output statements and the necessary basic properties of Fortran records and files for using input/output statements.

[Chapter 8 Debugging](#)

This chapter describes the debugging functions provided by this system and how to operate them.

[Chapter 9 Optimization Functions](#)

This chapter describes functions to use optimization functions effectively, as well as some points to note.

[Chapter 10 Fortran Modules and Submodules](#)

This chapter describes some of notes on compiling programs including modules and submodules.

[Chapter 11 Mixed Language Programming](#)

This chapter describes the specification and notes when linking a Fortran program and a C/C++ program.

Chapter 12 Multiprocessing

This chapter describes how to perform parallel processing of a Fortran program.

Appendix A Restrictions and Notes

This appendix describes restrictions and notes of this system.

Appendix B Endian Convert Command

This appendix describes endian convert commands provided by this system.

Appendix C Preprocessor

This appendix describes the C language preprocessor and the Fortran preprocessor.

Appendix D GNU Compatible Options

This appendix describes GNU compatible options supported by this system.

Appendix E Data and Memory Regions

This appendix describes the attributes of data and memory regions in this system.

Appendix F Intrinsic Procedures and I/O Statements Compatible with Half-Precision Type

This appendix describes intrinsic procedures and input/output statements compatible with the half-precision type.

Appendix G Code Coverage

This appendix describes the code coverage.

Appendix H Runtime Information Output Function

This appendix describes the Runtime Information Output Function.

Appendix I Using High-Speed Facility on Job Operation Software

This appendix describes information on the compilation and execution processes for using the high-speed facility on FX system.

Appendix J Fujitsu OpenMP Library

This appendix describes how to perform parallel processing of a Fortran program using Fujitsu OpenMP libraries.

Notes of This Manual

The code examples of optimization in this manual are conceptual source that complements each explanation of functions. When the code examples are compiled and executed, optimizations may not work as expected. This is because optimizations depend on compilation options and other conditions.

Notation Used in This Manual

Syntax Description Symbols

A syntax description symbol is a symbol that has a specific meaning when used to describe syntax. The following symbols are used in this manual:

Symbol name	Symbol	Explanation
Selection symbols	{ }	Only one of the items enclosed in the braces must be selected (items are listed vertically).
		Multiple items are enumerated by this delimiter (items are listed horizontally).
Option symbol	[]	An item enclosed in brackets can be omitted. This symbol includes the meaning of the selection symbol "{ }".
Repeat symbol	...	The item immediately preceding the ellipsis can be specified repeatedly in the syntax.

Parallel

The way of parallelization is thread parallelization unless otherwise described.

Abbreviation

The following abbreviation is used in this manual:

Abbreviation	Explanation
OpenMP 3.1	Specification defined in "OpenMP Application Program Interface Version 3.1 July 2011"
OpenMP 4.0	Specification defined in "OpenMP Application Program Interface Version 4.0 July 2013"
OpenMP 4.5	Specification defined in "OpenMP Application Programming Interface Version 4.5 November 2015"
OpenMP 5.0	Specification defined in "OpenMP Application Programming Interface Version 5.0 November 2018"

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- Arm is trademark or registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
- OpenMP is a trademark of OpenMP Architecture Review Board.
- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- All other trademarks are the property of their respective owners.
- The trademark notice symbol (TM, (R)) is not necessarily added in the system name and the product name, etc. published in this material.

Acknowledgement

Guidance messages of the optimization is based on the result of the project with National Research and Development Agency Japan Aerospace Exploration Agency (JAXA).

Date of Publication and Version

Version	Manual code
September 2023, Version 2.11	J2UL-2473-02ENZ0(11)
March 2023, Version 2.10	J2UL-2473-02ENZ0(10)
September 2022, Version 2.9	J2UL-2473-02ENZ0(09)
March 2022, Version 2.8	J2UL-2473-02ENZ0(08)
November 2021, Version 2.7	J2UL-2473-02ENZ0(07)
July 2021, Version 2.6	J2UL-2473-02ENZ0(06)
March 2021, Version 2.5	J2UL-2473-02ENZ0(05)
January 2021, Version 2.4	J2UL-2473-02ENZ0(04)
November 2020, Version 2.3	J2UL-2473-02ENZ0(03)
September 2020, Version 2.2	J2UL-2473-02ENZ0(02)
June 2020, Version 2.1	J2UL-2473-02ENZ0(01)
March 2020, 2nd Version	J2UL-2473-02ENZ0(00)
January 2020, 1st Version	J2UL-2473-01ENZ0(00)

Copyright

Copyright FUJITSU LIMITED 2020-2023

Update History

Changes	Location	Version
Improved the explanation of the following options: - -K{ fp_relaxed nofp_relaxed } - -K{ preex nopreex } - -N{ Rtrap Rnotrap }	2.2.3	Version 2.11
Corrected description of OTHER METHODS.	4.1.3.1	
Added a note.	4.2	
Improved the explanation of the following environment variables: - OCL FISSION_POINT[(n)] - OCL FP_RELAXED - OCL NOPREEX	9.10.4	
Added a note on using THREADPRIVATE.	12.3.4.3	
Added notes on using the large page function.	2.2.3 6.5.8	Version 2.10
Corrected the table of return value during compilation.	2.5	
Corrected description of supported OpenMP specifications.	12.3.2.2	
Improved figures design.	-	
Improved the explanation of the compiler option -N{ Rtrap Rnotrap }.	2.2.3	
Corrected the explanation.	9.11	Version 2.9
Corrected the explanation.	I.1	
Improved the explanation.	-	
Added a note.	4.1.1	
Added the following optimization control specifier: - [NO]JLFUNC	9.10.4	Version 2.8
Improved the explanation of the following options: - -K{ array_declaration_opt noarray_declaration_opt } - -K{ region_extension noextension }	2.2.3	
Added the section "Intrinsic Functions that SIMD Extensions can be Applied to"	9.1.1.7.5	Version 2.7
Improved the explanation of the following environment variables: - OCL STRIPING[(n)] - OCL FISSION_POINT[(n)]	9.10.4	
Added the explanation.	9.11	
Improved "Table 12.1 List of optimization control specifiers for automatic parallelization".	12.2.6.4	
Deleted the section "Combination of Assumed-Rank and MPI Routines".	A.1	

Changes	Location	Version
Improved the explanation.	-	
Added a note.	1.2.1	Version 2.6
Deleted a note.	2.1.2	
Changed the -K{ array_declaration_opt noarray_declaration_opt } option default to the -Knoarray_declaration_opt.	2.2.3	
Added the following environment variable: - FCOMP_LINK_FJOB	2.3	
Added article on "User-Defined Derived-Type Input/Output Procedures".	7.18.1	
Added a note.	8.1.2.3	
Improved the explanation.	9.1.1.7.4 9.1.2.1 9.1.2.10	
Reviewed the explanation of "Summary of Mixed Language Programming", and changed to "Compile Commands and Required Options when Linking"	11.1	
Added the section "Combination of Assumed-Rank and MPI Routines".	A.1.8	
Added the section "Link Error (undefined reference to)".	A.2.4	
Improved the explanation.	-	
Improved the explanation of the following options: - -K{ fp_relaxed nofp_relaxed } - -K{ ilfunc[={loop procedure}] noilfunc }	2.2.3	
Added notes for specifying the -mldefault=cdecl option.	2.2.3 6.6.6 12.3.5.4	
Improved the explanation of the zfill optimization.	2.2.3 9.1.2.5 9.10.4	
Added the following environment variable: - FLIB_L1_SCCR_CNTL	9.17.2.2 I.2 J.5.4	
Added the explanation.	9.20	
Added the following option: - --linkstl={ libfjc++ libc++ libstdc++ }	2.2.1 2.2.2 2.2.3	
Added a table of commands and required options for interlanguage linking.	11.1	Version 2.4
Added the section "Linking with C++ Language Programs".	11.15	
Added a note.	12.2.2.5 12.3.2.1 J.2.2 J.3.2	
Improved the explanation.	-	
Added a note.	2.1.2	
Corrected the errors in the explanations of the -N{ reordered_variable_stack noreordered_variable_stack } option.	2.2.1 2.2.3 E.4	Version 2.3

Changes	Location	Version
Fix article for the allocatable assignment.	2.2.3 6.4.2	
Improved the explanation about the -N{ Rtrap Rnotrap } option.	2.2.3	
Corrected the errors in the explanations of detected errors.	8.1.7 8.2.2.1	
Improved the explanation about loop unrolling.	9.1.1.4	
Added the section "Integer Division Exception when the Divisor is Zero".	A.2.3	
Improved the explanation.	-	
Changed the way intrinsic functions and operations are converted with -Kmfunc option.	2.2.3	Version 2.2
Improved the explanation about data boundaries of Quadruple-precision real type and Quadruple-precision complex type.	5.2	
Improved the explanation about the optimization control specifier SIMD ALIGNED and SIMD UNALIGNED.	9.10.4 9.18	
Improved code examples.	9.10.4 9.17.2.1 9.17.2.2	
Corrected the errors in the explanations of result of atan2 when the option -Kifunc or -Kmfunc option is specified.	9.20	
Improved the explanation.	-	
Added the following options: - -K{ array_declaration_opt noarray_declaration_opt } - -K{ extract_stride_store noextract_stride_store } - -K{ fp_precision nofp_precision } - -K{ subscript_opt nosubscript_opt } - -Kswp_policy={ auto small large }	2.2.1 2.2.3	Version 2.1
Added CLONE optimization to the optimizations applied by the -O3 option.	2.2.3	
Added the following environment variable: - FLIB_TRACEBACK_MEM_SIZE	3.8	
Added jwe1655.	4.2.2 8.1.1 8.1.8	
Added the following optimization control specifiers: - [NO]ARRAY_DECLARATION_OPT - [NO]EXTRACT_STRIDE_STORE - [NO]FULLUNROLL_PRE_SIMD[(n)] - SWP_POLICY({AUTO SMALL LARGE})	9.10.4	
Added an array section to the optimization target for the PREFETCH_READ and PREFETCH_WRITE specifiers.	9.10.4	
Fix article for the kind type parameter of character type.	E.1	
Improved the explanation.	-	
Changed the supported OpenMP specifications.	1.1	2nd Version

Changes	Location	Version
Added the following option: - { -help --help }	2.1.1 2.2.1 2.2.2 2.2.3	
Added the following environment variable: - FCOMP_UNRECOGNIZED_OPTION	2.2 2.3	
Added/improved the following options: - -K{ loop_fission_stripmining[={ <i>N</i> <i>c-level</i> }] loop_nofission_stripmining } - -Kloop_fission_threshold= <i>N</i> - -K{ loop_perfect_nest loop_noperfect_nest } - -Koptions - -K{ prefetch_stride[= <i>kind</i>] prefetch_nostride } - -K{ preload nopreload } - -K{ simd_uncounted_loop simd_nouncounted_loop } - -K{ simd_use_multiple_structures simd_nouse_multiple_structures } - -Ktls_size={ 12 24 32 48 }	2.2.1 2.2.3	
Enhanced the function of the -Knoalias option.	2.2.1 2.2.3 9.7.1	
Added and changed some macros defined by default.	2.2.3	
Changed the default value of <i>kind</i> of the -Kifunc[= <i>kind</i>] option.	2.2.3	
Changed the target loops of the -Kloop_fission option.	2.2.3	
Added the loop identification number to the format of compilation diagnostic message.	4.1.1	
Added jwe1051 and jwe1052.	8.1.1 8.1.8	
Added the section "Strip-mining".	9.1.2.11.1	
Added the following optimization control specifiers: - LOOP_[NO]FISSION_STRIPMINING[({ <i>n</i> <i>c-level</i>)}] - LOOP_FISSION_TARGET[({ CL LS })] - LOOP_FISSION_THRESHOLD(<i>n</i>) - LOOP_[NO]PERFECT_NEST - PREFETCH_[NO]CONDITIONAL - PREFETCH_[NO]INDIRECT - PREFETCH_LINE(<i>n</i>) - PREFETCH_LINE_L2(<i>n</i>) - PREFETCH_NOSEQUENTIAL - PREFETCH_[NO]STRIDE[({ SOFT HARD_AUTO HARD_ALWAYS })] - [NO]PRELOAD	9.10.4	

Changes	Location	Version
<ul style="list-style-type: none"> - SCACHE_ISOLATE_ASSIGN(<i>array1</i>[,<i>array2</i>]...) END_SCACHE_ISOLATE_ASSIGN - SCACHE_ISOLATE_WAY(L2=<i>n1</i>[,L1=<i>n2</i>]) END_SCACHE_ISOLATE_WAY - SIMD_[NO]USE_MULTIPLE_STRUCTURES 		
Deleted the following optimization control specifier: <ul style="list-style-type: none"> - LOOP_[NO]FISSION 	9.10.4	
Added the section "Assumed type".	11.2.9 12.3.4.11	
Added the section "Assumed Rank".	11.2.10 12.3.4.12	
Added the section "ISO_Fortran_binding.h".	11.2.11	
Changed the default value of the environment variable OMP_PROC_BIND in LLVM OpenMP Library.	12.2.2.5 12.3.2.1 12.3.3	
Added the following environment variable: <ul style="list-style-type: none"> - OMP_MAX_TASK_PRIORITY 	12.3.2.2	
Added the following items: <ul style="list-style-type: none"> - TASKLOOP construct - CRITICAL construct 	12.3.3	
Added a note to "Runtime Library Definitions" for specifying the compiler option - CcdI18, -CcI4I8, -CcdLL8, -CcL4L8, -Ccd4d8, or -Cca4a8.	12.3.3	
Added information about when an error occurred.	12.3.5.2	
Added the section "Notes".	A.2	
Added the appendix "GNU Compatible Options".	Appendix D	
Added intrinsic procedures compatible with the half-precision type (CSHIFT, EOSHIFT, MAXVAL, MINVAL, NORM2, PACK, PRODUCT, RESHAPE, SPACING, SPREAD, SUM, TRANSFER, UNPACK).	F.1	
Added the appendix "Using High-speed Facility on Job Operating Software".	Appendix I	
Improved the explanation.	-	
Changed the look according to product upgrades.	-	

All rights reserved.
The information in this manual is subject to change without notice.

Contents

Chapter 1 Overview of the Fortran System.....	1
1.1 Configuration of the Fortran System.....	1
1.2 How to Use the Fortran System.....	2
1.2.1 Preparation.....	2
1.2.2 Compilation and Linking.....	2
1.2.3 File Names.....	3
1.2.4 Some Compilation Options.....	3
1.2.5 Execution.....	3
1.2.6 Debugging.....	4
1.2.7 Tuning.....	4
Chapter 2 Compiling and Linking Fortran Programs.....	5
2.1 Compile Command.....	5
2.1.1 Compile Command Format.....	5
2.1.2 Compile Command Input File.....	5
2.2 Compiler Options.....	6
2.2.1 List of Compiler Options.....	6
2.2.2 Syntax of Compiler Options.....	16
2.2.3 Description of Compiler Options.....	16
2.3 Compile Command Environment Variable.....	72
2.4 Compilation Profile File.....	74
2.5 Return Values during Compilation.....	75
2.6 Compiler Directive Lines.....	75
Chapter 3 Executing Fortran Programs.....	76
3.1 Execution Command.....	76
3.2 Execution Command Format.....	76
3.3 Runtime Options.....	76
3.4 Execution Command Environment Variable.....	81
3.5 Execution Profile File.....	81
3.6 Execution Command Return Values.....	82
3.7 Standard Input, Output, and Error Output for Execution.....	82
3.8 Using Environment Variables for Execution.....	82
Chapter 4 Messages and Listings.....	85
4.1 Compilation Output.....	85
4.1.1 Compilation Diagnostic Messages.....	85
4.1.2 Guidance Messages during Compilation.....	86
4.1.3 Compilation Information.....	87
4.1.3.1 Compilation Information Options.....	87
4.1.3.2 Example of Compilation Information.....	93
4.1.3.3 Notes on Compilation Information.....	102
4.2 Executable Program Output.....	103
4.2.1 Diagnostic Messages.....	103
4.2.2 Trace Back Map.....	104
4.2.3 Error Summary.....	105
Chapter 5 Data Types, Kinds, and Representations.....	107
5.1 Internal Data Representation.....	107
5.1.1 Integer Type Data.....	107
5.1.2 Logical Type Data.....	107
5.1.3 Real and Complex Type Data.....	107
5.1.4 Character Type Data.....	111
5.1.5 Derived Type Data.....	111
5.2 Correct Data Boundaries.....	111
5.3 Precision Conversion.....	112

5.3.1 Precision Improving.....	112
5.3.1.1 Precision-Raising Options.....	112
5.3.2 Precision Lowering and Analysis of Errors.....	113
5.3.2.1 Defining Precision-Lowering Function.....	114
5.3.2.2 Using Precision-Lowering Function.....	114
Chapter 6 Notes on Programming.....	117
6.1 Fortran Versions.....	117
6.2 Data.....	119
6.2.1 COMMON Statement.....	119
6.2.1.1 Boundaries of Variables in Common Blocks.....	119
6.2.1.2 Common Blocks that Have Initial Values.....	119
6.2.2 EQUIVALENCE Statement.....	120
6.2.3 Initialization.....	120
6.2.3.1 Character Initialization.....	120
6.2.3.2 Initialization with Hexadecimal, Octal, and Binary Constants.....	121
6.2.3.3 Overlapping Initialization.....	121
6.2.4 Using CRAY Pointers.....	121
6.2.5 SAVE Attribute for Initialized Variables.....	122
6.2.6 Operation of Real or Complex Type for Constant Expression.....	122
6.3 Expressions.....	122
6.3.1 Integer Data Operations.....	122
6.3.2 Real Data Operations.....	123
6.3.3 Logical Data Operations.....	123
6.3.4 Order of Evaluation of Data Entities.....	123
6.3.5 Evaluation of a Part of Expression.....	124
6.3.6 Definition and Undefined by Function Reference.....	124
6.3.7 Variable Enclosed by Parenthesis.....	124
6.4 Assignment Statement.....	124
6.4.1 Assignment Statements with Overlapping Character Positions.....	124
6.4.2 Allocatable Assignment.....	125
6.5 Control Statements.....	125
6.5.1 GO TO Statement.....	125
6.5.2 Arithmetic IF Statement.....	125
6.5.3 Logical IF Statement.....	125
6.5.4 DO Statement.....	126
6.5.4.1 DO Loop Iteration Count Method.....	126
6.5.4.2 DO CONCURRENT.....	126
6.5.5 CASE Statement.....	126
6.5.6 PAUSE Statement.....	126
6.5.7 STOP/ERROR STOP Statement.....	127
6.5.8 ALLOCATE/DEALLOCATE Statement.....	128
6.5.9 Simple Output List Enclosed in Parentheses.....	129
6.5.10 Output Statement that Starts by TYPE Keyword.....	129
6.6 Procedures.....	129
6.6.1 Statement Function Reference.....	129
6.6.2 An Intrinsic Function that Specifies EXTERNAL Attribute.....	130
6.6.3 REAL and CMPLX Used as Generic Names.....	130
6.6.4 Intrinsic Procedures.....	130
6.6.5 Intrinsic Function Names with Type Declared.....	130
6.6.6 Service Routines.....	131
6.6.7 Returned Values of Intrinsic Function.....	131
6.6.8 Precision of Intrinsic Function.....	131
6.6.9 EXECUTE_COMMAND_LINE Intrinsic Subroutine.....	131
Chapter 7 Input/Output Processing.....	133
7.1 Files.....	133
7.1.1 Old and New Files.....	133

7.1.2 File Connection.....	133
7.1.2.1 Connection.....	133
7.1.2.2 File Disconnection.....	133
7.2 Unit Numbers and File Connection.....	134
7.2.1 Priority of Unit and File Connection.....	134
7.2.2 Environment Variables to Connect Units and Files.....	135
7.3 Fortran Records.....	135
7.3.1 Formatted Fortran Records.....	136
7.3.1.1 Formatted Sequential Record.....	137
7.3.1.2 Formatted Direct Record.....	137
7.3.1.3 Internal File Record.....	138
7.3.2 Unformatted Fortran Records.....	138
7.3.2.1 Unformatted Sequential Record.....	138
7.3.2.2 Unformatted Direct Record.....	140
7.3.3 List-Directed Fortran Records.....	141
7.3.4 Namelist Fortran Records.....	141
7.3.5 Endfile Records.....	141
7.3.6 Binary Fortran Records.....	141
7.3.7 STREAM Fortran Records.....	141
7.3.7.1 Formatted Stream Record.....	141
7.3.7.2 Unformatted Stream Record.....	142
7.4 OPEN Statement.....	142
7.4.1 OPEN Statement Specifiers.....	142
7.4.1.1 FILE= Specifier.....	143
7.4.1.2 STATUS= Specifier.....	143
7.4.1.3 RECL= Specifier.....	144
7.4.1.4 ACTION= Specifier.....	145
7.4.1.5 BLOCKSIZE= Specifier.....	146
7.4.1.6 CONVERT= Specifier.....	146
7.4.1.7 NEWUNIT= Specifier.....	146
7.4.2 Executing an OPEN Statement on a Connected File.....	146
7.4.3 Input/Output of Fortran Records.....	147
7.4.4 Default Value of ACCESS= Specifier with RECL= Specifier.....	148
7.5 CLOSE Statement STATUS= Specifier.....	148
7.5.1 STATUS= Specifier.....	148
7.6 INQUIRE Statement.....	149
7.6.1 INQUIRE Statement Parameters.....	149
7.6.2 Difference in Language Version of INQUIRE Statement.....	153
7.6.2.1 Return Value of ACTION= Specifier.....	153
7.6.2.2 Char-Constant Returned by Inquiry Specifiers of INQUIRE Statement.....	153
7.6.2.3 Return Value of PAD= Specifier.....	154
7.6.2.4 Return Value of Inquire by File of INQUIRE Statement.....	154
7.6.2.5 Return Value of Inquire by Unit of INQUIRE Statement.....	154
7.7 Data Transfer Input/Output Statements.....	154
7.7.1 Control Information for Input/Output Data Transfer Statements.....	154
7.7.1.1 Input/Output UNIT Specifier.....	154
7.7.1.2 Format Specifier.....	155
7.7.1.3 IOSTAT= Specifier.....	155
7.7.1.4 Error Branch.....	156
7.7.1.5 End-of-File Branch.....	156
7.7.1.6 End-of-Record Branch.....	157
7.7.1.7 IOMSG= Specifier.....	157
7.7.2 Sequential Access Input/Output Statements.....	157
7.7.2.1 Formatted Sequential Access Input/Output Statements.....	157
7.7.2.2 Unformatted Sequential Access Input/Output Statements.....	157
7.7.3 Direct Access Input/Output Statements.....	158
7.7.3.1 Direct Access READ Statement.....	158

7.7.3.2 Direct Access WRITE Statement.....	158
7.7.3.3 Error of Data Transfer in a DIRECT Access Input/Output.....	158
7.7.4 Namelist Input/Output Statements.....	159
7.7.4.1 Namelist READ Statements.....	159
7.7.4.1.1 Namelist Data in Namelist Input.....	161
7.7.4.2 Namelist WRITE Statements.....	162
7.7.4.3 NAMELIST Input/Output Version Differences.....	163
7.7.5 List-Directed Input/Output Statements.....	166
7.7.5.1 List-Directed READ Statements.....	166
7.7.5.2 List-Directed WRITE Statements.....	168
7.7.5.2.1 Form of Nondelimited Character Constants Produced by List-Directed Output.....	169
7.7.5.3 List-Directed Input/Output Version Differences.....	170
7.7.6 Internal File Input/Output Statements.....	170
7.7.7 Nonadvancing Input/Output Statements.....	170
7.8 Combining Input/Output Statements.....	171
7.8.1 Combinations of Input/Output Statements Not Allowed.....	174
7.8.1.1 Formatted Sequential Access Input/Output Statements.....	174
7.8.1.2 Unformatted Sequential Access Input/Output Statements.....	174
7.8.1.3 Direct Access Input/Output Statements.....	174
7.8.1.4 Stream Access Input/Output Statements.....	174
7.8.1.5 File Positioning Statements.....	175
7.9 File Positioning Statements.....	175
7.9.1 BACKSPACE Statement.....	175
7.9.2 ENDFILE Statement.....	176
7.9.3 REWIND Statement.....	179
7.10 FLUSH Statement.....	179
7.11 Conversion between Big Endian Data and Little Endian Data on Input/Output.....	179
7.11.1 Relationship between Big Endian Data and Input/Output Statements.....	179
7.12 Standard Files.....	180
7.12.1 Standard Files and Open Modes.....	180
7.12.2 Restrictions on Input/Output Statements for Standard Files.....	180
7.12.3 Relationship between Input/Output Statements and Seekable and Nonseekable Files.....	181
7.13 Restrictions Related to ACTION= Specifier.....	181
7.14 Number of Significant Digits.....	181
7.15 I/O Buffer.....	182
7.16 Edit Descriptors.....	182
7.16.1 Generalized Integer Editing.....	182
7.16.2 Exponent Character in Formatted Output.....	182
7.16.3 Result of G Editing.....	182
7.16.4 Diagnostic Messages for Character String Edit Descriptor.....	183
7.16.5 Effect of X Edit Descriptor.....	183
7.16.6 L Edit Descriptor.....	183
7.16.7 I, L, F, E, D, G, B, O, and Z Edit Descriptor without w or d Indicators.....	184
7.16.8 Editing of Special Real-Type Data.....	184
7.17 Asynchronous Input/Output Statements.....	185
7.17.1 Execution of Asynchronous Input/Output Statements.....	185
7.17.2 ID= Specifier in Data Transfer Input/Output Statements.....	185
7.18 User-Defined Derived-Type Input/Output.....	186
7.18.1 User-Defined Derived-Type Input/Output Procedures.....	186
7.18.2 Data Transfer Input/Output List.....	186
7.18.3 Namelist Input/Output.....	187
7.18.4 INQUIRE Statement.....	187
Chapter 8 Debugging.....	188
8.1 Error Processing.....	188
8.1.1 Error Monitor.....	188
8.1.2 Error Subroutines.....	194

8.1.2.1 ERRSAV Subroutine.....	194
8.1.2.2 ERRSTR Subroutine.....	195
8.1.2.3 ERRSET Subroutine.....	195
8.1.2.4 ERRTRA Subroutine.....	196
8.1.3 Using Error Monitor.....	197
8.1.3.1 User-Defined Error Subroutines.....	197
8.1.3.2 Handling of Incorrect Arguments in Intrinsic Functions.....	197
8.1.3.3 User Error Processing.....	198
8.1.4 Input/Output Error Processing.....	198
8.1.5 Intrinsic Function Error Processing.....	201
8.1.6 Intrinsic Subroutine Error Processing.....	219
8.1.7 Exception Handling Processing.....	221
8.1.8 Other Error Processing.....	222
8.2 Debugging Functions.....	224
8.2.1 Debugging Check Functions.....	224
8.2.1.1 Checking Argument Validity (ARGCHK).....	226
8.2.1.1.1 Checking Number of Arguments.....	226
8.2.1.1.2 Checking Argument Types.....	227
8.2.1.1.3 Checking Argument Attributes.....	227
8.2.1.1.4 Checking Function Types.....	227
8.2.1.1.5 Checking Argument Sizes.....	228
8.2.1.1.6 Checking Explicit Interface.....	228
8.2.1.1.7 Checking Rank of Assumed-Shape Array.....	228
8.2.1.2 Checking Subscript and Substring Values (SUBCHK).....	228
8.2.1.2.1 Checking Array and Substring References.....	229
8.2.1.2.2 Checking Array Section References.....	229
8.2.1.3 Checking References for Undefined Data Items (UNDEF).....	229
8.2.1.3.1 Checking for Undefined Data.....	229
8.2.1.3.2 Checking for Undefined Data Using Invalid Operation Exception.....	230
8.2.1.3.3 Checking for an Allocatable Variable.....	231
8.2.1.3.4 Checking for an Optional Argument.....	231
8.2.1.4 Shape Conformance Check (SHAPECHK).....	231
8.2.1.5 Extended Checking of UNDEF (EXTCHK).....	231
8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP).....	232
8.2.1.6.1 Overlapping Dummy Arguments.....	232
8.2.1.6.2 Undefined Assumed-Size Array with INTENT(OUT).....	233
8.2.1.6.3 Undefined Variable of SAVE Attribute.....	233
8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN).....	233
8.2.1.7.1 Checking Connection of a File to a Unit.....	233
8.2.1.7.2 I/O Recursive Call CHECK.....	234
8.2.2 Debugging Programs for Abend.....	234
8.2.2.1 Causes of Abend.....	234
8.2.2.2 Information Generated when an Abend Occurs.....	235
8.2.2.2.1 Information Generated for a General Abend.....	235
8.2.2.2.2 Information of SIGXCPU.....	235
8.2.2.2.3 Information Generated by RunTime Option -a.....	235
8.2.2.2.4 Information when an Abend Occurs Again during Abend Processing.....	235
8.2.3 Hook Function.....	235
8.2.3.1 User-Defined Subroutine Format.....	235
8.2.3.2 Notes on Hook Function.....	236
8.2.3.3 Calling a User-Defined Subroutine from a Specified Location.....	237
8.2.3.3.1 Notes on Calling from a Specified Location.....	238
8.2.3.4 Calling a User-Defined Function at Regular Time Interval.....	238
8.2.3.4.1 Notes on Calling by Regular Time Interval.....	238
8.2.3.5 Calling from Any Location in the Program.....	239
Chapter 9 Optimization Functions.....	240

9.1 Overview of Optimization.....	240
9.1.1 Standard Optimization.....	240
9.1.1.1 Elimination of Common Expression.....	240
9.1.1.2 Movement of Invariant Expressions.....	240
9.1.1.3 Reducing Strength of Operators.....	241
9.1.1.4 Loop Unrolling.....	241
9.1.1.5 Loop Blocking.....	242
9.1.1.6 Software Pipelining.....	242
9.1.1.7 SIMD.....	243
9.1.1.7.1 Normal SIMD.....	243
9.1.1.7.2 SIMD Extensions for Loop Containing IF Construct.....	243
9.1.1.7.3 List Vector Conversion.....	243
9.1.1.7.4 SIMD with Redundant Executions for the SIMD Width.....	244
9.1.1.7.5 Intrinsic Functions that SIMD Extensions can be Applied to.....	245
9.1.1.8 Loop Unswitching.....	246
9.1.2 Extended Optimization.....	246
9.1.2.1 Inline Expansion.....	246
9.1.2.2 Advance evaluation of Invariant Expressions.....	246
9.1.2.3 Optimization by Modifying Evaluation Methods.....	248
9.1.2.4 Loop Striping.....	248
9.1.2.5 zfill.....	249
9.1.2.6 Loop Versioning.....	249
9.1.2.7 CLONE Optimization.....	250
9.1.2.8 Link Time Optimization.....	250
9.1.2.8.1 Overview of Link Time Optimization.....	250
9.1.2.8.2 Notes on Link Time Optimization.....	251
9.1.2.9 Unroll-and-Jam.....	251
9.1.2.10 Tree-Height-Reduction Optimization.....	252
9.1.2.11 Loop Fission.....	253
9.1.2.11.1 Strip-Mining.....	253
9.2 Notes of Wrong Erroneous Program.....	254
9.3 Effects of Environmental Change on Optimization.....	255
9.4 Effects of Changing Execution Order.....	255
9.5 Branching Target of Assigned GO TO Statements.....	255
9.6 Effect of Dummy Arguments.....	255
9.7 Restrictions on Inline Expansion.....	258
9.7.1 Restrictions on Called User-Defined Procedures.....	258
9.7.2 Restrictions on Relationship between Calling and Called Program Units.....	259
9.7.3 Restrictions on User-Defined External Procedure Names.....	260
9.7.4 Restrictions on Option.....	260
9.7.5 Restrictions on Elemental Procedures.....	261
9.8 Effects of Cray Pointer Variables.....	261
9.9 Effects of Compiler Option -Kcommonpad[=N].....	261
9.10 Using Optimization Control line (OCL).....	262
9.10.1 Notation Rules for Optimization Control Lines.....	262
9.10.2 Description Position of Optimization Control Line.....	262
9.10.3 Wild Card Specification.....	263
9.10.4 Optimization Control Specifiers.....	263
9.11 Effects of Allocating on Stack.....	307
9.12 Effects of Applying Optimization to Change Shape of Array.....	308
9.13 Effects of Generating Prefetch on Program Performance.....	309
9.14 Dimension Shift of Array Declaration.....	309
9.14.1 Effects of Dimension Shift of Array Declaration.....	309
9.14.2 Restrictions of Target Variable of Dimension Shift of Array Declaration.....	310
9.14.3 Restriction for Ancestor Module of Submodule.....	310
9.15 Merging Array Variables.....	310
9.15.1 Effects of Merging Array Variables.....	310

9.15.2 Restrictions of Merging Local Array Variables.....	312
9.15.3 Restrictions of Target Variables on Common Block Object Array Merge.....	312
9.16 Multi-Operation Function.....	312
9.16.1 Calling of Multi-Operation Function.....	312
9.16.2 Effects of Compiler Option -Kmfunc=3.....	315
9.17 Software Control of Sector Cache.....	315
9.17.1 Using Sector Cache.....	315
9.17.2 How to Control Sector Cache by Software.....	315
9.17.2.1 Software Control with Optimization Control Lines.....	316
9.17.2.2 Software Control with Environment Variables and Optimization Control Lines.....	317
9.17.2.3 Behavior when an Invalid Value Is Specified.....	318
9.18 Incorrectly Specified Optimization Indicators.....	318
9.18.1 Example of NOVREC Specifier.....	318
9.18.2 Example of CLONE Specifier.....	318
9.18.3 Example of UNROLL_AND_JAM_FORCE Specifier.....	319
9.18.4 Example of ITERATIONS Specifier.....	319
9.19 Attentions for Mixed Language Programming.....	319
9.19.1 Address Parameters Received in C language.....	320
9.19.2 Pointers Received in the C language.....	321
9.20 Side Effect of Optimizations for Floating-Point Operation.....	321
9.21 Notes on Specified SVE Vector Register Size.....	323
Chapter 10 Fortran Modules and Submodules.....	324
10.1 Compilation of Modules and Module References.....	324
10.2 Recompiling a Program that Contains a Reference to a Module Information that Has Changed.....	326
10.3 Restrictions on Modules.....	326
10.4 Submodule.....	327
10.5 Intrinsic Modules.....	327
10.5.1 COMPILER_OPTIONS Intrinsic Module Function.....	328
10.5.2 COMPILER_VERSION Intrinsic Module Function.....	328
Chapter 11 Mixed Language Programming.....	329
11.1 Compile Commands and Required Options when Linking.....	329
11.1.1 Linking C++ Trad Mode with C++ Clang Mode.....	333
11.1.2 Linking C++ Clang Mode with C++ Clang Mode.....	334
11.1.3 Linking MPI Object Programs and Use Profiler.....	334
11.1.4 Objects Generated in Clang Mode with -flto Option.....	334
11.2 Features for Mixed Language Programming.....	334
11.2.1 Passing Arguments by Value.....	335
11.2.1.1 Passing Arguments by Value in Procedure Interface.....	335
11.2.1.2 Passing Arguments by Value in Intrinsic Function.....	335
11.2.2 Getting Arguments by Value.....	335
11.2.3 CHANGEENTRY Statement.....	335
11.2.4 \$pragma Specifier.....	336
11.2.5 Intrinsic Module ISO_C_BINDING.....	336
11.2.5.1 Named Constants by Intrinsic Module ISO_C_BINDING.....	336
11.2.5.2 Types by Intrinsic Module ISO_C_BINDING.....	338
11.2.5.3 Intrinsic Procedures by Intrinsic Module ISO_C_BINDING.....	338
11.2.6 BIND Statement, BIND Attribute Specifier, Language Binding Specifier and Procedure Language Binding Specifier.....	339
11.2.6.1 BIND Statement.....	340
11.2.6.2 BIND Attribute by Attribute Specifier of Type Declaration Statement.....	341
11.2.6.3 Procedure Language Binding Specifier.....	341
11.2.7 VALUE Statement, VALUE Attribute Specifier.....	342
11.2.8 Interoperability of Derive Type and C Language Structure Type.....	343
11.2.9 Assumed Type.....	343
11.2.10 Assumed Rank.....	344
11.2.11 ISO_Fortran_binding.h.....	346
11.2.11.1 CFI_cdesc_t Structure Type.....	346

11.2.11.2 CFI_dim_t Structure Type.....	347
11.2.11.3 Type and Macro Names in ISO_Fortran_binding.h.....	347
11.3 Rules for Processing Fortran Procedure Names.....	348
11.4 Correspondence of Type with C.....	349
11.5 Calling Procedures.....	350
11.5.1 Passing Control First to a C Program.....	351
11.5.2 Calling Standard C Libraries.....	352
11.5.2.1 Method of Specification of Procedure Language Binding Specifier.....	352
11.5.2.2 Method of No Specification of Procedure Language Binding Specifier.....	353
11.6 Passing and Receiving Function Values.....	353
11.6.1 Passing Fortran Function Values to C without Procedure Language Binding Specifier.....	353
11.6.2 Receiving Function Values from C without Procedure Language Binding Specifier.....	355
11.7 Passing and Getting Arguments.....	357
11.7.1 Passing Arguments from Fortran with Procedure Language Binding Specifier to C.....	357
11.7.2 Passing Arguments from C to Fortran without Procedure Language Binding Specifier.....	357
11.8 Passing Using External Variables.....	358
11.8.1 Passing External Variables with BIND Attribute.....	358
11.8.2 Passing External Variables without BIND Attribute.....	359
11.9 Passing Using Files.....	360
11.10 Array Storage Sequence.....	360
11.11 Passing a Character String to a C Program.....	360
11.12 Return Value of Executable Program.....	361
11.13 Notes of Compiler Option -Az.....	362
11.14 Restrictions on Linking with C Language Programs.....	362
11.15 Linking with C++ Language Programs.....	363
Chapter 12 Multiprocessing.....	365
12.1 Overview of Multiprocessing.....	365
12.1.1 Parallel Processing.....	365
12.1.2 Effect of Multiprocessing.....	366
12.1.3 Requirements for Effective Multiprocessing.....	366
12.1.4 Features of Parallel Processing in This Compiler.....	366
12.2 Automatic Parallelization.....	367
12.2.1 Compilation.....	367
12.2.1.1 Compiler Option for Automatic Parallelization.....	367
12.2.2 Execution Process.....	367
12.2.2.1 Number of Threads.....	367
12.2.2.2 Stack Size on Execution.....	367
12.2.2.3 Synchronization Process.....	367
12.2.2.4 Notes when Using Service Function and Intrinsic Subroutine.....	368
12.2.2.5 CPU Binding for Thread.....	368
12.2.3 Example of Compilation and Execution.....	371
12.2.4 Tuning of Parallelized Programs.....	371
12.2.5 Feature Details on Automatic Parallelization.....	371
12.2.5.1 Targets for Automatic Parallelization.....	371
12.2.5.2 What is Loop Slicing?.....	371
12.2.5.3 Array Operations and Automatic Parallelization.....	372
12.2.5.4 Automatic Loop Slicing by Compiler.....	372
12.2.5.5 Loop Interchange and Automatic Loop Slicing.....	372
12.2.5.6 Loop Distribution and Automatic Loop Slicing.....	373
12.2.5.7 Loop Fusion and Automatic Loop Slicing.....	373
12.2.5.8 Loop Reduction.....	374
12.2.5.9 Restrictions on Loop Slicing.....	375
12.2.5.10 Displaying the State of Automatic Parallelization.....	377
12.2.5.11 Extension of Parallel Region.....	377
12.2.5.12 Block Distribution and Cyclic Distribution.....	378
12.2.6 Optimization Control Line.....	379

12.2.6.1 Notation Rules for Optimization Control Lines.....	379
12.2.6.2 Description Position of Optimization Control Line.....	379
12.2.6.3 Automatic Parallelization and Optimization Control Specifiers.....	379
12.2.6.4 Optimization Control Specifiers for Automatic Parallelization.....	379
12.2.6.5 Wild Card Specification.....	395
12.2.7 Notes on Automatic Parallelization.....	395
12.2.7.1 Caution when Specifying Compile-Time Option -Kparallel,instance=N.....	395
12.2.7.2 Nested Parallel Processing.....	395
12.2.7.3 Caution when Specifying Compile-Time Option -Kparallel,reduction.....	396
12.2.7.4 Notes on Using Optimization Control Line.....	397
12.2.7.5 I/O Statement and Intrinsic Procedure Call on Parallel Processing.....	398
12.3 Parallelization by OpenMP Specifications.....	398
12.3.1 Compilation.....	398
12.3.1.1 Compiler Options for OpenMP Programs.....	399
12.3.1.2 Compiler Options for Obtaining Optimization Information.....	399
12.3.2 Execution Process.....	399
12.3.2.1 Environment Variable at Execution.....	399
12.3.2.2 Environment Variable for OpenMP Specifications.....	402
12.3.2.3 Notes during Execution.....	403
12.3.3 Implementation-Dependent Specifications.....	404
12.3.4 Clarifying OpenMP Specifications and Restrictions.....	408
12.3.4.1 Specifying Label with ASSIGN Statement and Assigned GO TO Statement.....	408
12.3.4.2 Additional Functions in ATOMIC Construct and Reduction Identifier.....	408
12.3.4.3 Notes on Using THREADPRIVATE.....	408
12.3.4.4 Inline Expansion.....	408
12.3.4.5 Internal Procedure Calling from PARALLEL Region.....	408
12.3.4.6 Polymorphic Variable.....	409
12.3.4.7 OPTIONAL Attribute.....	409
12.3.4.8 ASSOCIATE NAME.....	409
12.3.4.9 Selector.....	409
12.3.4.10 Derived Type Variable with Length Type Pparameter.....	409
12.3.4.11 Assumed Type Variable.....	410
12.3.4.12 Assumed Rank Variable.....	410
12.3.5 Notes on OpenMP Programming.....	410
12.3.5.1 Implementation of PARALLEL Region and Explicit TASK Region.....	410
12.3.5.2 Implementation of THREADPRIVATE Variable.....	411
12.3.5.3 Automatic Parallelization for OpenMP Programs.....	411
12.3.5.4 Runtime Library Routines.....	411
12.3.6 Debugging OpenMP Programs.....	412
12.4 Runtime Messages.....	412
Appendix A Restrictions and Notes.....	413
A.1 Restrictions.....	413
A.1.1 Nesting Level of Functions, Array Sections, Array Elements, and Substring References.....	413
A.1.2 DO, CASE, IF, SELECT TYPE, BLOCK, CRITICAL, and ASSOCIATE Constructs.....	413
A.1.3 Implied DO-Loops.....	413
A.1.4 INCLUDE Line.....	413
A.1.5 Array Declarations.....	414
A.1.6 Array Section.....	415
A.1.7 Distance between a Branch Instruction and Branch Destination Instruction.....	415
A.2 Notes.....	415
A.2.1 Debug Using Debugger.....	415
A.2.2 Changing SVE Vector Register Size.....	416
A.2.3 Integer Division Exception when Divisor Is Zero.....	416
A.2.4 Link Error (undefined reference to).....	416
Appendix B Endian Convert Command.....	417
B.1 Command Format.....	417

B.2 Return Value of fcvendian(1) and fcvendianpx(1).....	417
B.3 Diagnostic Messages of fcvendian(1) and fcvendianpx(1).....	417
B.4 Note.....	418
Appendix C Preprocessor.....	419
C.1 C Language Preprocessor.....	419
C.2 Fortran Preprocessor.....	419
C.2.1 Continuation Line of Fortran.....	419
C.2.2 Comment Line of Fortran.....	420
C.2.3 Comment of Fortran and Comment of C Language.....	420
C.2.4 Macro Substitution in Fortran Comment Construct.....	420
C.2.5 Treatment of Characters Specified after Column 72 in Fixed Source Form.....	420
Appendix D GNU Compatible Options.....	422
Appendix E Data and Memory Regions.....	425
E.1 Data Size and Alignment.....	425
E.2 Memory Regions.....	425
E.3 Data Allocation.....	426
E.4 Data Allocation to Stack Region.....	426
Appendix F Intrinsic Procedures and I/O Statements Compatible with Half-Precision Type.....	429
F.1 Intrinsic Procedures.....	429
F.2 Input/Output Statements.....	430
Appendix G Code Coverage.....	431
G.1 How to Use Code Coverage.....	431
G.2 Necessary Files to Use Code Coverage.....	432
G.2.1 The ".gcno" File.....	432
G.2.2 The ".gcode" File.....	433
G.3 Notes on Code Coverage.....	434
Appendix H Runtime Information Output Function.....	435
H.1 Usage of Runtime Information Output Function.....	435
H.1.1 Range of Obtainable Information.....	435
H.1.2 Runtime Environment Variables.....	436
H.2 Output of Runtime Information Output Function.....	436
H.2.1 Output Information.....	436
H.2.2 Output Format.....	437
H.2.3 Output Example.....	437
H.3 Notes on Runtime Information Output Function.....	439
Appendix I Using High-Speed Facility on Job Operation Software.....	441
I.1 Inter-Core Hardware Barrier.....	441
I.1.1 Compilation.....	441
I.1.2 Execution.....	441
I.2 Sector Cache.....	442
Appendix J Fujitsu OpenMP Library.....	444
J.1 Overview of Multiprocessing.....	444
J.2 Automatic Parallelization.....	444
J.2.1 Compilation (Automatic Parallelization).....	444
J.2.2 Execution Process (Automatic Parallelization).....	445
J.2.3 Example of Compilation and Execution.....	448
J.2.4 Tuning of Parallelized Programs.....	448
J.2.5 Automatic Parallelization.....	448
J.2.6 Optimization Control Line.....	448
J.2.7 Notes on Automatic Parallelization.....	448
J.2.8 Linkage with Other Multi-Thread Programs.....	449

J.3 Parallelization by OpenMP Specification.....	450
J.3.1 Compilation (OpenMP Parallelization).....	450
J.3.2 Execution Process (OpenMP Parallelization).....	451
J.3.3 Implementation-Dependent Specifications.....	456
J.3.4 Clarifying OpenMP Specifications and Restrictions.....	460
J.3.5 Notes on OpenMP Programming.....	460
J.3.6 Linking with Other Multi-Thread Programs.....	460
J.3.7 Debugging OpenMP Programs.....	461
J.4 I/O Buffer Parallel Transfer.....	461
J.4.1 Compilation.....	461
J.4.2 Execution Process.....	461
J.4.3 Conditions to Perform I/O Buffer Parallel Transfer.....	462
J.4.4 Notes on I/O Buffer Parallel Transfer.....	462
J.4.5 Example.....	462
J.5 Using High-Speed Facility on Job Operation Software.....	463
J.5.1 Management of CPU Resources.....	463
J.5.2 CPU Binding.....	464
J.5.3 Inter-Core Hardware Barrier.....	465
J.5.4 Sector Cache.....	466
Index.....	468

Chapter 1 Overview of the Fortran System

This chapter provides an overview of this system (or simply "the system") and illustrates the most common ways to use the system to develop and run a Fortran program.

1.1 Configuration of the Fortran System

This system is one of language processing programs. This system consists of the following components:

Compile command (Fortran driver)

The compile command is a program which calls the Fortran compiler, the preprocessor, the assembler, and the linker. There are two types of compile commands in this system:

Kind	Compile command	Description
Cross compiler	ftrpx	The command used on the login node.
Native compiler	ftr	The command used on the compute node.

In this manual, the ftrpx is used as the name of the compile command. Read the ftrpx as the ftr when you use the native compiler.

Fortran compiler

The compile command calls the Fortran compiler. The Fortran compiler creates object programs from Fortran source programs.

Fortran library

The object program calls the Fortran library to execute the Fortran program. The Fortran library provides commonly used start-up routines, input/output facilities, intrinsic procedures, service routines, and multiprocessing.

This system provides two libraries for multiprocessing:

Library Name	Description
LLVM OpenMP Library	This is the library for multiprocessing based on LLVM OpenMP Runtime Library which is an open source software. Supported specifications are OpenMP 4.5/Part of OpenMP 5.0. For specifications of LLVM OpenMP Library, see " Chapter 12 Multiprocessing ".
Fujitsu OpenMP Library	This is the library for multiprocessing based on Fujitsu OpenMP Library for K computer and PRIMEHPC FX100 or earlier system. This library is suitable for maintaining compatibility with the past Fujitsu OpenMP Library. Supported specifications are OpenMP 3.1/Part of OpenMP 4.5. For specifications of Fujitsu OpenMP Library, see " Appendix J Fujitsu OpenMP Library ".

Endian convert command

Endian convert command fcvendian/fcvendianpx are included in the system.

In this manual, the fcvendianpx is used as the name of the endian convert command. Read the fcvendianpx as the fcvendian when you use the native compiler.

Online manual

Online manual pages, accessed by the man command, provide information about ftrpx(1), ftr(1), fcvendian(1), fcvendianpx(1), intrinsic(3) and service(3).

1.2 How to Use the Fortran System

1.2.1 Preparation

To use the system, following environmental variables must be set correctly. "*installation_path*" is the product/software installation path. For "*installation_path*", contact the system administrator.

Environment variable	Value
PATH	<i>/installation_path/bin</i>
LD_LIBRARY_PATH	<i>/installation_path/lib64</i>
MANPATH	<i>/installation_path/man</i>

Examples:

```
$ PATH=/installation_path/bin:$PATH ; export PATH
$ LD_LIBRARY_PATH=/installation_path/lib64:$LD_LIBRARY_PATH ; export LD_LIBRARY_PATH
$ MANPATH=/installation_path/man:$MANPATH ; export MANPATH
```



When setting the environment variable LANG with a locale, ensure that the locale is installed on the node where the program is compiled.

A list of available locales can be found with the "locale -a" command.

Example:

```
$ locale -a
C
:
```

If the locale is not installed, a warning message may be output or a compilation error may occur. In this case, take one of the following actions:

- Specify C for the environment variable LANG
- Ask the system administrator to install the locale

The current locale can be found with the "locale" command.

Example:

```
$ locale
LANG=C
:
```

1.2.2 Compilation and Linking

The frtpx command is used for compilation and linking.

The simplest case is to compile a Fortran program, stored in a file such as prog.f95. The command

```
$ frtpx prog.f95
```

compiles and links the program, producing an executable file named a.out. Specifying -c option causes only object file to be made, without linking.

```
$ frtpx -c prog.f95
```

1.2.3 File Names

The `frtpx` command may specify source files, object files, or assembly language files. Source files may have a `.f90`, `.f95`, `.f`, `.for`, `.f03`, `.f08`, `.F90`, `.F95`, `.F`, `.FOR`, `.F03`, or `.F08` suffix. If it is `.f90`, `.f95`, `.f03`, or `.f08`, the statements are assumed to be in free source form; if it is `.f` or `.for`, the statements are assumed to be in fixed source form; in either case, this assumption may be overridden with the `-Free` or `-Fixed` compiler option. If the suffix is `.F90`, `.F95`, `.F`, `.FOR`, `.F03`, or `.F08`, the preprocessor is invoked before the compiler and the assumed source form is fixed and free, respectively.

Assembly language files have a `.s` suffix; they are not processed by the compiler but passed on to the assembler. An assembly language file is produced when the `-S` option is used.

Object files have a `.o` suffix. Such files are not processed by the compiler, but passed on to the linker.

1.2.4 Some Compilation Options

There are many compilation options; they are described in detail in Section "[2.2 Compiler Options](#)". Some of the most important and commonly used options are:

<code>-c</code>	Produce object files only and do not link to create an executable.
<code>-fw</code>	Output only warning and serious error messages (no informative messages).
<code>-fs</code>	Output only serious error messages.
<code>-fmsg_num</code>	Suppress particular error message specified by <i>msg_num</i> .
<code>-Nmaxserious=maxnum</code>	Stop the compilation if s-level (serious) error detected more <i>maxnum</i> .
<code>-oexe_file</code>	The executable is named <i>exe_file</i> instead of <code>a.out</code> . If the <code>-c</code> option is also specified, the object file is named <i>exe_file</i> .
<code>-Fixed</code>	The program is in fixed source form.
<code>-Free</code>	The program is in free source form.
<code>-H[a e f o s u x]</code>	Check for argument mismatches (a), shape conformance (e), simultaneous OPEN and I/O recursive call (f), overlapping dummy arguments and extend undefined (o), subscript and substring values out of bounds (s), undefined variables (u), or module, submodule, common, and pointer undefined check (x). These may be combined: <code>-Haesu</code> or <code>-Hsu</code> .
<code>-Mdirectory</code>	Names a <i>directory</i> to hold module information files.
<code>-Idirectory</code>	Names a <i>directory</i> to search for include and module information files.
<code>-O[{ 0 1 2 3 }]</code>	Specifies the optimization level. The default is the <code>-O2</code> .
<code>-Kfast</code>	Sets the many optimization parameters to reasonable values to achieve optimized performance.
<code>-X6</code>	Compiles Fortran source program as Fortran 66 source.
<code>-X7</code>	Compiles Fortran source program as Fortran 77 source.
<code>-X9</code>	Compiles Fortran source program as Fortran 95 source.
<code>-X03</code>	Compiles Fortran source program as Fortran 2003 source. It is same as <code>-X08</code> .
<code>-X08</code>	Compiles Fortran source program as Fortran 2008 source.

1.2.5 Execution

You can run an executable program created by `frtpx` command. An example of executing `a.out` file follows:

```
$ ./a.out
```

You can specify instructions to Fortran library on invoking an executable file (see "[Chapter 3 Executing Fortran Programs](#)").

1.2.6 Debugging

There are two methods of debugging Fortran programs; one is batch debug function in this Fortran system, another is source-level interactive debug function.

Use gdb to execute source-level debug function.

See Section "[8.2 Debugging Functions](#)" for batch debug function.

1.2.7 Tuning

Use the Profiler for tuning of Fortran programs.

See the "Profiler User's Guide" for the Profiler.

Chapter 2 Compiling and Linking Fortran Programs

This chapter tells you how to compile and link Fortran programs.

2.1 Compile Command

Use the compile command `frtpx` to compile and prepare Fortran programs for execution. The `frtpx` command analyzes its arguments. It then calls the `cpp` command for C preprocessing, the `fpp` command for Fortran preprocessing, Fortran compiler, the `as` command for the assembler, and the `ld` command for the linker, as necessary.

2.1.1 Compile Command Format

Specify the filename list and option list as arguments for the Fortran compiler, the `as` command, and the `ld` command. See Section "2.2 Compiler Options" for information on Fortran compiler options. For information on the `as`, and `ld` commands, use the `man` command to refer to the online manual.

The format of the `frtpx` command is as follows:

Command	Operand
<code>frtpx</code>	<code>[_option-list]_file-name-list</code>

`_`: At least one blank is required.

Notes:

1. The items in the option-list and file-name-list can be specified in any order. For example, you may specify the option-list after the file-name-list or mix option-list and file-name-list specifications.
2. At least one file name must be specified in the file-name-list, unless the entire operand is `-V` or `{ -help | --help }`.
3. See Section "2.1.2 Compile Command Input File" for details of file-name-list.

2.1.2 Compile Command Input File

The following table lists the suffixes allowed for `frtpx(1)` file names.

File type	File suffix	Program
Fortran source program	<code>.f</code>	Fortran compiler
	<code>.for</code>	
	<code>.f90</code>	
	<code>.f95</code>	
	<code>.f03</code>	
	<code>.f08</code>	
Preprocessing directive Fortran source program (*1)	<code>.F</code>	Preprocessor followed by Fortran compiler
	<code>.FOR</code>	
	<code>.F90</code>	
	<code>.F95</code>	
	<code>.F03</code>	
	<code>.F08</code>	
Assembler source program	<code>.s</code>	Assembler
Object program	<code>.o</code>	Linker
Other than above	Other than above	Linker

*1: Preprocessing directives may be included in Fortran source programs. They have the same form as directives for the C language; each line begins with the symbol "#".

2.2 Compiler Options

Compiler options may be specified as option arguments of `frtpx`, by the compilation profile file, and by the environment variable `FORT90CPX` or `FORT90C`.

See Section "2.3 Compile Command Environment Variable" for information about the environment variable `FORT90CPX` or `FORT90C`. See Section "2.4 Compilation Profile File" for information about the compilation profile file.

This section explains the format and function of compiler options.

Each of the system components may be affected by the options.

Usually, options that are unrecognizable to the compile command are ignored and warning message is output. The behavior for unrecognized options can be changed by specifying the environment variable `FCOMP_UNRECOGNIZED_OPTION`. Refer to "2.3 Compile Command Environment Variable" for the environment variable `FCOMP_UNRECOGNIZED_OPTION`.

The following table shows the relation between a file suffix and some compiler options.

File suffix	Option
.f	-X08 -Fixed (*1)
.for	
.F	
.FOR	
.f90	-X9 -Free
.f95	
.F90	
.F95	
.f03	-X03 -Free
.F03	
.f08	-X08 -Free
.F08	

*1: If the `-Xd7` option is specified, activates `-X7 -Fixed` options instead.

2.2.1 List of Compiler Options

Table 2.1 [Options for preprocessor]

Function overview	Option name
The <code>-Ccpp</code> option specifies to use the C preprocessor.	<code>-Ccpp</code>
The <code>-Cfpp</code> option specifies to use the Fortran preprocessor.	<code>-Cfpp</code>
The <code>-Cpp</code> option specifies to use preprocessor.	<code>-Cpp</code>
The <code>-D</code> option specifies to replace name by value.	<code>-Dname[=tokens]</code>
The <code>-P</code> option specifies to make preprocessor store results in the current directory.	<code>-P</code>
The <code>-E</code> option specifies to perform only preprocessing to the specified source file. The result of preprocessing is output to the standard-output.	<code>-E</code>
The <code>-U</code> option specifies to invalidate the definition of name.	<code>-Uname</code>

Table 2.2 [Options for source program]

Function overview	Option name
The -AU option specifies to treat characters in case sensitive manner.	-AU
The -Fixed option specifies to compile a Fortran source program as a fixed-form source.	-Fixed
The -Free option specifies to compile a Fortran source program as a free-form source.	-Free
The -Fwide option specifies to compile a Fortran source program as a fixed-form source whose length of all lines is 255.	-Fwide

Table 2.3 [Options for Fortran standards and language specifications]

Function overview	Option name
The -AE option specifies not to treat special characters such as backslash (\) as control-characters.	-AE
The -Ae option specifies to treat special characters such as backslash (\) as control-characters.	-Ae
The -AT option indicates that it does not apply implicit typing and specifies the null mapping for all the letters.	-AT
The -Ap option specifies to treat the default access permission of modules as PRIVATE.	-Ap
The -Aw option specifies to compile IACHAR, ACHAR, IBITS and ISHFTC functions as intrinsic procedures.	-Aw
The -Ay option specifies to assume type of an unsigned real constant on the right side of an assignment statement as type of a variable on the left side.	-Ay
The -Az option specifies to append a null character (\0) to the end of each string constant argument of an external procedure.	-Az
These options specify whether or not to enable external name which appears only in EXTERNAL statements.	-N{ allextput noallextput }
These options specify whether or not to process the allocatable assignment of the Fortran standard.	-N{ alloc_assign noalloc_assign }
These options specify to check whether a source code conforms to a specific Fortran standard.	-Ncheck_std={ 03d 03e 03o 03s 08d 08e 08o 08s }
These options specify whether or not to enable COARRAY specifications.	-N{ coarray nocoarray }
These options specify whether or not to copy constant actual argument of procedure.	-N{ copyarg nocopyarg }
These options specify whether or not to process the character assignment statements in Fortran standard.	-N{ f90move nof90move }
These options specify whether or not to assume MALLOC and FREE as intrinsic procedures.	-N{ mallocfree nomallocfree }
These options specify whether or not to compile particular functions as intrinsic functions.	-N{ obsfun noobsfun }
These options specify initial states of OpenMP private variables corresponding to allocated allocatable variables that are specified in the PRIVATE clause, etc.	-N{ privatealloc noprivatealloc }

Function overview	Option name
These options specify whether or not to append the RECURSIVE keyword to subroutine subprograms and function subprograms.	-N{ recursive norecursive }
These options specify whether or not to add SAVE statement in each program unit except main program.	-N{ save nosave }
These options specify whether or not to automatically set zero value by the procedure entrance and ALLOCATE statement in each variable.	-N{ setvalue[= <i>set_arg</i>] nosetvalue } <i>set_arg</i> : { { heap noheap } { stack nostack } { scalar noscalar } { array noarray } { struct nostruct } }
These options specify version of the language.	-X{ 6 7 9 03 08 }
The -Xd7 option specifies that -X7 option is the default of language-version option when the suffix of the Fortran source file is .f, .for, .F or .FOR.	-Xd7
Directs to search for a Standard Template Library (STL) required for linkage with C++ programs.	--linkstl= <i>stl_kind</i> <i>stl_kind</i> : { libfjcpp libc++ libstdc++ }

Table 2.4 [Options for degree of accuracy]

Function overview	Option name
These options specify to improve the precision of the variables.	-A{ d i q }
	-C{ cdII8 cI4I8 cdLL8 cL4L8 cdRR8 cR4R8 cd4d8 ca4a8 cdDR16 cR8R16 }
The -C/option specifies to verify whether or not precision of rounding errors is in specified range.	-C/ 0 <= I <= 15

Table 2.5 [Options for data allocation]

Function overview	Option name
The -AA option specifies not to align data in common blocks.	-AA
These options specify whether or not to free allocatable array.	-N{ freealloc nofreealloc }
These options specify whether or not to allocate the actual argument data to write inhibit area.	-N{ use_rodata nouse_rodata }
These options direct the compiler the order in which to allocate the automatic variables to the stack area.	-N{ reordered_variable_stack noreordered_variable_stack }

Table 2.6 [Options for math functions]

Function overview	Option name
These options specify whether or not to inline intrinsic functions and operations.	-K{ ilfunc[= <i>kind</i>] noilfunc } <i>kind</i> : { loop procedure }
These options specify whether or not to use multi-operation function.	-K{ mfunc[= <i>level</i>] nomfunc } <i>level</i> : { 1 2 3 }
These options specify whether or not to create executable program which the Fortran intrinsic function library is linked in statically.	-K{ static_fjlib nostatic_fjlib }
The -SSL2 option specifies to link SSL-II and sequential BLAS/LAPACK libraries.	-SSL2
The -SSL2 option specifies to link SSL-II and parallelized BLAS/LAPACK libraries.	-SSL2BLAMP

Table 2.7 [Options for performance and optimization]

Function overview	Option name
The -O option specifies the optimization level.	-O{ 0 1 2 3 }
The -x option specifies to apply inline expansion for user-defined procedures.	-x{ - <i>proc_name</i> [, <i>proc_name</i>]... <i>stmt_no</i> <i>dat_szK</i> <i>dir</i> = <i>dir_name</i> [, <i>dir</i> = <i>dir_name</i>]... <i>accept</i> = <i>accept_arg</i> 0 }
These options specify to output object files for each target processor.	-K{ A64FX GENERIC_CPU }
These options specify whether or not to adjust eight-byte boundary allocation to eight-byte integer data, double-precision real data, quadruple-precision real data, double-precision complex data, or quadruple-precision complex data.	-K{ <i>align_commons</i> <i>noalign_commons</i> }
These options specify to adjust loop alignment to a power of two byte boundary specified as <i>N</i> .	-K{ <i>align_loops</i> [= <i>N</i>] <i>noalign_loops</i> } 0 <= <i>N</i> <= 32768
These options specify to generate object files using instructions in each architecture.	-K{ ARMV8_A ARMV8_1_A ARMV8_2_A ARMV8_3_A }
These options specify to pad arrays.	-Karraypad_const[= <i>N</i>] 1 <= <i>N</i> <= 2147483647
	-Karraypad_expr= <i>N</i> 1 <= <i>N</i> <= 2147483647
These options specify whether or not to perform the optimization assuming that the array subscript does not exceed the range of the array declaration.	-K{ <i>array_declaration_opt</i> <i>noarray_declaration_opt</i> }
These options specify to merge arrays.	-Karray_merge
	-Karray_merge_common[= <i>name</i>]
	-Karray_merge_local
	-Karray_merge_local_size= <i>N</i> 2 <= <i>N</i> <= 2147483647
These options specify to shift dimensions of arrays.	-Karray_subscript
	-Karray_subscript element= <i>N</i> 2 <= <i>N</i> <= 2147483647
	-Karray_subscript_elementlast= <i>N</i> 2 <= <i>N</i> <= 2147483647
	-Karray_subscript_rank= <i>M</i> 2 <= <i>M</i> <= 30
	-Karray_subscript_variable=' <i>ary_nm</i> (<i>rank</i> , <i>rank</i> [, <i>rank</i>]...)' 1 <= <i>rank</i> <= 30
The -Karray_transform option is equivalent to -Karraypad_const,array_merge,array_subscript.	-Karray_transform
These options specify whether or not to control optimization considering features of the program.	-Kassume={ { <i>shortloop</i> <i>noshortloop</i> } { <i>memory_bandwidth</i> <i>nomemory_bandwidth</i> } { <i>time_saving_compilation</i> <i>notime_saving_compilation</i> } }
These options specify whether or not to allocate local variable on the stack.	-K{ <i>auto</i> <i>noauto</i> }

Function overview	Option name
These options specify whether to allocate automatic data on the stack or on the heap.	-K{ autoobjstack noautoobjstack }
These options specify whether to allocate the result area of user-defined array function statically in the caller program unit or dynamically in execution.	-K{ calleralloc[= <i>level</i>] nocalleralloc } <i>level</i> : { 1 2 }
The -Kcommonpad option specifies to pad array in common blocks.	-Kcommonpad[= <i>N</i>] $4 \leq N \leq 2147483644$
These options specify whether or not to perform optimization that modifies how operations are evaluated for object programs.	-K{ eval noeval }
These options specify to give priority to the instruction-level parallelism in the tree-height-reduction optimization.	-K{ eval_concurrent eval_noconcurrent }
These options specify whether or not to use scalar instructions for stride access store in the target loop when using SIMD extensions.	-K{ extract_stride_store noextract_stride_store }
The -Kfast option specifies to create an object program which runs at high speed on the target machine.	-Kfast
These options specify whether or not the program may access floating point rounding mode, and also specify whether or not the rounding mode is the default.	-K{ fenv_access nofenv_access }
These options direct whether or not to perform optimization using "Floating-Point Multiply-Add/Subtract" floating point operation instructions on source programs.	-K{ fp_contract nofp_contract }
These options specify whether or not to induce the combination of optimization options that do not cause calculation errors in floating-point operations.	-K{ fp_precision nofp_precision }
These options specify whether or not to convert a floating point division or a SQRT function into reciprocal approximation instructions.	-K{ fp_relaxed nofp_relaxed }
These options specify whether or not to apply optimizations which simplify floating-point operations.	-K{ fsimple nofsimple }
These options direct whether or not to use flush-to-zero mode.	-K{ fz nofz }
These options specify whether or not to use the HPC tag address override function of the A64FX processor.	-K{ hpctag nohpctag }
These options direct whether the optimization which uses the intent is performed in the program unit of procedure call which includes the actual argument associated with the dummy argument to which INTENT attribute is specified.	-K{ intentopt nointentopt }
These options specify whether or not to apply loop blocking optimization.	-K{ loop_blocking[= <i>N</i>] loop_noblocking } $2 \leq N \leq 10000$
These options specify whether or not to perform the loop fission optimization for promotion of the software	-K{ loop_fission loop_nofission }

Function overview	Option name
pipelining, improvement of the cache memory efficiency, and the resolution of register shortage.	
These options specify whether or not to perform the strip-mining optimization for fissioned loops.	-K{ loop_fission_stripmining[={ <i>N</i> <i>c-level</i> }] loop_nofission_stripmining } 2 <= <i>N</i> <= 100000000 <i>c-level</i> : {L1 L2}
These options specify the threshold to decide the granularity of loops after loop fission.	-Kloop_fission_threshold= <i>N</i> 1 <= <i>N</i> <= 100
These options specify whether or not to fuse loops into one loop.	-K{ loop_fusion loop_nofusion }
These options specify whether or not to exchange loops.	-K{ loop_interchange loop_nointerchange }
These options specify whether or not to perform optimization which divides loops and partially uses SIMD extensions.	-K{ loop_part_simd loop_nopart_simd }
These options specify whether or not to perform optimization which fissions the imperfectly nested loop into the perfectly nested loops.	-K{ loop_perfect_nest loop_noperfect_nest }
These options specify whether or not to perform optimization by the loop versioning.	-K{ loop_versioning loop_noversioning }
These options specify whether or not to perform link time optimization.	-K{ lto nolto }
The -Knoalias option specifies to perform optimization under the assumption that a pointer variable is not associated with other variable.	-Knoalias[= <i>spec</i>] <i>spec</i> : <i>s</i>
These options specify whether or not to make OCLs (Optimization Control Lines) effective.	-K{ ocl noocl }
These options specify whether or not to keep the frame pointer in a register from procedure calling.	-K{ omitfp noomitfp }
The -Koptions specifies to treat "!options" statements as compiler directive lines.	-Koptions
These options specify to limit the size of text data area to 1MB.	-K{ pc_relative_literal_loads nopc_relative_literal_loads }
These options specify to generate position-independent code (PIC).	-K{ pic PIC }
These options specify whether or not to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC).	-K{ plt noplt }
These options specify whether or not to evaluate invariant expressions in advance.	-K{ preex nopreex }
These options specify how to generate prefetch instructions.	-Kprefetch_cache_level={ 1 2 all }
	-K{ prefetch_conditional prefetch_noconditional }
	-K{ prefetch_indirect prefetch_noindirect }
	-K{ prefetch_infer prefetch_noinfer }
	-Kprefetch_iteration= <i>N</i>

Function overview	Option name
	1 <= N <= 10000
	-Kprefetch_iteration_L2=N
	1 <= N <= 10000
	-Kprefetch_line=M
	1 <= M <= 100
	-Kprefetch_line_L2=M
	1 <= M <= 100
	-K{ prefetch_sequential[= <i>kind</i>] prefetch_nosequential } <i>kind</i> : { auto soft }
	-K{ prefetch_stride[= <i>kind</i>] prefetch_nostride } <i>kind</i> : { soft hard_auto hard_always }
	-Knoprefetch
	-K{ prefetch_strong prefetch_nostrong }
	-K{ prefetch_strong_L2 prefetch_nostrong_L2 }
These options specify whether or not to perform speculative execution of load instructions.	-K{ preload nopreload }
These options specify whether or not to perform instruction scheduling after register allocation.	-K{ sch_post_ra nosch_post_ra }
These options specify whether or not to perform instruction scheduling before register allocation.	-K{ sch_pre_ra nosch_pre_ra }
These options specify to optimize sibling calls.	-K{ sibling_calls nosibling_calls }
These options specify whether or not to use SIMD extensions.	-K{ simd[= <i>level</i>] nosimd } <i>level</i> : { 1 2 auto }
These options specify whether or not to perform optimization promoting packed SIMD.	-K{ simd_packed_promotion simd_nopacked_promotion }
These options specify whether or not to use SIMD extensions to the reduction operation of product.	-K{ simd_reduction_product simd_noreduction_product }
This option specifies the size of the SVE vector register. Units are bits.	-Ksimd_reg_size={ 128 256 512 agnostic }
These options specify whether or not to use SIMD extensions to the uncounted loop.	-K{ simd_uncounted_loop simd_nouncounted_loop }
These options specify whether or not to use the SVE Load Multiple Structures instructions and the SVE Store Multiple Structures instructions.	-K{ simd_use_multiple_structures simd_nouse_multiple_structures }
These options specify whether or not to perform optimizations to prioritize neighboring data regarding array subscript.	-K{ subscript_opt nosubscript_opt }
These options specify whether or not to perform the striping optimization.	-K{ striping[= <i>N</i>] nostriping } 2 <= N <= 100
These options specify whether or not to output object files using SVE.	-K{ SVE NOSVE }
These options specify whether or not to apply software pipelining.	-K{ swp noswp }

Function overview	Option name
These options specify to change the condition of the register number of software pipelining.	-K{ swp_freg_rate= <i>N</i> swp_ireg_rate= <i>N</i> swp_preg_rate= <i>N</i> }
These options specify a policy to select an instruction scheduling algorithm used in software pipelining.	-Kswp_policy={ auto small large }
This option specifies to perform software pipelining for more loops by easing its applicable condition.	-Kswp_strong
These options specify the size of an offset necessary for the access to Thread-Local Storage.	-Ktls_size={ 12 24 32 48 }
These options specify whether or not to allocate temporary array data object on the stack.	-K{ temparraystack notemparraystack }
These options specify whether or not to apply loop unrolling.	-K{ unroll[= <i>N</i>] nounroll } 2 <= <i>N</i> <= 100
These options specify whether or not to apply unroll-and-jam.	-K{ unroll_and_jam[= <i>N</i>] nounroll_and_jam } 2 <= <i>N</i> <= 100
These options specify whether or not to perform zfill optimization.	-K{ zfill[= <i>N</i>] nozfill } 1 <= <i>N</i> <= 100

Table 2.8 [Options for parallelization]

Function overview	Option name
These options specify whether or not to localize arrays in loops.	-K{ array_private noarray_private }
These options specify whether or not to select a loop to be parallelized considering the number of threads when both an inner loop and an outer loop are candidates in a nested loop.	-K{ dynamic_iteration nodynamic_iteration }
The -Kindependent option specifies that the procedure named <i>proc_name</i> behave just the same regardless whether DO loops are parallelized or not.	-Kindependent= <i>proc_name</i>
The -Kinstance option specifies to create a parallelized object program executed in <i>N</i> threads.	-Kinstance= <i>N</i> 2 <= <i>N</i> <= 512
These options specify whether or not to perform automatic parallelization that requires dividing loops.	-K{ loop_part_parallel loop_nopart_parallel }
These options specify whether or not to enable directives of OpenMP standard.	-K{ openmp noopenmp }
These options direct whether or not to promote optimization by assuming that array elements of the chunk size have no data dependency over iteration.	-K{ openmp_assume_norecurrence openmp_noassume_norecurrence }
These options specify whether or not to remove the loop from the object of collapse in the loop construct of OpenMP.	-K{ openmp_collapse_except_innermost openmp_nocollapse_except_innermost }
These options specify whether or not to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the REDUCTION clause of OpenMP was specified.	-K{ openmp_ordered_reduction openmp_noordered_reduction }
These options specify whether or not to enable only the OpenMP SIMD construct, the DECLARE SIMD construct, the DECLARE REDUCTION directive, and the ORDERED construct with the SIMD clause.	-K{ openmp_simd noopenmp_simd }

Function overview	Option name
These options specify whether or not to apply automatic parallelization.	-K{ parallel noparallel }
These options specify whether or not to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads.	-K{ parallel_fp_precision parallel_nofp_precision }
The -Kparallel_iteration option specifies that only the loops which are analyzed to iterate N times or more are targets of parallelization.	-Kparallel_iteration= N $1 \leq N \leq 2147483647$
The -Kparallel_strong option specifies to parallelize all loops which can be parallelized automatically without estimating parallelization effects.	-Kparallel_strong
These options specify whether or not to apply reduction optimization.	-K{ reduction noreduction }
These options specify whether or not to make the parallel region extended.	-K{ region_extension noregion_extension }
These options specify whether or not to create the thread-safe object program.	-K{ threadsafe nothreadsafe }
The -Kvisimpact option specifies to create object programs optimized for VISIMPACT (Virtual Single Processor by Integrated Multicore Parallel Architecture).	-Kvisimpact
These options specify the libraries used for multiprocessing.	-N{ libomp fjomplib }

Table 2.9 [Options for debug and runtime information]

Function overview	Option name
These options specify whether or not to output information for debug into object programs.	{ -g -g0 }
These options specify to debug a serial program.	-H{ a e f o s u x }
The -Ncheck_global option specifies to check following characteristics in program units during compilation. <ul style="list-style-type: none"> - The procedure characteristics between external procedure definitions and references. - The procedure characteristics between external procedure definitions and interface body. - Size of common blocks. 	-N{ check_global nocheck_global }
The -Ncheck_intrfunc option specifies to perform intrinsic function error processing.	-Ncheck_intrfunc
These options specify whether or not to generate the information for the code coverage.	-N{ coverage nocoverage }
These options specify whether or not to enable the Profiler function.	-N{ fjprof nofjprof }
These options specify whether or not to use the hook function called from a specified location.	-N{ hook_func nohook_func }
These options specify whether or not to use the hook function called at regular time interval.	-N{ hook_time nohook_time }
These options specify whether or not to output the statement number where the user defined procedure which causes an error is	-N{ line noline }

Function overview	Option name
called and the statement number where an error occurred in the procedure.	
The -Nprofile_dir option specifies storage location directory of the .gda file which is necessary for the use of the code coverage.	-Nprofile_dir= <i>dir_name</i>
These options specify to debug Fortran source code.	-Eg
	-Nquickdbg [= <i>dbg_arg</i>] <i>dbg_arg</i> : { { argchk noargchk } { subchk nosubchk } { undef undefnan noundef } { inf_detail inf_simple } }
These options specify whether or not to output the runtime information.	-N{ rt_tune rt_notune }
	-Nrt_tune_func
	-Nrt_tune_loop[= <i>kind</i>] <i>kind</i> : { all innermost }
These options specify whether or not to detect the intrinsic instructions errors and floating-point exceptions.	-N{ Rtrap Rnotrap }

Table 2.10 [Options for messages]

Function overview	Option name
The -f option specifies the lowest error level of diagnostic messages.	-f{ i w s <i>msg_num</i> }
The -Ec option specifies to output diagnostic messages when expressions may not be evaluated accurately.	-Ec
These options specify whether or not to output information of applied optimizations and guidance.	-K{ optmsg[={ <i>level</i> guide }] nooptmsg } <i>level</i> : { 1 2 }
These options specify whether to cancel the compilation if the compiler forecasts that it takes a long time to compile the program.	-N{ cancel_overtime_compilation nocancel_overtime_compilation }
The -Ncheck_cache_arraysize option specifies to inspect the sizes of arrays during the compilation and issue a level <i>i</i> diagnostic message if the sizes could be a cause an impact to execution performance.	-Ncheck_cache_arraysize
These options specify whether or not to output the name of file and program unit which are currently processed.	-N{ compdisp nocompdisp }
The -Nmaxserious option specifies to stop compilation when serious errors are detected more than <i>maxnum</i> times.	-Nmaxserious= <i>maxnum</i>
These options specify to output information of compilation into .lst file.	-Nlst[={ a d i m p t x }]
	-Nlst_out= <i>file</i>

Table 2.11 [Options for linker]

Function overview	Option name
The -shared option specifies to create shared objects.	-shared
The -l option specifies to link <i>libname.so</i> or <i>libname.a</i>	-l <i>name</i>
These options specify whether or not to use large page function.	-K{ largepage nolargepage }
These options direct whether the compiler links the library of the optimized version on the string handling functions.	-K{ optlib_string nooptlib_string }
The -L option specifies to search libraries from <i>directory</i> .	-L <i>directory</i>

Table 2.12 [Other options]

Function overview	Option name
The -c option specifies to suppress calling a linker.	-c
These options specify to output help information.	{ -help --help }
These options specify to change method to rename external procedures.	-ml={ cdecl frt }
	-mldefault={ cdecl frt }
The -mlcmain option specifies to select C main program name.	-mlcmain={ main MAIN__ }
The -o option specifies to name output file <i>exe_file</i> .	-o <i>exe_file</i>
The -I option specifies to search include files and module information (.mod and .smod) files from <i>directory</i> .	-I <i>directory</i>
This option specifies the possible largest size of the text area and the static data area in an executable program or a shared object.	-Kcmodel={ small large }
The -M option specifies to output module information (.mod and .smod) files to <i>directory</i> , or to search .mod and .smod files from <i>directory</i> .	-M <i>directory</i>
The -P option specifies to perform only preprocessing.	-P
The -S option specifies to output assembly source.	-S
The -V option specifies to output the version information of each command.	-V
The -W option specifies to use <i>arg1</i> , <i>arg2</i> , ... as options for <i>tool</i> .	-W <i>tool, arg1[, arg2]...</i>
These options specify to output absolute path of tools and options used by frtpx command.	-#
	-###

2.2.2 Syntax of Compiler Options

```
[ -c ] [ -fmsg_lvl ] [ { -g | -g0 } ] [ { -help | --help } ] [ -lname ] [ -ml=target ] [ -mlcmain=main_program ] [ -mldefault=target ]
[ -oexe_file ] [ -shared ] [ -xinl_arg ]
[ -Aobj_arg ] [ -Ctyp_arg ] [ -Dname=tokens ] [ -E ] [ -Emsg_arg ] [ -Fixed ] [ -Free ] [ -Fwide ]
[ -Hdbg_arg ] [ -Idirectory ] [ -Kopt_arg ] [ -Ldirectory ] [ -Mdirectory ]
[ -Nsrc_arg ] [ -O[opt_lvl] ] [ -P ] [ -S ] [ -SSL2 ] [ -SSL2BLAMP ]
[ -Uname ] [ -V ] [ -Wtool,arg1[,arg2]... ] [ -Xlan_lvl ] [ -# ] [ -### ] [ --linkstl=stl_kind ]
```

2.2.3 Description of Compiler Options

-c

The -c (compile only) option suppresses calling the linking phase. If the -c option is specified, object programs are created, but an executable program is not created. It is ignored if the -P and -S option is specified. If only object files are specified by file, it is meaningless to invoke the compile command with the -c option.

-f *msg_lvl* *msg_lvl*: { i | w | s | *msg_num* }

The -f option specifies the lowest error level of diagnostic messages output for Fortran compiler errors.

Each diagnostic message has a prefix of the form *jwdxxxxz-y* message, where *xxxx* is the message serial number, *z* is the message kind, and *y* is the message error level. The *y* can be either i (information message), w (warning message), s (serious error), or u (unrecoverable error). The -f option is the default.

See Section "4.1 Compilation Output" for information about diagnostic messages output during compilation.

If -f*msg_num* is specified, the particular error messages whose message number is *msg_lvl* are suppressed.

The *msg_num* argument can coexist with other arguments (including other *msg_num* arguments). To specify *msg_num* and other arguments for the -f option, separate the arguments by a comma (,).

Example:

```
$ frtprx -fw,1040,2005 a.f95
```

i

The -fi option specifies to output all levels of diagnostic messages.

w

The -fw option specifies to output diagnostic messages only for error levels w, s, and u.

s

The -fs option specifies to output diagnostic messages only for error levels s and u.

msg_num

The -f*msg_num* suppresses particular information and warning diagnostic messages specified by the *msg_num* argument. The *msg_num* argument is the four-digit message number; more than one *msg_num* may be specified. The *msg_num* argument can coexist with any other arguments, but is ignored when -fs is specified.

{ -g | -g0 }

These options direct whether to generate the debugging information used by the debugger. The -g0 option is default.

-g

When this option is effective, the debugging information is generated in object programs.

The user can perform source-level debugging of programs compiled with this option.

When this option is effective, and the option which makes the -O1 or higher option effective is not specified, the -O0 option is effective.

When this option is effective, the -Karray_merge_common[=*name*], -Karray_merge_local, -Karray_subscript, and -Klto options are ignored.

Refer to "[A.2.1 Debug Using Debugger](#)" about the notes on the debugger.

-g0

This option invalidates the -g option.

{ -help | --help }

These options specify to output help information.

-lname

The -l option specifies to search the library *libname.so* or *libname.a*. The position of this option within the command line is important. This is because libraries are searched for in the order in which the other libraries and object files appear within the command line. This option and its argument are passed to the linker.

-ml=*target* *target*: { cdecl | frt }

The -ml option specifies to change how to process external procedure name. *cdecl* or *frt* can be specified for *target*. See Section "[11.3 Rules for Processing Fortran Procedure Names](#)" for details.

-mlcmain=*main_program* *main_program*: { main | MAIN__ }

The -mlcmain option specifies C main program name. C main program name is a function name specified for *main_program*. *main* or *MAIN__* can be specified for the function name. See Section "[11.5.1 Passing Control First to a C Program](#)" for details.

-mldefault=*target* *target*: { cdecl | frt }

The -mldefault option specifies to change how to process default external procedure names according to *target*. *cdecl* or *frt* can be specified for *target*. See Section "[11.3 Rules for Processing Fortran Procedure Names](#)" for details.

See Section "[6.6.6 Service Routines](#)" for precaution to use service routines.

See Section "[12.3.5.4 Runtime Library Routines](#)" for notes on using OpenMP runtime library routines.

See "Profiler User's Guide" for notes when specifying -mldefault=cdecl and using Profiler.

This option for external procedure names of routines in the math library only supports -mldefault=frt. To call the math library from a program compiled with -mldefault=cdecl, append an underscore to the routine name.

-o *exe_file*

The **-o** option specifies to name the output file *exe_file*. Unless either the **-c** or **-S** option is specified, the output file is the executable program. If the **-c** option is specified, the output file is the object program. If the **-S** option is specified, the output file is the assembly program.

-shared

The **-shared** option specifies to create a shared library. This option is passed for linker program.

-xinl_arg *inl_arg*: { - | *proc_name*[,*proc_name*].... | *stmt_no* | *dat_szK* | *dir=dir_name*[,*dir=dir_name*].... | *accept=accept_arg* | 0 }

The **-x** option specifies to apply inline expansion to user-defined procedures. Inline-expansion improves the execution performance as **-K** series options. Either of the **-O1** option or higher must be specified together. To specify plural arguments for the **-x** option, separate the arguments by a comma.

The **-x0** option is default.

This optimization increases the compilation time and memory requirements. The more optimizations are applied, the larger the object program size is. Specify the **-Koptmsg=2** option to know whether or not this optimization is applied.

See Section "9.1.2.1 Inline Expansion" for details.

-

The **-x** option specifies to choose the targets of inline expansion automatically from the user defined functions.

proc_name[,*proc_name*]....

The **-x*proc_name*[,*proc_name*]....** option specifies to apply inline expansion only to user-defined procedures which is specified by *proc_name* arguments. Specifying only the names of frequently called procedures can reduce compilation time and memory requirements significantly. To specify *proc_name* and other arguments (including other *proc_name* arguments) for the **-x** option, separate the arguments by a comma. If *stmt_no* or *dat_szK* is specified in addition to *proc_name*, the compiler applies inline expansion for procedures which meet either condition.

To specify module procedure name as *proc_name*, the module name and a period must be specified immediately before without intervening blanks.

Similarly, to specify internal procedure name as *proc_name*, the host procedure name and a period must be specified immediately before without intervening blanks.

Moreover, if the host procedure is module procedure, the module name and a period must be specified immediately before the host name without intervening blanks.

A procedure name that is implemented in a submodule cannot be specified to this option.

Example: Specifying internal procedure name

```
$ frtpx -xsub.insub a.f90
```

stmt_no 1 <= *stmt_no* <= 2147483647

The **-x*stmt_no*** option specifies to apply inline expansion for user-defined procedures having *stmt_no* or fewer executable statements. The argument *stmt_no* must be an integer value between 1 and 2147483647.

dat_szK 1 <= *dat_sz* <= 2147483

The **-x*dat_szK*** option specifies to apply inline expansion only to user-defined procedures whose all local arrays are up to *dat_szK*. The argument *dat_sz* must be an integer value between 1 to 2147483. The letter K, represents kilobytes (not kibibytes), must be added after the value of *dat_sz*. The default *dat_sz* is unlimited.

The **-x*dat_szK*** option applies inline expansion for user-defined procedures whose local array size is less than the specified *dat_szK* value.

If inline expansion for user-defined procedures is applied, the generated object program includes the data referenced in the user-defined procedure. If this data includes a huge local array, the generated object program can be quite large. To control this, specify the **-x *dat_szK*** option.

`dir=dir_name[,dir=dir_name]..`

The `-xdir=dir_name` option specifies to apply inline expansion for procedures defined in the file under the directory specified by the argument `dir_name` or referenced in the file currently compiled. However, these files are necessary to be the following conditions:

- These files whose suffix is `.f`, `.for`, `.f90`, `.f95`, `.f03`, or `.f08`. And,
- These files must not need to specify `-Cpp` option.

When the `-xdir=dir_name` option is specified, files under the `dir_name` directory are compiled temporarily by the compiler options which are enabled during the compilation of input file, and these are target of inline expansion.

To specify `dir=dir_name` and other arguments (including other `dir=dir_name` arguments) for the `-x` option, separate the arguments by a comma.

In that case, the other sub options are effective on the all files under the specified directory.

Specifying the `-xdir=dir_name` option, needs plenty time and large work area for compilation.

`accept=accept_arg accept_arg: { module_allocatable | nomodule_allocatable }`

These options specify whether to apply inline expansion to the procedure which is the target of inline expansion by other arguments (`-`, `proc_name`, `stmt_no`, `dat_szK`, `dir=dir_name`), and the procedure which contains either of the following conditions.

- Module procedures with a host module or submodule that has an allocatable variable
- Procedures using associated allocatable variable

Note that the execution performance may decrease because some optimizations are suppressed when the procedure includes a variable with `ALLOCATABLE` attribute which meets the following conditions.

1. The variable is of derived type which does not have the `POINTER` attribute. And,
2. A derived type which has the `ALLOCATABLE` attribute exists in the component of 1. And,
3. A variable which has the `POINTER` attribute exists in the component of 2.

This option must be specified at the same time with at least one or more other arguments (`-`, `proc_name`, `stmt_no`, `dat_szK`, `dir=dir_name`) activated.

This option is ignored when specified individually. The default is `-xaccept=nomodule_allocatable`.

`module_allocatable`

This option specifies to perform inline expansion.

`nomodule_allocatable`

This option specifies not to perform inline expansion.

Example: The compiler option "`-xaccept=module_allocatable`"

```
$ frtpx -x- -xaccept=module_allocatable a.f90
```

0

The `-x0` option specifies not to apply inline expansion.

`-A obj_arg obj_arg: { A | E | e | T | U | d | i | p | q | w | y | z }`

The arguments of the `-A` option are `A`, `E`, `e`, `T`, `U`, `d`, `i`, `m`, `p`, `q`, `w`, `y` and `z`. These arguments can be specified at the same time.

A

The `-AA` option specifies not to align data and derived type in common blocks. See Section "[5.2 Correct Data Boundaries](#)" for details.

E

The `-AE` option specifies not to treat special characters such as backslash (`\`) character as control-characters. The `-AE` option is default.

e

The `-Ae` option specifies to treat special characters such as backslash (`\`) as control-characters.

T

The -AT option does not apply implicit typing and specifies the null mapping for all the letters. This does not affect any IMPLICIT statements.

U

The -AU option indicates that names are to be interpreted in a case-sensitive manner. Note the following:

- This causes the program to be interpreted in a nonstandard manner. In ISO standard Fortran, letters are not case sensitive.
- References to intrinsic module entities, service subroutines, service functions or the external procedure names of the multi-operation functions must be in lowercase letters.
- The spelling of all declarations and references to an intrinsic procedure must be the same.
- In the IMPLICIT statement, the lowercase letters specified in a letter specifier list are equivalent to the corresponding uppercase letters.
- When you debug using the debugger, user-defined procedure names are case sensitive and variable names are not case sensitive.

d

The -Ad option improves constants, variables, and functions (intrinsic, statement, external, module and internal functions) of default real and default complex types to double precision. See Section "5.3.1 Precision Improving" for information about this function.

The -CcR8R16, -CcR4R8, -CcdRR8, -CcdDR16, -Ccd4d8, and -Cca4a8 options cannot be used with the -Ad option.

i

The -Ai option changes the default for integer type data from four-byte to two-byte. Two-byte integer type data will be used for:

- Integer constants with absolute values from 0 to 32767
- Data typed using the default implicit typing rules and having symbolic names beginning with I, J, K, L, M, or N
- Data without kind selector declared in an INTEGER statement
- Data resulting from intrinsic functions INT (without one-byte integer, four-byte integer, and eight-type integer argument), NINT, IDNINT, ICHAR, MAX1, MIN1, LEN, and INDEX

The -CcI4I8, -CcdII8, -Ccd4d8, and -Cca4a8 options cannot be used with the -Ai option.

p

The -Ap option specifies a default of private accessibility. It is same as explicitly specifies PRIVATE statement.

q

The -Aq option improves constants, variables, and functions (intrinsic, statement, external, module and internal functions) of double-precision real and double-precision complex types to quadruple precision. See Section "5.3.1 Precision Improving" for information about this function.

The -CcR8R16, -CcR4R8, -CcdRR8, -CcdDR16, -Ccd4d8, and -Cca4a8 options cannot be specified with this option.

w

The -Aw option causes the IACHAR, ACHAR, IBITS, and ISHFTC to be intrinsic procedures even if the -X6 or -X7 option is in effect.

y

The -Ay option improves the kind (precision) of an unsigned real literal constant on the right side of an assignment statement to the kind of the variable on the left side. For example,

```
REAL(8) A
A=1.23456789012345
```

The above program is equivalent to the following program if the -Ay option is specified.

```
REAL(8) A
A=1.23456789012345_8
```


Z

The `-Az` option appends the null character (`\0`) to the end of each character argument passed to an external procedure. This allows a character string to be passed correctly to a C function, such as `strlen`. The length of the character string argument does not include the null character. For example, the result of following program is `1234 4`, even if the `-Az` option is specified.

```
CALL SUB('1234')
END
SUBROUTINE SUB(ARG)
CHARACTER(LEN=*) ARG
PRINT *,ARG , LEN(ARG)
END
```

`-C typ_arg / typ_arg { / | cpp | fpp | pp | cdll8 | cl4l8 | cdll8 | cl4l8 | cdrr8 | cr4r8 | cd4d8 | ca4a8 | cdDR16 | cR8R16 }`

The `-C` option controls numerical accuracy and data alignment, the C preprocessor, the Fortran preprocessor, and the evaluation of data types. To specify more than one argument for the `-C` option, separate the arguments by a comma (,).

`/ 0 <= / <= 15`

The `/` arguments of the `-C` option determine the effect of rounding errors on the results of floating-point data operations. The `/` must be a number between 0 to 15. When the `-C/option is specified, the right most / bits of the mantissa are set to zero each time a value is assigned to a default real, double-precision real, quadruple-precision real, default complex, double-precision complex, or quadruple-precision complex type variable by an assignment statement. (For complex types, these bits are set to zero in both the real and imaginary parts.) See Section "5.3.2 Precision Lowering and Analysis of Errors" for information about this function.`

cpp

When it meets either of the following requirements, the `-Ccpp` option calls the C language preprocessor.

- The suffix of Fortran source file is `.F`, `.FOR`, `.F90`, `.F95`, `.F03`, or `.F08`.
- The `-Cpp` option is specified.

The C language preprocessor processes Fortran source as follows.

- The names are to be interpreted in a case-sensitive manner.
- The preprocessor ignores the comment for Fortran, the continuing line for Fortran, and the column number in fixed source form for Fortran.

If you want to know more detail, see "[Appendix C Preprocessor](#)".

fpp

When it meets either of the following requirements, the `-Cfpp` option calls the Fortran preprocessor.

- The suffix of Fortran source file is `.F`, `.FOR`, `.F90`, `.F95`, `.F03`, or `.F08`.
- The `-Cpp` option is specified.

The Fortran source is processed as follows.

- The names are to be interpreted in a case-sensitive manner.
- The comment for Fortran is given to priority more than the comment for C language.
- The preprocessor recognizes the comment for Fortran, the continuing line for Fortran, and the column number in fixed source form for Fortran.

If you want to know more detail, see "[Appendix C Preprocessor](#)".

pp

The `-Cpp` option calls the C language or Fortran preprocessor. The preprocessor of default is Fortran. If you want to use the C preprocessor, specify `-Cpp` and `-Ccpp` options.

cdll8

The `-Ccdll8` option interprets the default integer variables, constants, and functions as eight-byte integers.

If the `-CcdI18` option is specified, `INT`, `IFIX`, `IDINT`, `IQINT`, `NINT`, `IDNINT`, and `IQNINT` specific intrinsic functions must not be used as an actual argument.

If the `-CcdI18` option is specified, the specific intrinsic functions `NINT` and `IDNINT` must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The `-Ai` option cannot be used with this option. For example,

```
INTEGER      :: A
INTEGER(4)   :: B
A=2
B=2_4
```

is evaluated as follows if the `-CcdI18` option is specified.

```
INTEGER(8)   :: A
INTEGER(4)   :: B
A=2_8
B=2_4
```

cl418

The `-Ccl418` option interprets any four-byte integer type (whether default four-byte or explicitly declared four-byte) as eight-byte integer type. It applies to variables, constants, and functions.

If the `-Ccl418` option is specified, `INT`, `IFIX`, `IDINT`, `IQINT`, `NINT`, `IDNINT`, and `IQNINT` specific intrinsic functions must not be used as an actual argument.

If the `-Ccl418` option is specified, the specific intrinsic functions `NINT` and `IDNINT` must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The `-Ai` option cannot be used with this option. For example,

```
INTEGER      :: A
INTEGER(4)   :: B
A=2
B=2_4
```

is evaluated as follows if the `-Ccl418` option is specified.

```
INTEGER(8)   :: A
INTEGER(8)   :: B
A=2_8
B=2_8
```

cdLL8

The `-CcdLL8` option interprets default logical variables, constants, and functions as eight-byte logicals.

If the `-CcdLL8` option is specified, the specific intrinsic function `BTEST` must not be used as an actual argument.

For example,

```
LOGICAL      :: A
LOGICAL(4)   :: B
A=.TRUE.
B=.TRUE._4
```

is evaluated as follows if the `-CcdLL8` option is specified.

```
LOGICAL(8)   :: A
LOGICAL(4)   :: B
A=.TRUE._8
B=.TRUE._4
```

cL4L8

The `-CcL4L8` option interprets any four-byte logical type (whether default four-byte or explicitly declared four-byte) as an eight-byte logical type. It applies to variables, constants, and functions.

If the `-CcL4L8` option is specified, the specific intrinsic function `BTEST` must not be used as an actual argument. For example,

```
LOGICAL      :: A
LOGICAL(4)   :: B
A = .TRUE.
B = .TRUE._4
```

is evaluated as follows if the `-CcL4L8` option is specified.

```
LOGICAL(8)   :: A
LOGICAL(8)   :: B
A = .TRUE._8
B = .TRUE._8
```

cdRR8

The `-CcdRR8` option interprets the default real type as double-precision real type and interprets the default complex type as double-precision complex type. It applies to variables, constants, and functions.

If the `-CcdRR8` option is specified, the specific intrinsic functions `REAL`, `FLOAT`, `SNGL`, and `SNGLQ` must not be used as an actual argument.

The `-Ad` and `-Aq` option cannot be used with this option. For example,

```
REAL          :: A
REAL(4)       :: B
COMPLEX       :: C
COMPLEX(4)    :: D
A = 2.0
B = 2.0_4
C = (2.0, 2.0)
D = (2.0_4, 2.0_4)
```

is evaluated as follows if the `-CcdRR8` option is specified.

```
REAL(8)       :: A
REAL(4)       :: B
COMPLEX(8)    :: C
COMPLEX(4)    :: D
A = 2.0_8
B = 2.0_4
C = (2.0_8, 2.0_8)
D = (2.0_4, 2.0_4)
```

cR4R8

The `-CcR4R8` option is equivalent to specifying the `-Ad` option. It interprets any default real or complex type (whether default four-byte or explicitly declared four byte) as double-precision real or complex type. It applies to variables, constants, and functions.

If the `-CcR4R8` option is specified, the specific intrinsic functions `REAL`, `FLOAT`, `SNGL`, and `SNGLQ` must not be used as an actual argument.

The `-Ad` and `-Aq` option cannot be used with this option. For example,

```
REAL          :: A
REAL(4)       :: B
COMPLEX       :: C
COMPLEX(4)    :: D
A = 2.0
B = 2.0_4
```

```
C=(2.0,2.0)
D=(2.0_4,2.0_4)
```

is evaluated as follows if the -CcR4R8 option is specified.

```
REAL(8)      ::A
REAL(8)      ::B
COMPLEX(8)   ::C
COMPLEX(8)   ::D
A=2.0_8
B=2.0_8
C=(2.0_8,2.0_8)
D=(2.0_8,2.0_8)
```

cd4d8

The -Ccd4d8 option is equivalent to specifying all of the -CcdII8, -CcdLL8, and -CcdRR8 options. For example,

```
INTEGER      ::A
INTEGER(4)   ::B
LOGICAL      ::C
LOGICAL(4)   ::D
REAL         ::E
REAL(4)      ::F
COMPLEX      ::G
COMPLEX(4)   ::H
A=2
B=2_4
C=.TRUE.
D=.TRUE._4
E=2.0
F=2.0_4
G=(2.0,2.0)
H=(2.0_4,2.0_4)
```

is evaluated as follows if the -Ccd4d8 option is specified.

```
INTEGER(8)   ::A
INTEGER(4)   ::B
LOGICAL(8)   ::C
LOGICAL(4)   ::D
REAL(8)      ::E
REAL(4)      ::F
COMPLEX(8)   ::G
COMPLEX(4)   ::H
A=2_8
B=2_4
C=.TRUE._8
D=.TRUE._4
E=2.0_8
F=2.0_4
G=(2.0_8,2.0_8)
H=(2.0_4,2.0_4)
```

ca4a8

The -Cca4a8 option is equivalent to specifying all of the -CciI4I8, -CclL4L8, and -CcrR4R8 options. For example,

```
INTEGER      ::A
INTEGER(4)   ::B
LOGICAL      ::C
LOGICAL(4)   ::D
REAL         ::E
REAL(4)      ::F
```

```

COMPLEX      ::G
COMPLEX(4)  ::H
A=2
B=2_4
C=.TRUE.
D=.TRUE._4
E=2.0
F=2.0_4
G=(2.0,2.0)
H=(2.0_4,2.0_4)

```

is evaluated as follows if the -Cca4a8 option is specified.

```

INTEGER(8)  ::A
INTEGER(8)  ::B
LOGICAL(8)  ::C
LOGICAL(8)  ::D
REAL(8)     ::E
REAL(8)     ::F
COMPLEX(8)  ::G
COMPLEX(8)  ::H
A=2_8
B=2_8
C=.TRUE._8
D=.TRUE._8
E=2.0_8
F=2.0_8
G=(2.0_8,2.0_8)
H=(2.0_8,2.0_8)

```

cdDR16

The -CcdDR16 option interprets the double-precision real type as quadruple-precision real type. It applies to variables, constants, and functions.

If the -CcdDR16 option is specified, the specific intrinsic functions DFLOAT, DBLE, DBLEQ, DREAL, and DPROD must not be used as an actual argument.

If the -CcdDR16 option is specified, the specific intrinsic function DPROD must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The -Ad, and -Aq option cannot be used with this option. For example,

```

DOUBLE PRECISION ::A
REAL(8)          ::B
A=2.0D0
B=2.0_8

```

is evaluated as follows if the -CcdDR16 option is specified.

```

REAL(16) ::A
REAL(8)  ::B
A=2.0_16
B=2.0_8

```

cR8R16

The -CcR8R16 option is equivalent to specifying the -Aq option. It interprets any double-precision real type as quadruple-precision type and interprets any double-precision complex type as quadruple-precision complex type. It applies to variables, constants, and functions.

If the -CcR8R16 option is specified, the specific intrinsic functions DFLOAT, DBLE, DBLEQ, DREAL, and DPROD must not be used as an actual argument.

If the -CcR8R16 option is specified, the specific intrinsic function DPROD must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The -Ad, and -Aq option cannot be used with this option. For example,

```
DOUBLE PRECISION ::A
REAL(8)          ::B
COMPLEX(8)       ::C
A=2.0D0
B=2.0_8
C=(2.0_8,2.0_8)
```

is evaluated as follows if the -CcR8R16 option is specified.

```
REAL(16)        ::A
REAL(16)        ::B
COMPLEX(16)     ::C
A=2.0_16
B=2.0_16
C=(2.0_16,2.0_16)
```

-D *name*[=*tokens*]

The -D option specifies to define *name* as #define directives do. If *tokens* is omitted, *name* is defined to be 1.

If both -D*name* and -U*name* are specified, the -U*name* option invalidate -D*name* option regardless of the order. Even if -D is not specified, the following macros are defined by default.

Macro	Value	Note
__FRT_major__	The major version number of the compiler	
__FRT_minor__	The minor version number of the compiler	
__FRT_patchlevel__	The patch level of the compiler	
__FRT_version__	The string which represents the version of the compiler	
__FUJITSU	1	
__arch64__	1	
__unix	1	
__linux	1	
unix	1	
linux	1	
_OPENMP	201511	This macro is defined when the -Kopenmp option is effective

-E

The -E option specifies to perform only preprocessing to the specified source file.

The result of preprocessing is output to the standard-output.

The -Cpp option must be specified together unless a suffix of the source file is .F, .FOR, .F90, .F95, .F03, or .F08.

The -P option cannot be specified together.

-E *msg_arg* *msg_arg* { c | g }

The -E option tells the Fortran compiler to output diagnostic messages about operations for which round off may affect the results.

The arguments of the -E option are c and g. These arguments may be combined, e.g., -Ecg.

c

The -Ec option specifies to output diagnostic messages when expressions may not be evaluated accurately. During compilation, this option also outputs level i diagnostic messages for arithmetic IF statements with real expressions.

See Section "[4.1.1 Compilation Diagnostic Messages](#)" for information about diagnostic messages output during compilation. See Section "[6.3.2 Real Data Operations](#)" for information about relational operations.

g

This function increases the compilation time. When the -Eg option is specified, the -Ncheck_global, -Haefosux and -O0 options are in effect.

-Fixed

The -Fixed option specifies to compile a Fortran source program as a fixed-form source.

-Free

The -Free option specifies to compile a Fortran source program as a free-form source.

-Fwide

The -Fwide option specifies to compile a Fortran source program as a fixed-form source whose length of all lines is 255.

-H *dbg_arg* *dbg_arg*: { a | e | f | o | s | u | x }

The -H option specifies the level of run-time error checking. During compilation some errors can be detected in Fortran object programs and error messages are generated. Program execution automatically detects some errors, and the arguments a, e, f, o, s, u and x can be specified to broaden the range of errors checked. Arguments a, e, f, o, s, u and x can be combined. These arguments can be specified at the same time as -Haefosu.

The -Eg option is supported to check the program error, too. It checks characteristics of a procedure between procedure definition and reference, between procedure definition and interface body, and size of common block during compilation.

If the -H option is specified, the default optimization is -O0. The -O option may be specified after the -H option to make the -O option effective. See Section "[8.2 Debugging Functions](#)" for information about the debug function.

a

The -Ha option compares the numbers, types, attributes, and sizes of actual and dummy arguments. The types of function results are also checked during execution. If a mismatch occurs during execution, the corresponding diagnostic message is output. See Section "[8.2.1.1 Checking Argument Validity \(ARGCHK\)](#)" for information about the debug function.

e

The -He options checks shape conformance when an array assignment statement is executed. If a mismatch of shape conformance occurs during execution, the corresponding diagnostic message is output. See Section "[8.2.1.4 Shape Conformance Check \(SHAPECHK\)](#)" for information about the debug function.

f

The -Hf option checks whether the file is connected with two devices or more at the same time in the input/output statement, and whether the input/output statement is called in addition is inspected by the function while executing the input/output statement.

If a mismatch occurs during execution, the corresponding diagnostic message is output. See Section "[8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK \(I/O OPEN\)](#)" for information about the debug function.

o

The -Ho option checks the follows:

- Two dummy arguments are overlapped and the part of overlap is changed.
- When an assumed-size array with INTENT(OUT) attribute is referenced during execution, checks to see if the variable is defined.
- When a variable with SAVE attribute is referenced during execution, checks to see if the variable is defined.

If a mismatch occurs during execution, the corresponding diagnostic message is output. For information about the debug function, see Section "[8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK \(OVERLAP\)](#)".

When the -Ho option is specified, -Ha option and -Hu option are in effect. See Section "[8.2.1.1 Checking Argument Validity \(ARGCHK\)](#)" for information about the -Ha debug function. See Section "[8.2.1.3 Checking References for Undefined Data Items \(UNDEF\)](#)" for information about the -Hu debug function.

S

When an array section, array element, or substring is referenced during execution, the `-Hs` option checks the value of each subscript or substring expression to see if it is within the declared range. The same check is done during compilation when possible. If the value is not within the declared range, a diagnostic message is output. See Section "8.2.1.2 Checking Subscript and Substring Values (SUBCHK)" for information about the debug function.

U

When a variable outside a common block, a module, and a submodule is referenced during execution, the `-Hu` option checks to see if the variable is defined. If the referenced variable is undefined, a diagnostic message is output. See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)" for information about the `u` argument.

X

The `-Hx` option checks to see if the variable declared in a module, in a submodule, or in a common block is defined when the variable is referenced during execution, to see if the pointer is associated when a pointer is referenced during execution, and to see if the incrementation parameter of the DO construct, stride of the FORALL, incrementation parameter of array constructor implied do control and stride of subscript-triplet is not 0.

If the `-Hx` option is specified, the `-Hu` option is in effect.

Note the following when the `-Hx` option using:

- All program units that have a definition or initialization for common block object shall be compiled with `-Hx` option.
- All program units that uses the module or the submodule shall be compiled with `-Hx` option, if the module or the submodule is compiled with `-Hx` option.

See Section "8.2.1.5 Extended Checking of UNDEF (EXTCHK)" for information about the `x` argument.

-I *directory*

The `-I` option specifies directory names to be searched for files specified in an INCLUDE line or for module information (`.mod` and `.smod`) files.

More than one `-I` option may be specified. If multiple `-I` options are specified, the path names are searched in the order specified.

INCLUDE file retrieval is performed in the following order:

1. The directory of the files containing the INCLUDE lines
2. The directory in which `frtpx` is being executed
3. Directories specified as the argument of an `-I` option

The module information files are searched in the following order:

1. Directories specified as the argument of a `-M` option
2. The directory in which `frtpx` is being executed
3. Directories specified as the argument of an `-I` option

-Kopt_arg

`opt_arg`: { { align_commons | noalign_commons } | { align_loops[=*N*] | noalign_loops } | *archi* | arraypad_const[=*N*] | arraypad_expr=*N* | { array_declaration_opt | noarray_declaration_opt } | array_merge | array_merge_common[=*name*] | array_merge_local | array_merge_local_size=*N* | { array_private | noarray_private } | array_subscript | array_subscript_element=*N* | array_subscript_elementlast=*N* | array_subscript_rank=*N* | array_subscript_variable='ary_nm(rank,rank[,rank]...)' | array_transform | assume={ { shortloop | noshortloop } | { memory_bandwidth | nomemory_bandwidth } | { time_saving_compilation | notime_saving_compilation } } | { auto | noauto } | { autoobjstack | noautoobjstack } | { calleralloc[=*level*] | nocalleralloc } | cmodel={ small | large } | commonpad[=*N*] | *cpu* | { dynamic_iteration | nodynamic_iteration } | { eval | noeval } | { eval_concurrent | noeval_concurrent } | { extract_stride_store | noextract_stride_store } | fast | { fenv_access | nofenv_access } | { fp_contract | nofp_contract } | { fp_precision | nofp_precision } | { fp_relaxed | nofp_relaxed } | { fsimple | nofsimple } | { fz | nofz } | { hpctag | nohpctag } | { ilfunc[=*kind*] | noilfunc } | independent=proc_name | instance=*N* | { intentopt | nointentopt } | { largepage | nolargepage } | { loop_blocking[=*N*] | loop_noblocking } | { loop_fission | loop_nofission } | { loop_fission_stripmining[=*N* | *c-level*] } | loop_nofission_stripmining | loop_fission_threshold=*N* | { loop_fusion | loop_nofusion } | { loop_interchange | loop_nointerchange } | { loop_part_parallel | loop_nopart_parallel } | { loop_part_simd | loop_nopart_simd } | { loop_perfect_nest | loop_noperfect_nest } | { loop_versioning


```
{ loop_noversioning } | { lto | nolto } | { mfunc[=level] | nomfunc } | noalias[=spec] | noprefetch | { ocl | noocl } | { omitfp | noomitfp }
| { openmp | noopenmp } | { openmp_assume_norecurrence | openmp_noassume_norecurrence } |
{ openmp_collapse_except_innermost | openmp_nocollapse_except_innermost } | { openmp_ordered_reduction |
openmp_noordered_reduction } | { openmp_simd | noopenmp_simd } | options | { optlib_string | nooptlib_string } |
{ optmsg[=level | guide ] } | nooptmsg | { parallel | noparallel } | { parallel_fp_precision | parallel_nofp_precision } |
parallel_iteration=N | parallel_strong | { pc_relative_literal_loads | nopc_relative_literal_loads } | { pic | PIC } | { plt | noplt } |
{ preex | nopreex } | prefetch_cache_level=N | { prefetch_conditional prefetch_noconditional } | { prefetch_indirect |
prefetch_noindirect } | { prefetch_infer | refetch_noinfer } | prefetch_iteration=N | prefetch_iteration_L2=N | prefetch_line=N |
prefetch_line_L2=N | { prefetch_sequential[=kind] | prefetch_nosequential } | { prefetch_sequential | prefetch_nosequential }
| { prefetch_stride[=kind] | prefetch_nostride } | { prefetch_strong | prefetch_nostrong } | { prefetch_strong_L2 |
prefetch_nostrong_L2 } | { preload | nopreload } | { reduction | noreduction } | { region_extension | noregion_extension } |
{ sch_post_ra | nosch_post_ra } | { sch_pre_ra | nosch_pre_ra } | { sibling_calls | nosibling_calls } | { simd[=level] | nosimd } |
{ simd_packed_promotion | simd_nopacked_promotion } | { simd_reduction_product | simd_noreduction_product } |
simd_reg_size={ 128 | 256 | 512 | agnostic } | { simd_uncounted_loop | simd_nouncounted_loop } |
{ simd_use_multiple_structures | simd_nouse_multiple_structures } | { subscript_opt | nosubscript_opt } | { static_fjlib |
nostatic_fjlib } | { striping[=M] | nostriping } | { SVE | NOSVE } | { swp | noswp } | { swp_freq_rate=N | swp_ireg_rate=N |
swp_preg_rate=N } | swp_policy={ auto | small | large } | swp_strong | swp_weak | { temparraystack | notemparraystack } |
{ threadsafe | nothreadsafe } | tls_size={ 12 | 24 | 32 | 48 } | { unroll[=M] | nounroll } | { unroll_and_jam[=M] | nounroll_and_jam }
| visimpact | { zfill[=M] | nozfill }
```

The `-K` options controls various aspects of optimization in order to improve execution performance of object programs. It is specified using arguments, each controlling a different area of optimization. Multiple arguments can be specified, separated with commas.

The `-N` option can be specified with arguments. The arguments may be separated by comma as in `-Nlst=a,lst=d,lst=x`.

```
{ align_commons | noalign_commons }
```

When allocating the variables in common block, these options specify whether or not to adjust eight-byte boundary allocation to eight-byte integer data, double-precision real data, quadruple-precision real data, double-precision complex data, or quadruple-precision complex data. The default is `-Kalign_commons`.

`align_commons`

When allocating the variables in common block, the `-Kalign_commons` option adjusts eight-byte boundary allocation to eight-byte integer data, double-precision real data, quadruple-precision real data, double-precision complex data, or quadruple-precision complex data.

`noalign_commons`

When allocating the variables in common block, the `-Knoalign_commons` option adjusts appropriate boundary allocation determined by the compiler to eight-byte integer data, double-precision real data, quadruple-precision real data, double-precision complex data, or quadruple-precision complex data.

If the `-Knoalign_commons` option is specified, all program units shall be compiled with `-Knoalign_commons` option.

```
{ align_loops[=N] | noalign_loops }      0 <= N <= 32768
```

The `align_loops` option directs to align loops to a power of two byte boundary. *N* is the number of bytes for the alignment boundary. The value that can be specified for *N* is a power of two (from 1 to 32768) or 0. If a value is not specified or 0 is specified for *N*, the compiler will automatically determine a value. `-Knoalign_loops` is set by default when the `-O0` or `-O1` option is set, and `-Kalign_loops` option is set by default when the `-O2` option or higher is set.

`align_loops[=N]`

The `align_loops` option directs to align loops to a power of two byte boundary. If a value is not specified or 0 is specified for *N*, the compiler will automatically determine a value. The `-Kalign_loops` option is effective when `-O1` option or higher is effective.

`noalign_loops`

The `noalign_loops` option directs not to adjust loop alignment. The `-Kalign_loops=1` is same meaning as this option.

archi

Specifies to generate the object file that is suitable for the given name of architecture and extension. Specify `ARMV8_A`, `ARMV8_1_A`, `ARMV8_2_A`, or `ARMV8_3_A` for *archi*. The default is `-KARMV8_3_A`.

ARMV8_A

This option specifies to generate the object file using Armv8-A instruction set.

ARMV8_1_A

This option specifies to generate the object file using Armv8-A and Armv8.1-A instruction set.

ARMV8_2_A

This option specifies to generate the object file using Armv8-A, Armv8.1-A, and Armv8.2-A instruction set

ARMV8_3_A

This option specifies to generate the object file using Armv8-A, Armv8.1-A, Armv8.2-A, and Armv8.3-A instruction set.

`arraypad_const[=N]` $1 \leq N \leq 2147483647$

If the first dimension of an array has explicit bounds and the bounds expression is constant, `-Karraypad_const` directs to execute padding to align *N* elements. When *N* is not specified, the compiler will determine the padding volume for each array. Padding is the act of creating spaces within arrays.

This cannot be specified at the same time as the `-Karraypad_expr=N`.

See Section "9.12 Effects of Applying Optimization to Change Shape of Array" for points to note when specifying this option.

`arraypad_expr=N` $1 \leq N \leq 2147483647$

Regardless of whether the bounds expression is a constant, `-Karraypad_expr` directs to execute padding to align *N* elements in arrays where the first dimension has explicit bounds.

This cannot be specified at the same time as the `-Karraypad_const[=N]` option.

See Section "9.12 Effects of Applying Optimization to Change Shape of Array" for points to note when specifying this option.

{ `array_declaration_opt` | `noarray_declaration_opt` }

These options specify whether or not to perform the optimization assuming that the array subscript does not exceed the range of the array declaration.

The `-Knoarray_declaration_opt` is default.

`array_declaration_opt`

The `-Karray_declaration_opt` option directs to perform the optimization assuming that the array subscript does not exceed the range of the array declaration.

This option is effective only when the `-O1` option or higher is set.

`noarray_declaration_opt`

The `-Knoarray_declaration_opt` option directs to perform the optimization without assuming that the array subscript does not exceed the range of the array declaration.

`array_merge`

The `-Karray_merge` option is equivalent to specifying the `-Karray_merge_local` and `-Karray_merge_common` options.

See Section "9.15 Merging Array Variables" for points to note when specifying this option.

`array_merge_common[=name]`

The `-Karray_merge_common` option directs that multiple arrays within common blocks are to be merged. If the `-Ncheck_global`, `-Nquickdbg=argchk`, `-Nquickdbg=undef`, `-Nquickdbg=undefnan`, `-Nquickdbg=subchk`, `-g` or `-H` option is set, `-Karray_merge_common[=name]` option will be ignored. Specify the common block name in the *name*.

If this *name* is omitted, the operation will target all arrays within named common blocks.

`array_merge_local`

The `-Karray_merge_local` option specifies to whether multiple local arrays are to be merged. If the `-Ncheck_global`, `-Nquickdbg=argchk`, `-Nquickdbg=undef`, `-Nquickdbg=undefnan`, `-Nquickdbg=subchk`, `-g` or `-H` option is set, the `-Karray_merge_local` option will be invalid. `-Karray_merge_local_size=1000000` will also take effect at the same time.

`array_merge_local_size=N` $2 \leq N \leq 2147483647$

The `-Karray_merge_local_size` option directs that the size of the local arrays to be merged are N bytes or greater. The `-Karray_merge_local` option must be specified together.

{ `array_private` | `noarray_private` }

These options direct whether or not to localize arrays in loops. This option is effective when `-Kparallel` option is effective. The default is `-Knoarray_private`.

`array_private`

The `-Karray_private` option directs to localize arrays in loops. The `-Karray_private` option is effective when `-Kparallel` option is effective.

`noarray_private`

The `-Knoarray_private` option directs not to localize arrays in loops.

`array_subscript`

The `-Karray_subscript` option causes an array dimension shift to be performed on:

- Allocatable arrays with 4 or more dimensions.
- Arrays with 4 or more dimensions that have 10 or fewer elements in the last dimension but more than 100 elements in the other dimensions.

See Section "9.14 Dimension Shift of Array Declaration" for points to note when specifying this option. If the `-Ncheck_global`, `-Nquickdbg=argchk`, `-Nquickdbg=undef`, `-Nquickdbg=undefnan`, `-Nquickdbg=subchk`, `-g` or `-H` option is set, the `-Karray_subscript` option will be invalid.

`-Karray_subscript_element=100`, `-Karray_subscript_elementlast=10`, and `-Karray_subscript_rank=4` will take effect at the same time.

`array_subscript_element=N` $2 \leq N \leq 2147483647$

The `-Karray_subscript_element` option directs that the number of elements other than the last dimension of the array (for which dimension shift is to be performed) is N or more. `-Karray_subscript_element` is effective only when `-Karray_subscript` is set. This option is not effective in allocatable arrays.

`array_subscript_elementlast=N` $2 \leq N \leq 2147483647$

The `-Karray_subscript_elementlast` option directs that the number of elements of the last dimension of the array (for which dimension shift is to be performed) is N or fewer. `-Karray_subscript_elementlast` is effective only when the `-Karray_subscript` option is set. This option is not effective in allocatable arrays.

`array_subscript_rank=N` $2 \leq N \leq 30$

The `-Karray_subscript_rank` option directs that the number of elements of the array (for which dimension is to be performed) is N or more. This is effective only when the `-Karray_subscript` option is set.

`array_subscript_variable='ary_nm(rank,rank[,rank]...)'` $1 \leq rank \leq 30$

The `-Karray_subscript_variable` option specifies to perform dimension shift on the array specified by `'ary_nm(rank,rank[,rank]...)'`. The argument `ary_nm` is the name of array to optimize, and the argument `rank` is the position of the dimension to be shifted. The number of specified dimension positions must be the same as the dimension number of the argument `ary_nm`. The same position of the dimension cannot be specified two times or more. This is effective only when the `-Karray_subscript` option is set, and will invalidate the values specified in the `-Karray_subscript_element=N`, `-Karray_subscript_elementlast=N`, and `-Karray_subscript_rank=N` options.

`array_transform`

The `-Karray_transform` option is equivalent to specifying the `-Karraypad_const`, `-Karray_merge`, and `-Karray_subscript`.

`assume={ { shortloop | noshortloop } | { memory_bandwidth | nomemory_bandwidth } | { time_saving_compilation | notime_saving_compilation } }`

These options specify whether or not to control optimization considering features of the program. Multiple `-Kassume` options can be specified at the same time.

The default is `-Kassume=noshortloop,assume=nomemory_bandwidth,assume=notime_saving_compilation`. The compiler performs optimizations under the following policy.

- Assume that the iteration count is large when the iteration count of the innermost loop is unknown at compilation.
- Solve the bottleneck of CPU operation preferentially.
- Give priority to increase the speed of the executable program rather than decrease of the compilation time.

These options are effective only when the `-O1` option or higher is set.

{ `shortloop` | `noshortloop` }

These options specify whether or not to perform the optimization that assuming that iteration counts of the innermost loops in the program are small. The default is `-Kassume=noshortloop`.

`assume=shortloop`

Optimization is performed assuming that iteration counts of the innermost loops in the program are small. Loop optimizations such as SIMD extensions, automatic parallelization, loop unrolling and software prefetch may be controlled or invalidated.

`assume=noshortloop`

Optimization is performed assuming that iteration counts of the innermost loops in the program are not small.

{ `memory_bandwidth` | `nomemory_bandwidth` }

These options specify whether or not to perform the optimization that assuming that innermost loops in the program has a memory bandwidth bottleneck. The default is `-Kassume=nomemory_bandwidth`.

`assume=memory_bandwidth`

Optimization is performed assuming that innermost loops in the program has a memory bandwidth bottleneck. Loop optimizations such as SIMD extensions, automatic parallelization and loop unrolling may be controlled or invalidated, and `zfill` optimization may be controlled or promoted.

`assume=nomemory_bandwidth`

Optimization is performed assuming that the innermost loop in the program does not have a memory bandwidth bottleneck.

{ `time_saving_compilation` | `notime_saving_compilation` }

These options specify whether or not to control the optimization to decrease compilation time of the program. The default is `-Kassume=notime_saving_compilation`.

`assume=time_saving_compilation`

Optimization is controlled to decrease compilation time of the program.

`assume=notime_saving_compilation`

Optimization is performed with giving priority to the speed of the executable program rather than decrease of the compilation time.

{ `auto` | `noauto` }

These options direct whether to treat local variables (other than those with the `SAVE` attribute or initial values) as automatic variables. The automatic variables become undefined when the procedure ends. The default is `-Kauto`.

`auto`

Local variables, other than those with the `SAVE` attribute or initial values, are treated as automatic variables and allocated to the stack. However, when the `-Kopenmp` option is effective, the local variables of the main program are not allocated to the stack. See Section "9.11 Effects of Allocating on Stack" for points to note when specifying this option.

`noauto`

Local variables, other than those with the `SAVE` attribute or initial values, are not treated as automatic variables.

If this option is to be specified at the same time as the `-Kopenmp` option, it must be specified after the `-Kopenmp` option. Specifying this option with `-Knothreadsafesize` option reduces the stack area size of the procedure being compiled.

This option can be specified for the main program or a subprogram called outside of the dynamic `PARALLEL` region, and must be with OpenMP directives other than the following:

- PARALLEL constructs that do not include private variable declarations
- THREADPRIVATE directing statement

If this is specified for a procedure for which it cannot be specified, the program may not work correctly.

{ autoobjstack | noautoobjstack }

These options specify whether to allocate automatic data on the stack or the heap. The default is -Kautoobjstack.

autoobjstack

The -Kautoobjstack option directs that automatic data objects are allocated to the stack. See Section "9.11 Effects of Allocating on Stack" for points to note when specifying this option.

noautoobjstack

The -Knoautoobjstack option directs that automatic data objects are allocated to the heap.

{ calleralloc[=*level*] | nocalleralloc } *level*: { 1 | 2 }

These options direct whether the function result area is allocated by the caller or called, when referencing user-defined array function. The default is -Knocalleralloc.

Specify 1 or 2 for *level*. The default value is 1. Function results matching the conditions are allocated by the caller according to *level*. With function results that do not meet the condition, the area is allocated dynamically during execution of the function, and the caller deallocates the area after return.

Specifying this option may reduce overhead at execution caused by the dynamic allocation and deallocation of the function result area.

The same option must be specified for all files in which the array function is defined or referenced. If this option is specified differently or different *levels* are specified across the caller and called, the program may not work correctly.

calleralloc=1

When a user-defined array function is referenced, the caller of the function allocates the area for function results that meet the following conditions:

- The result variables of the function are primitive data types.
- The result variables of the function are explicit-shape arrays and all explicit bounds declarations are constant expressions.
- When the result variables of the function are character types, the character length parameter of those variables are constant expressions.

calleralloc=2

In addition to the conditions of calleralloc=1, the result area that meets the following conditions will be allocated by the caller of the function.

- -Karraypad_expr=*N* is not valid.
- The result variables of the function are primitive data types.
- When the result variables of the function are character types, the character length parameter of those variables are constant expressions.
- At the declaration of the function, all the explicit bounds declarations are constant expressions or intrinsic function SIZE references.
- The argument of the intrinsic function SIZE is a derived array name without an OPTIONAL attribute.
- The intrinsic function SIZE does not have a second argument.

nocalleralloc

The -Knocalleralloc option directs to allocate the area of the function result dynamically whenever the function is executed with user-defined array function references. It also deallocates the area at the caller after return.

cmode={ small | large }

This option specifies the possible largest size of the text area and the static data area in an executable program or a shared object. -Kcmode=small is set by default.

`cmodel=small`

The total size of the text area and the static data area is limited to 4GB at linking. This option creates an efficient object program.

`cmodel=large`

Only the size of the text area is limited to 4GB at linking. This option is used when the static data area is large and an error occurs at linking.

Note that the `-K{ pic | PIC }` and `-Kcmodel=large` options cannot be set simultaneously.

`commonpad[=N]` $4 \leq N \leq 2147483644$

The `-Kcommonpad` option specifies to pad array in common blocks to use cache efficiently.

If the source file which includes the common block is compiled with the `-Kcommonpad` option, other source files which include the same common block must be compiled with same `-Kcommonpad`. *N* is specified by bytes.

When *N* is omitted, the compiler determines automatically suitable value. When the common block contain only array variables, the pads are effective.

cpu

Specifies the target processor. In *cpu*, specify A64FX or GENERIC_CPU. `-KA64FX` is set by default.

A64FX

`-KA64FX` option specifies to output object files for the A64FX processor. The `-Khpctag` option is effective when the `-KA64FX` option is set.

GENERIC_CPU

`-KGENERIC_CPU` option specifies to output the object file for the Arm processor.

{ `dynamic_iteration` | `nodynamic_iteration` }

These options direct whether to dynamically select a loop to be parallelized considering the number of threads when both the inner loop and the outer loop are candidates in a nested loop. This option is effective only when the `-Kparallel` option is set. The default is `-Knodynamic_iteration`.

dynamic_iteration

Loops are dynamically selected to be parallelized considering the number of threads when both the inner loop and the outer loop are candidates in a nested loop. The nest level for the loops is up to three.

nodynamic_iteration

Loops are not dynamically selected to be parallelized when both the inner loop and the outer loop are candidates in a nested loop.

{ `eval` | `noeval` }

These options direct whether to perform optimization that modifies how operations are evaluated for source programs. The default is `-Knoeval`.

Note that performing this optimization may cause side effects in the execution result. See Section "[9.20 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

eval

The `-Keval` option modifies how operations are evaluated for source programs. When `-Keval` is set, `-Kfsimple` is valid.

In addition, when this option and `-Kparallel` are set, `-Kfsimple` and `-Kreduction` are valid.

When `-Keval` option and `-Ksimd[={ 1 | 2 | auto }]` option are set, `-Kfsimple` option and `-Ksimd_reduction_product` option are valid.

This option is effective only when the `-O1` option or higher is set.

noeval

The `-Knoeval` option does not modify how operations are evaluated for source programs. When `-Knoeval` is set, `-Knofsimple`, `-Knoreduction`, and `-Ksimd_noreduction_product` options are valid.

{ eval_concurrent | eval_noconcurrent }

-Keval_concurrent option directs to give priority to the instruction-level parallelism in the tree-height-reduction optimization.

-Keval_noconcurrent option is set by default. See Section "[9.1.2.10 Tree-Height-Reduction Optimization](#)" for the tree-height-reduction optimization.

eval_concurrent

-Keval_concurrent option directs to give priority to the instruction-level parallelism in the tree-height-reduction optimization.

This option is effective when the -O1 option or higher and the -Keval option is effective.

The effect of the -Keval_concurrent option is expected when the loop iteration count is small.

eval_noconcurrent

-Keval_noconcurrent option directs to suppress the instruction-level parallelism and specifies to give priority to utilizing FMA instructions in the tree-height-reduction optimization.

{ extract_stride_store | noextract_stride_store }

These options specify whether or not to use scalar instructions for stride access store in the target loop when using SIMD extensions.

The default is -Knoextract_stride_store.

extract_stride_store

The -Kextract_stride_store option directs to use scalar instructions for stride access store in the target loop when using SIMD extensions.

When this option is specified, the function of "SIMD with redundant executions for the SIMD width" is suppressed for loops to which this function is applied. For details, see "[9.1.1.7.4 SIMD with Redundant Executions for the SIMD Width](#)".

Specifying this option may increase the object program size and translation time because SIMD instruction for the stride access store is expanded and scalar instructions are extracted as many as SIMD width.

This option is effective only when the -O2 option or higher is set.

noextract_stride_store

The -Kextract_stride_store option directs to use SIMD instructions for stride access store in the target loop when using SIMD extensions.

fast

The -Kfast option specifies the optimizations for high speed execution on the target machine.

This option is equivalent to the following:

-O3 -Keval,fp_contract,fp_relaxed,fz,ilfunc,mfunc,omitfp,simd_packed_promotion

This option should be specified when programs are compiled and linked.

When this option is set, side effects may occur in the execution results due to effect of the -Keval, -Kfp_contract, -Kfp_relaxed, -Kilfunc, and -Kmfunc options. See Section "[9.20 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

{ fenv_access | nofenv_access }

These options direct whether or not the program may access floating point rounding mode and also direct whether or not the rounding mode is the default. The default is -Knofenv_access.

fenv_access

The -Kfenv_access specifies that the program can access floating point rounding mode or the rounding mode is not the default. Some floating point optimization may be suppressed when this option is valid.

nofenv_access

The -Knofenv_access specifies that the program does not access the floating point rounding mode and this directs that the rounding mode is the default.

{ fp_contract | nofp_contract }

These options direct whether or not to perform optimization using "Floating-Point Multiply-Add/Subtract" floating point operation instructions on source programs. The default is -Knofp_contract.

When -Kfp_contract option is specified, side effects (rounding errors) may occur in the execution results. See Section ["9.20 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

fp_contract

Optimization using Floating-Point Multiply-Add/Subtract floating point operation instructions is performed on source programs.

This option is effective only when the -O1 option or higher is set.

nofp_contract

Optimization using Floating-Point Multiply-Add/Subtract floating point operation instructions is not performed on source programs.

{ fp_precision | nofp_precision }

These options specify whether or not to induce the combination of optimization options that do not cause calculation errors in floating-point operations. The default is -Knofp_precision.

fp_precision

The -Kfp_precision option induces the combination of optimization options that do not cause calculation errors in floating-point operations.

This option is equivalent to replacing this option with the following:

-Knoeval,nofp_contract,nofp_relaxed,nofz,noifunc,nomfunc,parallel_fp_precision

If this option is effective, some optimizations are limited and the execution performance may decrease.

nofp_precision

The -Knofp_precision option does not induce the combination of optimization options that do not cause calculation errors in floating-point operations.

This option invalidates the -Kfp_precision option but does not affect the individual options induced by -Kfp_precision option, even if they are specified together.

{ fp_relaxed | nofp_relaxed }

These options specify whether or not to perform optimizations using reciprocal approximation operation instructions on single-precision or double precision floating point divisions or SQRT functions. The default is -Knofp_relaxed.

This optimization may introduce some side effects. Also, this optimization may result in generating a floating-point exception when -NRtrap is not valid. See Section ["9.20 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

fp_relaxed

The -Kfp_relaxed option directs that reciprocal approximation operation instructions and Floating-Point Multiply-Add/Subtract floating point operation instructions are used on single-precision or double precision floating point division or SQRT functions. The -Kfp_relaxed option is effective only when the -O1 option or higher is set.

When the -NRtrap option and either of the -Knosimd option or the -KNOSVE option are effective, the optimization to convert a SQRT function into reciprocal approximation instructions is suppressed. Therefore, the execution performance may decrease as compared with when the -NRnotrap option is effective. Note that the reciprocal approximation instructions generated by the -Kfp_relaxed option may be evaluated in advance when the -Knopreex option is enabled. And a floating-point exception may occur when the -Knopreex, -Kfp_relaxed, and -NRtrap options are valid.

nofp_relaxed

The -Knofp_relaxed option directs that normal division instructions or SQRT instructions are used for single-precision or double precision floating point division or SQRT functions.

{ fsimple | nofsimple }

These options direct whether to perform simplification of floating point operation on source programs. For example, operations such as "x*0" will be simplified to "0". The default is -Knofsimple.

When `-Kfsimple` option is specified, side effects may occur in the execution results. See Section ["9.20 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

`fsimple`

Simplification of floating point operation on source programs is performed. `-Kfsimple` is effective only when the `-O1` option or higher is set.

`nofsimple`

Simplification of floating point operation on source programs is not performed.

`{ fz | nofz }`

These options direct whether to use flush-to-zero mode. These options must be set at linking. The default is `-Knofz`.

When `-Kfz` option is specified, side effects may occur in the execution results. See Section ["9.20 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

`fz`

The `-Kfz` option directs to use flush-to-zero mode. If a result or a source operand is a denormalized number, flush-to-zero mode replaces it with zero with the same sign. Some programs may get more performance. `-Kfz` option is effective only when the `-O1` option or higher is set at linking.

`nofz`

The `-Knofz` option directs not to use flush-to-zero mode.

`{ hpctag | nohpctag }`

These options specify whether to use the HPC tag address override function of the A64FX processor.

Using the HPC tag address override function, the Sector cache and the hardware prefetch assistance function become effective.

The `-Khpctag` and `-Knohpctag` options require that the `-KA64FX` option is set.

In addition, these options should be specified when programs are compiled and linked.

`-Khpctag` is set by default.

`hpctag`

The `-Khpctag` option directs to use the HPC tag address override function of the A64FX processor.

`nohpctag`

The `-Knohpctag` option directs not to use the HPC tag address override function of the A64FX processor.

`{ ifunc[=kind] | noifunc }` *kind*: { loop | procedure }

These options direct whether to perform inline expansion for intrinsic functions and operations. The following shows the intrinsic functions and exponentiations for which inline expansion can be performed.

- Intrinsic functions of the single-precision and double-precision real type: `ATAN`, `ATAN2`, `COS`, `EXP`, `EXP10`, `LOG`, `LOG10`, `SIN`, and `TAN`
- Intrinsic functions of the single-precision and double-precision complex type: `ABS` and `EXP`
- Exponentiations of the single-precision and double-precision real type which the base and exponent are the same type.

Specify `loop` or `procedure` for *kind*. The default value of *kind* is `procedure`. The default is `-Knoifunc`.

If the compiler determines that the execution performance may decrease, do not apply inline expansion.

Inline expansion is applied by the `-Kifunc` options, side effects may occur in the execution result. Also, this optimization may result in generating a floating-point exception when `-NRtrap` is not valid. See Section ["9.20 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

`ifunc=loop`

Inline expansion is performed for intrinsic functions and operations in a loop. Inline expansion is not performed for intrinsic functions and exponentiations which are not included in loops. The `-Kifunc=loop` option is effective only when the `-O1` option or higher is set. Inline expansion is applied when the optimization may be promoted by checking a data type, an existence of function call and so on.

ifunc=procedure

Inline expansion is performed for all the intrinsic functions and exponentiations in a program unit. The -Kifunc=procedure option is effective only when the -O1 option or higher is set.

noifunc

Inline expansion is not performed for intrinsic functions and exponentiations.

independent=pgm_nm

The -Kindependent option directs that the procedure reference specified in *pgm_nm* always performs as a sequential process, even if the procedure reference is specified within the parallelized DO-loop, and would be subject to automatic parallelization. -Kindependent is effective only when the -Kparallel option is set.

Procedure *pgm_nm* must be compiled with the -Kthreadsafe option specified. If the -Kthreadsafe option is not specified, the results of the execution cannot be guaranteed.

To specify module procedure name as *pgm_nm*, the module name and a period must be specified immediately before without intervening blanks.

Similarly, to specify internal procedure name as *pgm_nm*, the host procedure name and a period must be specified immediately before without intervening blanks.

For an internal procedure-name that is within module procedures, the module-name must also be specified.

A procedure name that is implemented in a submodule cannot be specified to this option.

Example: To specify the internal procedure-name "insub" that is within the parent procedure sub

```
$ frtpx -Kparallel,independent=sub.insub a.f90
```

This option shares points to note with the optimization indicator INDEPENDENT. See Section "[12.2.7.2 Nested Parallel Processing](#)" and Section "[12.2.7.4 Notes on Using Optimization Control Line](#)" for details.

instance=N 2 <= N <= 512

The -Kinstance=N option specifies the number of threads at execution. Specify a value between 2 and 512 for *N*. -Kinstance is effective only when the -Kparallel option is set. If the value of *N* differs from the number of threads at execution, execution will be aborted. See Section "[12.2.7.1 Caution when Specifying Compile-Time Option -Kparallel,instance=N](#)" for details.

{ intentopt | nointentopt }

These options direct whether the optimization which uses the intent is performed in the program unit including procedure call, which procedure call includes the actual argument associated with the dummy argument to which INTENT attribute is specified.

When the -O0 option is set, the default is -Knointentopt. When the -O1 option or higher is set, the default is -Kintentopt.

intentopt

The optimization which uses the intent is performed in the program unit including procedure call, which procedure call includes the actual argument associated with the dummy argument to which INTENT attribute is specified.

This option is effective only when the -O1 option or higher is set.

nointentopt

The optimization which uses the intent is not performed in the program unit including procedure call, which procedure call includes the actual argument associated with the dummy argument to which INTENT attribute is specified.

{ largepage | nolargepage }

These options specify whether or not to create executable program where use large page function. These options must be set at linking.

The -Klargepage option is default.

largepage

-Klargepage option specifies to create the executable program which the large page function is applied.

See Section "[6.5.8 ALLOCATE/DEALLOCATE Statement](#)" for notes on using the ALLOCATE statement.

nolargepage

-Knolargepage option specifies to create the executable program which the large page function is not applied.

{ loop_blocking[=*N*] | loop_noblocking } 2 <= *N* <= 10000

These options direct whether to perform the loop blocking optimization. Specify the block size between 2 and 10000 for *N*. If a value was not specified for *N*, the compiler will automatically determine a suitable value. The default is -Kloop_blocking, when the -O2 option or higher is set.

See Section "9.1.1.5 Loop Blocking" for details.

loop_blocking[=*N*]

Loop blocking is performed. -Kloop_blocking is valid only when the -O2 option or higher is set.

loop_noblocking

Loop blocking is not performed.

{ loop_fission | loop_nofission }

These options direct whether to perform the loop fission optimization for promotion of the software pipelining, improvement of the cache memory efficiency, and the resolution of register shortage. The -Kloop_nofission is set by default when the -O0 or -O1 option is set, and the -Kloop_fission is always applied when the -O2 option or lower is set.

The target of loop fission and the fission point are specified by the following optimization control specifiers.

LOOP_FISSION_TARGET

This optimization control specifier fissions the specified loop automatically.

FISSION_POINT

This optimization control specifier fissions the loop at the specified executable statement.

See Section "9.10.4 Optimization Control Specifiers" for information about LOOP_FISSION_TARGET and FISSION_POINT specifiers. Note that you must specify the -Kocl option to validate optimization control specifiers.

See Section "9.1.2.11 Loop Fission" for details.

loop_fission

Loop fission is performed. -Kloop_fission is valid only when the -O1 option or higher is set. This option works only when used in combination with the above optimization control specifiers.

loop_nofission

Loop fission is not performed.

{ loop_fission_stripmining[={ *N* | *c-level* }] | loop_nofission_stripmining } 2 <= *N* <= 100000000 *c-level*: { L1 | L2 }

These options direct whether to perform strip-mining optimization at the loop fission. There are two ways to specify the strip length: by specifying the length (*N*) directly and by specifying the cache level (*c-level*) to adjust the strip length to the size of level 1 cache or level 2 cache. L1 or L2 can be specified at *c-level*. If the argument "={ *N* | *c-level* }" is omitted, the compiler chooses the strip length automatically.

By this optimization, improvement of the cache memory efficiency is expected for the data accessed between fissioned loops. The -Kloop_fission_stripmining option requires that the optimization control specifier LOOP_FISSION_TARGET is specified in the optimization control line and the -Kloop_fission option, -Kocl option, and -O2 option or higher are set.

-Kloop_nofission_stripmining is set by default.

For details about this function, see Section "9.1.2.11.1 Strip-Mining".

loop_fission_stripmining=*N*

N indicates the strip length. *N* should be from 2 to 100000000.

loop_fission_stripmining=L1

The strip length is adjusted to the size of the first level cache for the cache memory efficiency.

loop_fission_stripmining=L2

The strip length is adjusted to the size of the second level cache for the cache memory efficiency.

loop_nofission_stripmining

The -Kloop_nofission_stripmining specifies not to perform strip-mining optimization.

loop_fission_threshold=N 1 <= N <= 100

This option directs the threshold *N* to decide the granularity of loops after loop fission. *N* should be from 1 to 100. If the value of *N* is reduced, the fissioned loops tends to become small and the number of the fissioned loops tends to increase. The -Kloop_fission_threshold option requires that the optimization control specifier LOOP_FISSION_TARGET is specified in the optimization control line and the -Kloop_fission option, -Kocl option, and -O2 option or higher are set.

This option requires that the -O2 option or higher is set and the -Kloop_fission option or the optimization control specifier LOOP_FISSION is set. -Kloop_fission_threshold=50 is set by default.

{ loop_fusion | loop_nofusion }

These options direct whether to perform loop fusion, which combines adjacent loops. The default when the -O2 option or higher is set is -Kloop_fusion.

loop_fusion

Loop fusion is performed. -Kloop_fusion is valid only when the -O2 option or higher is set.

loop_nofusion

Loop fusion is not performed.

{ loop_interchange | loop_nointerchange }

These options specify whether or not to perform loop interchange. The default when the -O2 option or higher is set is -Kloop_interchange.

loop_interchange

Loop interchange is performed. -Kloop_interchange is valid only when the -O2 option or higher is set.

loop_nointerchange

Loop interchange is not performed.

{ loop_part_parallel | loop_nopart_parallel }

When a loop contains statements to which parallelization can be applied and statements to which parallelization cannot be applied, the loop is divided into two or more loops and automatic parallelization is applied to one of the loops. This optimization is applied to the innermost loop. Note that compile time and execution time may increase. The -Kloop_part_parallel option is effective only when the -Kparallel option is set. The default is -Kloop_nopart_parallel.

loop_part_parallel

The automatic parallelization which needs dividing loops is performed.

loop_nopart_parallel

The automatic parallelization which needs dividing loops is not performed.

{ loop_part_simd | loop_nopart_simd }

When a loop contains instructions to which SIMD extensions can be applied and instructions to which SIMD extensions cannot be applied, the loop is divided into two or more loops and SIMD extensions are applied to one of the loops. This optimization is applied to the innermost loop. Note that compile time and execution time may increase. The -Kloop_part_simd option is effective only when the -Ksimd[={ 1 | 2 | auto }] option is set. The default is -Kloop_nopart_simd option.

loop_part_simd

The optimization using SIMD extensions which needs dividing loops is performed.

loop_nopart_simd

The optimization using SIMD extensions which needs dividing loops is not performed.

{ loop_perfect_nest | loop_noperfect_nest }

These options specify whether to perform optimization which fissions the imperfectly nested loop into the perfectly nested loops. This optimization may promote optimizations such as the loop interchange and loop collapsing. The -Kloop_noperfect_nest is set by default when the -O2 option or less is set, and the -Kloop_perfect_nest is set by default when the -O3 option is set.

loop_perfect_nest

The -Kloop_perfect_nest option specifies to fission the imperfectly nested loop into the perfectly nested loops. This option is valid only when the -O2 option or higher is set.

loop_noperfect_nest

The -Kloop_noperfect_nest option specifies not to perform fission the imperfectly nested loop to the perfectly nested loops.

{ loop_versioning | loop_noversioning }

These options specify whether to perform optimization by the loop versioning.

The loop versioning may promote optimizations such as SIMD, software pipelining, or automatic parallelization.

Note that the size of the object program and compile time may increase because the loop versioning generates two loops. Moreover, the execution performance of the program may decrease due to the overhead of judgement for the choice of generated loops.

These options are effective only when the -O2 option or higher is set. The default is -Kloop_noversioning.

For details about the loop versioning, see Section "9.1.2.6 Loop Versioning".

loop_versioning

The -Kloop_versioning option specifies to perform optimization by the loop versioning.

loop_noversioning

The -Kloop_noversioning option specifies not to perform optimization by the loop versioning.

{ lto | nolto }

These options direct whether to perform link time optimization. -Klto option should be specified when programs are compiled and linked. The default is -Knolto.

If the -g or -Ncoverage option is set, -Klto option will be invalid. If -Klto option and -xdir=*dir_name* option are specified, -xdir=*dir_name* option will be invalid.

See Section "9.1.2.8 Link Time Optimization" for this function.

lto

Link time optimization is performed. -Klto option is effective only when the -O1 option or higher is set.

nolto

Link time optimization is not performed.

{ mfunc[=*level*] | nomfunc } *level*: { 1 | 2 | 3 }

These options direct whether to perform optimization by converting intrinsic functions and operations to multi-operation functions. "Multi-operation functions" are optimized intrinsic functions which perform same operations for multiple arguments at a single calling to improve performance. However, they may introduce side effects (rounding errors) in the execution results due to a difference in algorithms. See Section "9.20 Side Effect of Optimizations for Floating-Point Operation" for the side effect of optimization.

This option is effective only when the optimization *level*-O2 option or higher is set. Specify 1, 2, or 3 for *level*. The default value for *level* is 1. The default is -Knomfunc.

The following shows the intrinsic functions and operations that can be converted.

Function/Operation name	Single precision real type	Double precision real type	Single precision complex type	Double precision complex type	Integer type
ACOS	o (*1)	o (*1)	o (*2)	o (*2)	-
ACOSH	x	x	o (*2)	o (*2)	-

Function/Operation name	Single precision real type	Double precision real type	Single precision complex type	Double precision complex type	Integer type
ASIN	o (*1)	o (*1)	o (*2)	o (*2)	-
ASINH	x	x	o (*2)	o (*2)	-
ATAN	o	o	o (*2)	o (*2)	-
ATAN2	o	o	-	-	-
ATANH	x	x	o (*2)	o (*2)	-
COS	o	o	o (*2)	o (*2)	-
COSH	x (*3)	x (*3)	o (*2)	o (*2)	-
COSQ	x	x	o (*2)	o (*2)	-
ERF	o (*1)	o (*1)	-	-	-
ERFC	o (*1)	o (*1)	-	-	-
EXP	o	o	o (*2)	o	-
EXP10	o	o	-	-	-
LOG	o	o	o (*2)	o (*2)	-
LOG10	o	o	-	-	-
SIN	o	o	o (*2)	o (*2)	-
SINH	x (*3)	x (*3)	o (*2)	o (*2)	-
SINQ	x	x	o (*2)	o (*2)	-
TAN	x	x	o (*2)	o (*2)	-
TANH	x (*3)	x (*3)	o (*2)	o (*2)	-
exponentiation	o	o	x	x	x (*3)
ISHFT	-	-	-	-	o (*2)

o: To be target

x: Not to be target

-: No Intrinsic function

*1) These multi-operation functions execute sequential instructions internally. The execution performance improves because optimizations, such as branch optimization, are applied to the internal instructions. The improvement is inferior to that of the normal multi-operation functions.

*2) These multi-operation functions call the corresponding intrinsic functions multiple times internally. Although the performance of Multi-operation functions is the same as when calling the intrinsic functions multiple times, SIMD optimization to loops including intrinsic functions is promoted by using Multi-operation functions.

*3) These functions are replaced with a sequence of corresponding intrinsic functions.

mfunc=1

A multi-operation function of multiplicities which is the same as the SIMD width is used.

mfunc=2

In addition to the function of -Kmfunc=1, a multi-operation function of multiplicities automatically determined by the compiler is also used. This function needs a large stack area.

mfunc=3

In addition to the function of -Kmfunc=2, a multi-operation function of multiplicities automatically determined by the compiler is also used in a loop which contains an IF construct or similar. The execution performance may be worse than with -Kmfunc=1 or -Kmfunc=2 when the true ratio of the IF construct is low. Moreover, this optimization has side effects which may cause abend.

See Section "[9.16.2 Effects of Compiler Option -Kmfunc=3](#)" for information on the side effects. This function needs a large stack area.

nomfunc

Optimization by converting intrinsic functions and operations to multi-operation functions is not performed.

noalias[=*spec*] *spec*: *s*

The **-Knoalias** directs the compiler to perform optimization under the assumption that a pointer variable or pointer component is not associated with other variable.

s can be specified in *spec*. When *s* is specified for *spec*, the compiler performs optimization under the assumption that an entity with POINTER attribute that meet Fortran standard is not associated with other variable.

In the following context, an entity with POINTER attribute may be associated with other variable. Note that the execution program may output an incorrect result.

- Pointer assignment statement
- Assignment statement of derived type with pointer component
- ALLOCATE statement with SOURCE= specifier in which derived type with pointer component appears
- Dummy argument with POINTER attribute or pointer component
- Initialization of a variable with POINTER attribute or pointer component

noprefetch

This option directs that objects using prefetch instructions are not created.

{ **ocl** | **noocl** }

These options direct whether to enable optimization control lines. See Section "[9.10 Using Optimization Control line \(OCL\)](#)" for information on optimization control lines. The default is **-Knoocl**.

ocl

Optimization control lines are enabled. **-Kocl** is effective only when the **-O1** option or higher is set.

noocl

Optimization control lines are disabled.

{ **omitfp** | **noomitfp** }

These options specify whether or not to keep the frame pointer in a register from procedure calling. The **-Knoomitfp** is default.

omitfp

When the **-Komitfp** option is effective, the frame pointer in a register is not kept. When this option is set, the trace back information is not kept. Either of the **-O1** option or higher must be specified together.

noomitfp

When the **-Knoomitfp** option is effective, the frame pointer in a register is kept.

{ **openmp** | **noopenmp** }

These options direct whether to enable OpenMP directives. See Section "[12.3 Parallelization by OpenMP Specifications](#)" for information on this function. The default is **-Knoopenmp**.

openmp

The OpenMP directives are enabled. When the **-Kopenmp** option is set, the **-Kthreadsafe** and **-Kauto** options will also be set. However, the local variable of the main program will not be allocated to the stack even if the **-Kauto** option is set.

The **-Knoauto** and **-Knothreadsafe** options are valid if they are specified after the **-Kopenmp** option. Caution should be taken when the **-Knoauto** and **-Knothreadsafe** options are specified at the same time as the **-Kopenmp** option.

When only linking is to be performed, **-Kopenmp** option must be specified if an object program that was compiled with the **-Kopenmp** option specified is included.

When the -O1 option or less is effective, SIMD Extensions are not used even if SIMD construct or DECLARE SIMD construct is effective.

noopenmp

OpenMP directives are treated as comments.

{ openmp_assume_norecurrence | openmp_noassume_norecurrence }

These options direct whether or not to promote optimizations by assuming that array elements of the chunk size have no data dependency over iteration for the loop with the OpenMP DO directive.

When -Kopenmp_assume_norecurrence is effective, SIMD extensions, software pipelining, and other optimizations are promoted.

These options are applied to the innermost loops with the OpenMP DO directive.

These options are effective only when the -Kopenmp option and -O2 option or higher is set. The default is -Kopenmp_noassume_norecurrence.

openmp_assume_norecurrence

This option directs to assume that array elements of the chunk size have no data dependency over iteration to promote optimizations.

openmp_noassume_norecurrence

This option directs not to assume that array elements of the chunk size have no data dependency over iteration.

{ openmp_collapse_except_innermost | openmp_nocollapse_except_innermost }

In the loop construct of OpenMP, these options specify whether or not to remove the innermost loop from the object of collapse when it meets all the following conditions.

- The COLLAPSE clause that includes the innermost loop is specified.
- When the compiler can judge that there is a possibility that the execution performance decreases by collapse that includes the innermost loop at the compile time.

The -Kopenmp_collapse_except_innermost and -Kopenmp_nocollapse_except_innermost options are effective only when the -Kopenmp option is set. -Kopenmp_nocollapse_except_innermost is set by default.

openmp_collapse_except_innermost

The innermost loop is removed from the object of collapse. The decrease of execution performance by collapse might be able to be prevented by -Kopenmp_collapse_except_innermost option. The loop removed from the object of collapse can be known from the diagnostic message by setting the -Koptmsg=2 option at the compile time.

openmp_nocollapse_except_innermost

Associated loops are collapsed as it was specified by the user.

{ openmp_ordered_reduction | openmp_noordered_reduction }

These options specify whether or not to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the REDUCTION clause of OpenMP was specified. If the number of threads used is identical, by fixing the order of the reduction operations same as in the numerical order of the threads, identical results will be always obtained. However, rounding errors may occur when the operation order is changed by the effect of the scheduling of a loop construct with a DYNAMIC or a GUIDED schedule kind, or SECTIONS construct.

The -Kopenmp_noordered_reduction option is default.

openmp_ordered_reduction

The -Kopenmp_ordered_reduction option specifies to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the REDUCTION clause was specified. Note that execution performance may decrease compared to when the -Kopenmp_ordered_reduction is invalidated. This option is valid only when the -Kopenmp option is in effect.

openmp_noordered_reduction

The -Kopenmp_noordered_reduction option specifies not to fix the operation order of the reduction operation at the end of the region for which the REDUCTION clause was specified.

{ `openmp_simd` | `noopenmp_simd` }

-`Openmp_simd` specifies that only the OpenMP SIMD construct, the DECLARE SIMD construct, the DECLARE REDUCTION directive, and the ORDERED construct with the SIMD clause are enabled.

-`Knoopenmp_simd` is set by default.

`openmp_simd`

Only the OpenMP SIMD construct, the DECLARE SIMD construct, the DECLARE REDUCTION directive, and the ORDERED construct with the SIMD clause are enabled.

Only the SIMD Extensions based on the OpenMP specifications are applied, and the parallelization is not applied.

When the `-O1` option or less is effective, SIMD Extensions are not used even if SIMD construct or DECLARE SIMD construct is effective.

`noopenmp_simd`

The `-Kopenmp_simd` option is disabled.

options

The `-Koptions` directs that lines starting with "`!options`" are treated as compiler directives. See Section ["2.6 Compiler Directive Lines"](#) for information on compiler directives.

{ `optlib_string` | `nooptlib_string` }

These options direct whether the compiler links the library of the optimized version on the following string handling functions:

`bcopy`, `bzero`, `memchr`, `memcmp`, `memcpy`, `memmove`, `memset`, `strcat`, `strcmp`, `strcpy`, `strlen`, `strncmp`, `strncpy`, `strncat`

These options must be specified at linking. The default is `-Knooptlib_string`.

`optlib_string`

The compiler links the library of the optimized version statically.

A produced executable file by specifying `-Koptlib_string` can be executed only if the execution environment supports SVE.

`-Koptlib_string` option is effective only when the `-KSVE` option and `-KA64FX` option are set.

`nooptlib_string`

Prevents the compiler from linking the library of the optimized version.

{ `optmsg`[= `level` | `guide`] | `nooptmsg` } *level*: { 1 | 2 }

These options direct whether to output messages about optimization status and guidance. Specify 1 or 2 for *level*. If the argument =`{ level | guide }` is omitted, `-Koptmsg=1` is set. The default is `-Koptmsg=1`.

`optmsg=1`

Messages are output indicating that the optimization performed may cause side effects in the execution results.

`optmsg=2`

Along with the messages from `optmsg=1`, messages are output indicating that optimization functions for automatic parallelization, SIMD optimization, loop unrolling, and inline expansion have performed.

`optmsg=guide`

Along with the messages from `optmsg=2`, guidance messages about inhibiting factor and coping method are output when the following optimizations are prohibited to apply.

- SIMD instruction
- Automatic Parallelization
- Software pipelining
- Inline expansion

Notes about this option:

- The guidance messages are not output when coping method can be determined only with the messages from `optmsg=2`.
- The coping method from the guidance messages may not work as expected with some programs.

`nooptmsg`

Output of optimization status messages and guidance messages are suppressed.

{ `parallel` | `noparallel` }

These options direct whether or not performed automatic parallelization. However, if the effect of parallel execution is not expected, automatic parallelization is not performed. This option induces the `-O2`, `-Kregion_extension`, `-Kloop_part_parallel`, and `-Kloop_perfect_nest` options. If the `-O3` option is set, however, the optimization level will be 3. If the `-H` or `-Eg` option is set, the `-Kparallel` option will be ignored. The default is `-Knoparallel`.

See Section "[12.2 Automatic Parallelization](#)" for information on this option.

When only linking is to be performed, `-Kparallel` option must be specified if an object program that was compiled with the `-Kparallel` option specified is included.

`parallel`

The `-Kparallel` option directs that automatic parallelization is performed.

`noparallel`

The `-Knoparallel` option directs that automatic parallelization is not performed.

{ `parallel_fp_precision` | `parallel_nofp_precision` }

These options specify whether or not the compiler controls to apply optimizations considering calculation error of a floating type or a complex type operation that is caused by the difference of the parallel number of threads. The default is `-Kparallel_nofp_precision`.

`parallel_fp_precision`

The `-Kparallel_fp_precision` option specifies to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads.

This option is effective when `-Kparallel` or `-Kopenmp` option is effective.

When this option and `-Kopenmp` option are set, `-Kopenmp_ordered_reduction` option is valid.

When this option is set, the execution performance may decrease because the part of optimization is restricted.

Note that the calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads even if this option is valid when the `REDUCTION` clause of OpenMP is specified.

`parallel_nofp_precision`

The `-Kparallel_nofp_precision` option specifies not to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads.

`parallel_iteration=N` $1 \leq N \leq 2147483647$

The `-Kparallel_iteration` option directs that only loops determined at compilation to have a number of iterations exceeding N will be subject to parallelization. `-Kparallel_iteration` is effective only when the `-Kparallel` option is set.

Example:

```
REAL(KIND=4),DIMENSION(10000,5) :: A,B,C
DO J=1,5
  DO I=2,10000
    A(I,J) = B(I,J) + C(I,J)
  END DO
END DO
```

When `-Kparallel_iteration=6` option is set, the outer loop will not be subject to parallelization because its iteration is 5.

`parallel_strong`

The `-Kparallel_strong` option directs to parallelize all loops for which it is detected that parallelization can be performed without estimating the effects of the parallelization. This option includes the `-Keval` and `-Kpreex` options.

Aside from this point, the functions and notes are the same as for the `-Kparallel` option.

{ `pc_relative_literal_loads` | `nopc_relative_literal_loads` }

The `-Kpc_relative_literal_loads` option directs to limit the size of text data area in the function to 1MB, and access to the literal pool in one instruction. This option is effective at compilation time only. `-Knopc_relative_literal_loads` is set by default.

`pc_relative_literal_loads`

`-Kpc_relative_literal_loads` enables PC-relative literal loads.

`nopc_relative_literal_loads`

`-Knopc_relative_literal_loads` disables PC-relative literal loads.

{ `pic` | `PIC` }

Specifies to generate position-independent code (PIC). The `-Kpic` and `-KPIC` options are equivalent.

Note that the `-K{ pic | PIC }` and `-Kcmodel=large` options cannot be set simultaneously.

This option takes effect only during compilation.

{ `plt` | `noplt` }

These options specify whether to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC). `-Kplt` is set by default.

This option is effective only if the `-Kpic` option or the `-KPIC` option is set.

`plt`

`-Kplt` specifies to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC).

`noplt`

`-Knoplt` specifies not to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC).

{ `preex` | `nopreex` }

These options direct whether to evaluate invariant expressions in advance. Note that this optimization may cause errors, as instructions that may not have been executed based on the logic of the program are likely to be executed. To identify whether or not this optimization was performed, see the diagnostic messages at compilation. See Section "9.1.2.2 Advance evaluation of Invariant Expressions" for information on the side effects.

The default is `-Knopreex`.

`preex`

Invariant expressions are evaluated in advance. `-Kpreex` is effective only when the `-O1` option or higher is set.

`nopreex`

Invariant expressions are not evaluated in advance. Note that a floating-point exception may occur when the `-Knopreex`, `-Kfp_relaxed`, and `-NRtrap` options are valid.

`prefetch_cache_level=N` $N: \{ 1 \mid 2 \mid \text{all} \}$

The `-Kprefetch_cache_level` option directs which cache level is to have data prefetched. `-Kprefetch_cache_level` is effective only when the `-Kprefetch_sequential`, `-Kprefetch_stride`, or `-Kprefetch_indirect` option is set. Specify 1, 2, or all for N . The default is `-Kprefetch_cache_level=all`.

`prefetch_cache_level=1`

The `-Kprefetch_cache_level=1` option directs to prefetch data in the first level cache. Normal prefetch instructions are used.

`prefetch_cache_level=2`

The `-Kprefetch_cache_level=2` option directs to prefetch data only using the second level cache.

`prefetch_cache_level=all`

Functionality of `prefetch_cache_level=1` and `prefetch_cache_level=2` are valid simultaneously. Faster prefetch can be achieved by using the two types of prefetch instruction in combination.

{ prefetch_conditional | prefetch_noconditional }

These options specify whether or not to generate prefetch instructions for array data used in blocks in IF constructs or CASE constructs. This option requires that the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect option is set. The default is -Kprefetch_noconditional.

prefetch_conditional

Prefetch instructions are generated for array data used in blocks in IF constructs and CASE constructs.

prefetch_noconditional

Prefetch instructions are not generated for array data used in blocks in IF constructs and CASE constructs.

{ prefetch_indirect | prefetch_noindirect }

These options direct whether to create an object that uses prefetch instructions for array data that accessed indirectly (list access) within a loop. This option is effective only when the -O1 option or higher is set. The default is -Kprefetch_noindirect.

prefetch_indirect

Prefetch instructions are generated for array data accessed indirectly (list access) within a loop.

prefetch_noindirect

Prefetch instructions are not generated for array data accessed indirectly (list access) within a loop.

{ prefetch_infer | prefetch_noinfer }

These options direct whether to create prefetch instructions even if the prefetching distance is unknown. This option requires that the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect option is set. The default is -Kprefetch_noinfer.

prefetch_infer

Prefetch instructions for sequential access are created, even if the prefetching distance is unknown.

prefetch_noinfer

Prefetch instructions for sequential access are not created if the prefetching distance is unknown.

prefetch_iteration=N 1 <= N <= 10000

The -Kprefetch_iteration option specifies to prefetch data that is referenced or defined after N iterations of the loop. If the optimization using SIMD extensions is applied to a loop, specifies the loop iteration count after using SIMD extensions for N .

As this option is only for prefetch instructions that prefetch to the first level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=1 or the -Kprefetch_cache_level=all option is set.

This cannot be specified with the -Kprefetch_line=N option.

prefetch_iteration_L2=N 1 <= N <= 10000

The prefetch_iteration_L2 option specifies to prefetch data that is referenced or defined after N iterations of the loop. If the optimization using SIMD extensions is applied to a loop, specifies the loop iteration count after using SIMD extensions for N .

As this option is only for prefetch instructions that prefetch to the second level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=2 or the -Kprefetch_cache_level=all option is set.

This cannot be specified at the same time as the -Kprefetch_line_L2=N option.

prefetch_line=N 1 <= N <= 100

The -Kprefetch_line option specifies to prefetch data which are referenced or defined after N cache line(s).

As this option is only for prefetch instructions that prefetch to the first level cache, it is valid only when the -Kprefetch_sequential or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=1 or the -Kprefetch_cache_level=all option is set.

This cannot be specified at the same time as the -Kprefetch_iteration=N option.

prefetch_line_L2=N 1 <= N <= 100

The -Kprefetch_line_L2 option specifies to prefetch data that is referenced to or defined in a line after N lines.

As this option is only for prefetch instructions that prefetch to the second level cache, it is valid only when the `-Kprefetch_sequential` or `-Kprefetch_indirect` option is valid, and, the `-Kprefetch_cache_level=2` or the `-Kprefetch_cache_level=all` option is set.

This cannot be specified at the same time as the `-Kprefetch_iteration_L2=N` option.

{ `prefetch_sequential[=kind]` | `prefetch_nosequential` } *kind*: { `auto` | `soft` }

These options direct whether to generate prefetch instructions for array data accessed sequentially within a loop. Specify `auto` or `soft` for *kind*. The default value of *kind* is `auto`. The default when the `-O0` or `-O1` option is set is `-Kprefetch_nosequential`. The default when the `-O2` option or higher is set is `-Kprefetch_sequential`.

`prefetch_sequential=auto`

The compiler automatically selects whether to use hardware-prefetch or to create prefetch instructions for array data that is accessed sequentially within a loop. `-Kprefetch_sequential=auto` is effective only when the `-O1` option or higher is set.

`prefetch_sequential=soft`

The compiler does not use hardware-prefetch, but rather creates prefetch instructions for array data that is accessed sequentially within a loop. `-Kprefetch_sequential=soft` is effective only when the `-O1` option or higher is set.

`prefetch_nosequential`

Prefetch instructions are not generated for array data that is accessed sequentially within a loop.

{ `prefetch_stride[=kind]` | `prefetch_nostride` } *kind*: { `soft` | `hard_auto` | `hard_always` }

These options direct whether to perform prefetching array data that is accessed with a stride larger than the cache line size used in the loop. This includes loops with prefetch addresses not defined at compilation. Specify `soft`, `hard_auto` or `hard_always` for *kind*. The default value of *kind* is `soft`. The default is `-Kprefetch_nostride`.

The `-Kprefetch_stride=soft` and `-Kprefetch_nostride` options are effective only when the `-O1` option or higher is set.

The `-Kprefetch_stride=hard_auto` and `-Kprefetch_stride=hard_always` options are effective only when the `-Khpctag` option and `-O1` option or higher are set.

`prefetch_stride=soft`

The `-Kprefetch_stride=soft` option specifies to generate prefetch instructions and perform the prefetch.

`prefetch_stride=hard_auto`

The `-Kprefetch_stride=hard_auto` option specifies to perform the prefetch using the hardware stride prefetcher. This option sets the stride prefetcher to prefetch only data that is not on cache.

`prefetch_stride=hard_always`

The `-Kprefetch_stride=hard_always` option specifies to perform the prefetch using the hardware stride prefetcher. In contrast with the `-Kprefetch_stride=hard_auto` option, this option sets the stride prefetcher to always prefetch.

`prefetch_nostride`

The `-Kprefetch_nostride` option specifies not to generate prefetch instructions.

{ `prefetch_strong` | `prefetch_nostrong` }

These options direct whether to create strong prefetch instructions for the first level cache.

As this option is only for prefetch instructions that prefetch to the first level cache, it is valid only when any option of the `-Kprefetch_sequential`, `-Kprefetch_stride`, or `-Kprefetch_indirect` is set, and, the `-Kprefetch_cache_level=1` or the `-Kprefetch_cache_level=all` option is set.

The default is `-Kprefetch_strong`.

`prefetch_strong`

Prefetch instructions created for the first dimension cache are strong prefetch.

`prefetch_nostrong`

Prefetch instructions created for the first dimension cache are not strong prefetch.

This option is effective only when the `-Khpctag` option is set.

{ prefetch_strong_L2 | prefetch_nostrong_L2 }

These options direct whether to create strong prefetch instructions for the second level cache.

As this option is only for prefetch instructions that prefetch to the second level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=2 or the -Kprefetch_cache_level=all option is set.

The default is -Kprefetch_strong_L2.

prefetch_strong_L2

Prefetch instructions created for the second level cache are strong prefetch.

prefetch_nostrong_L2

Prefetch instructions created for the second level cache are not strong prefetch.

This option is effective only when the -Khpctag option is set.

{ preload | nopreload }

These options direct whether to perform the speculative execution of load instructions. By specifying the -Kpreload option, the optimization of instruction scheduling is promoted, and the execution performance is expected to improve. Note that this optimization may cause interruption of execution of the program by the execution of load instructions referring to illegal areas that are not to be executed logically in the program. Specify the -Koptmsg=2 option to know whether or not this optimization is applied.

The default is -Knopreload.

preload

Speculative execution of load instructions is performed. The -Kpreload option is effective only when the -O1 option or higher is set.

nopreload

Speculative execution of load instructions is not performed.

{ reduction | noreduction }

These options direct whether to perform the reduction optimization. This option is effective only when the -Kparallel option is set. The default is -Knoreduction.

See Section "[9.20 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

See Section "[12.2 Automatic Parallelization](#)" and Section "[12.2.5.8 Loop Reduction](#)" for information about this function.

reduction

Reduction optimization is performed.

noreduction

Reduction optimization is not performed.

{ region_extension | noregion_extension }

These options specify whether or not to expand parallel region to reduce parallelization overhead. This option is effective only when the -Kparallel option is set. See Section "[12.2.5.11 Extension of Parallel Region](#)" for information on this function. The default is -Knoregion_extension.

region_extension

Parallelization overhead is reduced by expanding parallelization.

The execution performance may be degraded on loops with a small parallelization effect.

noregion_extension

Parallelization is not expanded.

{ sch_post_ra | nosch_post_ra }

These options specify whether or not to perform instruction scheduling after register allocation. The compiler rearranges execution instructions to improve the execution performance. This optimization does not increase saving and restoring instructions for registers to and from the memory.

When the -O0 option is set, the default is -Knosch_post_ra. When the -O1 option or higher is set, the default is -Ksch_post_ra.

sch_post_ra

Instruction scheduling after register allocation is performed.

This option is effective only when the -O1 option or higher is set.

nosch_post_ra

Instruction scheduling after register allocation is not performed.

{ sch_pre_ra | nosch_pre_ra }

These options specify whether or not to perform instruction scheduling before register allocation. The compiler rearranges the order of execution instructions to improve the execution performance. Note that the execution performance may decrease because of the increase of saving and restoring instructions for registers to and from the memory.

When the -O0 option is set, the default is -Knosch_pre_ra. When the -O1 option or higher is set, the default is -Ksch_pre_ra.

sch_pre_ra

Instruction scheduling before register allocation is performed.

This option is effective only when the -O1 option or higher is set.

nosch_pre_ra

Instruction scheduling before register allocation is not performed.

{ sibling_calls | nosibling_calls }

The -Ksibling_calls option directs to optimize the sibling call. The -Knosibling_calls option is set by default when the -O0 or -O1 option is set. The -Ksibling_calls option is set by default when the -O2 option or higher is set.

sibling_calls

The -Ksibling_calls option directs to optimize sibling calls. When this option is effective, the trace back information is not kept.

This option is effective when the -O1 option or higher is set.

nosibling_calls

The -Knosibling_calls option directs not to optimize sibling calls.

{ simd[=*level*] | nosimd } *level*: { 1 | 2 | auto }

These options direct whether to create objects that use SIMD extension instructions. The -Ksimd[={ 1 | 2 | auto }] option is effective only when the optimization level -O2 option or higher is set. Specify 1, 2, or auto for *level*. The default value for *level* is auto. The default when the -O2 option or higher is set is the -Ksimd=auto option. When the -Ksimd[={ 1 | 2 | auto }] option is specified, the -Kloop_part_simd is in effect.

See Section "9.1.1.7 SIMD" for details.

simd=1

Objects that use SIMD extension instructions are created.

simd=2

In addition to the functions of the simd=1 option, objects that use SIMD extension instructions are created for loops that include IF constructs. In order to redundantly execute instructions within IF constructs, performance may be reduced depending on the true ratio of the IF construct. In addition, as with the -Kpreex option, speculative executions are performed within an expression in IF constructs, so this optimization may execute an instruction which would not be executed based on the logic of the program, causing an error.

`simd=auto`

The compiler automatically determines whether to use SIMD extensions for the loop. SIMD extensions are promoted for loops that contain IF construct.

`nosimd`

Objects are created that do not use SIMD extension instructions.

{ `simd_packed_promotion` | `simd_nopacked_promotion` }

These options direct whether to perform optimization promoting packed SIMD assuming that index calculations of array elements of both single-precision floating-point type and 4-byte integer type do not exceed 4-byte range. The `-Ksimd_packed_promotion` option is effective when the `-Ksimd[={ 1 | 2 | auto }]` option is set.

The `-Ksimd_nopacked_promotion` option is set by default.

When the `-Ksimd_packed_promotion` is effective and the index calculations exceed 4-byte range, the program execution may be aborted or the result may be incorrect by referring illegal area.

`simd_packed_promotion`

Perform packed SIMD extensions.

`simd_nopacked_promotion`

Not perform packed SIMD extensions.

{ `simd_reduction_product` | `simd_noreduction_product` }

These options specify whether or not to use SIMD extensions to the reduction operation of product. These options are effective only when the `-Ksimd[={ 1 | 2 | auto }]` option is set. The default is `-Ksimd_noreduction_product`.

`-Ksimd_reduction_product` option cannot be specified with the `-Ksimd_reg_size=agnostic` option at the same time.

See Section "[9.20 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

`simd_reduction_product`

SIMD extensions are used to the reduction operation of product.

`simd_noreduction_product`

SIMD extensions are not used to the reduction operation of product.

`simd_reg_size={ 128 | 256 | 512 | agnostic }`

These options specify the size of the SVE vector register. Units are bits.

`-Ksimd_reg_size=512` is set by default. This option is effective when the `-KSVE` option is set.

`simd_reg_size={ 128 | 256 | 512 }`

When the `-Ksimd_reg_size={ 128 | 256 | 512 }` option is specified, optimizations are performed on the assumption that the SVE vector register size is a fixed value specified as the option at compilation time.

Therefore, optimizations are promoted and the improvement of the execution performance is expected.

However, the generated executable program works normally only on CPU architecture which has the same size of the SVE vector register as the size specified at compilation time.

For details, see Section "[9.21 Notes on Specified SVE Vector Register Size](#)".

`simd_reg_size=agnostic`

When specifies `-Ksimd_reg_size=agnostic`, the SVE vector register is not considered to be a specific size, and the executable program decides the SVE vector register size at execution time. The executable program does not depend on the vector register size on the CPU architecture. Note that the execution performance might decrease compared with the case of `-Ksimd_reg_size={ 128 | 256 | 512 }`.

{ `simd_uncounted_loop` | `simd_nouncounted_loop` }

These options specify whether or not to create objects that use SIMD extension instructions for statements in a DO WHILE loop, a DO UNTIL loop, and a DO loop that contains statements terminating the loop,

The `-Ksimd_uncounted_loop` option is effective when the `-Ksimd[={ 1 | 2 | auto }]` and `-KSVE` option are set. The default is `-Ksimd_nouncounted_loop`.

`simd_uncounted_loop`

Objects that use SIMD extension instructions are created.

`simd_nouncounted_loop`

Objects are created that do not use SIMD extension instructions.

{ `simd_use_multiple_structures` | `simd_nouse_multiple_structures` }

These options specify whether or not to use the SVE Load Multiple Structures and SVE Store Multiple Structures instructions when using SIMD extensions.

This option is effective when the `-Ksimd[={ 1 | 2 | auto }]` and the `-KSVE` option are set.

The `-Ksimd_use_multiple_structures` option is default.

`simd_use_multiple_structures`

Use the SVE Load Multiple Structures instructions and Store Multiple Structures instructions. The execution performance is improved by utilizing the above instructions for the load and store when using SIMD extensions. Note that the performance may decrease depending on data alignment.

`simd_nouse_multiple_structures`

The `-Ksimd_nouse_multiple_structures` option do not use the SVE Load Multiple Structures instructions and the SVE Store Multiple Structures instructions.

{ `subscript_opt` | `nosubscript_opt` }

These options specify whether or not to perform optimizations to prioritize neighboring data regarding array subscript, which is applicable to stencil calculation.

The `-Knosubscript_opt` option is default.

`subscript_opt`

The `-Ksubscript_opt` option specifies to perform optimizations to prioritize neighboring data regarding array subscript.

Note that the `-Ksubscript_opt` option may decrease execution performance because more registers are used.

`nosubscript_opt`

The `-Knosubscript_opt` option specifies not to perform optimizations to prioritize neighboring data regarding array subscript.

{ `static_fjlib` | `nostatic_fjlib` }

These options specify whether or not to create executable program which the Fortran intrinsic function library is linked in statically. These options must be set at linking. The `-Knostatic_fjlib` option is default.

`static_fjlib`

`-Kstatic_fjlib` option specifies to create the executable program which the Fortran intrinsic function library is linked in statically. The size of executable program may increase when specifying this option.

`nostatic_fjlib`

`-Knostatic_fjlib` option specifies to create the executable program which the Fortran intrinsic function library is linked in dynamically.

{ `striping[=N]` | `nostriping` } $2 \leq N \leq 100$

These options specify whether or not to perform the striping optimization. The striped length (number of expansions) can be specified for N as a value between 2 and 100. The default value for N is 2. When the iteration count of a loop in the source program is apparent, the expansion number determined by the compiler will be used even if a number that exceeds the iteration count is specified for N . The default is `-Knostriping`.

See Section "9.1.2.4 Loop Striping" for information on striping.

In striping, the statements in a loop are unrolled many times, so, as with loop unrolling, the size of the object module will increase, along with the required compilation time. Caution should be taken when using it, as performance may also be reduced due to an increased number of registers used.

`striping[=N]`

The striping optimization is performed. `-Kstriping` is effective only when the `-O2` option or higher is set.

`nostriping`

The striping optimization is not performed.

`{ SVE | NOSVE }`

These options specify whether or not to output object files using SVE, which is an Armv8-A extension. `-KSVE` is set by default.

Specify the `-KNOSVE` option to generate object files for a processor that does not support SVE.

`SVE`

`-KSVE` option specifies to output object files using SVE.

`NOSVE`

`-KNOSVE` option specifies to output object files not using SVE.

`{ swp | noswp }`

These options direct whether to perform the software pipelining optimization. However, if the effect of software pipelining is not expected, software pipelining is not performed. When the `-O0` or `-O1` option is set, the default is `-Knoswp`. When the `-O2` option or higher is set, the default is `-Kswp`.

The `-Kswp` and `-Knoswp` options require that the `-O1` option or higher is set.

If this option is specified with the `-Kswp_weak` or `-Kswp_strong` option, the one specified last is effective.

See Section "[9.1.1.6 Software Pipelining](#)" for information on software pipelining.

`swp`

Software pipelining is performed. `-Kswp` is effective only when the `-O2` option or higher is set.

`noswp`

Software pipelining is not performed.

`{ swp_freg_rate=N | swp_ireg_rate=N | swp_preg_rate=N }`

These options specify the rate (percentage) about the following registers that can be used by software pipelining.

- Floating-point register and SVE vector register
- Integer register
- SVE predicate register

N should be from 1 to 1000. `-Kswp_freg_rate=100,swp_ireg_rate=100,swp_preg_rate=100` is set by default.

The application of the software pipelining can be adjusted by changing the condition of the number of registers.

If the software pipelining is not applied due to shortage of register, it may be applied by specifying integer values that are larger than 100.

Specifying this option may increase saving and restoring instructions for registers to and from the memory, and the execution performance may decrease.

This option is effective only if the `-O2` option or higher is set.

`swp_freg_rate=N`

Specifies that *N*% of the floating-point register and the SVE vector register is available when applying the software pipelining.

`swp_ireg_rate=N`

Specifies that *N*% of the integer register is available when applying the software pipelining.

swp_preg_rate=*N*

Specifies that *N*% of the SVE predicate register is available when applying the software pipelining.

swp_policy={ auto | small | large }

These options specify a policy to select an instruction scheduling algorithm used in software pipelining.

Software pipelining is performed by the compiler option -Kswp, -Kswp_weak, or -Kswp_strong is specified, or the optimization control line corresponding to each option is enabled.

The default is -Kswp_policy=auto.

swp_policy=auto

The compiler automatically selects a fit algorithm for each loop.

swp_policy=small

An algorithm fit for a small loop, such as a loop with low register pressure, is used.

swp_policy=large

An algorithm fit for a large loop, such as a loop with high register pressure, is used.

swp_strong

The -Kswp_strong option specifies to perform software pipelining for more loops by easing its applicable condition.

This option may increase compilation time and memory requirement significantly.

If this option is specified with the -Kswp or -Kswp_weak option, the one specified last is effective.

Aside from this point, the functions and notes are the same as for the -Kswp option.

swp_weak

The -Kswp_weak option specifies to adjust software pipelining for the target loop and reduce overlap of instructions in the loop.

When the loop iteration count is uncertain and small, the effect of the optimization is expected because the loop iteration count required to execute the software-pipelined loop becomes small.

This option may decrease the execution performance because the overlap of instructions become small.

If this option is specified with the -Kswp or -Kswp_strong option, the one specified last is effective.

Aside from this point, the functions and notes are the same as for the -Kswp option.

{ temparraystack | notemparraystack }

These options direct whether to allocate the following results of the stack area. The default is -Ktemparraystack.

- Intermediate results of the array operation
- Mask expression evaluation result when iteration count of DO CONCURRENT is constant.

temparraystack

Intermediate results of the array operation and mask expression evaluation result are allocated to the stack area. See Section "[9.11 Effects of Allocating on Stack](#)" for points to note when specifying this option.

notemparraystack

Intermediate results of the array operation and mask expression evaluation result are not allocated to the stack area.

{ threadsafe | nothreadsafesafe }

These options direct whether or not to create thread-safe object programs. See Section "[9.11 Effects of Allocating on Stack](#)" for points to note about the stack area size.

The default is -Knothreadsafesafe.

threadsafe

Thread-safe object programs are created.

nothreadsafe

Created object programs not thread-safe.

If this option is to be specified at the same time as the `-Kopenmp` option, it must be specified after the `-Kopenmp` option. When this option is specified the instructions generated are not those that guarantee the process to be thread-safe.

This option can be specified for the main program or a subprogram called outside of the dynamic parallel region, and must be with OpenMP directives other than the following:

- PARALLEL constructs that do not include private variable declarations
- THREADPRIVATE directives

If this is specified for a procedure for which it cannot be specified, the program may not work correctly.

`tls_size={ 12 | 24 | 32 | 48 }`

These options specify the size of an offset necessary for the access to Thread-Local Storage. Units are bits.

As the size of Thread-Local Storage, `-Ktls_size=12`(4K bytes), `-Ktls_size=24`(16M bytes), `-Ktls_size=32`(4G bytes) or `-Ktls_size=48`(256T bytes) can be specified.

When the size of the Thread-Local Storage exceeds the range of the offset, an error occurs at link time. These options are invalidated when the `-Kpic` or `-KPIC` option is specified simultaneously.

`{ unroll[=N] | nounroll }` $2 \leq N \leq 100$

These options direct that the loop unrolling optimization is performed. Specify the upper bound for loop unrolling in *N*, as a value between 2 and 100. If a value was not specified for *N*, the compiler will automatically determine the best value. If the `-O0` or `-O1` option is set, the default is `-Knounroll`. If the `-O2` option or higher is set, the default is `-Kunroll`.

See Section "9.1.1.4 Loop Unrolling" for details.

`unroll[=N]`

Loop unrolling is performed. `-Kunroll` is effective only when the `-O1` option or higher is set.

`nounroll`

Loop unrolling is not performed.

`{ unroll_and_jam[=N] | nounroll_and_jam }` $2 \leq N \leq 100$

These option direct to apply unroll-and-jam. *N* is the upper limit of the unrolling expansion number which should be from 2 to 100. If `=N` is omitted, the compiler automatically determines a suitable value for *N*. `-Knounroll_and_jam` option is set by default. If the effect of unroll-and-jam is not expected, unroll-and-jam is not performed.

This optimization is not expected to be effective for all the loop in the program, so it is recommended to use optimization control specifier `UNROLL_AND_JAM` or `UNROLL_AND_JAM_FORCE` rather than `-Kunroll_and_jam` option.

For details about unroll-and-jam, see Section "9.1.2.9 Unroll-and-Jam".

`unroll_and_jam[=N]`

Unroll-and-jam is applied. `-Kunroll_and_jam` is effective when the `-O2` option or higher is set.

`nounroll_and_jam`

Unroll-and-jam is not applied.

visimpact

The `-Kvisimpact` option directs that the most appropriate automatically parallelized object for VISIMPACT (Virtual Single Processor by Integrated Multicore Parallel Architecture) is created. This performs the same optimization as when the `-Kfast,parallel` option is specified.

This option should be specified when programs are compiled and linked.

See Section "9.20 Side Effect of Optimizations for Floating-Point Operation" for the side effect of optimization.

`{ zfill[=N] | nozfill }` $1 \leq N \leq 100$

The `-Kzfill` option directs to perform zfill optimization. The zfill optimization speeds up write operations for array data that is only written in a loop, by using an instruction that allocates space on the cache for writing (DC ZVA) without loading data from the

memory. The zfill optimization works on the data N blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and N is an integer value between 1 and 100. If a value is not specified for N , the compiler will automatically determine a value. The `-Knozfill` is set by default.

The `-Kzfill` option requires that the `-KA64FX` and the `-O2` option or higher are set.

Note that if an object program compiled with the `-Kzfill` option is executed on a CPU other than the one on which a single cache write operation of a DC ZVA instruction is 256 bytes, the execution may be terminated abnormally or an incorrect result may occur. The amount of the cache write operation for a DC ZVA instruction on an A64FX is 256 bytes.

Performance may also be reduced under the following conditions:

- The program is not affected by memory bandwidth bottleneck.
- When the loop iteration count is too few.
- When the block size is explicitly specified with the `-Kzfill= N` option and the memory size where the data is written in the loop is smaller than N blocks.

This optimization is not expected to be effective for all the loop in the program, so it is recommended to use optimization control specifier `ZFILL` rather than `-Kzfill` option for the entire program.

For details about zfill, see Section "[9.1.2.5 zfill](#)".

`zfill[= N]`

The zfill optimization is performed. By specifying N , the data N blocks ahead are used for optimization.

`nozfill`

The zfill optimization is not performed.

`-L directory`

The `-L` option add directory to the list of directories in which the linker searches for libraries. This option and its argument are passed to the linker.

Libraries are searched in the following order:

1. Directories specified as the argument of the `-L` option
2. The directory installed libraries provided by the compiler
3. The directory installed standard libraries
4. Directories specified to the environment variable `LIBRARY_PATH`

`-M directory`

The `-M` option specifies an alternate directory for module information files (`.mod` and `.smod` files).

`.mod` and `.smod` files are created in directory during compilation. For information about the compilation of modules and submodules, see "[Chapter 10 Fortran Modules and Submodules](#)".

`-N src_arg`

`src_arg`: { { allextput | noallextput } | { alloc_assign | noalloc_assign } | { cancel_overtime_compilation | nocancel_overtime_compilation } | check_cache_arraysize | { check_global | nocheck_global } | check_intrfunc | check_std=`std_arg` | { coarray | nocoarray } | { compdisp | nocompdisp } | { copyarg | nocopyarg } | { coverage | nocoverage } | { f90move | nof90move } | { fjprof | nofjprof } | { freealloc | nofreealloc } | { hook_func | nohook_func } | { hook_time | nohook_time } | { libomp | fjomplib } | { line | noline } | lst[=`lst_arg`] | lst_out=`file` | { mallocfree | nomallocfree } | maxserious=`maxnum` | { obsfun | noobsfun } | { privatealloc | noprivatealloc } | profile_dir=`dir_name` | quickdbg[=`dbg_arg`] | { recursive | norecursive } | { reordered_variable_stack | noreordered_variable_stack } | { rt_tune | rt_notune } | rt_tune_func | rt_tune_loop[=`kind`] | { save | nosave } | { setvalue[=`set_arg`] | nosetvalue } | { use_rodata | nouse_rodata } | { Rtrap | Rnotrap } | { allextput | noallextput }

The `-Nallextput` option creates an external name which appears only on EXTERNAL statement. The `-Nallextput` option is default.

The `-Nnoallextput` option does not create an external name which appears only on EXTERNAL statement.

{ alloc_assign | noalloc_assign }

The -Nalloc_assign is specified, the allocatable assignment (Refer to Section "6.4.2 Allocatable Assignment") is processed in Fortran standard when the -X9, -X03, or -X08 option is in effect. If the -X6 or -X7 is in effect, the -Nalloc_assign is not in effect.

When the -X03 or -X08 option is in effect, the -Nalloc_assign option is set by default, and when the -X9 option is in effect, the -Nnoalloc_assign option is set by default.

alloc_assign

The -Nalloc_assign is specified, the allocatable assignment is processed in Fortran standard when the -X9, -X03, or -X08 option is in effect. The program has less effect on execution performance in Fortran 95 standard program when the -Nalloc_assign is specified.

noalloc_assign

The -Nnoalloc_assign is specified, the allocatable assignment is not processed in Fortran standard.

{ cancel_overtime_compilation | nocancel_overtime_compilation }

These options specify whether to cancel the compilation if the compiler forecasts that it takes a long time (24 hours or more as a guide) to compile the program.

The -Ncancel_overtime_compilation option is default.

cancel_overtime_compilation

The -Ncancel_overtime_compilation specifies to cancel the compilation if the compiler forecasts that it takes a long time to compile the program.

nocancel_overtime_compilation

The -Nnocancel_overtime_compilation specifies not to cancel the compilation even if it takes a long time to compile the program.

check_cache_arraysize

The -Ncheck_cache_arraysize option specifies to check the sizes of arrays during compilation and issue level-i diagnostic messages if the array sizes may cause execution performance decrease.

This function decides that an array could be a cause of a cache conflict if its size is a multiple of that of second level cache.

The array shape could be rewritten so as not to have a size of multiple of that of the cache in order to avoid the impact.

The function cannot judge correctly when the Sector cache is active.

When the compiler option -Karraypad_const[=*N*] or -Karraypad_expr=*N* is specified for notes when this function is used, the size of the array becomes the size which adds the size of the padding.

See Section "4.1.1 Compilation Diagnostic Messages" for information about diagnostic messages output during compilation.

{ check_global | nocheck_global }

The -Ncheck_global option specifies to check the size of common blocks, and the procedure characteristics between external procedure definitions and references, and between external procedure definitions and interface body in program units during compilation.

If an error occurs, a diagnostic message is output. When -Ncheck_global is valid, compilation time may increase.

The -Nnocheck_global option specifies not to check in between each program units during compilation.

check_intrfunc

The -Ncheck_intrfunc option specifies to make the intrinsic function error processing effective for object programs.

If this option is in effect, error processing is performed when the object program is executed.

When it suits the error as the result of checking, when the following processing is executed:

- The output of the diagnostic message, traceback and error summaries
- user control of error by ERRSET and ERRSAV subroutine
- execution of user-defined subroutine

See Section "8.1.5 Intrinsic Function Error Processing" for details about the error number is detected by `-Ncheck_intrfunc` option and intrinsic function error processing.

`check_std=std_arg std_arg: { 03d | 03e | 03o | 03s | 08d | 08e | 08o | 08s }`

The arguments may be separated by comma as in `-Ncheck_std=03d,check_std=03o`.

The arguments 03d and 08d, 03e and 08e, 03o and 08o, or 03s and 08s cannot be specified at the same time.

03d

The 03d specifies to issue diagnostic messages if the source code contains any deleted features in Fortran 2003.

03e

The 03e specifies to issue diagnostic messages if the source code contains any Fortran 2003 features not also in Fortran 95.

03o

The 03o specifies to issue diagnostic messages if the source code contains any obsolescent features in Fortran 2003.

03s

The 03s specifies to issue diagnostic messages if the source code contains any nonstandard features in Fortran 2003.

08d

The 08d specifies to issue diagnostic messages if the source code contains any deleted features in Fortran 2008.

08e

The 08e specifies to issue diagnostic messages if the source code contains any Fortran 2008 features not also in Fortran 2003.

08o

The 08o specifies to issue diagnostic messages if the source code contains any obsolescent features in Fortran 2008.

08s

The 08s specifies to issue diagnostic messages if the source code contains any nonstandard features in Fortran 2008.

`{ coarray | nocoarray }`

These options direct whether to enable COARRAY specifications. See the "Fortran User's Guide Additional Volume COARRAY" for information on this function. The default is `-Nnocoarray`.

coarray

The `-Ncoarray` option validates COARRAY specification.

When only linking is to be performed, `-Ncoarray` option must be specified if an object program that was compiled by with the `-Ncoarray` option specified is included.

nocoarray

The `-Nnocoarray` option invalidates COARRAY specification.

If you compile COARRAY programs with this option, error messages are output and the compilation stops.

`{ compdisp | nocompdisp }`

These options specify whether or not to output file name and program name which are currently compiled. The `-Nnocompdisp` option is default.

compdisp

The `-Ncompdisp` option specifies to output file name and program name which are currently compiled.

nocompdisp

The `-Nnocompdisp` option specifies not to output file name and program name which are currently compiled.

`{ copyarg | nocopyarg }`

These options specify whether or not to copy scalar constant argument to generated variable argument. The `-Nnocopyarg` option is default.

copyarg

The `-Ncopyarg` option specifies to copy scalar constant argument to generated variable argument. When the `-Ncopyarg` is specified, even if the value of dummy argument is updated in procedure, it does not affected by the constant value of actual argument. However, this option enables programs that do not conform to the Fortran standard.

nocopyarg

The `-Nnocopyarg` option specifies that the scalar constant specified in the actual argument of the procedure interface is used as the argument.

Example:

```
CALL SUB(1)
PRINT *,1
END
SUBROUTINE SUB(I)
I=2
END
```

When `-Ncopyarg` is specified, this program outputs 1.

{ coverage | nocoverage }

These options specify whether or not to generate the information for the code coverage. `-Ncoverage` option should be specified when programs are compiled and linked. `-Nnocoverage` is set by default.

When the `-Ncoverage` option is effective, the `-Klto` option invalid.

For details about the code coverage, see section "[Appendix G Code Coverage](#)".

coverage

The `-Ncoverage` option specifies to generate the information for the code coverage.

If the `-Ncoverage` option specifies, the execution performance may decrease due to the instructions that measures the execution time are added in the object program.

nocoverage

The `-Nnocoverage` option specifies not to generate the information for the code coverage.

{ f90move | nof90move }

These options direct the behavior of "[6.4.1 Assignment Statements with Overlapping Character Positions](#)". If the `-X9`, `-X03`, or `-X08` option is in effect, the `-Nf90move` is always in effect and the `-Nnof90move` option cannot be specified. If the `-X7` or `-X6` option is in effect, the `-Nnof90move` option is default.

f90move

The `-Nf90move` option is specified, the character assignment statements are processed in Fortran standard even if the `-X7` or `-X6` option is in effect.

nof90move

The `-Nnof90move` option is specified, the character assignment statements are not processed in Fortran standard.

Example:

```
CHARACTER(LEN=5) C
C='12345'
C(2:5)=C(1:4)
PRINT *,C ! output is 11234
END
```

When the `-Nf90move` option is specified, this program outputs "11234" even if the `-X7` or `-X6` option is in effect.

{ fjprof | nofjprof }

These options direct whether to enable the Profiler function. These options must be set at linking. The default is `-Nfjprof`.

See the "Profiler User's Guide" for information on the Profiler.

fjprof

The -Nfjprof option validates the Profiler function.

nofjprof

The -Nnofjprof option invalidates the Profiler function.

{ freealloc | nofreealloc }

These options specify whether or not to deallocate arrays which do not have save attribute and are allocated when the procedure exits. The -Nfreealloc option is default.

freealloc

The -Nfreealloc option specifies to deallocate arrays which do not have save attribute and are allocated when the procedure exits.

nofreealloc

If -Nnofreealloc option is specified and an unsaved allocatable array has a status of currently allocated when a procedure is exited, the array is not deallocated. This is not the behavior specified by the Fortran 95 standard.

{ hook_func | nohook_func }

These options specify whether or not to use the hook function that is called from a specified location.

See Section "[8.2.3 Hook Function](#)", for information about the hook function.

The default is -Nnohook_func.

When only linking is to be performed, -Nhook_func option must be specified if an object program that was compiled with the -Nhook_func option specified is included.

hook_func

The -Nhook_func option is specified, a user-defined subroutine is called from the following locations:

- Program entry and exit
- Procedure entry and exit
- Parallel region (OpenMP or automatic parallelization) entry and exit

nohook_func

This option specifies not to use the hook function.

{ hook_time | nohook_time }

These options specify whether or not to use the hook function called at regular time interval.

Refer to Section "[8.2.3 Hook Function](#)", for information about the hook function.

The default is -Nnohook_time. This option must be set at linking.

hook_time

If the -Nhook_time option is enabled, a user-defined subroutine is called at regular time interval.

The time interval can be specified using the environment variable FLIB_HOOK_TIME. If the environment variable FLIB_HOOK_TIME is not specified, the user-defined subroutine is called once every minute.

Refer to Section "[3.8 Using Environment Variables for Execution](#)", for information about the environment variable FLIB_HOOK_TIME.

nohook_time

This option specifies not to use the hook function.

{ libomp | fjomplib }

These options specify libraries used for multiprocessing. These options must be set at linking. -Nlibomp option is set by default.

libomp

The -Nlibomp option specifies to use LLVM OpenMP Library for multiprocessing. See "[Chapter 12 Multiprocessing](#)" for LLVM OpenMP Library.

fjomplib

The `-Nfjomplib` option specifies to use Fujitsu OpenMP Library for multiprocessing. See "[Appendix J Fujitsu OpenMP Library](#)" for Fujitsu OpenMP Library.

{ line | noline }

These options specify whether or not to output the statement number where the user defined procedure which causes an error is called and the statement number where an error occurred in the procedure. This information is output to standard error output when an error occurs during execution of Fortran programs. The `-Nline` option is default. See Section "[4.2.2 Trace Back Map](#)" for information about the trace back map.

line

The `-Nline` option specifies to output the statement number where the user defined procedure which caused an error is called and the statement number where an error occurred in the procedure. These numbers are output into the trace back map.

noline

The `-Nnoline` option specifies that even if the error occurs in the procedure, the statement number is not displayed in the trace back map.

lst[=*lst_arg*] *lst_arg*: { a | d | i | m | p | t | x }

The `-Nlst` option outputs compilation information to a file with suffix `.lst`. If more than one Fortran source program file is specified, it is output to the first-file-name.lst.

The arguments may be specified at the same time, separated by comma as in `-Nlst=a,lst=d,lst=x`. If arguments are omitted, the source program list and diagnostic error messages are output.

a

If the `-Nlst=a` option is specified, the attributes of names are output in addition to the compilation information produced by the `-Nlst` option.

d

If the `-Nlst=d` option is specified, the layout of derived types is output in addition to the compilation information produced by the `-Nlst` option.

i

If the `-Nlst=i` option is specified, include files are listed in addition to the compilation information produced by the `-Nlst` option.

m

If the `-Nlst=m` option is specified, the Fortran program which expresses the situation of automatic parallelization with OpenMP directives is output in addition to the compilation information produced by the `-Nlst=p` option. Fortran program to be output is stored in the file having name that `.omp` is inserted before suffix (`.f`, `.F`, etc.) of an input file. It is ignored if the `-Nlst=i` option is specified, or if `-Kparallel` option is not effective.

p

The `-Nlst=p` option specifies optimization information to be output along with listing of `-Nlst` option. Optimization information shows the situation of automatic parallelization, inlining and unrolling.

t

If the `-Nlst=t` option is specified, the details optimization information and the statistics information is output in addition to the compilation produced by the `-Nlst=p` option.

x

If the `-Nlst=x` option is specified, the cross reference list of name and label is output in addition to the compilation produced by the `-Nlst` option.

lst_out=*file*

The compilation information is output to the *file*.

{ mallocfree | nomallocfree }

These options specify whether or not to evaluate `MALLOC` and `FREE` as intrinsic procedures. The `-Nnomallocfree` is default.

mallocfree

-Nmallocfree option specifies to evaluate MALLOC and FREE as intrinsic procedures.

nomallocfree

-Nnomallocfree option specifies to evaluate MALLOC and FREE as service routines.

maxserious=*maxnum*

Stop the compilation if s-level (serious) errors are detected more than *maxnum* times. *maxnum* must be greater than or equal to 1. When this option is omitted, compilation does not stop even if s-level (serious) errors are detected.

{ obsfun | noobsfun }

These options specify whether or not to interpret the following functions as intrinsic functions. The -Nnoobsfun option is default.

obsfun

The -Nobsfun option specifies that the following functions are interpreted as intrinsic functions.

noobsfun

The -Nnoobsfun option specifies that the following functions are not interpreted as intrinsic functions.

Function Name:

AIMAX0, AJMAX0, I2MAX0, IMAX0, JMAX0, IMAX1, JMAX1, AIMIN0, AJMIN0, I2MIN0, IMIN0, JMIN0, IMIN1, JMIN1, FLOATI, FLOATJ, DFLOTI, DFLOTJ, IABS, JIABS, I2ABS, IIDIM, JIDIM, I2DIM, IIFIX, JIFIX, JFIX, INT1, INT2, INT4, INT8, IINT, JINT, ININT, JNINT, IIDNNT, I2NINT, JIDNNT, IIDINT, JIDINT, IMOD, JMOD, I2MOD, IISIGN, JISIGN, I2SIGN, BITEST, BJTEST, IIBCLR, JIBCLR, IIBITS, JIBITS, IIBSET, JIBSET, IBCHNG, ISHA, ISHC, ISHL, IAND, JIAND, IIEOR, JIEOR, IIOR, JIOR, INOT, JNOT, IISHFT, JISHFT, IISHFTC, JISHFTC, IZEXT, JZEXT, IZEXT2, JZEXT2, JZEXT4, VAL

{ privatealloc | noprivatealloc }

These options specify initial states of OpenMP private variables corresponding to allocated allocatable variables that are specified in the PRIVATE clause, etc.

These options are valid only when -Kopenmp option is in effect. The default is -Nnoprivatealloc.

privatealloc

The initial states of such private variables are unallocated.

This option may cause unworkable behavior to the OpenMP 3.0 standard or later to maintain backward compatibility.

noprivatealloc

The initial states of such private variables are allocated.

profile_dir=*dir_name*

This option specifies storage location directory of the .gcca file which is necessary for the use of the code coverage.

For *dir_name*, the storage location directory name is specified by the relative path or the absolute path.

The .gcca file is generated when the executable program linked with object programs containing information for the code coverage is executed. If the directory specified as *dir_name* does not exist at the execution time, the directory is generated.

This option requires that the -Ncoverage option and the -S or -c option are set.

For details about the code coverage, see section "[Appendix G Code Coverage](#)".

quickdbg[=*dbg_arg*] *dbg_arg*: {{ argchk | noargchk }} | {{ subchk | nosubchk }} | {{ undef | undefnan | noundef }} | {{ inf_detail | inf_simple }}

This function is used to debug Fortran source code. If the debug function is enabled, it checks for Fortran source errors during compilation. It also embeds in object programs the information required for debugging and performs automatic checks during execution.

Checks are performed if the -Nquickdbg=argchk, -Nquickdbg=subchk, -Nquickdbg=undef, or -Nquickdbg=undefnan options are set.

For *dbg_arg*, either *argchk*, *noargchk*, *subchk*, *nosubchk*, *undef*, *undefnan*, *noundef*, *inf_detail*, or *inf_simple* can be specified. These options can be used in combination by specifying the `-Nquickdbg[=dbg_arg]` option multiple times.

The `-Nquickdbg=inf_detail` and `-Nquickdbg=inf_simple` options are enabled when set together with either the `-Nquickdbg`, `-Nquickdbg=argchk`, `-Nquickdbg=subchk`, `-Nquickdbg=undef`, or `-Nquickdbg=undefnan` option.

Omitting *dbg_arg* is equivalent to specifying `-Nquickdbg=argchk,fastdbg=subchk,fastdbg=undef`.

If the `-Nquickdbg=argchk`, `-Nquickdbg=subchk`, `-Nquickdbg=undef`, or the `-Nquickdbg=undefnan` option is enabled, the `-Nquickdbg=inf_detail` and `-Ncheck_global` options are also enabled.

See "[8.2 Debugging Functions](#)" for detail.

If the `-H` option is enabled, disable the `-Nquickdbg` option.

{ *argchk* | *noargchk* }

These options specify whether or not to check the validity of the called and calling of procedure references.

This check is performed at compilation and at execution. See Section "[8.2.1.1 Checking Argument Validity \(ARGCHK\)](#)" for detail.

argchk

Checks the validity of arguments and result of procedure. If an error is detected, a diagnostic message is output.

noargchk

The validity of arguments and result of procedure are not checked.

{ *subchk* | *nosubchk* }

These options specify whether or not the validity of the declared expression and referenced expression are checked in array section, array element, and substring declarations and references.

This check is performed at execution. See Section "[8.2.1.2 Checking Subscript and Substring Values \(SUBCHK\)](#)" for details.

subchk

Checks the subscript and substring value. If an error is detected, a diagnostic message is output.

nosubchk

The subscript and substring value are not checked.

{ *undef* | *undefnan* | *noundef* }

These options specify whether or not to check that data is defined in references to module or submodule declaration parts and variables outside of common blocks.

This check is performed when the program is executed. See Section "[8.2.1.3 Checking References for Undefined Data Items \(UNDEF\)](#)" for details.

undef

Checks undefined data references. If an error is detected, a diagnostic message is output.

undefnan

Checks undefined data references. When this option is specified, real type and complex type errors are detected as invalid operation exceptions. When an error is detected, either a diagnostic message or an invalid operation exception message is output, depending on the variable type.

noundef

Undefined data reference checks are not performed.

{ *inf_detail* | *inf_simple* }

This option specifies the information to be included in diagnostic messages output when errors are detected.

See Section "[8.2.1 Debugging Check Functions](#)" for details.

inf_detail

In addition to the message and line number where the error occurred, information is output such as the variable name, procedure name, and element location in order to identify the cause.

However, when `-Nquickdbg=undefnan` option is specified, for errors detected as invalid operation exceptions, only the error and the line number where it occurred are output in diagnostic messages.

inf_simple

The error and the line number where it error occurred are output in diagnostic messages.

{ recursive | norecursive }

These options specify whether or not to append the `RECURSIVE` keyword to subroutine subprograms and function subprograms. The default is `-Nnorecursive`.

recursive

The `-Nrecursive` option specifies to append the `RECURSIVE` keyword to subroutine subprograms and function subprograms which are not elementary procedures.

When the `-Nrecursive` option is effective, the program does not conform to the Fortran standard. It is recommended to modify the source program to include the `RECURSIVE` keyword in the recursive invocation procedure.

norecursive

The `-Nnorecursive` option specifies not to append the `RECURSIVE` keyword to subroutine subprograms and function subprograms.

{ reordered_variable_stack | noreordered_variable_stack }

The `-Nreordered_variable_stack` option directs the compiler the order in which to allocate the automatic variables to the stack area. The default is `-Nnoreordered_variable_stack`. The order of allocation is not guaranteed when the `-Nnoline`, `-g0`, or `-Kocl` option is set. For details, see Section "[E.4 Data Allocation to Stack Region](#)".

reordered_variable_stack

The `-Nreordered_variable_stack` option specifies to determine the allocation order of automatic variables is determined in the following order.

1. Descending order of their alignments
2. Descending order of data sizes if the alignments are equal
3. The order of appearance of the declaration statements in the source program if both the alignments and data sizes are equal

The stack area of the entire program can be reduced by allocating the automatic variables in descending order of their alignments.

noreordered_variable_stack

The `-Nnoreordered_variable_stack` option specifies that automatic variables are allocated in order of the appearance of the declaration statements in the source program.

{ rt_tune | rt_notune }

These options specify whether or not to output the runtime information. The `-Nrt_notune` option is default.

The runtime information can be used for the tuning of user program. See "[Appendix H Runtime Information Output Function](#)" for details.

rt_tune

The `-Nrt_tune` option specifies to output the runtime information.

rt_notune

The `-Nrt_notune` option specifies not to output the runtime information.

rt_tune_func

In addition to the `-Nrt_tune` output, runtime information about a user-defined functions is output.

This option induces the `-Nrt_tune` option. When the `-Nrt_notune` option is specified after `-Nrt_tune_func` option, `-Nrt_tune_func` option will be invalid.

See "[Appendix H Runtime Information Output Function](#)" for details.

`rt_tune_loop[=kind] kind: { all | innermost }`

In addition to the `-Nrt_tune` output, runtime information about loops is output.

All or innermost can be specified in the argument *kind*. If the *kind* is not specified, all is used.

This option induces the `-Nrt_tune` option. When the `-Nrt_notune` option is specified after `-Nrt_tune_loop` option, `-Nrt_tune_loop` option will be invalid.

See "[Appendix H Runtime Information Output Function](#)" for details.

`rt_tune_loop=all`

Runtime information about all loops is output.

`rt_tune_loop=innermost`

Runtime information about the innermost loops is output.

`{ save | nosave }`

These options specify whether or not to add the SAVE statement without saved entity list in program unit. The `-Nnosave` option is default.

`save`

The `-Nsave` option specifies that the SAVE statement without saved entity list is added in each program unit except main program. It is ignored if the `-Nrecursive` or `-Kauto` option is specified.

`nosave`

The `-Nnosave` option specifies that the SAVE statement without saved entity list is not added in program unit. The `-Nnosave` option is default.

`{ setvalue[=set_arg] | nosetvalue }`

`set_arg: { { heap | noheap } | { stack | nostack } | { scalar | noscalar } | { array | noarray } | { struct | nostruct } }`

The options specify whether or not to automatically set zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to stack or heap. The `-Nnosetvalue` is default.

Note that execution time may increase for the initialization.

If the `-Eg`, `-H` or `-Nquickdbg` option is specified, the `-Nsetvalue` option is ignored.

`setvalue[=set_arg]`

The `-Nsetvalue` option specifies to automatically set zero value in variables.

Either heap, noheap, stack, nostack, scalar, noscalar, array, noarray, struct or nostruct can be specified in the *set_arg*.

These options can be used in combination by specifying the `-Nsetvalue=set_arg` option multiple times.

If the *set_arg* is omitted, it is equivalent to the following options:

`-Nsetvalue=heap,setvalue=stack,setvalue=scalar,setvalue=array,setvalue=struct`

`{ heap | noheap }`

These options specify whether or not to automatically set zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to heap.

The zero value is not set to the allocated area by service subroutine MALLOC and intrinsic function MALLOC. The zero value is not set in the allocatable variable at head of the OpenMP directive block when the allocatable variable is specified by PRIVATE or LASTPRIVATE clause of OpenMP directive.

The value is set in the following variables:

- Automatic data objects when `-Knoautoobjstack` option is effective.
- Allocatable variables.
- Pointer variables.

setvalue=heap

The -Nsetvalue=heap option specifies to automatically set the zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to heap.

When the setvalue=heap is specified, the -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct options are in effect.

setvalue=noheap

The -Nsetvalue=noheap option does not apply to automatically set the zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to heap.

{ stack | nostack }

These options specify whether or not to automatically set the zero value by the procedure entrance in variables that allocated to stack.

The value is set in the following variables:

- Automatic data objects when -Kautoobjstack option is effective.
- Local variables without SAVE attribute when -Kauto option is effective.
- Local variables in program unit that has RECURSIVE or PURE.
- Local variables that declare by AUTOMATIC attribute or AUTOMATIC statement.
- Local variables that specify by PRIVATE clause or LASTPRIVATE clause in OpenMP directive.

setvalue=stack

The -Nsetvalue=stack option specifies to automatically set the zero value by the procedure entrance in variables that are allocated to stack.

But, the zero value is set at head of the OpenMP directive block when the specifying variables by PRIVATE clause or LASTPRIVATE clause.

When the setvalue=stack is specified, the -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct options are in effect.

setvalue=nostack

The -Nsetvalue=nostack option does not apply to automatically set the zero value by the procedure entrance in variables that are allocated to stack.

{ scalar | noscalar }

These options specify whether or not to automatically set the zero value in scalar variables of numeric types and logical types.

setvalue=scalar

The -Nsetvalue=scalar option specifies to automatically set the zero value in scalar variables of numeric types and logical types.

setvalue=noscalar

The -Nsetvalue=noscalar option does not apply to automatically set the zero value in scalar variables of numeric types and logical types.

{ array | noarray }

These options specify whether or not to automatically set the zero value in array variables of numeric types and logical types.

setvalue=array

The -Nsetvalue=array option specifies to automatically set the zero value in array variables of numeric types and logical types.

setvalue=noarray

The -Nsetvalue=noarray option does not apply to automatically set the zero value in array variables of numeric types and logical types.

{ struct | nostruct }

These options specify whether or not to automatically set the zero value in scalar and array variables of derived type and character type.

setvalue=struct

The -Nsetvalue=struct option specifies to automatically set the zero value in scalar and array variables of derived type and character type.

The variable of derived type having length type parameter is not applied.

setvalue=nostruct

The -Nsetvalue=nostruct option does not apply to automatically set the zero value in scalar and array variables of derived type and character type.

nosetvalue

The -Nnosetvalue option does not apply to automatically set zero value in variables.

The option is equivalent to the following options:

-Nsetvalue=noheap,setvalue=nostack,setvalue=noscalar,setvalue=noarray,setvalue=nostruct

Note the following:

1. When the -Nsetvalue option is in effect, it is necessary to be the following conditions:
 - The -Nsetvalue=heap or -Nsetvalue=stack is in effect, and
 - The -Nsetvalue=scalar, -Nsetvalue=array or -Nsetvalue=struct is in effect.
2. If the -Nsetvalue=heap or -Nsetvalue=stack is specified, -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct are generated. Therefore, when the -Nsetvalue=noscalar, -Nsetvalue=noarray or -Nsetvalue=nostruct will be in effect, the option should be specified after -Nsetvalue=heap or -Nsetvalue=stack.
3. When the zero value is set in only scalar variable of numeric types and logical types for stack, it is necessary to be the following options:

-Nsetvalue=stack,setvalue=noarray,setvalue=nostruct

When the -Nsetvalue=stack is specified, the -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct are generated. Therefore, if -Nsetvalue=stack,setvalue=scalar are specified, the option specifying is not limited to scalar variables alone.

{ use_rodata | nouse_rodata }

These options specifies whether or not to allocate constants which are specified as actual arguments to write inhibit area when the actual arguments may be written through the associated dummy arguments. The -Nuse_rodata option is default.

use_rodata

The -Nuse_rodata option specifies to allocate constants which are specified as actual arguments to write inhibit area.

nouse_rodata

The -Nnouse_rodata option specifies not to allocate constants which are specified as actual arguments to write inhibit area.

{ Rtrap | Rnotrap }

The compiler option -NRtrap specifies to detect the intrinsic instructions errors and floating-point exceptions in execution, and if the intrinsic instructions errors or floating-point exceptions are found, the runtime messages are outputted. The compiler option -NRtrap should be specified when programs are compiled and linked. This option is not effective for a constant expression that is operated at compilation time. The compiler option -NRnotrap is the default.

See Section ["8.1.5 Intrinsic Function Error Processing"](#) for the intrinsic instructions errors.

See Section ["8.1.7 Exception Handling Processing"](#) and ["8.2.2.1 Causes of Abend"](#) for floating-point exceptions.

Note that the integer divide by zero is not detected in the Fujitsu CPU A64FX with Arm architecture. See Section ["A.2.3 Integer Division Exception when Divisor Is Zero"](#) for details.

Rtrap

If a main program unit is compiled with the compiler option -NRtrap, the following errors are detected.

- The intrinsic instructions errors (from jwe0259i-e to jwe0282i-e, jwe1397i-e, jwe1398i-e, jwe1413i-e, jwe1414i-e, jwe1416i-e, and jwe1417i-e):

The system checks arguments in execution, and outputs messages if any errors are found in the check results.

- The floating-point exceptions (jwe1017i-u):

The system handles with enabling traps corresponding to signals to be detect, and outputs a message when an exception is handled. The floating-point underflow exception is trapped when the environment variable FLIB_EXCEPT=u have been specified.

If the compiler option -NRtrap is used together with the compiler options -Kpreex and -Ksimd=2, the speculative execution may cause exceptions that would not normally occur.

When the compiler option -Kfp_relaxed is specified, if the compiler option -NRtrap is effective and either of the compiler options -Knosimd or -KNOSVE is effective, a floating-point exception may occur. However, to avoid a floating-point divide-by-zero exception for SQRT(0.0), the optimization to convert to reciprocal approximation instructions is suppressed for the SQRT function. Therefore, the execution performance may decrease as compared with when the compiler option -NRnotrap is effective.

For the messages, see "Fortran/C/C++ Runtime Messages" for details.

Rnotrap

If a main program is compiled with the compiler option -NRnotrap, the intrinsic instructions errors and the floating-point exceptions are not detected and runtime messages are not output.

-O [*opt_lv*] *opt_lv*: { 0 | 1 | 2 | 3 }

The -O option specifies the optimization level used by the compiler.

The system provides four optimization levels: 0, 1, 2 and 3. If the argument is omitted, level 3 is used. If the -O option is not specified, level 2 is used.

See "[Chapter 9 Optimization Functions](#)" for information about the optimization of source programs.

In addition, the following optimization facilities can be specified:

- The advance evaluation of invariant expressions and the evaluation strategy can be changed by specifying -K series option
- It can indicate the inline expansion of user-defined procedure by specifying -x option.

0

The -O0 option creates a source program without applying optimizations. A program compiled with the -O0 option requires the least compilation time and memory. Specify this argument to debug compilation errors in Fortran source programs.

1

The -O1 option creates a source program by applying basic optimization. The run time of an executable program will be shorter and the object program size will be smaller if it is created with the -O1 option than if it is created with the -O0 option.

2

When the -O2 option is effective, following optimizations are applied in addition to the optimizations applied by the -O1 option.

- Loop unrolling (-Kunroll)
- Software pipelining (-Kswp)
- Loop blocking (-Kloop_blocking)
- Loop fusion (-Kloop_fusion)
- Loop fission (-Kloop_fission)
- Exchange Loop (-Kloop_interchange)
- Using prefetch instructions (Equivalent to -Kprefetch_sequential,prefetch_cache_level=all)
- Using SIMD instructions (-Ksimd=auto)
- Adjusting loop alignment (-Kalign_loops)
- Optimize sibling calls (-Ksibling_calls)

- Repeat basic optimization

The -O2 option repeats optimizations applied by the -O1 option until it admits of no optimization.

Note that compile time and object program size may be increase compared to the -O1 option. To determine whether each optimization was applied, specify the -Koptmsg=2 option.

3

When the -O3 option is effective, following optimizations are applied in addition to the optimizations applied by the -O2 option.

- Unrolling nested loops
- Transforming loops into perfectly nested loops to promote optimization such as loop interchange (-Kloop_perfect_nest)
- Loop unswitching
- CLONE Optimization

Note that compile time and object program size may be increase compared to the -O2 option.

To determine whether each optimization was applied, specify the -Koptmsg=2 option.

-P

The -P option specifies to perform only preprocessing to the specified source file. The result of preprocessing is output to the temporary file.

The -Cpp option must be specified together unless a suffix of the source file is .F, .FOR, .F90, .F95, .F03, or .F08. The -E option cannot be specified together.

If this option is set, only preprocessor is performed. This option cannot be specified with -E.

The temporary files are generated are as follows:

Source file	Result file of preprocessing
<i>file.F</i>	<i>file.cpp.f</i>
<i>file.f</i>	
<i>file.FOR</i>	<i>file.cpp.for</i>
<i>file.for</i>	
<i>file.F90</i>	<i>file.cpp.f90</i>
<i>file.f90</i>	
<i>file.F95</i>	<i>file.cpp.f95</i>
<i>file.f95</i>	
<i>file.F03</i>	<i>file.cpp.f03</i>
<i>file.f03</i>	
<i>file.F08</i>	<i>file.cpp.f08</i>
<i>file.f08</i>	

-S

The -S option suppresses creation of the corresponding object programs and does not call the linker. Instead, assembly programs (files with the suffix .s) are created. If only assembly source files or object files are specified by file, it is meaningless to invoke the compile command.

{ -SSL2 | -SSL2BLAMP }

These options relate to linking with Fujitsu's math libraries (SSL II, BLAS, LAPACK). The details are described in "SSL II Online Documents", "SSL II Thread-Parallel Capabilities Online Documents" and "BLAS, LAPACK, ScaLAPACK Online Documents". Following is just an outline.

-SSL2

The whole set of routines from SSL II, SSL II Thread-Parallel Capabilities and BLAS/LAPACK becomes part of link libraries.

-SSL2BLAMP

The whole set of routines from SSL II, SSL II Thread-Parallel Capabilities and BLAS/LAPACK Thread-Parallel versions becomes part of link libraries (i.e. -SSL2BLAMP just replaces the sequential BLAS/LAPACK from -SSL2 with the corresponding thread-parallel versions.)

-U *name*

The -U option undefines *name*, which has the same effect as an #undef preprocessing directive. If the same name is specified for both -D and -U, *name* is not defined regardless of the order of the options.

-V

The -V option displays the version and release information on each command.

-W *tool, arg1[, arg2]...*

Passes each argument *arg1[, arg2]...* as a separate argument to a tool. The arguments must be separated by commas (A comma can be part of an argument by using a backslash as an escape character before it. The backslash is removed from the resulting argument).

tool can be one of the following:

Value	Meaning
p	Preprocessor
0	Compiler
a	Assembler
l	Linker

For example, -Wa,-oobjfile passes -o and objfile to the assembler, in that order. Also, -Wl,-lname causes the linking phase to override the default name of the dynamic linker. For other -W options, arguments passed to a tool may not be in order.

-X *lan_lvl lan_lvl*: { 6 | 7 | 9 | 03 | 08 | d7 }

The -X option directs the level of language specification. Either 6, 7, 9, 03, 08 or d7 can be specified as the argument of the -X option.

The interpretation of Fortran source programs is different for each level of language specification. To compile Fortran source programs, indicate the level of language specification used by the Fortran compiler. See Section "6.1 Fortran Versions" for information on the difference between languages.

6

The -X6 option specifies to compile Fortran source programs as Fortran 66 source.

7

The -X7 option specifies to compile Fortran source programs as Fortran 77 source.

9

The -X9 option specifies to compile Fortran source programs as Fortran 95 or Fortran 90 source. Activates -Nf90move option at the same time.

03

The -X03 option specifies to compile Fortran source programs as Fortran 2008 source. It is same as -X08 option. Activates -Nf90move option at the same time. When the -X03 option is in effect, the -Nalloc_assign option is set by default.

08

The -X08 option specifies to compile Fortran source programs as Fortran 2008 source. Activates -Nf90move option at the same time. When the -X08 option is in effect, the -Nalloc_assign option is set by default.

d7

If the version of the language is not specified, and the suffix of the Fortran source file is .f, .for, .F or .FOR, -X7 is the default.

-#

The -# option specifies to output the name of each path and options used by frtpx command, but does not execute any of the tools.

-###

The -### option specifies to output the name of each path and options used by frtpx command as it executes.

--linkstl=*stl_kind* *stl_kind* : { libfjcpp | libc++ | libstdc++ }

Directs to search for a Standard Template Library (STL) required for linkage with C++ programs. For details, see "[Chapter 11 Mixed Language Programming](#)" for information about Mixed Language Programming. This option must be set at linking.

libfjcpp

Search the STL used by C++ compiler in Trad Mode.

libc++

Search the libc++ used by C++ compiler in Clang Mode.

The libc++ is the STL used by C++ compiler in Clang Mode when specifying the -stdlib=libc++ option.

libstdc++

Search the libstdc++ used by C++ compiler in Clang Mode.

The libstdc++ is the STL used by C++ compiler in Clang Mode by default. Also, used when -stdlib=libstdc++ option is specified.

If C++ objects that linked with Fortran programs using STL and the STL specified in the suboption of this option does not match, one of the following may occur.

- Link error at linking
- Indefinite behavior at runtime

2.3 Compile Command Environment Variable

It explains the environment variables that the compile commands recognize as follows.

FORT90CPX

FORT90C

Environment variables to set compiler options. The environment variable FORT90CPX is for the cross compiler and the environment variable FORT90C is for the native compiler.

The user can specify compiler options for the value of these environment variables.

It is useful to set compiler options that you always specify for these environment variables.



Example

Example 1 and Example 2 are equivalent.

Example 1: Specifying compiler options to the environment variable FORT90CPX

```
$ export FORT90CPX="-fw -I/usr/prv/usri"
$ frtpx a.f90
```

Example 2: Specifying compiler options to the operand of the compile command

```
$ frtpx -fw -I/usr/prv/usri a.f90
```

FCOMP_LINK_FJOB

In this system, original objects of this system are usually combined at linking, but they are not combined under the following conditions.

- The -L option which is passed to the linker by this system is directly specified on the compile command to pass the linker by user.

This can result in a link error (undefined reference to), which can be avoided by setting the environment variable FCOMP_LINK_FJOB.

The environment variable FCOMP_LINK_FJOB can have any value. When the environment variable FCOMP_LINK_FJOB is set, original objects of this system are combined at linking.

For the example of a link error, see "A.2.4 Link Error (undefined reference to)".

Example

Example of using the environment variable FCOMP_LINK_FJOB

```
$ export FCOMP_LINK_FJOB=true
```

FCOMP_UNRECOGNIZED_OPTION

The behavior for unrecognized compiler options can be changed by specifying the environment variable FCOMP_UNRECOGNIZED_OPTION.

Specify either warning or error to the environment variable FCOMP_UNRECOGNIZED_OPTION. If the environment variable FCOMP_UNRECOGNIZED_OPTION is not set or an invalid value is set to it, warning is set.

Value	Description
warning	A warning message is output for an unrecognized compiler option and the compilation is continued.
error	An error message is output for an unrecognized compiler option and the compilation is canceled.

Example

Example 1: Specifying warning to the environment variable FCOMP_UNRECOGNIZED_OPTION

```
$ FCOMP_UNRECOGNIZED_OPTION=warning
$ export FCOMP_UNRECOGNIZED_OPTION
$ frtpx -unrecognized_option a.f90
frtpx: warning: -unrecognized_option is unrecognized option.
$ echo $?
0
```

Example 2: Specifying error to the environment variable FCOMP_UNRECOGNIZED_OPTION

```
$ FCOMP_UNRECOGNIZED_OPTION=error
$ export FCOMP_UNRECOGNIZED_OPTION
$ frtpx -unrecognized_option a.f90
frtpx: error: -unrecognized_option is unrecognized option.
$ echo $?
1
```

LIBRARY_PATH

The directory to be searched for libraries at linking can be added by specifying a value for the environment variable LIBRARY_PATH.

Multiple directories can be specified using colons to separate them.

Libraries are searched in the following order:

1. Directories specified as the argument of the -L option
2. The directory installed libraries provided by the compiler
3. The directory installed standard libraries
4. Directories specified to the environment variable LIBRARY_PATH

Example

Example to use the environment variable LIBRARY_PATH

```
$ export LIBRARY_PATH="/usr/local/lib64:/usr/local_2/lib64"
```

TMPDIR

The temporary directory used by the compile command can be changed by specifying a value for the environment variable TMPDIR.

If the environment variable TMPDIR is not set, /tmp is used. To avoid output to a common directory, set the local directory to the environment variable TMPDIR.

Example

Example to use the environment variable TMPDIR

```
$ export TMPDIR=/usr/local/tmp
```

2.4 Compilation Profile File

The defaults of compilation options can be changed by specifying the compilation profile file (/etc/opt/FJSVxtclanga/jwd_prof).

Note

For contents about the settings in the compilation profile file, contact the system administrator.

The format of a compilation profile file is as follows:

- Blank characters not enclosed quotation mark (") or single quotation mark (') are not significant.
- Either single or double quotation marks can be used to begin and end character strings. A character string is terminated by the same delimiter with which the string began. A character string is also terminated by the end of a line.
- A symbol (#) that is not part of a character string starts a comment. Any characters from the # to the end of the line are treated as part of the comment.

Example

Example of a compilation profile file specification

```
#Default options  
-fW -Kocl
```

Compiler options are determined by the following priorities:

1. Compiler directive lines (the -Koptions is specified)
2. Operand of the compile command
3. Environment variables to set compiler options
4. Compilation profile file
5. Default value

2.5 Return Values during Compilation

The following table lists the return values set by the frtpx(1) command.

Return value	Status
0	Normal termination
Non-zero	Compile or linker error

2.6 Compiler Directive Lines

If !options is specified the first in Fortran source program and compiler option -Koptions is effective, options in !options become effective.

Form of compiler directive lines:

```
!options opt [,opt] ...
```

opt: compiler options

The -O[1-3] options can be specified as compiler options.

Example:

```
1---5----10---15---20---25
!options -O2
subroutine sub
```

Notes:

!options cannot be specified the first in module procedure or internal subprogram.

Chapter 3 Executing Fortran Programs

This chapter describes the procedures for executing Fortran programs.

3.1 Execution Command

An execution command is used to execute the executable program file created by the `frtpx` compile command.

3.2 Execution Command Format

Runtime options and user-defined executable program options may be specified as command option arguments of an execution command. The runtime options use functions supported by the Fortran library.

See Section "3.3 Runtime Options", for information on the runtime option.

The format of runtime options is as follows:

```
exe_file [-Wl,-runtime option[,-runtime option]...] [-user-defined
          executable program option[,-user-defined executable program
          option]...]
```

Notes:

1. `exe_file` indicates the executable program file.
2. The `GETPARM` and `GETARG` service subroutines fetch user-defined executable program options other than runtime options.
3. If an option is specified more than once with different arguments, the last occurrence is used.

The following example shows how to specify the runtime option `-Wl,-i` and user-defined executable program options `-K` and `-L` as command option arguments of a command named `a.out`.

```
$ ./a.out -Wl,-i -K,-L
```

3.3 Runtime Options

Runtime options may be specified as arguments for execution commands or in the `FORT90L` environment variable. This section explains the format and functions of the runtime options.

The runtime option format is as follows:

```
-Wl[,-a] [,-dnum] [,-enum] [,-gnum] [,-i] [,-l $\ell$ lV] [,-mu_no] [,-n] [,-pu_no] [,-q] [,-ru_no] [,-tsec] [,-x] [,-Lb] [,-Li] [,-Lr] [,-Lu] [,-Q] [,-Re] [,-Rp] [,-Ry] [,-Tu_no]
```

When runtime options are specified, `-Wl` (l is lowercase) is required at the beginning of the runtime options, and the options must be separated by commas. If the same runtime option is specified more than once with different arguments, the last occurrence is used.

Example: Runtime option

```
$ ./a.out -Wl,-a,-p10,-x
```

`-a`

When the `-a` option is specified, an `abend` is executed forcibly following normal program termination. This processing is executed immediately before closing external files.

Example: Runtime option `-a`

```
$ ./a.out -Wl,-a
```


-dnum 1 <= num <= 32767

The -d option determines the size of the input/output buffer used by a direct access input/output statement. The -d option improves input/output performance when data is read from or written to files a record at a time in sequential record-number order. If the -d option is specified, the input/output buffer size is used for all units used during execution.

To specify the size of the input/output buffer for individual units, specify the number of Fortran records in the environment variable `fu xx bf` (xx is the unit number). See Section "3.8 Using Environment Variables for Execution" for details. When the -d option and the environment variable are specified at the same time, the environment variable becomes effective. The option argument `num` specifies the number of Fortran records, in fixed-block format, included in one block. The option argument `num` must be an integer from 1 to 32767. The other integer value is specified, the -d option is invalid. To obtain the input/output buffer size, multiply `num` by the value specified in the `RECL=` specifier of the `OPEN` statement. If the files are shared by several processes, the number of Fortran records per block must be 1. If the -d option is omitted, the size of the input/output buffer is 8M bytes.

Example: Runtime option -d

```
$ ./a.out -w1,-d10
```

-enum 0 <= num <= 32767

The -e option controls termination based on the total number of execution errors. The option argument `num`, specifies the error limit as an integer from 0 to 32767. When `num` is greater than or equal to 1, execution terminates when the total number of errors reaches the limit. If -enum is omitted or `num` is zero, execution is not terminated based on the error limit. However, program execution still terminates if the Fortran system error limit is reached.

Example: Runtime option -e

```
$ ./a.out -w1,-e10
```

-gnum 0 <= num <= 2097151

The -g option sets the size of the input/output buffer used by a sequential access input/output statement or a stream access input/output statement. This size is set in units of kilobytes for all unit numbers used during execution. The argument `num` must be an integer from 1 to 2097151. The other value is specified, the -g option is invalid. If the -g option is omitted, the size of the input/output defaults to 8M bytes.

The -g option improves input/output performance when a large amount of data is read from or written to files by an unformatted sequential access input/output statement or an unformatted stream access input/output statement. The argument `num` is used as the size of the input/output buffer for all units. To avoid using excessive memory, specify the size of the input/output buffer for individual units by specifying the size in the environment variable `fu xx bf` (xx is the unit number). See Section "3.8 Using Environment Variables for Execution" for details. When the -g option is and the environment variable are specified at the same time, the environment variable becomes effective.

Example: Runtime option -g

```
$ ./a.out -w1,-g10
```

-i

The -i option controls processing of runtime interrupts. When the -i option is specified, the Fortran library is not used to process interrupts. When the -i option is not specified, the Fortran library is used to process interrupts. Diagnostic messages are output. For the error check, see Section "8.2.2 Debugging Programs for Abend" for details.

For the interrupt, see Section "8.1.7 Exception Handling Processing" for details.

Example: Runtime option -i

```
$ ./a.out -w1,-i
```

-levl *evl*: { i | w | e | s }

The -l option controls the output of diagnostic messages during execution. The option argument `evl`, specifies the lowest error level, i, w, e, or s, for which diagnostic messages are to be output. If the -l option is not specified, diagnostic messages are output for error levels w, e, and s. However, messages beyond the print limit are not printed.

Example: Runtime option -l

```
$ ./a.out -w1,-le
```

i

The -li option outputs diagnostic messages for all error levels.

w

The -lw option outputs diagnostic messages for error levels w, e, s, and u.

e

The -le option outputs diagnostic messages for error levels e, s, and u.

s

The -ls option outputs diagnostic messages for error levels s and u.

-mu_no 0 <= *u_no* <= 2147483647

The -m option connects the specified unit number *u_no* to the standard error output file where diagnostic messages are to be written. If the -m option is omitted, unit number 0, the system default, is connected to the standard error output file. See Section "3.8 Using Environment Variables for Execution", and Section "7.2 Unit Numbers and File Connection" for details.

Example: Runtime option -m

```
$ ./a.out -w1,-m10
```

-n

The -n option controls whether prompt messages are sent to standard input. When the -n option is specified, prompt messages are output when data is to be entered from standard input using formatted sequential READ statements, including list-directed and namelist statements. If the -n option is omitted, prompt messages are not generated when data is to be entered from standard input using a formatted sequential READ statement.

Example: Runtime option -n

```
$ ./a.out -w1,-n
```

-pu_no 0 <= *u_no* <= 2147483647

The -p option connects the unit number *u_no* to the standard output file. If the -p option is omitted, unit number 6, the system default, is connected to the standard output file. See Section "3.8 Using Environment Variables for Execution", and Section "7.2 Unit Numbers and File Connection" for details.

Example: Runtime option -p

```
$ ./a.out -w1,-p10
```

-q

The -q option specifies whether to capitalize the E, EN, ES, D, Q, G, L and Z edit output characters produced by formatted output statements. This option also specifies whether to capitalize the alphabetic characters in the character constants used by the inquiry specifier (excluding the NAME= specifier) in the INQUIRE statement. If the -q option is specified, the characters appear in uppercase letters. If the -q option is omitted, the characters appear in lowercase letters. In Fortran 95 and Fortran 2003, the characters appear in uppercase letters so the -q option is not required.

Example: Runtime option -q

```
$ ./a.out -w1,-q
```

-ru_no 0 <= *u_no* <= 2147483647

The -r option connects the unit number *u_no* to the standard input file during execution. If the -r option is omitted, unit number 5, the system default, is connected to the standard input file. See Section "3.8 Using Environment Variables for Execution", and Section "7.2 Unit Numbers and File Connection" for details.

Example: Runtime option -r

```
$ ./a.out -w1,-r10
```

-tsec 1 <= sec <= 9999

The -t option sets the time limit in seconds for program execution. If -t20 is specified, for example, execution is canceled after 20 seconds.

Example: Runtime option -t

```
$ ./a.out -W1,-t20
```

-x

The -x option determines whether blanks in numeric edited input data are ignored or treated as zeros. If the -x option is specified, blanks are changed to zeros during numeric editing with formatted sequential input statements for which no OPEN statement has been executed. The result is the same as when the BLANK= specifier in an OPEN statement is set to zero. If the -x option is omitted, blanks in the input field are treated as null and ignored. The result is the same as if the BLANK= specifier in an OPEN statement is set to NULL or if the BLANK= specifier is omitted.

Example: Runtime option -x

```
$ ./a.out -W1,-x
```

-Lb

Service routines use eight-byte logicals instead of four-byte logicals as arguments and function results.

Example: Runtime option -Lb

```
$ ./a.out -W1,-Lb
```

-Li

Service routines use eight-byte integers instead of four-byte integers as arguments and function results.

Example: Runtime option -Li

```
$ ./a.out -W1,-Li
```

-Lr

Service routines use eight-byte reals instead of four-byte reals as arguments and function results.

Example: Runtime option -Lr

```
$ ./a.out -W1,-Lr
```

-Lu

If the length of the unformatted Fortran record to be transferred in a sequential access unformatted input/output statement is 2G bytes or more, input/output is by dividing it into multiple Fortran records. If the -Lu option is specified, the size of the field which exists in the top and the end of the logical record and which is assigned the length of Fortran record is expanded to 8 bytes from 4 bytes, and can be input/output as single Fortran record even if the length of the Fortran record is 2G bytes or more.

The data which has written to the -Lu option can be read only by unformatted sequential input statements with the -Lu option. And the data which has written without the -Lu option can be read only by that statements without the -Lu option.

If the -Lu option is specified differently across writing and reading the data, the unformatted sequential input statement may not work.

Example: Runtime option -Lu

```
$ ./a.out -W1,-Lu
```

-Q

The -Q option suppressed padding of an input field with blanks when a formatted input statement is used to read a Fortran record. This option applies to cases where the field width needed in a formatted input statement is longer than the length of the Fortran record and the file was not opened with an OPEN statement. The result is the same as if the PAD= specifier in an OPEN statement is set to NO. If the -Q option is omitted, the input record is padded with blanks. The result is the same as when the PAD= specifier in an OPEN statement is set to YES or when the PAD= specifier is omitted.

Example: Runtime option -Q

```
$ ./a.out -Wl,-Q
```

-Re

Disables the runtime error handler. The diagnostic message, traceback, error summaries, user control of errors by ERRSET and ERRSAV, and execution of user code for error correction are suppressed. The standard correction is processed if an error occurs.

This runtime option is invalidated for the runtime error handler of LLVM OpenMP Library. See "[12.4 Runtime Messages](#)" for the runtime error handler of LLVM OpenMP Library.

Example: Runtime option -Re

```
$ ./a.out -Wl,-Re
```

-Rp

The -Rp option issues a diagnostic message under certain circumstances for programs running on multiple processors.

If the -Rp option is specified, a diagnostic message (jwe1034i-w) is output when a program compiled with the compiler option -Kparallel runs on only one processor.

If the -Rp option is not specified, the diagnostic message (jwe1034i-w) is not output.

This runtime option is invalidated for LLVM OpenMP Library.

Example: Runtime option -Rp

```
$ ./a.out -Wl,-Rp
```

-Ry

A diagnostic message may be output when a program used service routines that return the year using only the last two digits.

The last two digits of the year are returned when one of the following procedures is used.

- DATE service subroutine
- JDATE service function

If the return values are compared by a user program, a malfunction may occur. Thus the program should receive proper consideration. By specifying the runtime option -Ry,-li, the diagnostic message is output when these routines are executed.

The routines that return a four-digit number for the year are listed below:

- CTIME service function
- FDATE service subroutine
- IDATE service subroutine
- GETDAT service subroutine
- DATE_AND_TIME intrinsic subroutine

Example: Runtime option -Ry

```
$ ./a.out -Wl,-Ry,-li
```

-T or -Tu_no 0 <= u_no <= 2147483647

Big endian integer data, logical data, and IEEE 754 floating-point data is transferred in an unformatted input/output statement.

The argument *u_no* is a unit number connected with an unformatted file. If *u_no* is omitted, -T takes effect for all unit numbers. If both -T and -Tu_no are specified as in the example, "-Wl,-T,-T10", -T takes effect for all unit numbers.

Example: Runtime option -T

```
$ ./a.out -Wl,-T10
```

3.4 Execution Command Environment Variable

The environment variable FORT90L may be used to specify runtime options. The runtime options specified in FORT90L are combined with the execution command option arguments. The execution command option arguments take precedence over the corresponding options specified in the environment variable FORT90L.

The following examples show how to use the environment variable FORT90L.

Example 1:

- a. When using the environment variable FORT90L

```
$ export FORT90L='-Wl,-e99,-le'  
$ ./a.out -Wl,-m99 -k
```

- b. When not the using environment variable FORT90L

```
$ ./a.out -Wl,-e99,-le,-m99 -k
```

Both two above examples have the same meaning. When executing the command a.out, runtime options -e99, -le, and -m99, and user-defined executable program option -k are in effect.

Example 2:

```
$ export FORT90L='-Wl,-e10'  
$ ./a.out -Wl,-e99
```

When executing execution command a.out, runtime option -e99 is in effect.

3.5 Execution Profile File

The defaults of runtime options can be changed by specifying the execution profile file (/etc/opt/FJSVxtclanga/jwe_prof).



For contents about the settings in the execution profile file, contact the system administrator.

The format of an execution profile file is as follows:

- Lines beginning with the pound sign (#) are treated as comment lines.
- A single runtime option cannot extend over multiple lines.
- The format of runtime options that can be specified must follow the conventions explained in Section "3.3 Runtime Options".



Example of an execution profile file specification

```
#Default options  
-Wl,-x,-le
```

Runtime options are determined by the following priorities:

1. Execution command operand
2. Environment variable
3. Execution profile file
4. Default value

3.6 Execution Command Return Values

The following table lists return values set by the execution command.

Return value	Status
0	No error or level i (information message)
4	Level w error (warning)
8	Level e error (medium)
12	Level s error (serious)
16	Limit exceeded for level w, e, s error, or a level u error (unrecoverable) was detected
240	Abnormal termination
Others	Forcible termination

3.7 Standard Input, Output, and Error Output for Execution

The default unit numbers for standard input, output, and error output for execution commands are:

Standard input : Unit number 5

Standard output : Unit number 6

Standard error output : Unit number 0

3.8 Using Environment Variables for Execution

This section describes environment variables that control execution.

fuxx filen

The *fuxx* environment variable connects units and files. The value *xx* is a unit number. The value *filen* is a filename to be connected to a unit number. See Section "7.2 Unit Numbers and File Connection" for information on connecting units and files. The standard input and output files (fu05 and fu06) and error file (fu00) must not be specified.

The following example shows how to connect data.file to unit number 10 prior to the start of execution.

```
$ export fu10="data.file"
```

fuxxbf size

The *fuxxbf* environment variable specifies the size of the input/output buffer used by a sequential, direct, or stream access input/output statement for every unit number. The value *xx* in the *fuxxbf* environment variable specifies the unit number. The *size* argument used for sequential access input/output statements or stream access input/output statements is in kilobytes; the *size* argument used for direct access input/output statements is in Fortran records. The *size* argument must be an integer with a value of 1 to 2097151. The *size* argument for direct access input/output statements is the number of Fortran records per block in fixed-block format. The *size* argument must be an integer from 1 to 2147483747/RECL= specifier for an OPEN statement that indicates the number of Fortran records per block. If the value of *size* argument is incorrect, the input/output buffer size is the default size. The default size is 8M bytes. If the environment variable and the runtime options -d or -g are specified at the same time, the environment variable is effective. However, if the *size* argument is not effective, the value specified with the runtime options is effective.

When sequential access input/output statements or stream access input/output statements are executed for unit number 10, the statements use an input/output buffer of 64 kilobytes.

```
$ export fu10bf="64"
```

When direct access input/output statements are executed for unit number 10, the number of Fortran records included in one block is 50. The input/output buffer size is obtained by multiplying 50 by the value specified in the RECL= specifier of the OPEN statement.

```
$ export fu10bf="50"
```

FLIB_ALLOC_ALIGN *size*

The boundary of allocated memory by an ALLOCATE statement is aligned on a specified alignment boundary size. The value of *size* must be 16 or more and must be a power of two.

The following example shows how to align 128 byte boundary to allocated memory.

```
$ export FLIB_ALLOC_ALIGN=128
```

FLIB_EXCEPT *u*

The environment variable FLIB_EXCEPT=*u* controls floating point underflow interrupt processing.

If the program was compiled with the -NRtrap compiler option and the program was executed setting the environment variable FLIB_EXCEPT=*u*, the system performs floating point underflow interrupt processing. The system outputs diagnostic message jwe0012i-*u* during execution.

If the program was compiled without the -NRtrap compiler option or the program was executed without setting the environment variable FLIB_EXCEPT=*u*, the system ignores the floating point underflow.

The following example shows how to specify FLIB_EXCEPT.

```
$ export FLIB_EXCEPT=u
```

FLIB_HOOK_TIME *time*

Specifies the interval at which the user-defined subroutine is called at regular time interval. When *time* is specified in the environment variable FLIB_HOOK_TIME, the user-defined subroutine is called at interval of *time* milliseconds.

The unit for specifying *time* is milliseconds, and the valid value range is $0 \leq \textit{time} \leq 2147483647$. If 0 is specified for *time*, calling at regular time interval is disabled. If the environment variable is not specified, the Fortran system default value takes effect. The default value is 60000 milliseconds (one minute).

Refer to Section "[8.2.3 Hook Function](#)", for information about the hook function.

An example of specifying the environment variable FLIB_HOOK_TIME is shown below.

```
$ export FLIB_HOOK_TIME=600000
```

In this example, the user-defined subroutine is called at 10 minutes interval.

FLIB_RTINFO

Outputs runtime information.

See Section "[Appendix H Runtime Information Output Function](#)" for details on runtime information output function.

FLIB_RTINFO_CSV file

In the Runtime Information Output Function, Outputs the fetched information to a CSV file.

file is optional. Any file name can be specified as file of the environment variable.

If file is omitted, flib_rtinfo.csv is assumed.

See Section "[Appendix H Runtime Information Output Function](#)" for details on runtime information output function.

FLIB_RTINFO_NOFUNC

In the Runtime Information Output Function, even if the compiler option -Nrt_tune_func is effective, the output of information for each procedure is suppressed.

See Section "[Appendix H Runtime Information Output Function](#)" for details on runtime information output function.

FLIB_RTINFO_NOLOOP

In the Runtime Information Output Function, Even if the compiler option -Nrt_tune_loop is effective, the output of information for each loop is suppressed.

See Section "[Appendix H Runtime Information Output Function](#)" for details on runtime information output function.

FLIB_TRACEBACK_MEM_SIZE *size*

This environment variable changes the size of the heap memory used to debug information output to the trace back map information.

The unit for specifying *size* is MiB, and *size* must be an integer from 1 to 128.

If no environment variable is specified, or if the value of *size* is invalid, the Fortran system default value takes effect. The default value is 16MiB.

See "[4.2.2 Trace Back Map](#)" for the trace back map.

FLIB_UNDEF_VALUE *hex*

This environment variable changes the values used for undefined data reference checks. Using *hex*, hexadecimal values can be specified. The valid range of values is $00 \leq hex \leq FF$. If the environment variable is not specified or if *hex* value is disabled, the Fortran system default value takes effect. The default value is 8B.

See Section "[8.2.1.3 Checking References for Undefined Data Items \(UNDEF\)](#)" for details on undefined data reference checks.

An example of specifying the FLIB_UNDEF_VALUE environment variable is shown below.

```
$ export FLIB_UNDEF_VALUE=8C
```

In this example, the value 8C is applied in each byte of the data value used in the undefined data reference check.

FLIB_USE_STOPCODE *n*

This environment variable specifies whether to reflect the stop-code of the STOP/ERROR STOP statement in the return code. The least significant byte (0 to 255) in a stop-code is reflected in the return code if the value of *n* is 1. The stop-code is not reflected in the return code if anything other than 1 is specified to *n* or this environment variable is not specified.

The following example shows how to specify the environment variable FLIB_USE_STOPCODE.

```
$ export FLIB_USE_STOPCODE=1
```

In this example, the stop-code of the STOP/ERROR STOP statement is reflected in the return code.

TMPDIR *dir*

The TMPDIR environment variable changes the directory where an unnamed file is created. If this is not effective, an unnamed file is created on the current directory.

The following example shows how to specify TMPDIR.

```
$ export TMPDIR='/tmp'
```

In this example, an unnamed file is created on /tmp.

Chapter 4 Messages and Listings

This chapter describes the system messages and listing produced as a result of compiling a Fortran source program or executing a Fortran program.

4.1 Compilation Output

4.1.1 Compilation Diagnostic Messages

The output from the `frtpx` command and the Fortran compiler includes diagnostic messages. If `frtpx` command line arguments contain errors or if errors or warnings are detected during the compilation of Fortran source programs, diagnostic messages are written.

Compiler Messages

Formats

Messages issued by `frtpx` are in the following format:

```
$ frtpx: Message-text
```

Messages issued by the Fortran compiler are in the following format:

```
jwdxxxxz-y filename line column loop-identification Message-text
```

The elements of such a compiler message are defined as follows:

`jwdxxxxz-y`

`jwd`: A diagnostic message from the Fortran compiler

`xxxx`: Message serial number

`z`: Message type (i, o, s, or p)

Message type	Explanation
i	This message is unrelated to the optimization.
o	This message reports the optimization status.
s	This message reports the optimization of using SIMD instructions.
p	This message reports the optimization status of automatic parallelization.

`y`: Message error level (i, w, s, or u) and return code

Error level	Return code	Explanation
i	0	No error.
w		The message is information.
	1	The message is a warning. Processing continues.
s		A serious error.
u		The system ignores the erroneous statement and continues processing other statements.
		An unrecoverable error.
		The system terminates processing.

`filename`

The filename where the message event was detected.

line

The line number where the message event was detected.

Note

Due to the influence of optimizations, incorrect line numbers may output.

column

The column number of the line where the message event was detected. The column number may be added to diagnostic messages.

loop-identification

The loop identification number of the loop generated internally by the compiler optimizations, such as loop fission.

Note

The loop identification number does not guarantee the execution order of loops.

Message-text

The message text is issued in either English or Japanese depending on user environment variables, such as LC_MESSAGES and LANG.

4.1.2 Guidance Messages during Compilation

By specifying the `-Koptmsg=guide` compiler option, the guidance messages about inhibiting factor and coping method are output when the following optimizations are prohibited to apply.

- SIMD instruction
- Automatic Parallelization
- Software pipelining
- Inline expansion

Guidance messages are output after each diagnostic message.

Output format is showed below.

```
jwdxxxxz-y Diagnostic Message
 [Guidance]
 Guidance Messages-text
```

See the "[4.1.1 Compilation Diagnostic Messages](#)" for the details of the diagnostic message.

An example of a guidance message to be output is shown below.

```
External subroutine subprogram "sub"
 (line-no.)(nest)(optimize)
   1                subroutine sub(a,b,n1,n2)
   2                real * 8 a(10),b(10)
                   <<< Loop-information Start >>>
                   <<< [OPTIMIZATION]
                   <<< FULL UNROLLING
                   <<< Loop-information End >>>
   3      1         fs      do i=1,10
   4      1         fs      a(i+n1) = a(i+n2) + b(i)
   5      1         fs      enddo
   6                return
   7                end
```

```
Diagnostic messages: program name(sub)
 jwd8203o-i "guide.f", line 3: Loop unrolled fully.
```

```
jwd6204s-i "guide.f", line 4: SIMD conversion cannot be applied to DO loop: unknown relation
between the variables 'n2' and 'n1' may cause different results from serial execution.
```

[Guidance]

If it is certain that the data dependency of SIMD converted execution is the same as that of the serial execution, do one of the following:

- When the data has no data dependency over iteration, specify !OCL NORECURRENCE to the loop.
- When the data has data dependency over iteration, specify !OCL NOVREC to the loop.

4.1.3 Compilation Information

4.1.3.1 Compilation Information Options

Compilation information is output by specifying the `-Nlst` or `-Nlst_out=file` compiler option. The compilation information is output to a file with suffix `.lst`. If more than one Fortran source program file is specified, it is output to the first-file-name.lst. The compilation information can be controlled by specifying the arguments of `-Nlst` or `-Nlst_out=file`. See Section "4.1.3.2 Example of Compilation Information". The options and their meanings are as follows:

`-Nlst`

Outputs the source program list and diagnostic error messages.

`-Nlst=a`

Outputs attributes of names in addition to the compilation information specified by the `-Nlst` option.

`-Nlst=d`

Outputs information about derived type layout in addition to the compilation information specified by the `-Nlst` option.

`-Nlst=i`

Outputs include file listings and a list of the names of include files in addition to the compilation information specified by the `-Nlst` option.

`-Nlst=m`

If the `-Nlst=m` option is specified, the Fortran program which expresses the situation of automatic parallelization with OpenMP directives is output in addition to the compilation information produced by the `-Nlst=p` option. `!FRT` is added at the last of directives output by this function in order to distinguish from directives already described.

In addition, when the situation of automatic parallelization cannot be correctly expressed with OpenMP directive, the line which starts not in `!$OMP` but in `!!!! PARALLEL INFO:` is output. The situation of the optimization performed simultaneously with automatic parallelization may be added to the line.

1. The kind of OpenMP directive output by this function is shown below.

- `PARALLEL DO`
- `FIRSTPRIVATE(var_name,...)`
- `LASTPRIVATE(var_name,...)`
- `PRIVATE(var_name,...)`
- `REDUCTION(+:var_name)`
- `REDUCTION(-:var_name)`
- `REDUCTION(*:var_name)`
- `REDUCTION(MAX:var_name)`
- `REDUCTION(MIN:var_name)`
- `REDUCTION(.OR.:var_name)`
- `REDUCTION(.AND.:var_name)`

2. The kind of information output to the line which starts in "!!!! PARALLEL INFO:" is shown below. Referring to these information, user can easily rewrite the program into OpenMP form.

- LOOP INTERCHANGED
It means that loop was interchanged.
- LOOP FUSED
It means that loop was fused.
- LOOP FISSION
It means that loop was split.
- COLLAPSED
It means that nested do loops were collapsed into a single loop was performed.
- OTHER METHODS
It means that the situation of automatic parallelization cannot be correctly expressed with OpenMP directive by a reason other than the above. The example of the main factors is shown below.
 - The loop is not DO loop.
 - The loop including an induction variable.
 - The loop including a variable of derived type.
 - The loop including a variable that has the POINTER or ALLOCATABLE attribute.
 - The loop including a definition in the array element with an invariable subscript.
 - The variable of reduction operation is the array element with an invariable subscript.
 - The initial value, terminal value or incrementation value of loop including the array element, function reference, and so on.
- UNKNOWN VARIABLE(var_name)
It means that it is the variable that kind of OpenMP directive is not decided correctly. The example of the main factors is shown below.
 - The variable that kind of OpenMP directive is not decided correctly because of optimization.
 - The variable of complex type.
- Clause of OpenMP(var_name)
When the kind of OpenMP directive is not decided correctly, the information for the clause of OpenMP might be output as a reference information. The kind of clause is shown below:
 - FIRSTPRIVATE(var_name,...)
 - LASTPRIVATE(var_name,...)
 - PRIVATE(var_name,...)
 - REDUCTION(+:var_name)
 - REDUCTION(-:var_name)
 - REDUCTION(*:var_name)
 - REDUCTION(MAX:var_name)
 - REDUCTION(MIN:var_name)
 - REDUCTION(.OR.:var_name)
 - REDUCTION(.AND.:var_name)

3. Notes on use are shown below.

- When -Nlst=m option is specified to file created by -P option, the file having name that .omp is added to the file name before preprocessing is created.

```
$ frtprx -P file.F95
```

file.cpp.f95 was created.

```
$ frtpx -Nlst=m -Kparallel file.cpp.f95
```

file.omp.F95 was created.

- If OpenMP directive REDUCTION([MAX | MIN]:var_name) is created from Fortran program containing a variable or a program unit with the name MAX or MIN, the Fortran program can issue the error of S level and cannot be compiled. Correct Fortran program not to use the name MAX or MIN in that case.
- Even when parallelized, neither OpenMP directive nor "!!!! PARALLEL INFO:" line may be output by the influence of optimization.

-Nlst=p

This option specifies optimization information and automatic parallelization information to be output along with listing of -Nlst option. Optimization information shows status of parallelization, inlining and unrolling. To know the output on statements specified with the OpenMP Fortran directives, see "[12.3 Parallelization by OpenMP Specifications](#)".

1. Automatic parallelization information

The meanings of marks for DO statement or array expression are as follows.

These describe the parallelization status of whole DO loop or array expression.

p	:	DO loop or array expression is parallelized
m	:	DO loop or array expression is partially parallelized
s	:	DO loop or array expression is not parallelized
blank	:	DO loop or array expression is not target of parallelization

When the above mark "p", "m" or "s" is displayed for DO statement, "p", "m" or "s" is displayed for executable statement in loop. The meaning of marks for executable statements is as follows.

p	:	Executable statements can be parallelized
m	:	Executable statements can be partially parallelized
s	:	Executable statements cannot be parallelized

The following condition applies to statements targeted for parallelization. Non-executable statements and executable statements that process only transfer of control are displayed by symbols determined by the parallelization information of executable statements before and after these statements. Transfer of control statements include ELSE, END IF, CONTINUE, ENDDO, ELSEWHERE, and so on. Also parallelization information for these statements may be blank. Even when a DO loop or array expression is enabled for parallelization, for the sake of performance, the system may not parallelize it. In this case it displays "s" for the DO statement and displays the unchanged analysis results for statements inside the DO loop.

2. Optimization information

The meaning of optimization marks are as follows.

i	:	inlined the reference to procedure
number	:	unrolled expansion number
f	:	full unrolling the loop

-Nlst=t

If the -Nlst=t option is specified, the details optimization information and the statistics information is output in addition to the compilation produced by the -Nlst=p option.

The optimization information is output in loop units. The parallel dimension information is output in statement units. The statistics information is output approximate stack size is used in a reference of procedure.

1. The outputs information in loop units is defined as follows:

a. Displaying about automatic parallelization

- Standard iteration count : N

When the iteration count of a loop is N or more, the loop is brought to parallel execution. When less than N , it is brought to serial execution.

b. Displaying optimization information in loop units

- INTERCHANGED(nest:*nest-no*)

It means that loop was interchanged.

nest-no is the nesting level of the loop that was moved by loop interchange optimization.

(nest:) may not be displayed if the other optimization like loop collapsing was applied.

- FUSED(lines: *num1,num2[,num3]...*)

It means that loops were fused.

Loops at line *num2* and subsequent lines were fused into a loop at line *num1*.

(lines:) is not displayed at line *num2* and subsequent lines, that were fused to line *num1*.

- FISSION(num: N)

It means that loop was fissioned.

N is the number of loops after fission.

- COLLAPSED

It means that nested loops were collapsed into a single loop.

- SOFTWARE PIPELINING(IPC: *ipc*, ITR: *itr*, MVE: *mve*, POL: *pol*)

It means that software pipelining was applied to the loop.

- *ipc* is the expectation value of the number of executed instructions per cycle (Instructions Per Cycle) when the software pipelining is applied to the loop.

- *itr* is the minimum iteration count required to choose the software pipelined loop. It is the same as the value output by the optimization message jwd8205o-i.

- *mve* is the loop expansion number by the software pipelining.

- *pol* is the applied instruction scheduling algorithm. `S` means that the algorithm fit for a small loop is applied. `L` means that the algorithm fit for a large loop is applied. These algorithms correspond to the ones applied by -Kswp_policy=small and -Kswp_policy=large, respectively.

- SIMD

It means that SIMD instructions were generated for the loop. It is displayed by one of the following.

- SIMD(VL: *length[,length]...*)

It means that SIMD instructions were generated for the loop. A SIMD instruction treats *length* elements of array. When loop fission is applied to the loop and the values of *length* for each loop are different, two or more *lengths* are displayed.

- SIMD(VL: AGNOSTIC; VL: *length[,length]...* in 128-bit)

It means that SIMD instructions were generated for the loop without regarding the SVE vector register size as a specific size. A SIMD instruction treats *length* elements of array if the size of the SVE vector register is 128-bit. When loop fission is applied to the loop and the values of *length* for each loop are different, two or more *lengths* are displayed.

- STRIPING

It means that loop was performed striping.

- MULTI-OPERATION FUNCTION

It means that loop includes multi-operation function.

- PATTERN MATCHING(matmul)

It means that loop was changed library function call(matmul).

- UNSWITCHING

It means that loop was performed loop unswitching.

- FULL UNROLLING

It means that loop was fully unrolled.

- CLONE
It means that CLONE optimization was applied to the loop.
 - LOOP VERSIONING
It means that loop versioning was applied to the loop.
- c. Information that relates to the prefetch is displayed.
- c1. Hardware-prefetch.
- PREFETCH(HARD) Expected by compiler:
It indicates an array name which does not generate the prefetch instruction expecting to use hardware-prefetch for array data accessed sequentially within a loop.
 - array name,...
It indicates an array name expecting to use hardware-prefetch. The array which was generated by compiler is shown as "(unknown)"
- c2. Prefetch instruction.
- PREFETCH(SOFT) : N
It indicates the number of all prefetch instructions that exist in the loop.
In addition, the number and the array name of the prefetch instruction of each access type of the prefetch are shown in the form of the following.
access type: N
array name: N ,...
 - access type
 - SEQUENTIAL: N
It indicates the number of prefetch instructions for array data accessed sequentially within a loop.
 - STRIDE: N
It indicates the number of prefetch instructions for array data that is accessed with a stride larger than the cache line size used in the loop.
 - INDIRECT: N
It indicates the number of prefetch instructions for array data that is accessed indirectly (list access) within a loop.
 - SPECIFIED: N
It indicates the number of prefetch instructions generated by PREFETCH_READ or PREFETCH_WRITE optimization control specifier.
 - array name
 - array name: N ,...
It indicates the number of prefetch instructions for the array name. The array which was generated by compiler is shown as "(unknown)".
- d. Information that relates to the zfill optimization is displayed.
- ZFILL
Indicates that the zfill optimization is applied.
 - array name,...
Indicates array names to which the zfill optimization is applied. An array generated by compiler is shown as "(unknown)"
- e. Information that relates to the register is displayed.
- SPILLS:
Indicates the number of saving and restoring instructions for registers that exist in the innermost loop shown for each register type.
 - GENERAL : SPILL N FILL N
Indicates the number of saving and restoring instructions for general registers to and from the memory.

- SIMD&FP : SPILL *N*FILL *N*

Indicates the number of saving and restoring instructions for SIMD and floating point registers to and from the memory.

- SCALABLE : SPILL *N*FILL *N*

Indicates the number of saving and restoring instructions for scalable vector registers to and from the memory.

- PREDICATE : SPILL *N*FILL *N*

Indicates the number of saving and restoring instructions for predicate registers to and from the memory.

2. Marks for DO statement or array expression are as follows:

p : It means the beginning of parallelized range.(The mark "*p*" of -Nlst=*p* option is output together, so not *p* but "*pp*" is displayed)

3. The statistics information is defined as follows:

a. The statistics information is output approximate stack size is used in a reference of procedure.

Note:

- The stack size in internal procedure is displayed with stack size in parent procedure.
- The following data is allocated stack area, but the size does not include outputs in this function.
- The data is specified THREADPRIVATE directive.
- The data is automatic data object and -Kautoobjstack option in effect.

b. The number of prefetch instructions procedure is displayed.

4. SIMD information

The meanings of marks for DO statement or array expression are as follows.

These describe the SIMD optimization status of whole DO loop or array expression.

v : DO loop or array expression is applied SIMD optimization
m : DO loop or array expression is applied partially SIMD optimization
s : DO loop or array expression is not applied SIMD optimization
blank : DO loop or array expression is not target of SIMD optimization

When the above mark "*v*", "*m*" or "*s*" is displayed for DO statement, "*v*", "*m*" or "*s*" is displayed for executable statement in loop.

The meaning of marks for executable statements is as follows.

v : Executable statements can be applied SIMD optimization (*1)
m : Executable statements can be applied partially SIMD optimization (*1)
s : Executable statements cannot be applied SIMD optimization

The following condition applies to statements targeted for SIMD optimization.

Non-executable statements and executable statements that process only transfer of control are displayed by symbols determined by the SIMD optimization information of executable statements before and after these statements.

Transfer of control statements include ELSE, END IF, CONTINUE, ENDDO, ELSEWHERE, and so on.

Also SIMD optimization information for these statements may be blank.

Even when a DO loop or array expression is enabled for SIMD optimization, for the sake of performance, the system may not SIMD optimization it. In this case it displays "*s*" for the DO statement and displays the unchanged analysis results for statements inside the DO loop.

*1: When SIMD optimization is applied to a DO loop and the mark "*v*" or "*m*" is displayed for the DO statement, the mark "*v*" could be displayed for statements in the loop though SIMD extensions are not used for them.

The mark "*s*" is displayed for a statement preventing SIMD optimization, which is a clue to tune programs.

-Nlst=*x*

Outputs a cross reference listing of names and labels in addition to the compilation information specified by the -Nlst option.

These arguments can be specified at the same time, separated by comma as in `-Nlst=a,lst_out=file,lst=x`.

`-Nlst_out=file`

Outputs the information to a file named *'file'*.

4.1.3.2 Example of Compilation Information

The following is an example of the output produced by the `-Nlst` option.

```
Main program "CHECK"
(line-no.)(nest)
   1          PROGRAM CHECK
   2          INTEGER,DIMENSION(10,10) :: A = 0
   3          INTERFACE
   4              SUBROUTINE SUB(I,J,A)
   5                  INTEGER,DIMENSION(:,:) :: A
   6                  END SUBROUTINE
   7          END INTERFACE
   8      1      DO I=1,10
   9          2      DO J=MOD(I,2)+1,10,2
  10              2      CALL SUB(I,J,A)
  11          2      END DO
  12      1      END DO
  13          WRITE (*,"(10I3)") (A(:,I),I=1,10)
  14          END PROGRAM

Procedure information
  Lines      : 14
  Statements : 14

External subroutine subprogram "SUB"
(line-no.)(nest)
  15
  16          SUBROUTINE SUB(I,J,A)
  17          INTEGER,DIMENSION(:,:) :: A
  18          A(I,J) = 1
  19          END SUBROUTINE

Procedure information
  Lines      : 5
  Statements : 4

Total information
  Procedures : 2
  Total lines : 19
  Total statements : 18
```

The following is an example of diagnostic error messages that are output.

```
Main program "ERROR"
(line-no.)(nest)
   1          PROGRAM ERROR
   2      1      DO 10 I=1,3
   3          2      DO 20 J=1,3
   4              2      CALL SUB(I*J)
   5          2      10  ENDDO
   6      1      END PROGRAM

Diagnostic messages: program name(ERROR)
  jwd1027i-s "error.f", line 3: Undefined statement label ' 20' referenced.
  jwd1131i-s "error.f", line 3, column 9: The program unit contains unmatched nest in DO construct,
  IF construct, CASE construct, SELECT TYPE construct, ASSOCIATE construct, CRITICAL construct, or
  BLOCK construct.
```

Procedure information

Lines : 6
Statements : 6

External subroutine subprogram "SUB"

```
(line-no.)(nest)
  7
  8          SUBROUTINE SUB(I)
  9          INTEGER,INTENT(IN) :: I
 10          PRINT *,I()
 11          END SUBROUTINE
```

Diagnostic messages: program name(SUB)

jwd2008i-i "error.f", line 8: Dummy argument 'I' not used in this subprogram.
jwd1771i-s "error.f", line 10, column 16: 'I' already has the INTENT attribute.

Procedure information

Lines : 5
Statements : 4

Total information

Procedures : 2
Total lines : 11
Total statements : 10

The following is an example from the -Nlst=i option.

```
Module "MOD"
(inc)(line-no.)(nest)
  1          MODULE MOD
  2          CHARACTER(LEN=15) CC
  3          END MODULE
```

Procedure information

Lines : 3
Statements : 3

Main program "INCLUDE"

```
(inc)(line-no.)(nest)
  4
  5          PROGRAM INCLUDE
  6          INCLUDE "INCL"
  1  1      use mod,only:cc
  7          CALL INIT()
  8          PRINT *,CC
  9          END PROGRAM
```

Procedure information

Lines : 7
Statements : 5

External subroutine subprogram "INIT"

```
(inc)(line-no.)(nest)
 10
 11          SUBROUTINE INIT()
 12          INCLUDE "INC2"
  2  1      include "incl"
  1  1      use mod,only:cc
  2  2
 13          CC = "FUJITSU FORTRAN"
 14          END SUBROUTINE
```

```

Procedure information
  Lines      : 5
  Statements : 3

Total information
  Procedures : 3
  Total lines : 14
  Total statements : 10

Include file name list
  1 : ./incl
  2 : ./inc2

```

The following is an example of the output from the `-Nlst=p -Kparallel -x` option.

```

Main program "MAIN"
(line-no.)(nest)(optimize)
  1              INTEGER,PARAMETER ::N=10000
  2              REAL,DIMENSION(N)::A
  3      1      p      6              DO I=1,N
  4      1      pi     6              A(I) = A(I) * FUNC(I)
  5      1      p      6              ENDDO
  6              END

Procedure information
  Lines      : 6
  Statements : 6

External function subprogram "FUNC"
(line-no.)(nest)(optimize)
  7              FUNCTION FUNC(I)
  8              FUNC=I+1
  9              END

Procedure information
  Lines      : 3
  Statements : 3

Total information
  Procedures : 2
  Total lines : 9
  Total statements : 9

```

The following is an example of the output from the `-Nlst=x` option.

```

Module "COMPLEX"
(line-no.)(nest)
  1              MODULE COMPLEX
  2              TYPE TYPE1
  3              SEQUENCE
  4              REAL :: R_PART
  5              REAL :: I_PART
  6              END TYPE TYPE1
  7              INTERFACE OPERATOR(+)
  8              MODULE PROCEDURE PLUS
  9              END INTERFACE
 10              PRIVATE PLUS
 11              CONTAINS
 12              TYPE(TYPE1) FUNCTION PLUS(OP1,OP2)
 13              TYPE(TYPE1),INTENT(IN) :: OP1,OP2
 14              PLUS%R_PART = OP1%R_PART + OP2%R_PART
 15              PLUS%I_PART = OP1%I_PART + OP2%I_PART
 16              END FUNCTION PLUS
 17              END MODULE

```

Procedure information

Lines : 17
Statements : 17

Scoping unit of module : COMPLEX

Cross reference of name

COMPLEX

|(Class and Type) : module name
|(Declaration) : 1
|(Definition) :
|(Reference) :

PLUS

|(Class and Type) : module function name, TYPE(TYPE1)
|(Declaration) : 8 10
|(Definition) :
|(Reference) :

TYPE1

|(Class and Type) : type name
|(Declaration) :
|(Definition) : 2
|(Reference) :

[OPERATOR(+)]

|(Class and Type) : user defined operator
|(Declaration) : 7
|(Definition) :
|(Reference) :

Scoping unit of module sub-program : PLUS

Cross reference of name

OP1

|(Class and Type) : variable name, TYPE(TYPE1)
|(Declaration) : 12 13
|(Definition) :
|(Reference) : 14 15

OP2

|(Class and Type) : variable name, TYPE(TYPE1)
|(Declaration) : 12 13
|(Definition) :
|(Reference) : 14 15

PLUS

|(Class and Type) : variable name, TYPE(TYPE1)
|(Declaration) :
|(Definition) : 14 15
|(Reference) :

PLUS

|(Class and Type) : module function name, TYPE(TYPE1)
|(Declaration) :
|(Definition) : 12
|(Reference) : 16

TYPE1

|(Class and Type) : type name
|(Declaration) :
|(Definition) :
|(Reference) : 12 13

Main program "MAIN"

(line-no.)(nest)

```
18
19          PROGRAM MAIN
20          USE COMPLEX
21          TYPE(TYPE1) CPX1, CPX2, RESULT
22          READ (*,*) CPX1, CPX2
```

```

23             RESULT = CPX1 + CPX2
24             PRINT *,RESULT
25             END PROGRAM MAIN

```

Procedure information

```

Lines       : 8
Statements  : 7

```

Scoping unit of program : MAIN

Cross reference of name

COMPLEX

```

|(Class and Type) : module name
|(Declaration)    :
|(Definition)     :
|(Reference)      : 20

```

CPX1

```

|(Class and Type) : variable name, TYPE(TYPE1)
|(Declaration)    : 21
|(Definition)     : 22
|(Reference)      : 23

```

CPX2

```

|(Class and Type) : variable name, TYPE(TYPE1)
|(Declaration)    : 21
|(Definition)     : 22
|(Reference)      : 23

```

MAIN

```

|(Class and Type) : program name
|(Declaration)    : 19
|(Definition)     :
|(Reference)      : 25

```

RESULT

```

|(Class and Type) : variable name, TYPE(TYPE1)
|(Declaration)    : 21
|(Definition)     : 23
|(Reference)      : 24

```

TYPE1

```

|(Class and Type) : type name
|(Declaration)    :
|(Definition)     :
|(Reference)      : 21

```

[OPERATOR(+)]

```

|(Class and Type) : user defined operator
|(Declaration)    :
|(Definition)     :
|(Reference)      : 23

```

Total information

```

Procedures      : 2
Total lines      : 25
Total statements : 24

```

The following is an example of the output from the -Nlst=a,lst=x option.

```

Module "COMPLEX"
(line-no.)(nest)
  1             MODULE COMPLEX
  2             TYPE TYPE1
  3             SEQUENCE
  4             REAL :: R_PART
  5             REAL :: I_PART
  6             END TYPE TYPE1
  7             INTERFACE OPERATOR(+)
  8             MODULE PROCEDURE PLUS

```

```

9          END INTERFACE
10         PRIVATE PLUS
11         CONTAINS
12         TYPE(TYPE1) FUNCTION PLUS(OP1,OP2)
13             TYPE(TYPE1), INTENT(IN) :: OP1,OP2
14             PLUS%R_PART = OP1%R_PART + OP2%R_PART
15             PLUS%I_PART = OP1%I_PART + OP2%I_PART
16         END FUNCTION PLUS
17     END MODULE

```

Procedure information

```

Lines      : 17
Statements : 17

```

Scoping unit of module : COMPLEX

Attribute and Cross reference of name

COMPLEX

```

|(Class and Type) : module name
|(Attributes)      :
|(Declaration)     : 1
|(Definition)      :
|(Reference)       :

```

PLUS

```

|(Class and Type) : module function name, TYPE(TYPE1)
|(Attributes)     : PUBLIC
|(Declaration)    : 8 10
|(Definition)     :
|(Reference)      :

```

TYPE1

```

|(Class and Type) : type name
|(Attributes)     : PUBLIC
|(Declaration)    :
|(Definition)     : 2
|(Reference)      :

```

[OPERATOR(+)]

```

|(Class and Type) : user defined operator
|(Attributes)     : PUBLIC
|(Declaration)    : 7
|(Definition)     :
|(Reference)      :

```

Scoping unit of module sub-program : PLUS

Attribute and Cross reference of name

OP1

```

|(Class and Type) : variable name, TYPE(TYPE1)
|(Attributes)     : INTENT(IN), dummy-argument
|(Declaration)    : 12 13
|(Definition)     :
|(Reference)      : 14 15

```

OP2

```

|(Class and Type) : variable name, TYPE(TYPE1)
|(Attributes)     : INTENT(IN), dummy-argument
|(Declaration)    : 12 13
|(Definition)     :
|(Reference)      : 14 15

```

PLUS

```

|(Class and Type) : variable name, TYPE(TYPE1)
|(Attributes)     : result-value
|(Declaration)    :
|(Definition)     : 14 15
|(Reference)      :

```

PLUS

```

|(Class and Type) : module function name, TYPE(TYPE1)

```

```
| (Attributes)      :  
| (Declaration)    :  
| (Definition)     : 12  
| (Reference)      : 16
```

TYPE1

```
| (Class and Type) : type name  
| (Attributes)     : host-associated  
| (Declaration)    :  
| (Definition)     :  
| (Reference)      : 12 13
```

Main program "MAIN"

(line-no.)(nest)

```
18  
19          PROGRAM MAIN  
20          USE COMPLEX  
21          TYPE(TYPE1) CPX1, CPX2, RESULT  
22          READ (*,*) CPX1, CPX2  
23          RESULT = CPX1 + CPX2  
24          PRINT *,RESULT  
25          END PROGRAM MAIN
```

Procedure information

```
Lines       : 8  
Statements  : 7
```

Scoping unit of program : MAIN

Attribute and Cross reference of name

COMPLEX

```
| (Class and Type) : module name  
| (Attributes)     :  
| (Declaration)    :  
| (Definition)     :  
| (Reference)      : 20
```

CPX1

```
| (Class and Type) : variable name, TYPE(TYPE1)  
| (Attributes)     :  
| (Declaration)    : 21  
| (Definition)     : 22  
| (Reference)      : 23
```

CPX2

```
| (Class and Type) : variable name, TYPE(TYPE1)  
| (Attributes)     :  
| (Declaration)    : 21  
| (Definition)     : 22  
| (Reference)      : 23
```

MAIN

```
| (Class and Type) : program name  
| (Attributes)     :  
| (Declaration)    : 19  
| (Definition)     :  
| (Reference)      : 25
```

RESULT

```
| (Class and Type) : variable name, TYPE(TYPE1)  
| (Attributes)     :  
| (Declaration)    : 21  
| (Definition)     : 23  
| (Reference)      : 24
```

TYPE1

```
| (Class and Type) : type name  
| (Attributes)     : use-associated  
| (Declaration)    :  
| (Definition)     :
```

```

|(Reference)      : 21
[OPERATOR(+)]
|(Class and Type) : user defined operator
|(Attributes)    : use-associated
|(Declaration)   :
|(Definition)    :
|(Reference)     : 23

```

Total information

```

Procedures      : 2
Total lines     : 25
Total statements : 24

```

The following is an example of the output from the -Nlst=d option.

```

Module "COMPLEX"
(line-no.)(nest)
  1          MODULE COMPLEX
  2          TYPE TYPE1
  3          SEQUENCE
  4          REAL :: R_PART
  5          REAL :: I_PART
  6          END TYPE TYPE1
  7          INTERFACE OPERATOR(+)
  8          MODULE PROCEDURE PLUS
  9          END INTERFACE
 10          PRIVATE PLUS
 11          CONTAINS
 12          TYPE(TYPE1) FUNCTION PLUS(OP1,OP2)
 13             TYPE(TYPE1),INTENT(IN) :: OP1,OP2
 14             PLUS%R_PART = OP1%R_PART + OP2%R_PART
 15             PLUS%I_PART = OP1%I_PART + OP2%I_PART
 16          END FUNCTION PLUS
 17          END MODULE

```

Procedure information

```

Lines      : 17
Statements : 17

```

Scoping unit of module : COMPLEX

Derived type construction

```

TYPE1
| (Attributes) : SEQUENCE
|R_PART
| (Type and Attributes) : REAL(4), PUBLIC
|I_PART
| (Type and Attributes) : REAL(4), PUBLIC

```

Scoping unit of module sub-program : PLUS

Main program "MAIN"

```

(line-no.)(nest)
 18
 19          PROGRAM MAIN
 20          USE COMPLEX
 21          TYPE(TYPE1) CPX1, CPX2, RESULT
 22          READ (*,*) CPX1, CPX2
 23          RESULT = CPX1 + CPX2
 24          PRINT *,RESULT
 25          END PROGRAM MAIN

```

Procedure information

```

Lines      : 8

```


Statements : 7

Scoping unit of program : MAIN

Total information

Procedures : 2

Total lines : 25

Total statements : 24

The following is an example of the output from the -Kparallel,reduction -Nlst=m option.(omp file)

```
      SUBROUTINE SUB(N,M)
      REAL,DIMENSION(10000) :: A
      REAL,DIMENSION(10000,100) :: C,D
      REAL::RRES=0.0

      CALL INIT(A,B,C)

!$OMP PARALLEL DO REDUCTION(MAX:RRES) !FRT
      DO I=1,N
          IF (RRES<A(I)) RRES = A(I)
      END DO
      CALL OUTPUT(RRES)
!$OMP PARALLEL DO PRIVATE(I) !FRT
      DO J=1,M
          DO I=1,N
              C(I,J) = C(I,J) + D(I,J)
          END DO
      END DO
      CALL OUTPUT(C)

      DO I=1,N
!!!! PARALLEL INFO: LOOP INTERCHANGED
          DO J=1,M
              D(I,J) = D(I,J) + C(I,J)
          END DO
      END DO
      CALL OUTPUT(D)
      END
```

The following is an example of the output produced by the -Kfast -Kparallel -Nlst=t option.

```
External subroutine subprogram "SUB"
(line-no.)(nest)(optimize)
 1          SUBROUTINE SUB(A,B,C,N)
 2          IMPLICIT NONE
 3          INTEGER(KIND=4) :: N
 4          REAL(KIND=8),DIMENSION(N,N) :: A,B
 5          REAL(KIND=8) :: C
 6          INTEGER(KIND=4) :: I,J
          <<< Loop-information Start >>>
          <<< [OPTIMIZATION]
          <<< INTERCHANGED(nest: 2)
          <<< SIMD(VL: 8)
          <<< SOFTWARE PIPELINING(IPC: 3.00, ITR: 192, MVE: 7, POL: S)
          <<< PREFETCH(HARD) Expected by compiler :
          <<< B, A
          <<< Loop-information End >>>
 7          1 p 2v          DO I=2,N
          <<< Loop-information Start >>>
          <<< [PARALLELIZATION]
          <<< Standard iteration count: 3
          <<< [OPTIMIZATION]
          <<< INTERCHANGED(nest: 1)
```

```

      <<<   PREFETCH(HARD) Expected by compiler :
      <<<       B, A
      <<< Loop-information End >>>
 8      2  pp  2          DO J=2,N
 9      2  p   2v       A(I,J) = A(I,J) + B(I,J) * C
10     2  p   2          END DO
11     1  p           END DO
12                                END SUBROUTINE

Procedure information
  Lines      : 12
  Statements : 12
  Stack(byte): 112
  Prefetch num: 0

Total information
  Procedures      : 1
  Total lines     : 12
  Total statements : 12
  Total stack(byte): 112
  Total prefetch num: 0

```

4.1.3.3 Notes on Compilation Information

Notes on Compilation Information are shown in the following.

- The `-Nlst=p` option and `-Nlst=t` option may output wrong compilation information (optimization information) in the following cases.
 - When a function is inline expanded, different optimizations may be applied to each line. In this case, the compiler may output information incorrectly as follows.
 - Output messages redundantly.
 - Output optimization messages inconsistent with compilation information (optimization information).
 - Output no compilation information (optimization information).
- In addition, `PREFETCH: N`, which is the number of prefetch instructions in the inlined function, includes all prefetch instructions expanded in multiple lines.
- The compiler applies multiple optimizations to one loop. In this case, the compiler may output information incorrectly as follows.
 - Output optimization information on line number to an incorrect line.
 - Output optimization messages inconsistent with compilation information (optimization information).
- Loop unswitching optimization creates a loop in each "TRUE" and "FALSE" blocks of the IF construct, but the compiler outputs optimization message for only one of the loops. In addition, optimization information on line number may not be output.
- When multiple loops are written in the same line in a program, the compiler outputs optimization information for only one of the loops. Write one loop in one line in order to output optimization information.
- Detailed optimization information for a loop which consists of IF constructs and GO TO statements is not output. Optimization information on line number may be output to an incorrect line.
- The compiler may generate loops when any of the following conditions is met. The optimization messages may be output for the generated loops, and compilation information (optimization information) may be output for the generated loops when `-Nlst=p` or `-Nlst=t` option is specified.
 - The actual argument is a pointer array, assumed shape array, or array section, and the corresponding dummy argument is a nonpointer and nonassumed shape array.
 - `-Nquickdbg=undef`, `-Nquickdbg=undefnan`, or `-Nsetvalue=array` option is effective.
 - An array variable name appears in `FIRSTPRIVATE`, `LASTPRIVATE`, `REDUCTION`, `COPYIN`, or `COPYPRIVATE` clause of OpenMP.

- There is a loop parallelized automatically by `-Karray_private` option or optimization control specifier `ARRAY_PRIVATE`, `FIRST_PRIVATE` or `LAST_PRIVATE`.
- When link time optimization is applied, the optimization which is different from the optimization displayed in compilation information may be applied at runtime.
- When a `#line` directive is included in the source program, compilation information (optimization information) output by `-Nlst=p` or `-Nlst=t` option is output based on the line number specified by the `#line` directive. Suppose, for example, a `#line` directive specifying the line 3 in the source program is attached to a DO statement which is in the line 10. In this case, the compilation information (optimization information) about the DO statement is output to the line 3 in the source program.

4.2 Executable Program Output

Program execution generates the following output:

- Print data from output statements
- Messages from PAUSE or STOP/ERROR STOP statements
- Diagnostic messages
- Trace back map
- Error summary information
- Debug output
- Runtime Information Output Function

This section describes the diagnostic messages, trace back map, and error summary information output when errors are detected during Fortran program execution. See "[7.7 Data Transfer Input/Output Statements](#)" for output statements. See "[6.5.6 PAUSE Statement](#)", and "[6.5.7 STOP/ERROR STOP Statement](#)" for PAUSE or STOP/ERROR STOP statements. See "[8.2 Debugging Functions](#)" for debugging function. See "[Appendix H Runtime Information Output Function](#)" for Runtime Information Output Function.



Note

Due to the influence of optimizations, incorrect line numbers may output in the executable program output.

4.2.1 Diagnostic Messages

A diagnostic message is output if PAUSE statements or STOP/ERROR STOP statements are executed or an error is detected during Fortran program execution.

Messages issued by the Fortran library are in the following format:

```
jwexxxx-t-y Additional-information Message-text
```

- `jwexxxx-t-y`
`jwe` : A diagnostic messages from the Fortran library
`xxxx` : A message serial number
`t` : Output is 'i' or 'a'. 'i' is output for messages not requiring a response. 'a' is output by a PAUSE statement, which requires a response. See "[6.5.6 PAUSE Statement](#)" for the PAUSE statement.

<i>y</i>	Explanation
i	Not an error. The message is information message.
w	The message is a warning. Processing continues.
e	A medium error. The system ignores the erroneous statement and continues processing. Up to 10 erroneous statements can be detected before the system terminates processing. The error number for system terminates can be controlled <code>-enum</code> runtime option and runtime error processing library.
s	A serious error. The system ignores the erroneous statement and terminates processing.

<i>y</i>	Explanation
u	An unrecoverable error. The system terminates processing.

b. Additional-information

Additional information may be added to diagnostic messages. The additional information indicates the line number of the Fortran source program where the error was detected.

c. Message-text

The message text is issued in English.

4.2.2 Trace Back Map

A trace back map is generated as specified in the error control table. This information indicates how the main program calls the program in which the error was detected.

Also, this information is obtained from the debug information stored in the heap memory allocated at startup. If the information is not output, the information may be output by setting the size of the heap memory larger than the default value using the environment variable FLIB_TRACEBACK_MEM_SIZE. If the information cannot be obtained due to insufficient heap memory, a diagnostic message jwe1655i-i will be output. This diagnostic message is output during error processing, so the order in which the diagnostic messages are output may be incorrect.

See "8.1 Error Processing" for information about the error control table. See "8.1.7 Exception Handling Processing", and "8.2.2 Debugging Programs for Abend" for exception handling. See "2.2 Compiler Options" for a compiler option -Nnoline. See "3.8 Using Environment Variables for Execution" for the environment variable FLIB_TRACEBACK_MEM_SIZE.

The trace back map is generated in the following format after a diagnostic message:

```
error occurs at name1 line s1 loc wwwwwwww offset xxxxxxxx
name1          at loc yyyyyyyy called from loc zzzzzzzz in name2 line s2
name2          at loc yyyyyyyy called from loc zzzzzzzz in name3 line s2
name3          at loc yyyyyyyy called from o.s
```

name1

Intrinsic function name or entry name of the program unit where the error was detected. If the name is unknown, '???????' is assigned to *name1*.

name2

Intrinsic function name or entry name of the program unit where the error was detected. If the name is unknown, '???????' is assigned to *name2*.

name3

Entry name of the main program; o.s indicates the control program. If the name is unknown, '???????' is assigned to *name3*.

s1

Line number of the statement where the error was detected. If the compiler option -Nnoline is specified, it does not appear.

s2

Line number of the statement where the program unit is called. If the compiler option -Nnoline is specified, it does not appear.

wwwwwwwww

Absolute address (hexadecimal) of the statement where the error was detected.

xxxxxxx

Relative address (hexadecimal) of the statement where the error was detected, starting at the beginning of the program unit; not output for intrinsic function errors; if this address is unknown, it does not appear.

yyyyyyyy

Absolute address (hexadecimal) of the beginning of the program unit; if this address is unknown, it is '0000000000000000'.

ZZZZZZZZ

Absolute address (hexadecimal) of the statement that called the program unit; if this address is unknown, it is '0000000000000000'.

name1, *name2*, and *name3* are processing names of the procedure names. The processing names and corresponding procedure names are shown in the following table:

Kind of procedure	Processing name
Main-program	MAIN__
Internal-subprogram of main-program	Main-program-name.internal-subprogram-name_
External-subprogram	External-subprogram-name_
Internal-subprogram of external-subprogram	External-subprogram-name.internal-subprogram-name_
Module-subprogram in a module, or separate module procedure whom corresponding separate interface body is in a module	Module-name.module-subprogram-name_
Internal-subprogram of module-subprogram in a module, or internal subprogram of a separate module procedure whom corresponding separate interface body is in a module	Module-name.module-subprogram-name.internal-subprogram-name_
Module-subprogram in a submodule	_Ancestor-module-name.submodule-name.module-subprogram-name_
Separate module procedure whom corresponding separate interface body is in a submodule	_Ancestor-module-name.submodule-name.module-subprogram-name_ (*)
Internal-subprogram of module-subprogram in a submodule	_Ancestor-module-name.submodule-name.module-subprogram-name.internal-subprogram-name_
Internal-subprogram of separate module procedure whom corresponding separate interface body is in a submodule	_Ancestor-module-name.submodule-name.module-subprogram-name.internal-subprogram-name_ (*)

*: The submodule-name is name of submodule who has corresponding separate interface body.

Note1: Processing names are uniformly lowercase unless the compiler option -AU is specified.

Note2: The information in shared objects may be not correct.

4.2.3 Error Summary

The following error summary information is output when program execution is completed.

However, the system errors whose error identification number is 900 to 1000 are not output.

```

error summary (Fortran)
error number error level error count
jwennnni      y          zzz
:             :          :
total error count = xxx

```

nnnn

Message serial number

y

Error level *y*(*y*:i, w, e, s, u) detected

ZZZ

Error count

XXX

Total error count of diagnostic messages

Chapter 5 Data Types, Kinds, and Representations

This section describes the Fortran data types and kinds, and describes the machine representations of these data.

5.1 Internal Data Representation

The following describes the internal representation for each data type.

5.1.1 Integer Type Data

Kind 1 (one-byte) integer type data is represented internally as an 8-bit two's complement binary number with values ranging from -128 to 127.

Kind 2 (two-byte) integer type data is represented internally as a 16-bit two's complement binary number with values ranging from -32768 to 32767.

Kind 4 (four-byte) integer type data is represented internally as a 32-bit two's complement binary number with values ranging from -2147483648 to 2147483647.

Kind 8 (eight-byte) integer type data is represented internally as a 64-bit two's complement binary number with values ranging from -9223372036854775808 to 9223372036854775807.

In two's complement representation, the first bit of each integer is the sign. For a positive integer, the sign bit is zero; for a negative integer, the sign bit is 1. For example, the kind 1 representation of -1 is Z'FF'. For integer zero, all bits are set to zero.

5.1.2 Logical Type Data

Kind 1 (one-byte) logical type data is represented internally as an 8-bit binary number. For TRUE, the value is Z'01'. For FALSE, all bits are zero.

Kind 2 (two-byte) logical type data is represented internally as a 16-bit binary number. For TRUE, the value is Z'0001'. For FALSE, all bits are zero.

Kind 4 (four-byte) logical type data is represented internally as a 32-bit binary number. For TRUE, the value is Z'00000001'. For FALSE, all bits are zero.

Kind 8 (eight-byte) logical type data is represented internally as a 64-bit binary number. For TRUE, the value is Z'0000000000000001'. For FALSE, all bits are zero.

5.1.3 Real and Complex Type Data

The internal representations of all kinds of real type data conform to IEEE 754 specifications.

The internal representations of the real and imaginary parts of all kinds of complex type data are the same as the internal representations of the corresponding kinds of real type data. A complex type value requires twice the storage of a real type value of the same kind.

The following describes in detail how floating-point data is stored in memory.

Kind 2 (half precision) real and complex

The figure below shows the bit composition for sign (s), exponent (e), and fraction (f).

Default real storage format:



The table below shows the bit patterns for numbers stored as default real values.

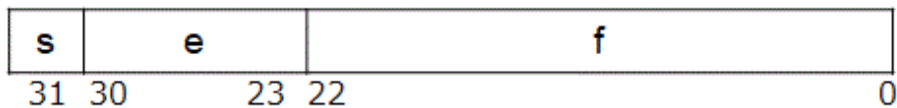
Table 5.1 Default real values

Common name	Storage format	Value
0	0000	0.0E+00
+ maximum normal number	7BFF	65504
- maximum normal number	FBFF	-65504
+ minimum normal number	0400	6.10351562E-05
- minimum normal number	8400	-6.10947609E-05
+ maximum denormalized number	03FF	6.09755516E-05
- maximum denormalized number	83FF	-6.09755516E-05
+ minimum denormalized number	0001	5.96046448E-08
- minimum denormalized number	8001	-5.96046448E-08
+infinity	7C00	Inf (Infinity)
-infinity	FC00	-Inf
+quiet_NaN	7C01 - 7DFF	NaN (Not a number)
-quiet_NaN	FC01 - FDFF	-NaN
+signaling_NaN	7E01 - 7FFF	NaN
-signaling_NaN	FE01 - FFFF	-NaN

Kind 4 (single precision) real and complex

The figure below shows the bit composition for sign (s), exponent (e), and fraction (f).

Default real storage format:



The table below shows the bit patterns for numbers stored as default real values.

Table 5.2 Default real values

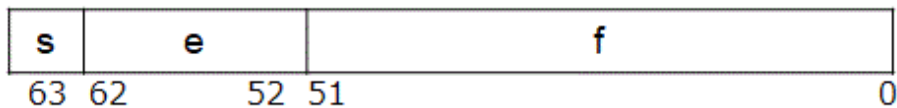
Common name	Storage format	Value
0	00000000	0.0E+00
+ maximum normal number	7F7FFFFFFF	3.40282347E+38
- maximum normal number	FF7FFFFFFF	-3.40282347E+38
+ minimum normal number	00800000	1.17549435E-38
- minimum normal number	80800000	-1.17549435E-38
+ maximum denormalized number	007FFFFFFF	1.17549421E-38
- maximum denormalized number	807FFFFFFF	-1.17759421E-38

Common name	Storage format	Value
+ minimum denormalized number	00000001	1.40129846E-45
- minimum denormalized number	80000001	-1.40129846E-45
+infinity	7F800000	Inf (Infinity)
-infinity	FF800000	-Inf
+quiet_NaN	7FC00000 - 7FFFFFFF	NaN (Not a number)
-quiet_NaN	FFC00000 - FFFFFFFF	-NaN
+signaling_NaN	7F800001 - 7FBFFFFFFF	NaN
-signaling_NaN	FF800001 - FFBFFFFFFF	-NaN

Kind 8 (double precision) real and complex

The figure below shows the bit composition of sign (s), exponent (e), and fraction (f).

Double-precision real storage format:



The table below shows the bit patterns for numbers stored as double-precision real values.

Double-precision real values

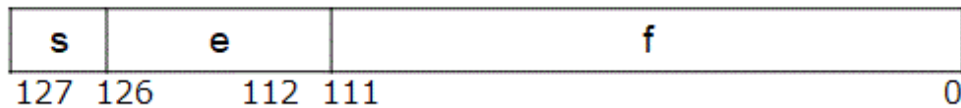
Common name	Storage format	Value
0	00000000 00000000	0.0E+00
+ maximum normal number	7FEFFFFFF FFFFFFFF	1.797693134862316D+308
- maximum normal number	FFEFFFFFF FFFFFFFF	-1.797693134862316D+308
+ minimum normal number	00100000 00000000	2.225073858507201D-308
- minimum normal number	80100000 00000000	-2.225073858507201D-308
+ maximum denormalized number	000FFFFFF FFFFFFFF	2.225073858507201D-308
- maximum denormalized number	800FFFFFF FFFFFFFF	-2.225073858507201D-308
+ minimum denormalized number	00000000 00000001	4.940656458412465D-324
- minimum denormalized number	80000000 00000001	-4.940656458412465D-324
+infinity	7FF00000 00000000	Inf
-infinity	FFF00000 00000000	-Inf
+quiet_NaN	7FF80000 00000000 - 7FFFFFFF FFFFFFFF	NaN
-quiet_NaN	FFF80000 00000000 - FFFFFFF FFFFFFFF	-NaN

Common name	Storage format	Value
+signaling_NaN	7FF00000 00000001 - 7FF7FFFF FFFFFFFF	NaN
-signaling_NaN	FFF00000 00000001 - FFF7FFFF FFFFFFFF	-NaN

Kind 16 (quadruple precision) real

The figure below shows the bit composition of sign (s), exponent (e), and fraction (f).

Quadruple-precision real storage format:



The table below shows the bit patterns for numbers stored as quadruple-precision real values.

Quadruple-precision real values

Common name	Storage format	Value
0	00000000 00000000 00000000 00000000	0.0E+00
+ maximum normal number	7FFeFFFF FFFFFFFF FFFFFFFF FFFFFFFF	1.189731495357231765085 7593266280070Q+4932
- maximum normal number	FFFeFFFF FFFFFFFF FFFFFFFF FFFFFFFF	-1.189731495357231765085 7593266280070Q+4932
+ minimum normal number	00010000 00000000 00000000 00000000	3.362103143112093506262 6778173217526Q-4932
- minimum normal number	80010000 00000000 00000000 00000000	-3.362103143112093506262 6778173217526Q-4932
+ maximum denormalized number	0000FFFF FFFFFFFF FFFFFFFF FFFFFFFF	3.362103143112093506262 6778173217519Q-4932
- maximum denormalized number	8000FFFF FFFFFFFF FFFFFFFF FFFFFFFF	-3.362103143112093506262 6778173217519Q-4932
+ minimum denormalized number	00000000 00000000 00000000 00000001	6.475175119438025110924 4389582276465Q-4966
- minimum denormalized number	80000000 00000000 00000000 00000001	-6.475175119438025110924 4389582276465Q-4966
+infinity	7FFF0000 00000000 00000000 00000000	Inf
-infinity	FFFF0000 00000000 00000000 00000000	-Inf
+quiet_NaN	7FFF8000 00000000 00000000 00000000 - 7FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	NaN
-quiet_NaN	FFFF8000 00000000 00000000 00000000	-NaN

Common name	Storage format	Value
	- FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	
+signaling_NaN	7FFF0000 00000000 00000000 00000001 - 7FFF7FFF FFFFFFFF FFFFFFFF FFFFFFFF	NaN
-signaling_NaN	FFFF0000 00000000 00000000 00000001 - FFFF7FFF FFFFFFFF FFFFFFFF FFFFFFFF	-NaN

Complex

Half-precision complex type expresses the approximate value of the complex number, and is expressed by a pair of half-precision real type data. The data size of this type is 4 bytes, and 2 bytes in the first half mean the real part, and 2 bytes in the latter half mean the imaginary part. Precision and absolute value of the real part and imaginary part are the same as half-precision real type.

Single-precision complex type expresses the approximate value of the complex number, and is expressed by a pair of single-precision real type data. The data size of this type is 8 bytes, and 4 bytes in the first half mean the real part, and 4 bytes in the latter half mean the imaginary part. Precision and absolute value of the real part and imaginary part are the same as single-precision real type.

Double-precision complex type expresses the approximate value of the complex number, and is expressed by a pair of double-precision real type data. The data size of this type is 16 bytes, and 8 bytes in the first half mean the real part, and 8 bytes in the latter half mean the imaginary part. Precision and absolute value of the real part and imaginary part are the same as double-precision real type.

Quadruple-complex type expresses the approximate value of the complex number, and is expressed by a pair of quadruple-precision real type data. The data size of this type is 32 bytes, and 16 bytes in the first half mean the real part, and 16 bytes in the latter half mean the imaginary part. Precision and absolute value of the real part and imaginary part are the same as quadruple-precision real type.

5.1.4 Character Type Data

Character data is represented internally using ASCII codes. Each character is represented as an 8-bit binary number without a parity bit.

All ASCII characters, both printable and nonprintable, are allowed in character constants and comments.

5.1.5 Derived Type Data

A derived type is a data type derived from the intrinsic data types, whose ultimate components are of intrinsic type. The derived type scalar data, which is a set of intrinsic type data, is called a structure.

If a derived type is not a sequence type, the storage sequence of the structure is undefined.

5.2 Correct Data Boundaries

The compiler usually aligns the data on the correct boundaries.

For elements of derived type and variable in a common block, the compiler assigns the data to the correct boundaries. However, if the compiler option `-AA` is specified, the compiler may not align data on the correct boundaries. In this case, the boundaries are those specified by the user for variables that share storage space because of an `EQUIVALENCE` statement. For a derived type definition, the boundary that follows the last component must be a correct boundary for the derived type.

For example, consider the following derived type definition when the compiler option `-AA` is specified:

```
TYPE TAG
  REAL(8)      R
  INTEGER(4)   I
END TYPE
```

The correct boundary for this derived type (TAG) is an address that is a multiple of 8. But because the boundary for the component I is a multiple of 4, but not 8, this derived type definition is incorrect.

Table 5.1 lists the correct data boundaries.

Table 5.3 Correct boundaries for data types

Data type	Correct boundary
One-byte integer type	Arbitrary address
Two-byte integer type	Address in multiples of 2
Four-byte integer type	Address in multiples of 4
Eight-byte integer type	Address in multiples of 8
One-byte logical type	Arbitrary address
Two-byte logical type	Address in multiples of 2
Four-byte logical type	Address in multiples of 4
Eight-byte logical type	Address in multiples of 8
Half-precision real type	Address in multiples of 2
Single-precision real type	Address in multiples of 4
Double-precision real type	Address in multiples of 8
Quadruple-precision real type	Address in multiples of 16
Half-precision complex type	Address in multiples of 2
Single-precision complex type	Address in multiples of 4
Double-precision complex type	Address in multiples of 8
Quadruple-precision complex type	Address in multiples of 16
Character type	Arbitrary address
Derived type	Address in multiples of the largest required multiple for all of the components

Boundary of array

The first element of an array whose element type's size is 8 bytes or less is allocated on 8-byte boundary, while whose element type's size is more than 8-byte is allocated on 16-byte boundary.

However, arrays passed as arguments, arrays in a COMMON block which are not first element of the block and pointer arrays may not be allocated on above boundary.

5.3 Precision Conversion

The precision of the arithmetic data that Fortran handles is restricted by the hardware. This section explains how to use the precision-improving and precision-lowering functions that the system provides.

5.3.1 Precision Improving

The precision-improving function increases the precision in programs. This function converts constants, variables, and functions of the specified type to the next higher precision.

5.3.1.1 Precision-Raising Options

Precision improving can be executed for real and complex except half-precision constants, variables, and functions.

If compiler option `-Ad` is specified, the data types are converted as follows:

```

Single-precision real      ->    Double-precision real
Single-precision complex  ->    Double-precision complex

```

If compiler option `-Aq` is specified, the data types are converted as follows:

```

Double-precision real      ->    Quadruple-precision real
Double-precision complex  ->    Quadruple-precision complex

```

An example of precision improving for constants, variables, and functions when compiler option `-Ad` is specified as follows.

Example: Precision improving for constants, variables, and functions

```

IMPLICIT REAL(8)(D)
REAL KNOTE,KIN
COMPLEX ARY(10),CDATA
KNOTE = 1.0
KIN = 3.111-2.5D0*KNOTE
DEX = 0.5D0-1.111111111111
ARY(1) = (1.1,1.0)
KIN = KIN-SQRT(KIN/2.0)

```

The above program is equivalent to the following program.

```

IMPLICIT REAL(8)(D)
REAL(8) KNOTE,KIN
COMPLEX(8) ARY(10),CDATA
KNOTE =1.0_8
KIN =3.111_8-2.5D0*KNOTE
DEX =0.5D0-1.111111111111_8
ARY(1) =(1.1_8,1.0_8)
KIN =KIN-DSQRT(KIN/2.0_8)

```

If precision is improved for an intrinsic function, the type of the function result is changed to the higher precision type. The function name is not changed if an `EXTERNAL` statement is used to declare a user-defined function.

For user-defined external functions, precision is improved for the function result. If the precision is improved for a function reference, functions must also be recompiled using the precision-improving function.

For intrinsic functions, precision improving affects function names, the types of arguments, and function results. Intrinsic functions with specific names are changed to the appropriate name. If an intrinsic function has a generic name, the precision-improving function is executed for the arguments, and an appropriate version of the function is selected. If an intrinsic function is used as an actual argument, the name is changed to reflect precision improving.

5.3.2 Precision Lowering and Analysis of Errors

The precision-lowering function determines the effect that a rounding error has on the result. Rounding errors occur because of the limit on floating-point data precision.

Numerical programs can be debugged using the precision-lowering function to check whether algorithm errors are tolerable.

For example, consider programs that solve simultaneous linear equations or obtain eigenvalues or eigenvectors of matrices. In these programs, a slight input error in the matrix may cause an extremely large error. Thus, the user may not be able to determine whether the result is correct. The precision-lowering function is provided to improve and debug numerical programs by detecting sections easily affected by errors. This facility helps the user select the appropriate solution to resolve the errors and minimize the effect of input errors.

Generally, the following relationship can be determined for the number of arithmetic digits and the precision of the answer:

$$d = (p - a)/m$$

d: Precision of answer (digits)

p: Number of arithmetic digits

a: Missing digits
m: Values for the problem and solution method

The value of m is usually 1 and corresponds to the order of multiple roots when roots of algebraic equations are obtained. The preceding equation expresses the problem conditions precisely. However, the equation does not apply when:

- Input data or a constant is erroneous.
- A problem equation contains an error.
- A calculation terminates inappropriately.

That is, the above expression comes into existence when rounding error a prime cause. Incrementing and decrementing the digits by one increments and decrements the solution precision by $1/m$ digits. Thus, a slight shifting of the number of digits shifts the solution precision by the same amount. The precision-lowering function uses this principle to obtain the solution precision.

Compiler option `-C/` enables the precision-lowering function.

5.3.2.1 Defining Precision-Lowering Function

Specifying the following option enables the precision-lowering function:

```
-C1 0 <= I <= 15
```

The precision-lowering function can be used on all kinds (precisions) of real and complex data except half-precision.

If the precision-lowering function is effect, when an assignment statement defines the all kinds of real and complex data except half-precision, the low-order I bits are set to zero.

The precision-lowering number must be between 0 and 15, inclusive. The result when I is zero is equivalent to specifying compiler option `-C/`. The PRNSET service subroutine can be called to change I dynamically. Thus, the precision-lowering function operates when I is changed dynamically to a value other than zero.

The precision-lowering number I is determined for each executable program. Therefore, when the PRNSET service subroutine is called from parallel region, the dynamic change by a specific thread may affect the execution of other threads. The initial value of I is specified when the main program is compiled. Compiler option `-C/` only starts the precision-lowering function in subprograms; I itself is meaningless in this case.

5.3.2.2 Using Precision-Lowering Function

In floating-point calculations, I can be changed, the same calculation re-executed, and the results compared.

First time:

The program is compiled and executed using `-C0`.

Second time:

The program is compiled and executed using `-C1`.

If the digits from two consecutive attempts are identical, do not assume that either result is correct. If the digits from two consecutive attempts are not identical, assume that one or the other of the two results is incorrect.

The number of executions can be increased to improve the accuracy. If you change the value of I to a number from 2 to 10 and increase the number of executions, the precision of the result can be determined accurately. Generally, two executions are sufficient.

However, it is more efficient to use the computer for only one execution to determine the precision. Source code to determine the precision from only one execution must be created. This program must use compiler option `-C/` and include the PRNSET service subroutine. Use the following procedure to create this source code:

1. Save the data to be changed in the check program section.
2. Execute `CALL PRNSET(I)`.
3. Use the saved data in the calculations. Store the result in I .
4. Change I . Return to step 2 if the specified count is not reached.
5. Compare the results for each I . Print the information that matches.

An example of a program that uses the precision-lowering function as follows.

Example: Program that uses the precision-lowering function

```

REAL,DIMENSION(10,11)::A,B
REAL,DIMENSION(10,10)::X
1 READ (5,FMT='(I2,I2)',END=99) N,M ! Enter data
PRINT '(I2,I2)',N,M ! Print data
DO I=1,N
  READ (5,'(5F12.8)') (A(I,J),J=1,N+1)
  WRITE(6,'(5F12.8)') (A(I,J),J=1,N+1)
END DO
DO L=1,M
  CALL PRNSET(L-1)
  CALL SWEEP(A,B,N) ! Calculate while changing
! precision lowering
  X(:N,L)=B(:N,N+1)
END DO
WRITE(UNIT=*,FMT='(//,A,5X,3(A,10X),A)') &
& ' -C1 ', 'X1=1.0', 'X2=2.0', 'X3=1.0', 'X4=-1.0'
DO L=1,M
  WRITE(6,'(A,I1,A,1P8E16.7)') &
  & ' -C',L-1,' ',X(:N,L) ! Print the result
END DO
WRITE(6,'(//,A,7X,3(A,14X),A)') ' -C1:-C0', 'X1', 'X2', 'X3', 'X4'
DO L=2,M
  X(:N,L)=X(:N,L)-X(:N,1)
END DO
DO L=2,M ! Result of -C1 and -C0
  WRITE(6,'(A,I1,A,1P8E16.7)') ' -C',L-1,' :-C0',X(:N,L)
END DO
GO TO 1
99 END
SUBROUTINE SWEEP(A,Z,N)
REAL,DIMENSION(10,11)::A,Z
Z(:N,:N+1)=A(:N,:N+1)
DO K=1,N
  Z(K,K+1:N+1) = Z(K,K+1:N+1)/Z(K,K)
  DO I=1,N ! Use the sweep method to solve the
! simultaneous linear equations
    IF(K.EQ.I) CYCLE
    Z(I,K+1:N+1)=Z(I,K+1:N+1)-Z(I,K)*Z(K,K+1:N+1)
  END DO
END DO
END SUBROUTINE SWEEP

```

In the printed input data, the first line indicates the original values and test number. The second to fifth lines indicate the four coefficients and one constant.

```

4 9
3.20000005  1.25979996  -2.01999998  5.13980007  -1.44019997
1.02010000  -1.35010004  3.09999990  -2.12010002  3.53999996
-2.02979994  2.54999995  -1.37020004  3.64000010  -1.94000006
3.21000004  1.11020005  2.80999994  4.53999996  3.70040011

```

```

-C1      X1=1.0      X2=2.0      X3=1.0      X4=-1.0
-C0      9.9909294E-01  1.9976463E+00  1.0000743E+00  -9.9882913E-01
-C1      9.9581611E-01  1.9891458E+00  1.0003431E+00  -9.9459982E-01
-C2      9.9198890E-01  1.9792151E+00  1.0006566E+00  -9.8966026E-01
-C3      1.0101604E+00  2.0263634E+00  9.9916744E-01  -1.0131168E+00
-C4      1.0316334E+00  2.0820694E+00  9.9740791E-01  -1.0408325E+00
-C5      1.0174370E+00  2.0452271E+00  9.9856758E-01  -1.0225105E+00
-C6      8.9227295E-01  1.7205429E+00  1.0088348E+00  -8.6096954E-01

```

-C7	1.0482483E+00	2.1251831E+00	9.9604034E-01	-1.0623016E+00
-C8	9.1392517E-01	1.7766724E+00	1.0070496E+00	-8.8891602E-01

Difference between -C/and -C0

-C1:-C0	X1	X2	X3	X4
-C1:-C0	-3.2768250E-03	-8.5005760E-03	2.6881695E-04	4.2293072E-03
-C2:-C0	-7.1040392E-03	-1.8431187E-02	5.8233738E-04	9.1688633E-03
-C3:-C0	1.1067390E-02	2.8717041E-02	-9.0682507E-04	-1.4287710E-02
-C4:-C0	3.2540321E-02	8.4423065E-02	-2.6663542E-03	-4.2002678E-02
-C5:-C0	1.8343925E-02	4.7580719E-02	-1.5066862E-03	-2.3681164E-02
-C6:-C0	-1.0681915E-01	-2.7709961E-01	8.7604523E-03	1.3785934E-01
-C7:-C0	4.9155235E-02	1.2753677E-01	-4.0339231E-03	-6.3470840E-02
-C8:-C0	-8.5166931E-02	-2.2097397E-01	6.9752932E-03	1.0991287E-01

Notes:

1. This problem uses the sweep method to solve simultaneous linear equations that have poor characteristics (conditions).
2. When the number of precision-lowering bits varies from 0 to 8, this program prints the difference between the solution and the result of -C0.

The difference in the result of using -C0 almost indicates an error, which shows that debugging is sufficient.

Chapter 6 Notes on Programming

This chapter describes some of notes on programming on the Fujitsu Fortran system.

Input/output features are discussed in "[Chapter 7 Input/Output Processing](#)".

6.1 Fortran Versions

Four versions of Fortran, all compiled using the frtpx command, are available:

1. Fortran 66: Programs written in this version are called Fortran 66 programs. Use compiler option -X6.
2. Fortran 77: Programs written in this version are called Fortran 77 programs. The compiler treats programs as Fortran 77 programs under the following conditions:
 - The compiler option -X7 is specified.
 - The file suffix is ".f", ".for", ".F" or ".FOR", and the compiler option -Xd7 is specified.

Example:

The compiler treats the following program as a Fortran 77 program:

```
$ frtpx a.f90 -X7
```

```
$ frtpx b.f -Xd7
```

3. Fortran 95 (including Fortran 90): Programs written in this version are called Fortran 95 programs. The compiler treats programs as Fortran 95 programs under the following conditions:
 - The compiler option -X9 is specified.
 - The file suffix is ".f90", ".f95", ".F90" or ".F95", and the compiler option -X is not specified.

Example:

The compiler treats the following program as a Fortran 95 program:

```
$ frtpx b.f -X9
```

```
$ frtpx a.f90
```

```
$ frtpx a.f95
```

4. Fortran 2008 (including Fortran 2003): This compiler compiles the Fortran 2003 and Fortran 2008 by the same operations. Programs written in this version are called Fortran 2008 programs. The compiler treats programs as Fortran 2008 programs when any of the following conditions is met:
 - The compiler option -X03 or -X08 is specified.
 - The file suffix is ".f03", ".F03", ".f08", or ".F08", and the compiler option -X is not specified.
 - The file suffix is ".f", ".for", ".F" or ".FOR", and the compiler option -X7 is not specified.

Example:

```
$ frtpx b.f
```

```
$ frtpx a.f90 -X03
```

```
$ frtpx a.f80
```

The compiler treats this program as a Fortran 2008 program:

Each Fortran Version, some items have different interpretations.

They are interpreted in accordance with any of the following Fortran version, and run.

- The Fortran version when the main program is compiled.
- The Fortran version of a program that includes a specification that is fixed when compiling a program

This compiler compiles all Fortran 66, Fortran 77, Fortran 95 (including Fortran 90) and Fortran 2008 (including Fortran 2003) programs. However, differences in the four versions can cause incorrect results.

The following items are interpreted differently, depending on whether compiler option -X6, -X7, -X9, -X03, or -X08 is specified.

The items and their interpretations are shown in the following.

These items are interpreted differently by language specification	The interpretation is decided when the each program is compiled	The interpretation is decided when the main program is compiled
EXTERNAL statement that specifies an intrinsic function (Refer to Section " 6.6.2 An Intrinsic Function that Specifies EXTERNAL Attribute ")	*	
DO loop iteration count method (Refer to Section " 6.5.4.1 DO Loop Iteration Count Method ")	*	
Effect of the X edit descriptor (Refer to Section " 7.16.5 Effect of X Edit Descriptor ")		*
REAL and CMPLX used as generic names (Refer to Section " 6.6.3 REAL and CMPLX Used as Generic Names ")	*	
Intrinsic procedures (Refer to Section " 6.6.4 Intrinsic Procedures ")	*	
Intrinsic function names with the type declared (Refer to Section " 6.6.5 Intrinsic Function Names with Type Declared ")	*	
Assignment statements with overlapping character positions (Refer to Section " 6.4.1 Assignment Statements with Overlapping Character Positions ")	*	
Allocatable assignment (Refer to Section " 6.4.2 Allocatable Assignment ")	*	
SAVE attribute for initialized variables (Refer to Section " 6.2.5 SAVE Attribute for Initialized Variables ")	*	
OPEN statement with RECL= and with no ACCESS= specifier (Refer to Section " 7.4.4 Default Value of ACCESS= Specifier with RECL= Specifier ")	*	*
INQUIRE statement (Refer to Section " 7.6.2 Difference in Language Version of INQUIRE Statement ")		*
Generalized integer editing (Refer to Section " 7.16.1 Generalized Integer Editing ")		*
Output character in formatted output statement (Refer to Section " 7.16.2 Exponent Character in Formatted Output ")		*
Result of G editing (Refer to Section " 7.16.3 Result of G Editing ")		*
NAMelist input/output (Refer to Section " 7.7.4.3 NAMelist Input/Output Version Differences ")		*
Diagnostic messages for a character string edit descriptor (Refer to Section " 7.16.4 Diagnostic Messages for Character String Edit Descriptor ")		*
The form of nondelimited character constants produced by list-directed output (Refer to Section " 7.7.5.2.1 Form of Nondelimited Character Constants Produced by List-Directed Output ")		*

These items are interpreted differently by language specification	The interpretation is decided when the each program is compiled	The interpretation is decided when the main program is compiled
The specified value of the OPEN statement (Refer to Section " 7.4.2 Executing an OPEN Statement on a Connected File ")		*
Input/output of Fortran records (Refer to Section " 7.4.3 Input/Output of Fortran Records ")		*
Data transfer in direct access I/O (Refer to Section " 7.7.3.3 Error of Data Transfer in a DIRECT Access Input/Output ")		*
Input result of L editing (Refer to Section " 7.16.6 L Edit Descriptor ")		*
Variable enclosed by parenthesis (Refer to Section " 6.3.7 Variable Enclosed by Parenthesis ")	*	
The form of simple output list enclosed in parentheses (Refer to Section " 6.5.9 Simple Output List Enclosed in Parentheses ")	*	
The output statement that starts by TYPE keyword (Refer to Section " 6.5.10 Output Statement that Starts by TYPE Keyword ")	*	
List-directed input/output statements (Refer to Section " 7.7.5.3 List-Directed Input/Output Version Differences ")		*
The returned values of ATAN2, ATAN2D and ATAN2Q when negative zero is specified for the argument (Refer to Section " 6.6.7 Returned Values of Intrinsic Function ")		*
The returned values of LOG and SQRT when negative zero is specified for imaginary part of the argument of complex type (Refer to Section " 6.6.7 Returned Values of Intrinsic Function ")		*

6.2 Data

This section discusses features related to data.

6.2.1 COMMON Statement

When using a COMMON statement, note the following items:

6.2.1.1 Boundaries of Variables in Common Blocks

The variables in common blocks must be aligned on the correct boundaries. For more information, see Section "[5.2 Correct Data Boundaries](#)". If data is not aligned on the correct boundaries, the compiler inserts space in the storage area and aligns the data correctly. However, there are exceptions, as described below.

- If the compiler option -AA is specified, the compiler does not insert space.
- Variables of common entities with BIND attribute have interoperable boundary.
- When double precision or quadruple precision real or complex variables are not aligned on addresses that are multiples of eight, these variables are aligned on addresses that are multiples of four. But if the compiler option -Kalign_commons is specified, they are aligned on addresses that are multiples of eight.

The beginning address of a common block is always a multiple of eight.

6.2.1.2 Common Blocks that Have Initial Values

When either of the following is specified, an error is detected at linking:

1. In different files, common blocks with the same name or blank commons are initialized more than once.

Example 1:

Error 1 at linking:

! file name : a.f PROGRAM MAIN COMMON /COM/A,B DATA A/1.0/ CALL SUB END	! file name : b.f SUBROUTINE SUB COMMON /COM/A,B DATA B/2.0/ PRINT *,A,B END
----------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------

2. In different files, common blocks with the same name or blank commons that have different sizes are used, and the common block that does not have the maximum size of the same common blocks is initialized.

Example 2:

Error 2 at linking:

! file name : a.f PROGRAM MAIN COMMON /COM/A,B PRINT *,A END	! file name : b.f BLOCK DATA BLK COMMON /COM/A DATA A/1.0/ END
--------------------------------------------------------------------------	----------------------------------------------------------------------------

6.2.2 EQUIVALENCE Statement

Each equivalence object must be aligned on the correct boundary.

6.2.3 Initialization

This section describes items related to initialization.

6.2.3.1 Character Initialization

If a character constant is specified for the corresponding noncharacter type variable, the system outputs a level w diagnostic message. If a character string that exceeds the length of an element is specified as an initial value, the excess is not used to initialize the subsequent element.

Example1: Initializing array names

```
CHARACTER(3) A(2)
DATA A/'ABCDEF' /
```

Initial values in this case:

```
A(1): 'ABC'
A(2): No initial value (undefined)
```

Example2: Initializing array element

```
CHARACTER(LEN=2) B(4)
DATA B(2) /'ABCDE' /
```

Initial values in this case:

```
B(1) No initial value (undefined)
B(2) 'AB'
B(3) No initial value (undefined)
B(4) No initial value (undefined)
```

The following examples give 'CD' and 'E' as initial values to B(3) and B(4).

```
CHARACTER(LEN=2) B(4)
DATA B(2),B(3),B(4) / 'AB', 'CD', 'E' /
```

or

```
CHARACTER(LEN=2) B(4)
CHARACTER(LEN=6) BB
EQUIVALENCE (B(2),BB)
DATA BB / 'ABCDE' /
```

6.2.3.2 Initialization with Hexadecimal, Octal, and Binary Constants

To initialize an array, the number of hexadecimal, octal, or binary constants must match the number of array elements. If these numbers do not match, the remaining part is undefined. Thus, one hexadecimal, octal, or binary constant corresponds to one array element. If a constant exceeds an element in size, the excess is truncated.

Example: Initializing array elements with hexadecimal values

```
INTEGER A(3)
DATA A/Z'F0F0A100F2F2A202' /
```

Initial value of A(1)	:	F2F2A202
Initial value of A(2)	:	No initial value (undefined)
Initial value of A(3)	:	No initial value (undefined)

If a hexadecimal, octal, or binary constant that exceeds the size of an array element is specified, the high-order excess is truncated.

If a hexadecimal, octal, or binary constant is shorter than an array element, the space to the left of the constant is padded with zeroes.

6.2.3.3 Overlapping Initialization

If multiple initial values are specified for the same variable, the system outputs a level w diagnostic message and uses the last value specified in the source code. However, the code should be corrected.

6.2.4 Using CRAY Pointers

When using CRAY pointers, note the following items:

- Do not save the address of a dummy argument in a subprogram.

Example: Incorrect use of a pointer (1)

```
SUBROUTINE SUB(A)
  POINTER (IP, IPP)
  INTEGER A(10), IPP(10)
  SAVE IP
  :
  IP = LOC(A(2))
  :
  END
```

- Do not return the address of an actual argument as a result.

Example: Incorrect use of a pointer (2)

```
FUNCTION ADDR(I)
  INTEGER ADDR, I(5)
  ADDR = LOC(I(2))
  END
  POINTER (IP, POINTER)
  INTEGER POINTER, DATA(5), ADDR
  IP = ADDR(DATA)
```

```
POINTER = 1
END
```

- c. The data referenced by a pointer must not overwrite other data, except when the address is explicitly defined in a program unit by the intrinsic function LOC. Do not use the address in an assignment statement for which the value is copied.

Example: Incorrect use of a pointer (3)

```
POINTER (IP1,ARRAY1)
POINTER (IP2,ARRAY2)
INTEGER DATA(5),ARRAY1(5),ARRAY2(5)
IP1 = LOC(DATA)
ARRAY1 = 1
IP2 = IP1
ARRAY2 = 2 ! cannot use the address defined
END ! in assignment statement
```

6.2.5 SAVE Attribute for Initialized Variables

In Fortran 95 or later, initialized variables have the SAVE attributes.

In Fortran 66 and Fortran 77, initialized variables do not have the SAVE attribute unless they are given the SAVE attribute explicitly.

Example: The SAVE attribute for a variable with an initial value

```
SUBROUTINE S
INTEGER::I=1
I=I+1
END
```

In Fortran 95, the previous program is the same as the following:

```
SUBROUTINE S
INTEGER::I=1
SAVE I
I=I+1
END
```

6.2.6 Operation of Real or Complex Type for Constant Expression

When real or complex type is operated in constant expression of nonexecutable statement, the result may have some difference within the margin of error even if the same operation appears in executable statement.

6.3 Expressions

This section discusses features related to expressions and operations.

6.3.1 Integer Data Operations

If integer data is assigned, the size is not checked. If the result of integer operations exceeds the range allowed, the result is undefined. This rule is especially critical for integer comparisons.

Example:

```
IF(I-J)1,2,3
```

Note:

In this example, the result of I-J can overflow. Change the expression to a comparison using relational operators in a logical IF statement.

If overflow occurs when evaluating the constant expression that defines a parameter, the value of the named constant is undefined.

6.3.2 Real Data Operations

Real data of any precision is processed as accurately as possible. However, machine language does not always represent data exactly.

Example: Decisions with no tolerance are allowed, but the result may not be what is expected

```
IF (A-0.1) 1,2,1
3 IF (B.EQ.C) GO TO 4
```

Note:

Even if 0.1 is the expected result of calculating A, processing does not always go to the statement with label 2. Even if B and C are expected to be equal, processing does not always go to the statement with label 4.

Do not execute equality or inequality value comparisons with relational operations using real or complex data. Level i diagnostic messages are produced for relational expressions that do not allow tolerance if the -Ec compiler option is specified. Relational expressions with no tolerance allowed are "e1 relop e2", where relop (relational operator) is .EQ., ==, .NE., or /=, and variables e1 and e2 are type complex or real of any kind (precision). See Section "2.2 Compiler Options" for information on the -Ec option. The following is an example.

Example: Preferred decision

```
IF (ABS(A-0.1)-0.00001) 2,1,1
3 IF (ABS(B-C).LT.D) GO TO 4
```

In the constant expression defining a parameter, do not specify a constant expression whose evaluated result might be a denormalized number (see Section "5.1.3 Real and Complex Type Data"). Such a number may cause unexpected results.

6.3.3 Logical Data Operations

Do not assign a value other than the result of a logical expression (.TRUE. or .FALSE.) for logical data and treat that value as logical. For example, do not initialize logical variable with a hexadecimal number and treat the value as logical. Also, do not equivalence a logical and integer variable and treat the integer as logical.

Example: Logical operation using other than logical values

```
LOGICAL A/Z'0000002'/,B
INTEGER I/2/
EQUIVALENCE (I,B)
IF (A) GO TO 10
IF (B) GO TO 20
IF (A.AND.B) GO TO 30
IF (A.OR.B) GO TO 40
```

Note:

In this case, do not expect the logical IF statement to execute the GO TO statement.

The result of a logical expression is TRUE or FALSE. Logical operators do not perform logical operations for each bit.

Example:

```
LOGICAL(1) L1,L2
DATA L1/Z'FF'/
L2=.NOT.L1
```

Note:

In this example, the result of L2 is not necessarily false.

6.3.4 Order of Evaluation of Data Entities

The order of evaluation of data entities connected by an operator is undefined.

For example, it is not allowed in a program to assume that one of F(X) and F(Y) should be evaluated previous to the other in the expression F(X) + F(Y).

6.3.5 Evaluation of a Part of Expression

When an expression is evaluated, some parts of the expression may not be evaluated.

For example, in evaluating the logical expression $L1.AND.L2$, $L2$ may not be evaluated if $L1$ is false because the value of the expression is clearly false without evaluating $L2$.

Therefore, such a function reference as a part of an expression must be written carefully.

6.3.6 Definition and Undefined by Function Reference

Evaluation of a function reference may not change the value of the data entity referred outside the function reference in the same statement.

For example, in the expression $F(I).AND.I=1$, evaluation of the function reference $F(I)$ may not change the value of I .

Evaluation of the function reference in the logical expression of the logical IF statement can exceptionally change the value of the data entity referred in the executable statement of the logical IF statement.

Additionally, evaluation of a function reference in a statement may not change the value of the common block element whose value might affect evaluation of any other function references in the same statement.

For example, it is not allowed evaluating the expression $FA() + FB()$ where functions FA and FB are defined.

Example:

```
FUNCTION FA()  
COMMON /C/A  
FA=A  
END  
FUNCTION FB()  
COMMON /C/A  
A=1  
FB=A  
END
```

6.3.7 Variable Enclosed by Parenthesis

An actual argument is different in Fortran versions when a variable is enclosed by parenthesis in an actual argument of user's procedure.

In Fortran 66 and Fortran 77, the actual argument has the described variable.

In Fortran 95 or later, the value of the variable is copied to the temporary area, and the actual argument has temporary area.

6.4 Assignment Statement

6.4.1 Assignment Statements with Overlapping Character Positions

In Fortran 66 and Fortran 77, if the `-Nf90move` compiler option is not specified, overlapping data (in a storage area) may not be specified on both sides of a character assignment statement. If the `-Nf90move` compiler option is specified, overlapping character data is permitted.

In Fortran 95 or later, overlapping character data is permitted.

Example: In Fortran 95 or later, assignment of data with overlapping character positions

```
CHARACTER(LEN=5) C  
C='12345'  
C(1:4)=C(2:5)  
PRINT *,C ! output is '23455'
```

Note:

In Fortran 66 and Fortran 77, if the `-Nf90move` option is not specified, the assignment is not permitted.

6.4.2 Allocatable Assignment

When `-Nalloc_assign` compiler option is effective and left side of the assignment statement is an allocatable variable, diagnostic message `jwd2881i-i` is output at compilation time and the allocatable assignment of the Fortran 2003 or later standard is operated at execution time.

When `-Nnoalloc_assign` compiler option is effective and left side of the assignment statement is an allocatable variable, diagnostic message `jwd2754i-i` is output at compilation time and the assignment of the Fortran 95 standard is operated at execution time.

The execution performance decreases when the `-Nalloc_assign` compiler option is specified for the source program of the Fortran 95 language specification.

The execution performance never decreases when the `-Nnoalloc_assign` compiler option is specified or the left side does not become an allocatable variable as the following example:

Example: No allocatable variable whose the section subscript

Before: left part is allocatable variable

```
INTEGER,ALLOCATABLE::ARRAY(:)
ALLOCATE(ARRAY(2))
ARRAY=[1,2]
END
```

After: left part is not allocatable variable

```
INTEGER,ALLOCATABLE::ARRAY(:)
ALLOCATE(ARRAY(2))
ARRAY(:)=[1,2]
END
```

6.5 Control Statements

This section describes considerations related to control statements.

6.5.1 GO TO Statement

If a statement label list is present and the destination of an assigned `GO TO` statement is a statement label other than one in the list, no error is output during compilation. However, the program may not operate correctly.

To improve execution efficiency, specify all labels of statements that can be branch targets of the assigned `GO TO` statement.

6.5.2 Arithmetic IF Statement

If the arithmetic expression in an arithmetic `IF` statement is an integer expression and fixed-point overflow occurs, the next statement executed is unpredictable. For example, if the following `IF` statement is executed, the statement executed next could be 10, 20, or 30.

Example:

```
I=3
IF(2147483647+I)10,20,30
```

6.5.3 Logical IF Statement

If an expression in a logical `IF` statement has a value that does not conform to the internal logical data form, execution is unpredictable. For more information, see Section "5.1.2 Logical Type Data", and Section "6.3.3 Logical Data Operations".

Example:

```
LOGICAL TR
EQUIVALENCE(TR,JR)
DATA JR/16/
IF(TR) CALL S
```

Note:

The CALL statement may or may not be executed.

6.5.4 DO Statement

If there is a statement that branches into the range of a DO loop from outside the range of the DO loop, the system outputs a warning diagnostic message. In this case, program operation is unpredictable.

For the extended range of a DO loop defined using Fortran 66, the system outputs a diagnostic message as a warning, and the program operates normally. However, the program should be corrected to ensure future compatibility.

6.5.4.1 DO Loop Iteration Count Method

The following formula defines the DO loop iteration count $r7$ used in Fortran 77 or later.

$$r7 = \text{MAX} (\text{INT} ((m2 - m1 + m3) / m3) , 0)$$

In this formula, $m1$, $m2$, and $m3$ are the initial, terminal, and incrementation parameters of the DO loop. The body of the DO loop is not always executed.

The following formula defines the DO loop iteration count $r6$ used in Fortran 66.

$$r6 = \text{MAX} ((m2 - m1) / m3 + 1 , 1)$$

For Fortran 66, the body of the DO loop is executed at least once.

6.5.4.2 DO CONCURRENT

If a loop control of DO construct has CONCURRENT, the DO construct (DO CONCURRENT) is equal to DO construct with NORECURRENCE specifier in optimization control specifier. See Section "9.10.4 Optimization Control Specifiers" for information about NORECURRENCE specifier.

6.5.5 CASE Statement

If a case expression is an integer expression, the case value shall have the value that can be represent of case expression type and type parameter.

Example:

```
INTEGER(2)::I
...
SELECT CASE (I)
CASE (40000)      ! THE CASE VALUE HAS THE VALUE BETWEEN -32768 TO
...              ! 32767, BECAUSE THE CASE EXPRESSION IS OF TYPE
                  ! TWO-BYTE INTEGER
END SELECT
```

6.5.6 PAUSE Statement

The PAUSE statement writes diagnostic message jwe0001a to the standard error file. If the message is sent to the terminal and the standard input is from the terminal, execution of the program is suspended awaiting a response. Entering any standard input data at the terminal restarts the program. In cases other than the above, the PAUSE statement is ignored and the program continues.

Example: PAUSE statement diagnostic messages

```
PAUSE
jwe0001a pause
```

```
PAUSE n
jwe0001a pause n
```

n : A decimal number of one to five digits

```
PAUSE c
jwe0001a pause c
```

c: A character constant enclosed with apostrophes with a length not exceeding 255

6.5.7 STOP/ERROR STOP Statement

The STOP/ERROR STOP statement causes termination of the executing program, and writes the diagnostic message as follows to the standard error output file.

Statement kind	Stop-code specification	Diagnostic message
STOP	None	-
	Yes	jwe0002i
ERRO STOP	None	jwe0003i
	Yes	

-: No message

The stop-code of the STOP/ERROR STOP statement is reflected in the return code as follows if the environment variable FLIB_USE_STOPCODE has the value 1.

Statement kind	Stop-code	Return code
STOP	(Not specified)	0
	scalar-int-constant-expr	The least significant byte specified by a stop-code (0 to 255)
	scalar-default-char-constant-expr	0
ERROR STOP	(Not specified)	1
	scalar-int-constant-expr	The least significant byte specified by a stop-code (0 to 255)
	scalar-default-char-constant-expr	1

Example: STOP/ERROR STOP statement diagnostic messages

- STOP *scalar-int-constant-expr*

```
STOP 123456
jwe0002i stop 123456
```

```
STOP -3 + 4
jwe0002i stop 1
```

- STOP *scalar-default-char-constant-expr*

```
STOP 'NORMAL'
jwe0002i stop NORMAL
```

```
STOP ('AB'/'CD')/'EF'
jwe0002i stop ABCDEF
```

- ERROR STOP *scalar-int-constant-expr*

```
ERROR STOP 123456
jwe0003i error stop 123456
```

```
ERROR STOP -3 + 2
jwe0003i error stop -1
```

- ERROR STOP *scalar-default-char-constant-expr*

```
ERROR STOP 'ERROR'
jwe0003i stop ERROR

ERROR STOP ('AB'/'CD')/'EF'
jwe0003i stop ABCDEF
```

Example: Return code from the STOP/ERROR STOP statement

Statement kind	Stop-code	Example sentence	Diagnostic message	Return code
STOP	(Not specified)	STOP	-	0
	scalar-int-constant-expr	STOP -3 + 4	jwe0002i stop 1	1
		STOP 400	jwe0002i stop 400	144
	scalar-default-char-constant-expr	STOP 'ABC'	jwe0002i stop ABC	0
		STOP ('AB'/'CD')/'EF'	jwe0002i stop ABCDEF	0
ERROR STOP	(Not specified)	ERROR STOP	jwe0003i error stop	1
	scalar-int-constant-expr	ERROR STOP 222	jwe0003i error stop 222	222
		ERROR STOP -3 + 2	jwe0003i error stop -1	255
	scalar-default-char-constant-expr	ERROR STOP 'ERROR'	jwe0003i error stop ERROR	1
		ERROR STOP ('AB'/'CD')/'EF'	jwe0003i error stop ABCDEF	1

-: No message

6.5.8 ALLOCATE/DEALLOCATE Statement

If the STAT= specifier is specified for an ALLOCATE or DEALLOCATE statement, the following value is returned and execution continues.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end
912	Available memory was insufficient when the ALLOCATE statement was executed.
1001	The allocatable variable specified for an ALLOCATE statement was already allocated.
1003	The allocatable variable specified for a DEALLOCATE statement was not allocated. The deallocation is ignored.
1004	The pointer specified for a DEALLOCATE statement was not associated.
1005	The pointer specified for a DEALLOCATE statement is associated with a target that has not been created by an ALLOCATE statement.

When the large page function is used with the prepagging method, if the execution of an ALLOCATE statement causes out of memory, the process is killed, and the program terminates abnormally, regardless of whether it has a STAT= specifier.

See "Job Operation Software End-user's Guide for HPC Extensions" for information on the large page function.

6.5.9 Simple Output List Enclosed in Parentheses

In Fortran 66, when a simple output list that is enclosed in parentheses and contains named constant is specified, it is output as a simple output list enclosed in parentheses.

In Fortran 77 or later, it is output as a COMPLEX literal constant.

Example:

```
REAL A,B
PARAMETER (A=1.0,B=2.0)
WRITE (*,*) (A,B)
END
```

In Fortran 66, two REAL literals as a simple output list enclosed in parentheses are output as follows:

```
$ ./a.out
 1.00000000 2.00000000
$
```

In Fortran 77 or later, a COMPLEX literal constant is output as follows:

```
$ ./a.out
 (1.00000000,2.00000000)
$
```

6.5.10 Output Statement that Starts by TYPE Keyword

The output statement that starts by TYPE keyword can be compiled and executed in Fortran 66 and Fortran 77. In Fortran 95 or later, the output statement that starts by TYPE keyword is interpreted as a TYPE statement of derived type definition or a TYPE type declaration statement, and the diagnostic message at s level is output. Change the keyword TYPE to PRINT.

Example: The output statement that starts by TYPE keyword

```
A = 1.0
TYPE*, A ! The diagnostic message at s level is output in Fortran 95
          or later.
END
```

6.6 Procedures

This section describes various properties of procedures.

6.6.1 Statement Function Reference

If the types of actual and dummy arguments do not match in a statement function reference, the system outputs a diagnostic message. The actual argument type is used for evaluation. If an arithmetic or logical expression is used for the actual argument, however, the actual argument type must follow the expression conversion rules.

In the following example, the actual arguments are double-precision real for statement function SF. However, statement function ESF causes an error.

Example: Statement function reference

```
REAL(8) X,Y,Z
LOGICAL L,M,N,ESF
SF(I,J,K)=I**2+J**3+K**4
ESF(L,M,N)=.NOT. L .AND. M .AND. N
:
XX=3.0E1+SF(X,Y,Z)           ! Actual arguments of SF are
:                             ! double-precision real
```

```
L=X==Y .AND. ESF(X,Y,Z) ! These arguments not allowed
                        ! by conversion rules
```

6.6.2 An Intrinsic Function that Specifies EXTERNAL Attribute

In Fortran 77 or later, if the EXTERNAL attribute is specified to an intrinsic function name, the intrinsic function characteristic is lost, and the name is processed as a user-defined external procedure or block data program unit.

In Fortran 66, if a basic external function name is specified in an EXTERNAL statement, the name is processed as a basic external function.

Note:

The Fortran 77 or later, intrinsic function is the generic name for a Fortran 66 basic external or intrinsic functions.

6.6.3 REAL and CMPLX Used as Generic Names

The generic function REAL converts the data to type real in Fortran 77 or later, but it only fetches the real part from a complex argument in Fortran 66.

In Fortran 95 or later, if the first argument is of type complex and the second argument is not present in the generic function REAL, the result type parameter is the kind type parameter of the first argument.

The generic function CMPLX converts the data to type complex in Fortran 77 or later, but it only creates a complex number in Fortran 66.

6.6.4 Intrinsic Procedures

In Fortran 77 or later, the following names are interpreted as intrinsic functions. In Fortran 66, the same names are treated as user-defined functions and not as intrinsic functions (basic external functions).

```
ICHAR, CHAR, ANINT, DNINT, QNINT, NINT, IDNINT, IQNINT, DPROD, QPROD, LEN, INDEX, DASIN, QASIN,
DACOS, QACOS, LGE, LGT, LLE, LLT, NOT, IAND, IOR, Ieor, ISHFT, IBSET, IBCLR, BTEST
```

In Fortran 95 or later, the following names are interpreted as intrinsic procedures. In Fortran 66 and Fortran 77, the same names are treated as user-defined procedures.

```
ACHAR, ADJUSTL, ADJUSTR, ALL, ALLOCATED, ANY, ASSOCIATED, BIT_SIZE, CEILING,
COMMAND_ARGUMENT_COUNT, COUNT, CPU_TIME, CSHIFT, DATE_AND_TIME, DIGITS, DOT_PRODUCT, EOSHIFT,
EPSILON, EXPONENT, FLOOR, FRACTION, HUGE, GET_COMMAND, GET_COMMAND_ARGUMENT,
GET_ENVIRONMENT_VARIABLE, IACHAR, IBITS, ISHFTC, KIND, LBOUND, LEN_TRIM, LOGICAL, MATMUL,
MAXEXPONENT, MAXLOC, MAXVAL, MERGE, MINEXPONENT, MINLOC, MINVAL, MODULO, MOVE_ALLOC, MVBITS,
NEAREST, NEW_LINE, NULL, PACK, PRECISION, PRESENT, PRODUCT, RADIX, RANDOM_NUMBER, RANDOM_SEED,
RANGE, REPEAT, RESHAPE, RRSPPACING, SAME_TYPE_AS, SCALE, SCAN, SELECTED_CHAR_KIND,
SELECTED_INT_KIND, SELECTED_REAL_KIND, SET_EXPONENT, SHAPE, SIZE, SPACING, SPREAD, SUM, TINY,
SYSTEM_CLOCK, TRANSFER, TRANSPOSE, TRIM, UBOUND, UNPACK, VERIFY
```

If the -Nobsfun compiler option is specified, the following names are interpreted as intrinsic functions.

```
AIMAX0, AJMAX0, I2MAX0, IMAX0, JMAX0, IMAX1, JMAX1, AIMINO, AJMIN0, I2MIN0, IMIN0, JMIN0, IMIN1,
JMIN1, FLOATI, FLOATJ, DFLOTI, DFLOTJ, IIABS, JIABS, I2ABS, IIDIM, JIDIM, I2DIM, IIFIX, JIFIX,
IFIX, INT1, INT2, INT4, INT8, IINT, JINT, ININT, JNINT, IIDNNT, I2NINT, JIDNNT, IIDINT, JIDINT,
IMOD, JMOD, I2MOD, IISIGN, JISIGN, I2SIGN, BTEST, BJTEST, IBCLR, JIBCLR, IBITS, JIBITS, IIBSET,
JIBSET, IBCHNG, ISHA, ISHC, ISHL, IAND, JIAND, IIEOR, JIEOR, IIOR, JIOR, INOT, JNOT, IISHFT,
JISHFT, IISHFTC, JISHFTC, IZEXT, JZEXT, IZEXT2, JZEXT2, JZEXT4, VAL
```

6.6.5 Intrinsic Function Names with Type Declared

In Fortran 77 or later, even if an intrinsic function is declared with a different type, the intrinsic function characteristic is not lost.

In Fortran 66, if an intrinsic function is declared with a different type, the intrinsic function characteristic is lost. In addition, the intrinsic function is treated as a user-defined function whenever it is used.

6.6.6 Service Routines

If compiler option -AU is specified, a service subroutine name must be entered in lowercase characters.

The validity of the number and type of arguments is checked if the module SERVICE_ROUTINES that includes all service routines explicit interface is used. There are following notes to use the SERVICE_ROUTINES:

- Specify ONLY option and service routines that only using in program because SERVICE_ROUTINES includes all service subroutines and service functions explicit interface. For example,

```
USE SERVICE_ROUTINES, ONLY:ACCESS
```

- If compiler option -AU is specified, module name SERVICE_ROUTINES and each service routine name in USE statement must be in lowercase characters.
- If compiler option -CcdII8, -CciI4I8, -CcdRR8, -CcR4R8, -CcdLL8, -CcL4L8, -Ccd4d8, -Cca4a8, -Ad or -Aq is specified, the corresponding types of argument of service routines is changed. Specify the correspondence runtime option -Lb, -Li, or -Lr.

Note the following if the module SERVICE_ROUTINES is not used.

- The validity of the number and type of arguments is not checked for service routines.
- If compiler option -mldefault=cdecl is specified, the program may not run correctly as follows:
 - Link error may cause for some routines, such as GETTOD service subroutine.
 - If you use a service routine whose name is the same as that of system call or system function, such as FSTAT service function, the system call or system function is called.

For about module and use a module, see "[Chapter 10 Fortran Modules and Submodules](#)".

If the -Nmallocfree compiler option is specified, the MALLOC and the FREE service routines are interpreted as intrinsic procedures.

6.6.7 Returned Values of Intrinsic Function

When negative zero is specified for the argument, the following cases become the different returned value between Fortran 2003 or later and Fortran 95 or earlier.

- ATAN2(Y,X): When $X < 0$ and Y is negative zero, the returned value is -PI in Fortran 2003 or later and PI in Fortran 95 or earlier.
- ATAN2D(Y,X): When $X < 0$ and Y is negative zero, the returned value is -180.0 in Fortran 2003 or later and 180.0 in Fortran 95 or earlier.
- ATAN2Q(Y,X): When $X < 0$ and Y is negative zero, the returned value is -2.0 in Fortran 2003 or later and 2.0 in Fortran 95 or earlier.
- LOG(X): When X is complex with $\text{REAL}(X) < 0$ and the imaginary part of X is negative zero, the imaginary part of the returned value is -PI in Fortran 2003 or later and PI in Fortran 95 or earlier.
- SQRT(X): When X is complex with $\text{REAL}(X) < 0$ and the imaginary part of X is negative zero, the imaginary part of the returned value is negative value in Fortran 2003 or later and positive value in Fortran 95 or earlier.

6.6.8 Precision of Intrinsic Function

In an intrinsic function, the difference of about a rounding error may occur in each version and each architecture by the logic improvement and the optimization which is characteristic of the architecture.

Attention should be paid to precision difference in an intrinsic function.

6.6.9 EXECUTE_COMMAND_LINE Intrinsic Subroutine

If CMDSTAT is specified for an EXECUTE_COMMAND_LINE intrinsic subroutine, the following value is returned and execution continues.

Value 1307 is runtime diagnostic message number. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end
-2	COMMAND is executed synchronously although WAIT is specified with the value false. Asynchronous command execution is not supported in this system.
1307	An error was detected in EXECUTE_COMMAND_LINE intrinsic subroutine.

Chapter 7 Input/Output Processing

This section describes the basic properties of Fortran records and files and the Fortran input/output statements used to process files. Features that are ISO standard Fortran are not discussed in general, except to describe their specific characteristics in the Fujitsu Fortran system. Unless indicated otherwise, "file" means external file in this section.

7.1 Files

Input/output statements process internal and external files. Internal files are stored in main memory.

Internal files consist of scalar character variables, character array elements, character arrays, or substrings.

Internal input/output statements may read or write internal files. External files are stored on external devices and are connected for sequential, direct, and stream access.

Fortran programs process the following types of files:

- Standard input files (stdin)
- Standard output files (stdout)
- Standard error output files (stderr)
- Regular files

Files other than standard files are called regular files. Either sequential, direct, or stream access methods can be used to access regular files. Only the sequential access method can be used to access standard files. Unformatted input/output statements may not be used to access standard files.

7.1.1 Old and New Files

New files may be created when an executable program is started or at any time during its execution.

Old and new files may be accessed when connected to a unit number specified in an input/output statement.

7.1.2 File Connection

A file and unit must be connected to execute a data transfer or file positioning input/output statement.

7.1.2.1 Connection

Files are connected in two ways:

- Dynamic connection (during execution of a program)
- Preconnection

Dynamic connection is performed when an OPEN, READ, WRITE, BACKSPACE, ENDFILE, or PRINT statement is executed.

However, dynamic connection for direct or stream access may be performed only by an OPEN statement.

Preconnection is performed by an external specification for a Fortran program before an executable program is started.

The standard input file (stdin, unit 5), the standard output file (stdout, unit 6), and the standard error output files (stderr, unit 0) are always preconnected. Executing an OPEN statement with a FILE= specifier disconnects a standard file.

7.1.2.2 File Disconnection

Files are disconnected in the following cases.

- Execution of an CLOSE statement
- Termination of the execution program
- Execution of an OPEN statement with different file name to the same unit number.

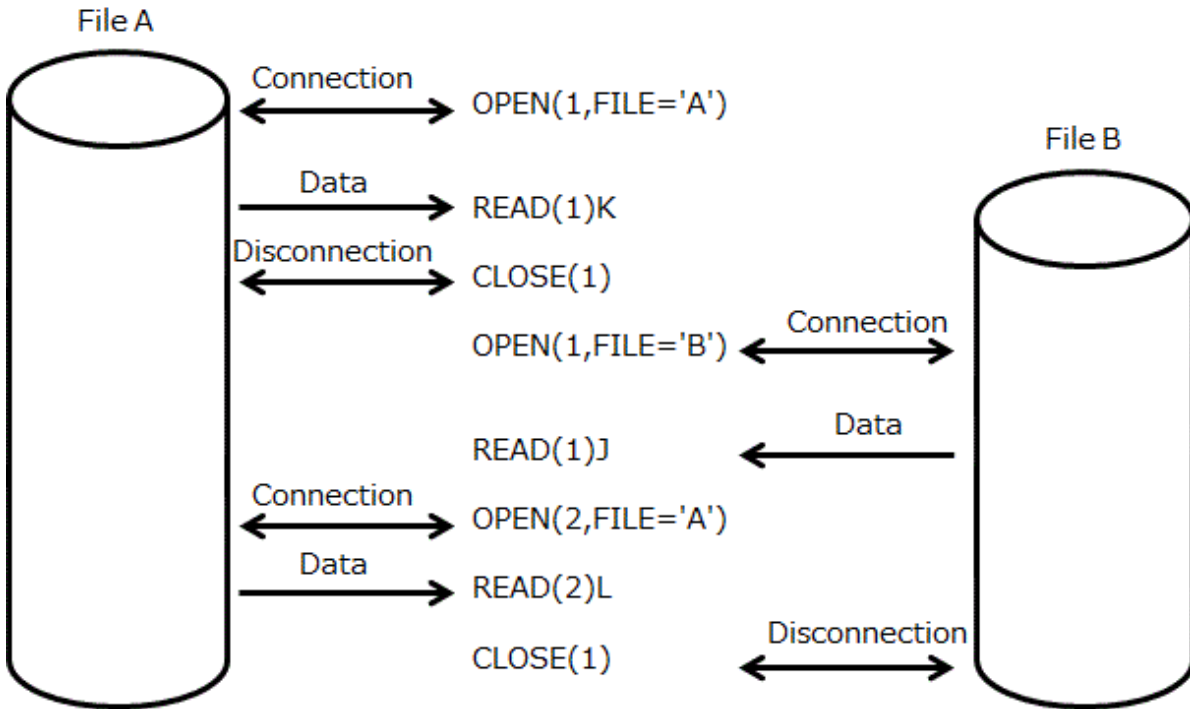
The following processes are done in the disconnection of the file.

- Termination of the connection of specified unit to an external file
- Deletion or keeping of the file that is connected to the specified unit

A file that has been disconnected may be connected by an OPEN statement.

The only means of connecting a file that has been disconnected is by appearance of its name in an OPEN statement.

There may be no means if reconnecting an unnamed file once it is disconnected.



7.2 Unit Numbers and File Connection

Units and files are connected as follows:

- Execute an OPEN statement with a filename specified.
- Use an environment variable outside the program to pre-connect a unit and file.

7.2.1 Priority of Unit and File Connection

The way of connecting it in the case of standard input, standard output and the standard error output file and that order of priority are shown in the following.

How to specify	Explanation	Priority
An OPEN statement with FILE= specifier	Connection by an OPEN statement with FILE= specifier	High
Runtime options	Connection by the runtime options. - <i>pu_no</i> (standard output) - <i>ru_no</i> (standard input) - <i>mu_no</i> (standard error output)	↕
Default value of Fortran system	Connection by Fortran System. Standard output (unit number 6)	Low

How to specify	Explanation	Priority
	Standard input (unit number 5) Standard error output (unit number 0)	

The way of connecting it and that order of priority are shown in the following in the case as the named file and the unnamed file.

How to specify	Explanation	Priority
An OPEN statement with FILE= specifier	Connection by an OPEN statement with FILE= specifier	High
The environment variable	Connection by the environment variable (fu xx).	Low

7.2.2 Environment Variables to Connect Units and Files

Environment variables may be used to connect units and files. How to specify file name by using an environment variable is shown in the following.

Name of Environment variable	Value
fu xx	<i>File-name</i>

fu: Fixed

xx: Unit number (00 to 2147483647)

File-name: Filename (full) pathname or filename

Unit numbers 0, 5, and 6 (set by runtime options -m, -r, and -p) are preconnected for error output, standard input, and standard output, respectively. These unit numbers may not be connected using an environment variable.

Environment variable file connections are disabled when a file is closed and remain disabled thereafter. The input/output statements that may close any of the standard files are as follows:

- CLOSE statement
- OPEN statement with a file specified

Example 1: Connect "test.data" to unit number 1

Environment variable setting:

```
$export fu01="test.data"
```

Example 2: Disabled connect by environment variable

Fortran program:

```
OPEN(1,FILE='b.file')
...
```

Commands:

```
$ export fu01=a.file
$ ./a.out
```

When a.out is executed, file connection by environment variable (connection to a.file) is disabled, and the filename specified an OPEN statement (connection to b.file) is enabled.

7.3 Fortran Records

Fortran records contain data in internal machine format or character strings. One Fortran record does not always correspond to one physical record. The Fortran record types are as follows:

- Formatted (see Section "7.3.1 Formatted Fortran Records")

- Unformatted (see Section "7.3.2 Unformatted Fortran Records")
- List-Directed (see Section "7.3.3 List-Directed Fortran Records")
- Namelist (see Section "7.3.4 Namelist Fortran Records")
- Endfile (see Section "7.3.5 Endfile Records")
- Binary (see Section "7.3.6 Binary Fortran Records")
- Stream (see Section "7.3.7 STREAM Fortran Records")

Fortran record characteristics are shown below.

Fortran record type	Input/output statement	Record format	One Fortran record
Formatted	Formatted sequential	Undefined length	One logical record
	Formatted direct	Fixed length	
	Internal file		One scalar character variable, array element, or substring. For an array, each element is a Fortran record.
Unformatted	Unformatted sequential		Variable length
	Unformatted direct	Fixed length	
List-directed	List-directed, Print statement	Undefined length	One logical record
	Internal file	Fixed length	One scalar character variable, array element, or substring. For an array, each element is a Fortran record.
Namelist	Namelist	Undefined length	One logical record from the &namelist name up to / or &end.
	Internal file	Fixed length	One scalar character variable, array element, or substring. For an array, each element is a Fortran record.
Endfile	Input/output statement other than direct	----	The last Fortran record does not have the length attribute.
Binary	Unformatted sequential	Fixed length (*1)	One logical record
	Unformatted direct	Fixed length (*1)	
Stream	formatted stream	Undefined length	One logical record
	Unformatted stream	Fixed length (*1)	One logical record

*1. The Fortran record length is determined according to the size of data for which input/output is to be executed.

Fortran records exist as logical records on external devices. An undefined length record consists of a Fortran record and the line-feed character (\n). A fixed length record consists of a Fortran record. A variable length record consists of a Fortran record and 8 bytes (a 4-byte length before and after the Fortran record).

7.3.1 Formatted Fortran Records

Formatted Fortran records can consist of any character string. Formatted Fortran records are accessed by formatted sequential, formatted direct, and internal file input/output statements.

7.3.1.1 Formatted Sequential Record

Files controlled by formatted sequential input/output statements have an undefined length record format.

One Fortran record corresponds to one logical record. The length of the undefined length record depends on the Fortran record to be processed. The max length may be assigned in OPEN statement RECL= specifier.

The line-feed character (\n) terminates the logical record. If the \$ edit descriptor or \ edit descriptor is specified for the format of the format of the formatted sequential output statement, the Fortran record does not include the line feed character (\n).

The relationship between formatted Fortran records and the undefined length record format is shown below.

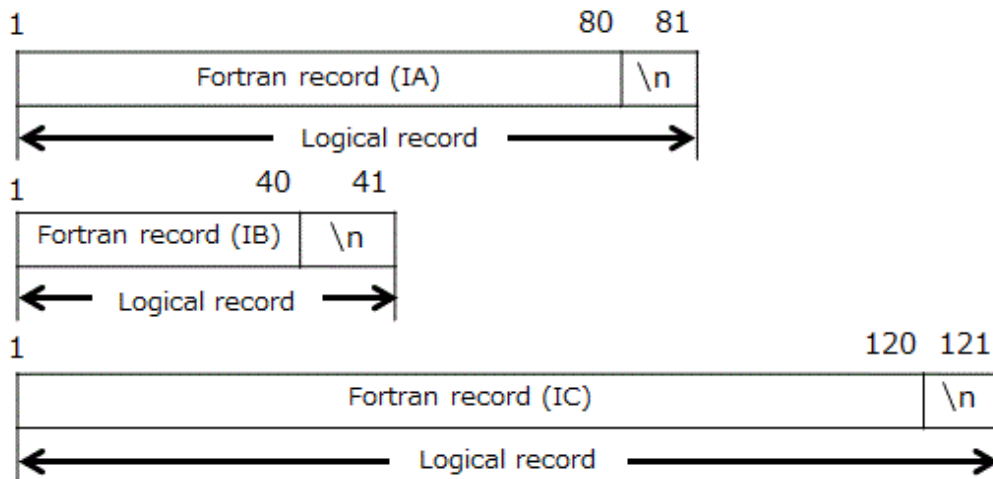
Fortran program:

```

    INTEGER,DIMENSION(20) :: IA
    INTEGER,DIMENSION(10) :: IB
    INTEGER,DIMENSION(30) :: IC
    WRITE(2,10) IA,IB
10  FORMAT(20I4,/,10I4)
    WRITE(2,20) IC
20  FORMAT(30I4)

```

Fortran records:



7.3.1.2 Formatted Direct Record

Files processed by formatted direct input/output statements have a fixed length record format. One Fortran record corresponds to one logical record. The length of the logical record must be assigned in the OPEN statement RECL= specifier. If the Fortran record is shorter than the logical record, the remaining part is padded with blanks. The length of the Fortran record must not exceed the logical record. This fixed length record format is unique to Fortran.

The relationship between formatted Fortran records and the fixed length record format is shown below.

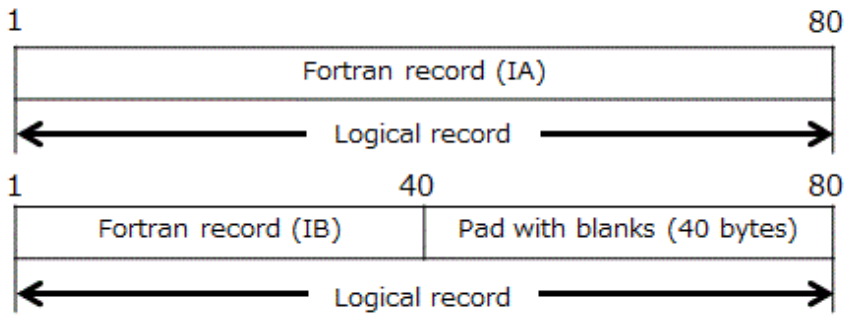
Fortran program:

```

    INTEGER,DIMENSION(20) :: IA
    INTEGER,DIMENSION(10) :: IB
    OPEN(UNIT=3,ACCESS='DIRECT',RECL=80,FORM='FORMATTED')
    WRITE(3,FMT=10,REC=1) IA,IB
10  FORMAT(20I4)

```

Fortran record:



7.3.1.3 Internal File Record

The fixed length records are processed by internal file input/output statements. The records are stored in internal files. Internal files that are a scalar character variable, character array element or substring contain only one Fortran record. The length of the Fortran record must not exceed the size of the scalar character variable, character array element, or substring. When an internal file is a character array, the length of a Fortran record must not exceed the size of an array element of the character array. If the length of the Fortran record is shorter than the size of the scalar character variable, character array element or substring, the remaining part is padded with blanks. For input, the lengths must be equal.

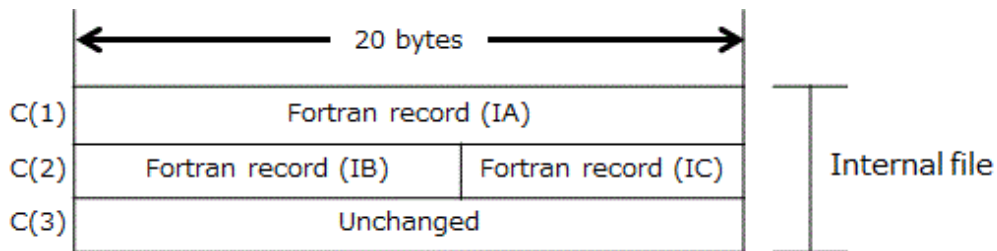
The relationship between formatted Fortran records and internal files is shown below.

Fortran program:

```

CHARACTER (LEN=20) C(3)
INTEGER, DIMENSION(5) :: IA
INTEGER, DIMENSION(3) :: IB
INTEGER, DIMENSION(2) :: IC
WRITE (C, 10) IA, IB, IC
10  FORMAT(5I4)
    
```

Fortran record (Internal file: C):



7.3.2 Unformatted Fortran Records

An unformatted Fortran record consists of a string of values (character or noncharacter data, and a string can be empty). The length depends on the input/output source program statement (zero length can be used).

Both unformatted sequential and unformatted direct input/output statements can use unformatted Fortran records.

7.3.2.1 Unformatted Sequential Record

Files processed using unformatted sequential input/output statements have a variable length record format.

One Fortran record corresponds to one logical record. The length of the variable length record depends on the length of the Fortran record. The length of the Fortran record includes 4 bytes added to the beginning and end of the logical record. If the runtime option -W1,-Lu is specified, the size of the field to put the length has 8 bytes. The max length may be assigned in the OPEN statement RECL= specifier. The beginning area is used when an unformatted sequential statement is executed. The end area is used when a BACKSPACE statement is

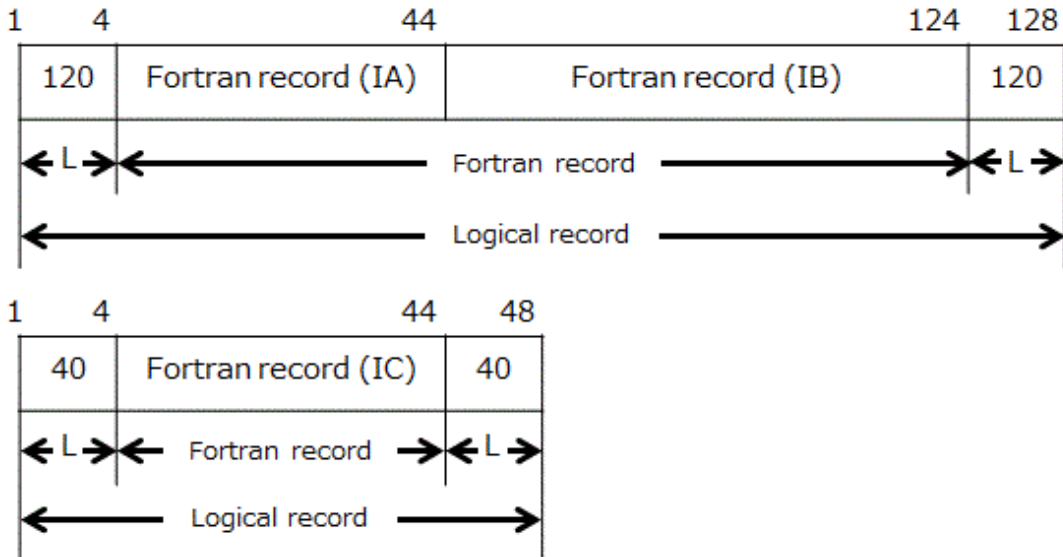
executed. This variable length record format is unique to Fortran. The relationship between unformatted Fortran records and the variable length record format is shown below.

Fortran program:

```

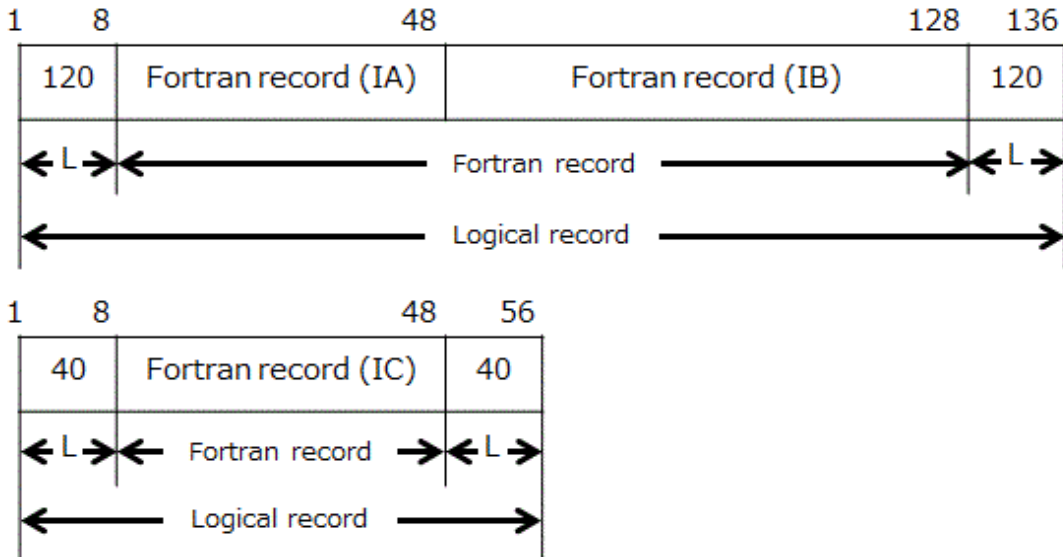
INTEGER, DIMENSION(10) :: IA, IC
INTEGER, DIMENSION(20) :: IB
WRITE(2) IA, IB
WRITE(2) IC
    
```

Fortran record (the runtime option -WL,-Lu is not specified):



L: Length of Fortran record

Fortran record (the compiler option is valid and the runtime option -WL,-Lu is specified):



L: Length of Fortran record

The Fortran record that length is 2G bytes (2147483648 bytes) or more is input/output by dividing the record. The length of each divided Fortran record is from 1 to 2147483647 bytes.

The value set to the top and the end of a logical record is as follows because the divided Fortran records are considered single Fortran record.

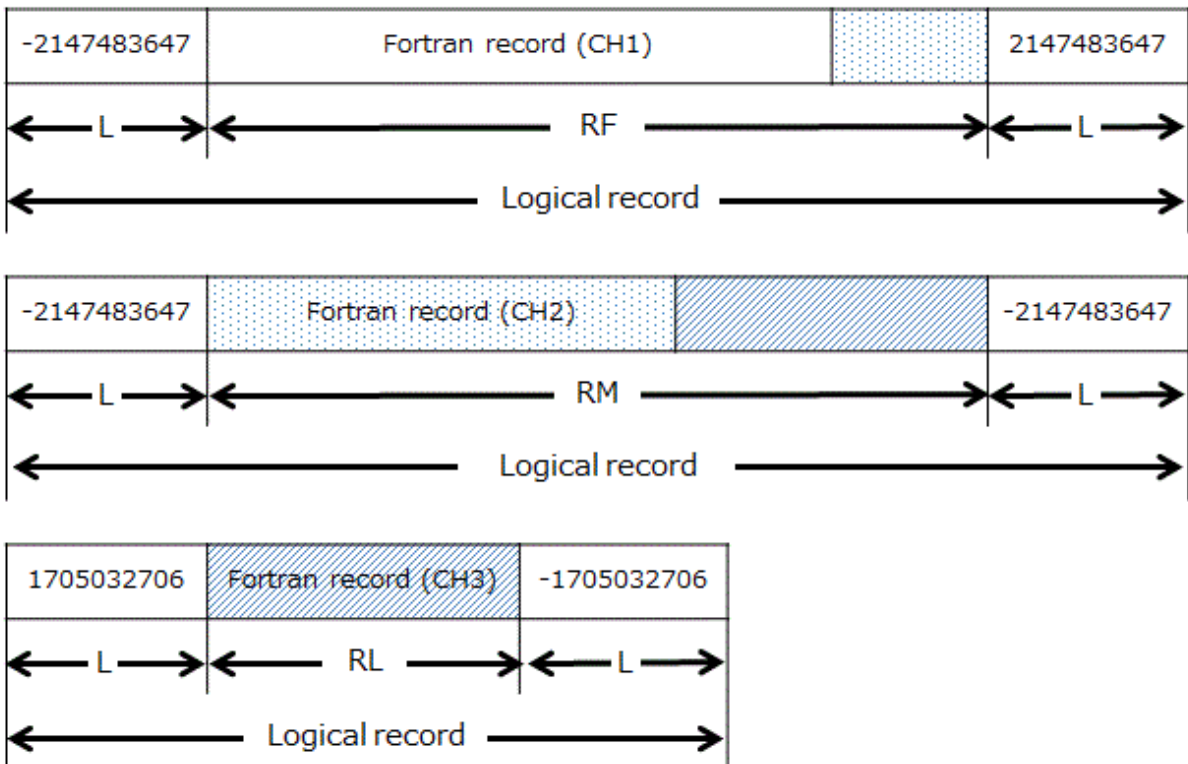
Divided Fortran record	The value of top of logical record	The value of end of logical record
First	A negative value that the absolute value is equal to the length of the divided Fortran record is set. (-2147483647 is set)	The length of the divided Fortran record is set. (2147483647 is set)
Middle	A negative value that the absolute value is equal to the length of the divided Fortran record is set. (-2147483647 is set)	A negative value that the absolute value is equal to the length of the divided Fortran record is set. (-2147483647 is set)
Last	The length of the divided Fortran record is set.	A negative value that the absolute value is equal to the length of the divided Fortran record is set.

The relationship between unformatted Fortran records that length is 2G bytes or more and the variable length record format is shown below.

Fortran program:

```
CHARACTER(LEN=2000000000) :: CH1,CH2,CH3
OPEN(10,FILE="work.data",FORM="UNFORMATTED")
WRITE(10) CH1,CH2,CH3
```

Fortran record:



- L: Length of Fortran record
- RF: Divided Fortran record (First)
- RM: Divided Fortran record (Middle)
- RL: Divided Fortran record (Last)

7.3.2.2 Unformatted Direct Record

Files processed by unformatted direct input/output statements have a fixed length record format. One Fortran record can correspond to more than one logical record. See Section "7.3.1.2 Formatted Direct Record" for details.

The record length must be assigned in the OPEN statement RECL= specifier. One Fortran record can consist of more than one logical record. If the Fortran record terminates within a logical record, the remaining part is padded with binary zeros. If the length of the Fortran record exceeds the logical record, the remaining data goes into the next record. This fixed length record format is unique to Fortran.

7.3.3 List-Directed Fortran Records

List-directed input/output statements, print statement, or internal file input/output statements are used to process list-directed Fortran records. List-directed Fortran records consist of data items and value separators. The data items are input/output character strings. The length of the Fortran record depends on the number and type of items. Data items from the beginning of a logical record up to the end of the input are handled as one Fortran record. Files processed using list-directed input/output statements have an undefined length record format or a fixed length record format. See Section "[7.3.1.1 Formatted Sequential Record](#)" for details. See Section "[7.3.1.3 Internal File Record](#)" for details.

7.3.4 Namelist Fortran Records

Namelist input/output statements and internal input/output statements access namelist Fortran records.

Namelist Fortran records consist of data items (character strings specified by a namelist name) from the &namelist name up to / or &END. The correspondence between namelist Fortran records and logical records is the same as for Fortran records handled using list-directed input/output statements. Files accessed by namelist input/output statements have an undefined length record format. See Section "[7.3.1.1 Formatted Sequential Record](#)" for details. See Section "[7.3.1.3 Internal File Record](#)" for details.

7.3.5 Endfile Records

An endfile record must be the last record of a file connected for sequential access. Endfile records do not have a length attribute.

The ENDFILE statement writes endfile records in files connected for sequential access. After at least one WRITE statements is executed, endfile records are output under the following conditions:

- A REWIND statement is executed.
- A BACKSPACE statement is executed.
- A CLOSE statement is executed.

7.3.6 Binary Fortran Records

A binary Fortran record consists of a string of values (character or noncharacter data). The length of the binary Fortran record depends on the input/output list (it may be zero length). One Fortran record corresponds to one logical record. An unformatted sequential access or an unformatted direct access input/output statement may access binary Fortran records. See Section "[7.3.1.2 Formatted Direct Record](#)" for details.

7.3.7 STREAM Fortran Records

A STREAM Fortran record is composed of the file storage units, and its storage position can be uniquely identified using positive integers.

STREAM Fortran records can be accessed by stream input/output statements.

7.3.7.1 Formatted Stream Record

Files controlled by formatted stream input/output statements have an undefined length record format.

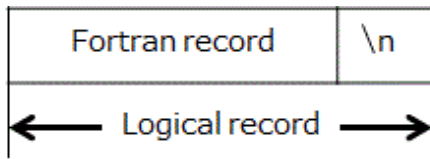
One Fortran record corresponds to one logical record. The length of the undefined length record depends on the Fortran record to be processed. The line-feed character (\n) terminates the logical record. If the \$ edit descriptor or \ edit descriptor is specified for the format of the formatted stream output statement, the Fortran record does not include the line feed character (\n).

Fortran program:

```
OPEN(10, FILE='x', ACCESS='stream', FORM='formatted')
DO I=1,10
  INQUIRE(10, POS=N)
  WRITE(10, '(I4)', POS=N) I
```

```
ENDDO
CLOSE (10)
```

Fortran record:



7.3.7.2 Unformatted Stream Record

Files controlled by unformatted stream input/output statements have a fixed length record format.

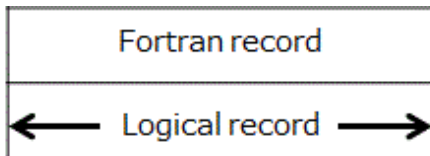
The length of the undefined length record depends on the Fortran record to be processed.

The relationship between unformatted Fortran records and the fixed length record format is shown below.

Fortran program:

```
OPEN(10, FILE='x', ACCESS='stream', FORM='unformatted')
DO I=1,10
WRITE(10,POS=4*I ) I
ENDDO
CLOSE(10)
```

Fortran record:



7.4 OPEN Statement

7.4.1 OPEN Statement Specifiers

These topics explain the parameters that can be specified in an OPEN statement. The parameters are:

- FILE= Specifier
- STATUS= Specifier
- RECL= Specifier
- ACTION= Specifier
- BLOCKSIZE= Specifier
- CONVERT= Specifier
- NEWUNIT= Specifier

The character strings that can be specified with STATUS=, ACCESS=, FORM=, BLANK=, ACTION=, PAD=, POSITION=, DELIM=, ASYNCHRONOUS=, DECIMAL=, ROUND=, SIGN=, and CONVERT= specifiers depend on the parameter.

The characters in the specifier may be any mixture of upper and lower case; however, ignoring case, they must match a legal specifier exactly. Blanks at the beginning are ignored. If the result is not an allowed value, diagnostic message jwe0086i-e is output, and the parameter is ignored. A specifier may contain trailing blanks.

Example values for the OPEN statement ACCESS= specifier are shown below. The other parameters are handled similarly.

Example:

Correct character expressions are:

```
OPEN(10,ACCESS='SEQUENTIAL')
OPEN(10,ACCESS='sequential')
OPEN(10,ACCESS='SeQuEnTial')
OPEN(10,ACCESS='SEQUENTIAL  ')
OPEN(10,ACCESS='    SEQUENTIAL')
```

Incorrect character expressions are:

```
OPEN(10,ACCESS='SEQUENTIALAB')
OPEN(10,ACCESS='SEQUEN')
OPEN(10,ACCESS='S EQUENTIAL')
OPEN(10,ACCESS='    ')
```

7.4.1.1 FILE= Specifier

The FILE= specifier specifies a file name. Any trailing blanks are ignored.

Filename specified in a FILE= specifier is shown below, where it is assumed that the current directory is /home/v1954.

Interpreted as /home/v1954/file.data1 are:

```
OPEN(10,FILE='/home/v1954/file.data1')
OPEN(10,FILE='file.data1')
OPEN(10,FILE=' file.data1')
```

Interpreted as /home/c1121/file.data1 is:

```
OPEN(10,FILE='../c1121/file.data1')
```

Filename is specified with the FILE= specifier and the STATUS= specifier. The tables below list the relationship between FILE= and STATUS= specifiers.

FILE= specifier	STATUS= specifier		
	NEW, OLD, SHR, REPLACE	SCRATCH	UNKNOWN
FILE= specifier	The system treats the specified value as a named file.	An Error.	The system treats the specified value as NEW or OLD, depending on the file status.
No FILE= specifier (no environment variable)	The system treats the specification as a named file with file name fort. <i>nn</i> (<i>nn</i> : unit number).	The system treats the specification as an unnamed file.	The system treats the specification as a named file with file name fort. <i>nn</i> (<i>nn</i> : unit number).
No FILE= specifier (environment variable allocated)	The system treats the file name specified by the shell variable as a named file.	An Error.	The system treats the file name specified by the environment variable as a named file.

Notes:

- If a FILE= specifier is not specified, processing depends on whether the units and file are connected by an environment variable (See Section "7.2 Unit Numbers and File Connection").
- If a name is not given, the name used is 'tF*nnn*', where *nnn* is a unique number. See "tempnam(3s)".

7.4.1.2 STATUS= Specifier

The STATUS= specifier must correspond to the file status. That is, NEW or SCRATCH must not be specified for an existing file, and if OLD or SHR is specified, the file must exist. When UNKNOWN is specified, the Fortran system determines the file status. If the STATUS= specifier is omitted, UNKNOWN is used by default.

If UNKNOWN and the FILE= specifier are specified, and the file does not already exist, the status is defined as NEW. If UNKNOWN and the FILE= specifier are specified, and the file does exist, the status is defined as OLD. If UNKNOWN and the environment variable are specified, and a FILE= specifier is not specified, the status is defined as OLD. If UNKNOWN is specified, and a FILE= specifier and the environment variable are not specified, the status is defined as SCRATCH. The FILE= specifier may be present when STATUS is SCRATCH.

The tables below list the relationship between file and STATUS= specifier.

SHR is the same as OLD. If SHR is specified, the file must exist, and a FILE= specifier must be specified at the same time. When SHR is specified, the connected file can be shared with other executable programs.

Relationship between File and STATUS= specifier

STATUS= specifier	File
NEW	Specified for a file that does not exist.
OLD, SHR	Specified for a file that exists.
SCRATCH	Specified for a file that does not exist. The file is deleted when the program is finished or the file is closed.
UNKNOWN	Specified if the file status is unknown. The Fortran system determines the status as follows: <ul style="list-style-type: none"> - If the FILE= specifier is specified, the status is NEW if the file does not exist. The status is OLD if the file exists. - If no FILE= specifier is specified, the status is SCRATCH if the file was not assigned using an environment variable. The status is OLD if the file was assigned using an environment variable.
REPLACE	If the file does exist, the file is deleted, and a new file is created with the same name.

7.4.1.3 RECL= Specifier

The RECL= specifier specifies the length of each Fortran record in files in which DIRECT is specified by the ACCESS= specifier. An OPEN statement with RECL= specified must be executed before any direct input/output statements are executed for that file. The value specified for an existing file must be the same as for each record in the file.

The RECL= specifier indicates the maximum length of a Fortran record in files with SEQUENTIAL specified by the ACCESS= specifier. If the RECL= specifier is omitted, 2147483647 is used by default.

The value of the RECL= specifier must be positive.

The RECL= specifier must not be specified for a file connected for stream access.

The tables below list the relationship between ACCESS= specifier and RECL= specifier.

Relationship between ACCESS= specifier and RECL= specifier

ACCESS= specifier	With RECL= specifier		Without RECL= specifier
	In Fortran or later	In Fortran 77 and Fortran 66	
SEQUENTIAL	Sequential access	Sequential access (*1)	Sequential access
DIRECT	Direct access	Direct access	Error
STREAM	Error	Error	Stream access
None	Sequential access	Direct access (*2)	Sequential access

*1: The diagnostic message (jwe0097i-w) is generated, and RECL= specifier is ignored.

*2: The compiler message (jwd1449i-i) is generated.

7.4.1.4 ACTION= Specifier

When READ is specified, "r" is the permission of the file; that is, WRITE and PRINT statements may not be executed for the file.

When WRITE is specified, "w" is the permission of the file; that is, READ, BACKSPACE, and REWIND statements cannot be executed for the file. For direct access, READWRITE or BOTH is assumed even if WRITE is specified.

BOTH is the same as READWRITE.

When READWRITE or BOTH is specified, "r+" or "w+" is the permission of the file; that is, all input/output is allowed for the file. When a file with read only permission is opened, READ must be specified. Therefore, when the WRITE statement is executed, a diagnostic message is output even if READWRITE or BOTH is specified. If the ACTION= specifier is omitted, READWRITE is used by default.

The tables below list the relationship between ACTION= specifier and file status.

Relationship between ACTION= specifier and Files

Access method	File STATUS	ACTION= specifier	Open mode	Notes			
Sequential	NEW	READ	Error	ACTION='READ' is not allowed for temporary files and new files.			
		WRITE	w				
		READWRITE or BOTH	w+				
	OLD, SHR	READ	r		ACTION='READ' is not allowed for temporary files and new files.		
		WRITE	w				
		READWRITE or BOTH	r+ (*1)				
	SCRATCH	READ	r			ACTION='READ' is not allowed for temporary files and new files.	
		WRITE	w				
		READWRITE or BOTH	w+ (*1)				
	REPLACE	READ	r				ACTION='READ' is not allowed for temporary files and new files.
		WRITE	w+				
		READWRITE or BOTH	W+ (*2)				
Direct	NEW	READ	Error	ACTION='READ' is not allowed for temporary files and new files.			
		WRITE	w				
		READWRITE or BOTH	w+				
	OLD, SHR	READ	r		ACTION='READ' is not allowed for temporary files and new files.		
		WRITE	r+				
		READWRITE or BOTH	r+ (*2)				
	SCRATCH	READ	r			ACTION='READ' is not allowed for temporary files and new files.	
		WRITE	w+				
		READWRITE or BOTH	r+ (*2)				
	REPLACE	READ	r				ACTION='READ' is not allowed for temporary files and new files.
		WRITE	w+				
		READWRITE or BOTH	w+ (*2)				

Access method	File STATUS	ACTION= specifier	Open mode	Notes	
Stream	NEW	READ	Error	ACTION='READ' is not allowed for temporary files and new files.	
		WRITE	w		
		READWRITE or BOTH	w+		
	OLD, SHR	READ	r		r+ (*1)
		WRITE	w		
		READWRITE or BOTH			
	SCRATCH	READ	r		w+ (*1)
		WRITE	w		
		READWRITE or BOTH			
	REPLACE	READ	r		w+ (*2)
		WRITE	w		
		READWRITE or BOTH			

*1: A file with the read-only attribute is opened in 'r' mode.

*2: A file with the read-only attribute causes an error.

7.4.1.5 BLOCKSIZE= Specifier

The BLOCKSIZE= specifier specifies the sequential or the stream access input/output buffer size in bytes. The default buffer size is 8M bytes.

7.4.1.6 CONVERT= Specifier

When BIG_ENDIAN is specified in the CONVERT= specifier, no asynchronous data transfer is performed when an asynchronous input/output statement is executed on that unit.

7.4.1.7 NEWUNIT= Specifier

The value defined for the variable specified in the NEWUNIT= specifier is as follows.

- If the type of the variable specified in the NEWUNIT specifier is a 1-byte integer type, -10 to -128 are assigned. More than 119 files cannot be connected at the same time by the execution of an OPEN statement specifying the NEWUNIT= specifier.
- If the type of the variable specified in the NEWUNIT= specifier is a 2-byte integer type, 4-byte integer type, or 8-byte integer type, -10 to -16384 are assigned. More than 16375 files cannot be connected at the same time by the execution of an OPEN statement specifying the NEWUNIT= specifier.

If the NEWUNIT= specifier appearance in an OPEN statement, either the FILE= specifier shall appear, or the STATUS= specifier shall appear with a value of SCRATCH.

7.4.2 Executing an OPEN Statement on a Connected File

If the file connected to a unit is the same as the file to be connected, only BLANK=, DELIM=, PAD=, DECIMAL=, ROUND=, SIGN=, ERR=, and IOSTAT= specifiers may have values different from those currently in effect. If the other specifiers have different value, the specified value is ignored in Fortran 66 and Fortran 77. In Fortran 95 or later, the runtime message (jwe1115i-w) is output and the specified value is ignored.

In addition, if the value of the STATUS= specifier is other than OLD, in the Fortran 2003 or later language specifications the diagnostic message jwe1115i-w is output, and the status of the external device that was connected immediately beforehand takes effect.

Example:

```
$ cat ap.f
OPEN(10,FILE='XX.DAT',POSITION='APPEND')
WRITE(10,*)123
OPEN(10,FILE='XX.DAT',POSITION='REWIND')
WRITE(10,*)456
END
```

In Fortran 66 and Fortran 77:

```
$ ./a.out
$ more XX.DAT
123
456
$
```

In Fortran 95 or later:

```
$ ./a.out
jwell115i-w line 7 The value of POSITION specifier in an OPEN statement is different from that
currently in effect (unit=10).
error occurs at MAIN__ line 7 loc 0000000000400b51 offset 0000000000000041
MAIN__ at loc 0000000000400b10 called from o.s.
taken to (standard) corrective action, execution continuing.
error summary (Fortran)
error number error level error count
jwell115i w 1
total error count = 1
$ more XX.DAT
123
456
$
```

7.4.3 Input/Output of Fortran Records

If an input/output statement is executed that is inconsistent with the file properties of the connection (open), the specifications in the input/output statement are used in Fortran 66 and Fortran 77. In Fortran 95 or later, the runtime message (jwe0113i-e) is output and the input/output statement is ignored.

Example:

```
$ cat fm.f
OPEN(10,FILE='XX.DAT',FORM='UNFORMATTED')
WRITE(10,*)123
END
```

In Fortran 66 and Fortran 77:

```
$ ./a.out
$ more XX.DAT
123
$
```

In Fortran 95 or later:

```
$ ./a.out
jwe0113i-e line 2 A(an) formatted I/O statement cannot be executed for a unit connected to
unformatted (unit=10).
error occurs at MAIN__ line 2 loc 0000000000400b42 offset 0000000000000032
MAIN__ at loc 0000000000400b10 called from o.s.
taken to (standard) corrective action, execution continuing.
error summary (Fortran)
error number error level error count
```

```

jwe0113i      e      1
total error count = 1
$ more XX.DAT
$

```

7.4.4 Default Value of ACCESS= Specifier with RECL= Specifier

In Fortran 66 and Fortran 77, if RECL= is specified and ACCESS= is not specified in an OPEN statement, ACCESS='DIRECT' is used by default. In Fortran 95 or later, if RECL= is specified and ACCESS= is not, ACCESS='SEQUENTIAL' is used by default.

Example: OPEN statement with RECL= and with no ACCESS= specifier:

```
OPEN (1, RECL=80)
```

7.5 CLOSE Statement STATUS= Specifier

This section explains the STATUS= specifier used in the CLOSE statement. See Section "[7.7.1 Control Information for Input/Output Data Transfer Statements](#)" for information on the other specifiers.

7.5.1 STATUS= Specifier

The STATUS= specifier indicates whether to keep or delete the file connected to the unit number when it is closed. This specifier and the OPEN statement STATUS= specifier determine whether the file is retained or deleted. The table below lists the relationship between STATUS= specifiers of OPEN and CLOSE statements.

Relationship between STATUS= specifiers of OPEN and CLOSE statements

OPEN statement	CLOSE statement		
	KEEP/FSYNC	DELETE	Default specification
NEW , OLD, SHR, REPLACE	Keep the file after CLOSE statement execution.	Delete the file after CLOSE statement execution.	Keep the file after CLOSE statement execution.
SCRATCH	Delete an unnamed file after CLOSE statement execution.		Delete the file after CLOSE statement execution.

Note:

If UNKNOWN is specified in the OPEN statement, the file status is based on whether the file is handled as NEW, OLD, or SCRATCH.

In case the 'FSYNC' is specified to the CLOSE statement, the data which is in memory for the file is synchronized with storage device by invoking the fsync(2) system call only if the following conditions are met:

1. The connected file is a named regular file.
2. The file listed in 1. above exists.
3. The file listed in 1. above is being used.
4. File system supports synchronous operation by invoking the fsync(2) system call.

If the condition listed in 1. above is not met, an error occurs when FSYNC is specified to CLOSE statement.

If the condition listed in 2. or 3. above is not met, the fsync(2) system call is not invoked, and processing is continued.

If the condition listed in 4. is not met, an error occurs.

7.6 INQUIRE Statement

7.6.1 INQUIRE Statement Parameters

The following table indicates the way the system treats returned INQUIRE values, depending on whether the length of the returned character value is less than the scalar character variable name or array element name to be set.

Character Strings:

Returned length >= Character variable length		Returned length < Character variable length	
NAME= specifier	Other specifier	NAME= specifier	Other specifier
The returned value is padded with blanks to fill the character variable.		A message is output indicating that the length of the character variable is too short. The character variable is set to all blanks.	A message is output indicating that the length of the character variable is too short. The returned value truncated to the length of the character variable.

The tables below list the returned values for the INQUIRE statement.

Returned values when a filename is specified

Inquiry specifier	File exists		File does not exist
	Being used (*1)	Not being used (*2)	
EXIST	True	True	False
OPENED	True	False	False
NUMBER	o	-1	-1
NAMED	True	True	True
NAME	o	o	o
FLEN	o	0	0
ACCESS	o	"UNDEFINED"	"UNDEFINED" (*8)
SEQUENTIAL	o	"UNKNOWN"	"UNKNOWN" (*8)
DIRECT	o	"UNKNOWN"	"UNKNOWN" (*8)
ACTION	o	"UNDEFINED"	"UNDEFINED" (*8)
READ	o	"UNKNOWN"	"UNKNOWN" (*8)
WRITE	o	"UNKNOWN"	"UNKNOWN" (*8)
READWRITE	o	"UNKNOWN"	"UNKNOWN" (*8)
FORM	o	"UNDEFINED"	"UNDEFINED" (*8)
FORMATTED	o	"UNKNOWN"	"UNKNOWN" (*8)

Inquiry specifier	File exists		File does not exist
	Being used (*1)	Not being used (*2)	
UNFORMATTED	o (*3)	"UNKNOWN"	"UNKNOWN" (*8)
BINARY	o	"UNKNOWN"	"UNKNOWN" (*8)
RECL	o (*3)	0	0
NEXTREC	o (*4)	0	0
BLANK	o (*5)	"UNDEFINED"	"UNDEFINED" (*8)
PAD	o (*5)	"UNDEFINED" (*7)	"UNDEFINED" (*13)
POSITION	o (*6)	"UNDEFINED"	"UNDEFINED" (*8)
DELIM	o (*5)	"UNDEFINED"	"UNDEFINED" (*8)
BLOCKSIZE	o (*9)	0	0
CONVERT	o (*10)	"UNKNOWN"	"UNKNOWN" (*8)
ASYNCHRONOUS	o (*11)	"UNDEFINED"	"UNDEFINED"
DECIMAL	o	"UNDEFINED"	"UNDEFINED"
ENCODING	o (*5)	"UNKNOWN"	"UNKNOWN"
ROUND	o	"UNDEFINED"	"UNDEFINED"
SIGN	o	"UNDEFINED"	"UNDEFINED"
ID	o	0	0
PENDING	o	FALSE	FALSE
POS	o (*12)	-1	-1
SIZE	o	o	-1
STREAM	o	"UNKNOWN"	"UNKNOWN"

True or False: Four-byte logical value

o : A defined value is assigned to the variable

*1. This column indicates the processing when input/output statements are being executed for the relevant unit number.

*2. This column indicates the processing when input/output statements are not being executed for the relevant unit number.

*3. The value of the RECL= specifier in the OPEN statement is assigned to the variable. If the RECL= specifier in the OPEN statement is omitted, the value 2 is assigned to the variable. However, the value 0 is assigned to the variable in Fortran 66 or Fortran 77

*4. Set only for direct access; otherwise, zero.

*5. Set only for formatted input/output; otherwise, it is assigned the value UNDEFINED.

*6. Set only for sequential access; otherwise, it is assigned the value UNDEFINED.

*7. Set the value YES in Fortran 95.

*8. Set blank characters in Fortran 66 and Fortran 77.

*9. Set only for sequential access or stream access; otherwise, zero.

*10. Set only for unformatted input/output; otherwise, it is assigned the value NATIVE.

*11. For a file or unit for which access other than unformatted sequential is specified, or when a special file has been connected, it is assigned the value NO.

*12. Set only for stream access; otherwise, it is assigned the value -1.

*13. Set the value YES in Fortran 95. Set blank characters in Fortran 66 and Fortran 77.

Note:

1. In Fortran 66 and Fortran 77, the return value of the character type is returned by the small letter.

2. The undefined value is the blank character or the value 0. The blank character is assigned to the character variable. 0 is assigned to the integer variable.

3. If the Fortran system cannot define a meaningful value, -1, 0, blanks, "UNKNOWN" or "UNDEFINED" is assigned to the variable.

Returned values when unit is specified.

Inquiry specifier	Unit number within the range (*1)			Unit specifier number outside the range (*2)
	File connected (*3)		File not connected (*4)	
	Being used (*5)	Not being used (*6)		
EXIST	True	True	True	False
OPENED	True	True	False	False
NUMBER	o	-1	-1	-1
NAMED	True (*7)	False	False	False
NAME	o (*8)	" "	" "	" "
FLEN	o	0	0	0
ACCESS	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
SEQUENTIAL	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
DIRECT	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
ACTION	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
READ	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
WRITE	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
READWRITE	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
FORM	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
FORMATTED	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)

Inquiry specifier	Unit number within the range (*1)			Unit specifier number outside the range (*2)
	File connected (*3)		File not connected (*4)	
	Being used (*5)	Not being used (*6)		
UNFORMATTED	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
BINARY	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
RECL	o (*9)	0	0	0 (*14)
NEXTREC	o (*10)	0	0	0 (*14)
BLANK	o (*11)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
PAD	o (*11)	"UNDEFINED" (*18)	"UNDEFINED" (*18)	"UNDEFINED" (*19)
POSITION	o (*12)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
DELIM	o (*11)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
BLOCKSIZE	o (*13)	0	0	0
CONVERT	o (*15)	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
ASYNCHRONOUS	o (*16)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
DECIMAL	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
ENCODING	o (*11)	"UNKNOWN"	"UNKNOWN"	"UNKNOWN"
ROUND	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
SIGN	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
ID	o	0	0	0
PENDING	o	FALSE	FALSE	FALSE
POS	o (*17)	-1	-1	-1
SIZE	o	o	-1	-1
STREAM	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN"

True or False: Four-byte logical value

o : A defined value is assigned to the variable

" ": Indicates blanks.

*1. The unit number range is 0 to 2147483647.

*2. Outside the unit number means less than 0 or greater than 2147483647.

*3. This column indicates the processing when a unit number and file are connected.

*4. This column indicates the processing when a unit number and file are not connected.

- *5. This column indicates the processing when input/output statements are being executed for the relevant unit number.
- *6. This column indicates the processing when input/output statements are not being executed for the relevant unit number.
- *7. For an unnamed file, false is set.
- *8. For an unnamed file, a blank is set.
- *9. The value of the RECL= specifier in the OPEN statement is assigned to the variable. If the RECL= specifier in the OPEN statement is omitted, the value 2 is assigned to the variable. However, the value 0 is assigned to the variable in Fortran 66 or Fortran 77.
- *10. Set only for direct access; otherwise, zero.
- *11. Set only for formatted input/output; otherwise, it is assigned the value UNDEFINED.
- *12. Set only for sequential access; otherwise, it is assigned the value UNDEFINED.
- *13. Set only for sequential access or stream access; otherwise, zero.
- *14. Set blank characters in Fortran 66 and Fortran 77.
- *15. Set only for unformatted input/output; otherwise, it is assigned the value NATIVE.
- *16. For a file or unit for which access other than unformatted sequential is specified, or when a special file has been connected, it is assigned the value NO.
- *17. Set only for stream access; otherwise, it is assigned the value -1.
- *18. Set the value YES in Fortran 95.
- *19. Set the value YES in Fortran 95. Set blank characters in Fortran 66 and Fortran 77.

Note:

1. In Fortran 66 and Fortran 77, the return value of the character type is returned by the small letter.
2. The undefined value is the blank character or the value 0. The blank character is assigned to the character variable. 0 is assigned to the integer variable.
3. If the Fortran system cannot define a meaningful value, -1, 0, blanks, "UNKNOWN" or "UNDEFINED" is assigned to the variable.

7.6.2 Difference in Language Version of INQUIRE Statement

See Section ["7.6.1 INQUIRE Statement Parameters"](#) for details of the value returned by the INQUIRE statement.

7.6.2.1 Return Value of ACTION= Specifier

In Fortran 66 and Fortran 77, the variable in the ACTION= specifier is assigned the value BOTH if the file is connected for both input and output.

In Fortran 95 or later, it is assigned the value READWRITE.

7.6.2.2 Char-Constant Returned by Inquiry Specifiers of INQUIRE Statement

In Fortran 66 and Fortran 77, char-constant returned by an inquiry specifier except NAME= specifier is returned by the small letter. However, when runtime option -q is specified, it is returned by the capital letter.

In Fortran 95 or later, it is returned by the capital letter.

Example:

```
CHARACTER(LEN=10) CH
OPEN(10,ACCESS='SEQUENTIAL')
INQUIRE(10,ACCESS=CH)
```

In Fortran 66 and Fortran 77, CH is assigned the value sequential.

In Fortran 95 or later, CH is assigned the value SEQUENTIAL.

7.6.2.3 Return Value of PAD= Specifier

In Fortran 66 and Fortran 77, the variable of PAD= specifier is assigned the value undefined or the blank character if the file is not being used.

In Fortran 95, it is assigned the value YES.

In Fortran 2003 or later, it is assigned the value UNDEFINED.

7.6.2.4 Return Value of Inquire by File of INQUIRE Statement

In Fortran 66 and Fortran 77, the variable of the following specifiers is assigned the value undefined, unknown, or the blank character if the file does not exist.

In Fortran 95 or later, it is assigned the value UNDEFINED or UNKNOWN.

Specifiers are ACCESS=, SEQUENTIAL=, DIRECT=, ACTION=, READ=, WRITE=, READWRITE=, FORM=, FORMATTED=, UNFORMATTED=, BINARY=, BLANK=, POSITION=, DELIM=, ASYNCHRONOUS=, DECIMAL=, ROUND=, SIGN= and STREAM=.

7.6.2.5 Return Value of Inquire by Unit of INQUIRE Statement

In Fortran 66 and Fortran 77, the variable of the following specifiers is assigned the value undefined, unknown, or the blank character if UNIT= specifier is out of range from 0 to 2147483647.

In Fortran 95 or later, it is assigned UNDEFINED or UNKNOWN.

Specifiers are ACCESS=, SEQUENTIAL=, DIRECT=, ACTION=, READ=, WRITE=, READWRITE=, FORM=, FORMATTED=, UNFORMATTED=, BINARY=, BLANK=, POSITION=, DELIM=, ASYNCHRONOUS=, DECIMAL=, ROUND=, SIGN= and STREAM=.

7.7 Data Transfer Input/Output Statements

Not every input/output statement may be used with all input/output devices. For example, a terminal device must not be accessed using unformatted I/O.

Input/output statement type	Terminal	Direct access storage device
Formatted sequential (including namelist and list-directed)	Allowed	
Unformatted sequential	Not allowed	Allowed
Direct		
Stream		

7.7.1 Control Information for Input/Output Data Transfer Statements

This section describes the control information (configuration elements) used in data transfer input/output statements.

The control specifiers are the UNIT=, FMT=, IOSTAT=, ERR=, END=, EOR=, and IOMSG= specifiers.

7.7.1.1 Input/Output UNIT Specifier

When the user specifies an asterisk as the unit number or omits the UNIT= specifier, the system uses a unit based on a runtime option. If no runtime option is specified, the system uses the default specification. The table below lists the standard unit numbers used in the conditions just described.

Standard unit numbers:

Input/output statement	Runtime option	Standard value
Diagnostic message output by the Fortran system	Value of <i>u_nos</i> specified by runtime option <i>-mu_no</i>	0
READ statement with * specified. READ statement with the unit specifier omitted.	Value of <i>u_nos</i> specified by runtime option <i>-ru_no</i>	5
WRITE statement with * specified. PRINT statement.	Value of <i>u_nos</i> specified by runtime option <i>-pu_no</i>	6

Note:

Values specified using runtime options have priority over the default values.

7.7.1.2 Format Specifier

Data is edited as real, complex, integer, logical, character, binary, octal, or hexadecimal according to an edit descriptor. For example, specifying the I edit descriptor for type real output edits data to produce integer data. The table below lists the edit descriptors and types of input/output.

Table 7.1 Edit descriptors and types of input/output

Types of input	Edit descriptor output fields							
	L	I	F,E,D,Q, EN,ES	A	G	B	Z	O
One-byte logical								
Two-byte logical								
Four-byte logical	o	+	+	o	o	o	o	o
Eight-byte logical								
One-byte integer								
Two-byte integer								
Four-byte integer	+	o	+	o	o	o	o	o
Eight-byte integer								
Default real								
Double-precision real	+	+	o	o	o	o	o	o
Quadruple-precision real								
Default complex								
Double-precision complex	+	+	o	o	o	o	o	o
Quadruple-precision complex								
Character	x	x	x	o	o	o	o	o

o: Combination is allowed.

+: Conflicting combination; a warning message is output and editing is performed according to the edit descriptor.

x: Combination is not allowed. A message is output, and processing terminates.

7.7.1.3 IOSTAT= Specifier

When a data transfer input/output statement is executed, the IOSTAT= specifier is set indicating one of the following:

- An error condition occurs.

- An end-of-file condition occurs.
- An end-of-record condition occurs.
- Neither an error condition, an end-of-file condition nor an end-of-record condition occurred.

The tables below list the value assigned to the IOSTAT= specifier after an error condition, an end-of-file condition, or an end-of-record condition occurs. The IOSTAT= specifier is set to zero if neither an error condition, an end-of-file condition, nor an end-of-record condition occurs. If an end-of-file condition or an end-of-record condition occurs for an input statement containing an IOSTAT= specifier, a diagnostic message is not output. The standard system and user-defined corrections for errors do not occur. See Section "8.1 Error Processing" for details.

Error, end-of-file and end-of-record condition

Input/output error conditions	Error cases	Corresponding input/output statement	Value of IOSTAT= specifier
Error condition (Unrecoverable error)	An unrecoverable error occurred during file input/output.	OPEN statement CLOSE statement Data transfer statements File positioning statements FLUSH statement	1
Error condition (except unrecoverable error)	An I/O error was detected.	OPEN statement CLOSE statement Data transfer statements File positioning statements FLUSH statement	Error number
End-of-file condition	An end-of-file record was detected.	Sequential READ List-directed READ Namelist READ Stream READ	-1
End-of-file condition	A Fortran record exceeded the end of the internal file.	Internal file READ	-1
End-of-record condition	The length of input list exceeds the length of a Fortran record.	Nonadvancing formatted sequential input	-2

7.7.1.4 Error Branch

If an error condition occurs during the execution of a data transfer input/output statement, execution continues with the statement specified in the ERR= specifier. If the input/output statement also contains an IOSTAT= specifier, the value of its error number is assigned to the variable. See Section "7.7.1.3 IOSTAT= Specifier" for details.

If an error condition occurs for an input/output statement that specifies the ERR= specifier, a diagnostic message is not output, and the standard system and user-defined error correction do not occur.

7.7.1.5 End-of-File Branch

If an end-of-file condition occurs and no error condition occurs during the execution of a data transfer input/output statement, execution continues with the statement specified in the END= specifier. If the input/output statement also contains an IOSTAT= specifier, the value -1 is assigned to the variable.

If an end-of-file condition occurs for an input statement that specifies the END= specifier, a diagnostic message is not output, and the standard system and user-defined error correction do not occur.

7.7.1.6 End-of-Record Branch

If an end-of-record condition occurs and no error condition occurs during the execution of a data transfer input/output statement, execution continues with the statement specified in the EOR= specifier. If the input/output statement also contains an IOSTAT= specifier, the value -2 is assigned to the variable.

If an end-of-record condition occurs for an input statement that specifies the EOR= specifier, a diagnostic message is not output, and the standard system and user-defined error correction do not occur.

7.7.1.7 IOMSG= Specifier

The IOMSG= specifier sets an error message when an error condition, end-of-file condition, or end-of-record condition occurs.

Note, however, that if an IOSTATE= specifier or an ERR= specifier is not specified simultaneously with the IOMSG= specifier, a message is not set. Further, if an error condition, end-of-file condition, or end-of-record condition does not occur, the value specified in the IOMSG= specifier is not changed.

7.7.2 Sequential Access Input/Output Statements

Sequential input/output statements read and write formatted and unformatted Fortran records from sequential files. Formatted sequential input/output statements are used to read and write formatted Fortran records. Unformatted sequential input/output statements are used to read and write unformatted Fortran records.

7.7.2.1 Formatted Sequential Access Input/Output Statements

Formatted sequential input/output statements sequentially read and write characters based on a format specification. Generally, these statements are used when the data must be displayed.

Formatted sequential READ statements read data from the current Fortran record based on a format. When an input list is specified, the Fortran record is edited into internal representations based on edit descriptors in the format.

Formatted sequential WRITE and PRINT statements write Fortran records to the current record based on a format. When an output list is specified, each item is edited based on the edit descriptors in the format. If the \$ edit descriptor is specified, a line-feed character is not written after the edited data.

Example: A formatted sequential input/output statement used to read and write data

```
REAL A,B
INTEGER I,J
A=1.5           !A is assigned the hexadecimal value 3fc00000
I=100          !I is assigned the hexadecimal value 00000064
WRITE(1,10) I,A !A Fortran record becomes the hexadecimal
                ! value 2031303020312e35
REWIND 1       !Position the file at the beginning.
READ(1,10) J,B !J is assigned the hexadecimal value 00000064.
                ! B is assigned the hexadecimal value 3fc00000.
10 FORMAT(1X,I3,1X,F3.1)
STOP
END
```

7.7.2.2 Unformatted Sequential Access Input/Output Statements

Unformatted sequential input/output statements read and write Fortran records consisting of internal representations. Generally, these statements are used when the data need not be displayed.

If unformatted sequential input/output statements are executed for devices (such as terminals) that process character codes, the results are not predictable.

Files written with variable-length Fortran records by Fujitsu Fortran implementation can be accessed by an unformatted input statement.

Unformatted sequential READ statements read the current record. When an input list is specified, the data is stored unchanged into each item in the input list in the order specified. An error occurs if the number of bytes in the record is less than in the input list items. If the number of bytes in the record is greater, the excess data is ignored. If an input list is not specified, the record is skipped.

Unformatted sequential WRITE statements write the output list items as a Fortran record in the order they are specified. The statement writes the record in the current file position. If an output list is not specified, a Fortran record of length zero is written.

The length of the Fortran record is the sum of the bytes in the output list items.

Example: An unformatted sequential input/output statement used to read and write data

```
REAL A,B
INTEGER I,J
A=1.5           !A is assigned the hexadecimal value 3fc00000
I=100          !I is assigned the hexadecimal value 00000064
WRITE(1) A,I   !A Fortran record becomes the hexadecimal value
               ! 3fc0000000000064
REWIND 1       !Position the file at the beginning.
READ(1) B,J    !B is assigned the hexadecimal value 3fc00000
               ! J is assigned the hexadecimal value 00000064

STOP
END
```

7.7.3 Direct Access Input/Output Statements

Direct input/output statements are used to read and write formatted or unformatted Fortran records to or from the record of the file indicated by the REC= specifier.

An OPEN statement must be executed before any direct input/output statements may be executed.

When TOTALREC is used in an OPEN statement, reading and writing can be executed only for the number of records specified. Sequential input/output statements must not be used to add records to direct files.

7.7.3.1 Direct Access READ Statement

A direct access READ statement reads Fortran records which are uniquely identified by the record number.

During formatted data transfer, data are transferred with editing between the file and the entities specified by the input list.

During unformatted data transfer, data are transferred without editing between current record and the entities specified by the input list. The number of values required by the input list shall be less than or equal to the number of values in the record.

Data are not transferred if the input list does not exist.

7.7.3.2 Direct Access WRITE Statement

A direct access WRITE statement writes Fortran records which are uniquely identified by the record number.

During formatted data transfer, data are transferred with editing between the file and the entities specified by the output list. Blank values are transferred if the output list does not exist.

During unformatted data transfer, data are transferred without editing between current record and the entities specified by the output list. The output list shall not specify more values than can fit into the record. If the values specified by the output list do not fill the record, the remainder of the record is undefined. Zero values are transferred if the output list does not exist.

7.7.3.3 Error of Data Transfer in a DIRECT Access Input/Output

The execution of Fortran program is different by the compiler option -X if the output list specifies more values than can fit into the record and the number of values required by the input list are more than the number of values in the record.

In Fortran 66 and Fortran 77, the program ends normally.

In Fortran 95 or later, a diagnostic message of w level is generated and the multiple records are written or read.

Example:

Program di.f

```
OPEN(10, FILE='XX.DATA', FORM='UNFORMATTED', ACCESS='DIRECT', RECL=4)
WRITE(10, REC=1) 1.0_8
INQUIRE(10, NEXTREC=i)
```

```
PRINT*, 'NEXTREC=', i
END
```

Result

In Fortran 66 and Fortran 77:

```
$ ./a.out
NEXTREC=3
$
```

In Fortran 95 or later:

```
$ ./a.out
jwell162i-w line 2 On output to a file connected for unformatted direct access, the output list must
not specify more values than can fit into the Fortran record (unit=10).
error occurs at MAIN__ line 2 loc 0000000000400be6 offset 0000000000000036
MAIN__ at loc 0000000000400bb0 called from o.s.
taken to (standard) corrective action, execution continuing.
NEXTREC= 3
error summary (Fortran)
error number error level error count
jwell162i w 1
total error count = 1
$
```

7.7.4 Namelist Input/Output Statements

Namelist input/output statements transfer data between memory and sequential formatted records in an internal or external file. The namelist names and their associated variable and array names used for data transfer must have been declared previously in a NAMELIST statement.

7.7.4.1 Namelist READ Statements

It is necessary for the type of the item in the namelist to match the type of the data in the record. The correspondence for type of element and type of constant is indicated in the following table.

Type of namelist group object	Type of constant					
	Logical	Integer	Real	Complex	Character	Hexadecimal
One-byte logical	o	+	+	x	+	+
Two-byte logical		(*1)	(*1)		(*2)	(*7)
Four-byte logical						(*8)
Eight-byte logical						
One-byte integer	+	o	+	x	+	o
Two-byte integer	(*3)		(*3)		(*2)	(*8)
Four-byte integer						
Eight-byte integer						
Default real	+	o	o	x	+	+
Double-precision real	(*4)	(*5)			(*2)	(*7)
Quadruple-precision real						(*8)
Default complex	+	+	+	o	+	+
Double-precision complex	(*4)	(*5)	(*6)		(*2)	(*7)
	(*6)	(*6)	(*9)			(*8)
	(*9)	(*9)				

Type of namelist group object	Type of constant					
	Logical	Integer	Real	Complex	Character	Hexadecimal
Quadruple-precision complex						
Character	x	x	x	x	o	x (*7) (*8)

o: Operation is normal.

+: Combination is not allowed, but operation continues; interprets to revise for element type.

x: Input is terminated because the combination is not allowed.

*1. Logical editing is performed.

*2. Character editing is performed.

*3. Integer editing is performed.

*4. Real editing is performed.

*5. Real editing is performed. The decimal point is assumed to be to the right of the constant.

*6. The edit result is entered in the real part. Zero is entered in the imaginary part.

*7. In Fortran 66 and Fortran 77, the operation for hexadecimal types is normal.

*8. In Fortran 95 or later, hexadecimal is assumed to be character. Therefore, if the type of element is character, the operation is normal. If the type of element is not character, the combination is not allowed but operation continues. See Section "2.2 Compiler Options" for compiler options.

*9. In Fortran 95 or later, a diagnostic message of w level is generated.

Note:

For input processing, editing is based on the type of namelist element. For hexadecimal or character types, editing is based on the type of constant in the file.

a. Processing of Character Constants for Array Name.

The processing of character constants for array names depends on whether the type of the array is character or not. If the type of the array is character, one character constant corresponds to one array element. Even if the character constant exceeds the array element length, there is no overflow to the next array element.

Example: Character constant processing for elements of type character Fortran program:

```
CHARACTER(4) HA(5)
NAMELIST/NAMEH/HA
READ(1,NML=NAMEH) !READ statement for namelist NAMEH
```

If the record to be read from the file connected to unit 1 is

```
&NAMEH HA=2*'NAMELIST READ STATEMENT' /
```

then the elements of HA will have the following values:

HA(1)	'NAME '
HA(2)	'NAME '
HA(3)	Unchanged
HA(4)	Unchanged
HA(5)	Unchanged

If the type of the array is not character, the character constant corresponding to the first array element is entered for the entire array. That is, if the length of the character constant exceeds one element, the character constant is stored in succeeding elements. If the length of the character constant is shorter than the entire array, the remaining elements are unchanged. If input ends within an element, the remaining part of the element is padded with blanks.

For the second and subsequent array elements, each character constant corresponds to a single element.

Even if the length of the character constant exceeds the array element length, no overflow occurs.

Example: The processing of character constants for elements of other than type character

Fortran program:

```
INTEGER IA(5)
NAMELIST/NAMEI/IA
READ(1,NML=NAMEI) !READ statement for namelist NAMEI
```

The namelist record for unit number 1 is set as follows:

```
&NAMEI IA=2* 'NAMELIST READ STATEMENT' /
```

The following is the result in character notation:

IA(1)	'NAME '
IA(2)	'NAME '
IA(3)	' REA '
IA(4)	'D ST '
IA(5)	'ATEM '

b. Input Editing Operation

The format of the namelist READ statement is determined based on the element type, constant type and constant size, as follows:

- If the type of constant and type of element are other than character, editing is performed according to the element type. The edit descriptor is Gw. The letter w indicates the size (between delimiters) of the constant to be entered.
- If the constant type is character, character editing is performed. The edit descriptor is Aw. The letter w indicates the number of characters.
- For noncharacter constants with elements of type character, character editing is performed. The edit descriptor is Aw.

7.7.4.1.1 Namelist Data in Namelist Input

The namelist data in the namelist input has one of following formats:

```
&name [v=c[,c]...[,v=c[,c]...]...] /
```

```
&name [v=c[,c]...[,v=c[,c]...]...] &end
```

name: The namelist-group name specified in the corresponding NAMELIST input/output statement.

v: The variable name or corresponding array element, array section, or character substring specified in the NAMELIST statement corresponding to the name.

c: NULL or one of the following forms:

```
const
```

```
r*const
```

r^*

const: A literal constant other than a Hollerith, binary, octal, or hexadecimal constant.

r: A repeat specification. The argument is an unsigned integer constant.

*r*const*: It is equivalent to the sequential appearance of the *const* *r* times.

*r**: It is equivalent to the sequential appearance of NULL *r* times.

If the input item is a character type and all following conditions are satisfied, the character constant need not be enclosed with apostrophes or quotation marks.

- A blank, comma, slash, or ampersand, which is considered to be a value separator, is not included in the character constant.
- A semicolon, which is considered to be a value separator, is not included in the character constant, when the decimal point symbol is a comma.
- Left parentheses, right parentheses, and percent sign are not included in the character constant.
- The character constant does not extend over more than one Fortran record.
- The first non-blank character is a character other than an apostrophe or quotation mark.
- The form of the input item is not *r*const*.

When a character constant does not end with a delimiter character (apostrophe or quotation mark), it is assumed to end with the first blank, comma, or slash. Moreover, an apostrophe or quotation mark in the character string is not treated as two consecutive apostrophes or two consecutive quotation marks.

7.7.4.2 Namelist WRITE Statements

Namelist WRITE statements are used for output editing. These statements write a record based on the types of variables and arrays specified in the namelist. The output format for variable names consists of a variable name, equal sign, and edit result, in that order. A comma separates constants. The output format for array names consists of an array name, equal sign, and edit result for each array element, in that order. A comma separates the edit results.

The output format for a namelist group object is as follows:

Type of namelist group object	Output format (Edit descriptor)
One-byte logical	G1,1H, (*1)
Two-byte logical	G1,1H, (*1)
Four-byte logical	G1,1H, (*1)
Eight-byte logical	G1,1H, (*1)
One-byte integer	G4,1H, (*1)
Two-byte integer	G6,1H, (*1)
Four-byte integer	G11,1H, (*1)
Eight-byte integer	G20,1H, (*1)
Default real	1P,E15.8,1H, (*1)
	0P,G16.9,1H, (*1)
Double-precision real	1P,E22.15,1H, (*1)
	0P,G23.16,1H, (*1)
Quadruple-precision real	1P,E43.34E4,1H, (*1)
	0P,G44.35E4,1H, (*1)
Default complex	1H,(1P,E15.8,1H,,E15.8,2H), (*2)
	1H,(0P,G16.9,1H,,G16.9,2H), (*2)

Type of namelist group object	Output format (Edit descriptor)
Double-precision complex	1H(,1P,E22.15,1H,,E22.15,2H), (*2)
	1H(,0P,G23.16,1H,,G23.16,2H), (*2)
Quadruple-precision complex	1H(,1P,E43.34E4,1H,,E43.34E4,2H), (*2)
	1H(,0P,G44.35E4,1H,,G44.35E4,2H), (*2)
<i>n</i> -byte character	1H',character-contents,2H', (*3)
	1H",character-contents,2H", (*4)
	character-contents,1H, (*5)

- *1. If an &end or a / comes immediately after the value generated by the G or E edit descriptor, 1H, is omitted.
- *2. If an &end or a / comes immediately after the value generated by the G or E edit descriptor, 1H) is generated instead of 2H),.
- *3. If an &end or a / comes immediately after the value delimited by quotes, 1H' is generated instead of 2H',.
- *4. If an &end or a / comes immediately after the value delimited by apostrophes, 1H" is generated instead of 2H",.
- *5. If an &end or a / comes immediately after the nondelimited character constants, 1H, is omitted.

Notes:

1. For types other than character, meaningless blanks in the edit result are deleted before output. The actual output length can therefore be smaller than the field width of the edit descriptor.
2. Whether 1P,Ew1.d1 or 0P,Gw2.d2 is used for type real or complex depends on the absolute value of the data. 0P,Gw2.d2 is used if the absolute value of x is 0 or $0.1 \leq x < 10^{**d2}$. Otherwise, 1P,Ew1.d1 is used.

7.7.4.3 NAMELIST Input/Output Version Differences

a. Character type output

In Fortran 66 and Fortran 77, character constants are written using apostrophes as delimiters. In Fortran 95 or later, DELIM= can be specified as the delimiter of character constants in an OPEN statement. If DELIM= specifier is omitted in an OPEN statement or an OPEN statement is not executed, the DELIM= specifier is treated as NONE. In this case, character constants are written without delimiters.

Example:

```
CHARACTER ( LEN=5 ) : : CH= ' XXXXX '
NAMELIST /X/CH
OPEN ( 10 )
WRITE ( 10 , NML=X )
```

In Fortran 66 and Fortran 77, the output is as follows:

```
&X
CH= ' XXXXX '
&end
```

In Fortran 95 or later, the output is as follows:

```
&X CH=XXXXX/
```

b. Format for terminating the namelist output statement

In Fortran 66 and Fortran 77, &end is written to terminate the namelist output statement. In Fortran 95 or later, / is written.

Example:

```
INTEGER : : IH=123
NAMELIST /X/ IH
WRITE ( 10 , NML=X )
```

In Fortran 66 and Fortran 77, the output is as follows:

```
&X  
IH=123  
&end
```

In Fortran 95 and Fortran 2003, the output is as follows:

```
&X IH=123/
```

c. Blanks in namelist input/output

Fortran 66 and Fortran 77 treat blanks as part of the input data. In Fortran 95 or later treat a blank as a delimiter. However, blanks between delimiters are treated as input data.

Example:

```
INTEGER, DIMENSION(3) :: IH  
NAMELIST /X/ IH  
READ(*, NML=X)
```

Assume that the input data is IH=1_2_3, (_ indicates a blank).

Fortran 66 and Fortran 77 treat this data as follows:

```
IH(1)=10203  
IH(2)=0  
IH(3)=0
```

Fortran 95 or later treat this data as follows:

```
IH(1)=1  
IH(2)=2  
IH(3)=3
```

d. Hexadecimal constants specified as namelist input data

Fortran 66 and Fortran 77 treat hexadecimal constants specified as namelist input data as hexadecimal constants. Fortran 95 or later treat these constants as character constants.

Example:

```
CHARACTER(LEN=5) CH  
NAMELIST /X/ CH  
READ(*, NML=X)
```

Assume that the input data is CH=Z3132333435,.

Fortran 66 and Fortran 77 treat this data as follows:

```
CH='12345'
```

Fortran 95 or later treat this data as follows:

```
CH='Z3132'
```

e. Hollerith constants specified as namelist input data.

Fortran 66 and Fortran 77 treat Hollerith constants specified as namelist input data as Hollerith constants. Fortran 95 or later treat Hollerith constants specified as namelist input data as character constants.

Example:

```
CHARACTER(LEN=5) CH  
NAMELIST /X/ CH  
READ(*, NML=X)
```

Assume that the input data is CH=3H123,.

Fortran 66 and Fortran 77 treat this data as follows:


```
CH= ' 123 '
```

Fortran 95 or later treat this data as follows:

```
CH= ' 3H123 '
```

- f. Output of the first byte of the second and subsequent records when Fortran records contain character constants.
In Fortran 66 and Fortran 77, a blank is written in the first byte of the second record. In Fortran 95 or later, the first byte contains a value.

Example:

```
CHARACTER(LEN=1000) CH  
NAMELIST /X/CH  
WRITE(10,NML=X)
```

In Fortran 66 and Fortran 77, a blank is written as follows:

```
CH= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX <- the first record  
*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' <- the second record  
^  
blank is written
```

In Fortran 95 or later, a value is written as follows:

```
CH= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX <- the first record  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' <- the second record  
^  
value is written
```

- g. Input data with the r^* form or $r^*\text{const}$ form.
In Fortran 66 and Fortran 77, when a r^* form appears in input data, it does not mean 'repeat the null value r times'. For those versions, it means the repeat is applied to the following input data. If the r^* constitutes the last characters of a Fortran record, the repeat is applied to the first input data of the next Fortran record. In Fortran 95 or later, r^* means 'repeat the null value r times'.

Example:

```
INTEGER :: I(4) = (/1, 2, 3, 4/)  
NAMELIST /X/I  
READ(*,NML=X)
```

Assume that the input data is $I=2^*_30,40$, ($_$ indicates a blank).

Fortran 66 and Fortran 77 treat this data as follows:

```
I(1)=30  
I(2)=30  
I(3)=40  
I(4)=4
```

Fortran 95 or later treat this data as follows:

```
I(1)=1  
I(2)=2  
I(3)=30  
I(4)=40
```

- h. The output of a group name or an object name of namelist output statement.
 In Fortran 66 and Fortran 77, a group name or an object name in the output is case sensitive.
 In Fortran 95 or later, it is in uppercase.

Example:

```
INTEGER :: Iu=1
NAMELIST /Xu/Iu
WRITE(10,NML=Xu)
```

In Fortran 66 and Fortran 77, the output is as follows:

```
&Xu
Iu=1
&end
```

In Fortran 95 or later, the output is as follows:

```
&XU IU=1/
```

- i. The output of a namelist Fortran record.
 In Fortran 66 and Fortran 77, the output is as follows.
 The first record : `_&name` (`_` indicates a blank, `name` indicates namelist group object name)
 The second record : `_v=c[,c]...[,v=c[,c]...]...`
 (`v` indicates namelist group object list item, `c` indicates namelist output value)
 The last record : `_&end`
 In Fortran 95 or later, the output is as follows.
 The first record : `_&name v=c[,c]...[,v=c[,c]...].../`

- j. Output of real number zero.

The result of executing a real number type of variable group output with a value of 0 varies according to the language specification level.

In the Fortran 66, Fortran 77, and Fortran 95 language specifications, output of real number zero is equivalent to the result output when E edit is used.

In the Fortran 2003 or later language specifications, output of real number zero is equivalent to the result output when F edit is used.

Example:

```
REAL :: R = 0
NAMELIST /DATA/R
WRITE(*, NML=DATA)
```

When the Fortran program above is executed, in the Fortran 95 language specifications, `&DATA R=0.0000000E+00/` is output.

When the Fortran program above is executed, in the Fortran 2003 or later language specifications, `&DATA R=0.0000000/` is output.

7.7.5 List-Directed Input/Output Statements

List-directed input/output statements transfer data between memory and formatted records based on the type of the data. Internal files may be used for list-directed formatting.

7.7.5.1 List-Directed READ Statements

List-directed READ statements transfer records from external and internal files to elements in input lists.

The types of elements in input lists must correspond to the types of elements in data items as indicated by the following table.

Type of element	Type of constant				
	Logical	Integer	Real	Complex	Character
One-byte logical	o	+ (*1)	+ (*1)	x	+ (*2)
Two-byte logical					

Type of element	Type of constant				
	Logical	Integer	Real	Complex	Character
Four-byte logical Eight-byte logical					
One-byte integer Two-byte integer Four-byte integer Eight-byte integer	+ (*3)	o	+ (*3)	x	+ (*2)
Default real Double-precision real Quadruple-precision real	+ (*4)	o (*5)	o	x	+ (*2)
Default complex Double-precision complex Quadruple-precision complex	+ (*4) (*6) (*7)	+ (*5) (*6) (*7)	+ (*6) (*7)	o	+ (*2)
Character	+ (*2)	+ (*2)	+ (*2)	+ (*2)	o

o: Operation is normal.

+: Combination is not allowed, but operation continues; interprets to revise for element type.

x: Input is terminated because the combination is not allowed.

*1. Logical editing is performed.

*2. Character editing is performed.

*3. Integer editing is performed.

*4. Real editing is performed.

*5. Real editing is performed. The decimal point is assumed to be to the right of the constant.

*6. The edit result is set in the real part. Zero is set in the imaginary part.

*7. In Fortran 95 or later, a diagnostic message of w level is generated.

a. Processing of Character Constants for Array Names

The processing of character constants for array names depends on whether the element is type character.

If the element is type character, input is performed so that one character constant corresponds to one array element. Even if the length of the character constant exceeds the array element, there is no overflow to the next array element.

If the element is not type character, the character constant corresponding to the first array element is entered for the entire array. That is, if the length of the character constant exceeds one element, the succeeding elements are also read. If the length of the character constant does not exceed the length of the entire array, the remaining elements are unchanged. If input ends within an element, the remaining part of the element is padded with blanks.

For second and subsequent array elements, the character constant corresponds to each element. Even if the length of the character constant exceeds that of the array element, the character constant does not overflow to the next element.

Example: How to enter data of type character for array names

Fortran program:

```
CHARACTER(LEN=4),DIMENSION(5) :: CHAR5
INTEGER,DIMENSION(5) :: I4
...
READ(5,*) CHAR5          ! List-directed READ statement for
                        ! character type array elements
```

```

READ(5,*) I4           ! List-directed READ statement for
                      ! other than character type array elements

```

The following is the format of the list-directed record for unit number 5:

```

3* 'ABCDEFGHIJKLMNO P' /
3* 'ABCDEFGHIJKLMNO P' /

```

The following is the result for CHAR5 and I4:

CHAR5(1)	' A B C D '	I4(1)	' A B C D '
CHAR5(2)	' A B C D '	I4(2)	' A B C D '
CHAR5(3)	' A B C D '	I4(3)	' A B C D '
CHAR5(4)	Unchanged	I4(4)	' M N O P '
CHAR5(5)	Unchanged	I4(5)	' Q '

b. Input Editing Operation

The format used by a list-directed READ statement is determined according to the element type, constant type, and constant size, as follows:

- If the constant and element are other than type character, editing is performed according to the element type. The edit descriptor is Gw . The letter w indicates the size (between delimiters) of the constant to be entered.
- If the constant is type character, character editing is performed. The edit descriptor is Aw . The letter w indicates the size between apostrophes.
- If the constant is other than type character and the element is type character, character editing is performed. The edit descriptor is Aw .

7.7.5.2 List-Directed WRITE Statements

List-directed WRITE statements generate formatted list-directed records. The following edit descriptors are used for output with a list-directed WRITE statement.

Type of element	Output format (edit descriptor)
One-byte logical	G1,1H_ (*1)
Two-byte logical	G1,1H_ (*1)
Four-byte logical	G1,1H_ (*1)
Eight-byte logical	G1,1H_ (*1)
One-byte integer	G4,1H_ (*1)
Two-byte integer	G6,1H_ (*1)
Four-byte integer	G11,1H_ (*1)
Eight-byte integer	G20,1H_ (*1)
Default real	1P,E15.8,1H_ or 0P,G16.9,1H_ (*1)
Double-precision real	1P,E22.15,1H_ or 0P,G23.16,1H_ (*1)
Quadruple-precision real	1P,E43.34E4,1H_ or 0P,G44.35E4,1H_ (*1)
Default complex	1H(,1P,E15.8,1H,,E15.8,2H)_ or (*2) 1H(,0P,G16.9,1H,,G16.9,2H)_ (*2)
Double-precision complex	1H(,1P,E22.15,1H,,E22.15,2H)_ or (*2)

Type of element	Output format (edit descriptor)
	1H(,0P,G23.16,1H,,G23.16,2H)_ (*2)
Quadruple-precision complex	1H(,1P,E43.34E4,1H,,E43.34E4,2H)_ or (*2) 1H(,0P,G44.35E4,1H,,G44.35E4,2H)_ (*2)
n-byte character	Gn,1H_ or (*3)(*6) 1H',Gn,2H'_ or (*4)(*6) 1H'',Gn,2H''_ (*5)(*6)

_ : Indicates blank

- *1. If the record terminates immediately after the value generated by the G or E edit descriptor, 1H_ is omitted.
- *2. If the record terminates 1 byte after the value generated by the G or E edit descriptor, 1H) is generated instead of 2H)_.
- *3. If the record terminates immediately after the nondelimited character constants generated by the G edit descriptor, 1H_ is omitted. If the compiler option -X9 is not in effect, the output format is Gn.
- *4. If the record terminates 1 byte after the value delimited by quotes generated by the G or E edit descriptor, 1H' is generated instead of 2H'_.
- *5. If the record terminates 1 byte after the value delimited by apostrophes generated by the G or E edit descriptor, 1H'' is generated instead of 2H''_.
- *6. In Fortran 66 and Fortran 77, the last blank is not output.

Notes:

1. For types other than character, meaningless blanks in the edit result are deleted before output. The actual output length can therefore be smaller than the field width of the edit descriptor.
2. Whether 1P,Ew1.d1 or 0P,Gw2.d2 is used for type real or complex depends on the absolute value of the data. 0P,Gw2.d2 is used if the absolute value of x is 0 or $0.1 \leq x < 10^{**d2}$. Otherwise, 1P,Ew1.d1 is used.

Example: List-directed WRITE statement

Fortran program:

```

INTEGER :: YY=2002, MM=09, DD=10
CHARACTER(15) :: CH='YEAR MONTH DATE'
INTEGER, DIMENSION(2,3) :: TABLE
WRITE(6,*) YY,CH(1:4),MM,CH(6:10),DD,CH(12:15)
OPEN(6,DELIM='QUOTE')
WRITE(6,*) YY,CH(1:4),MM,CH(6:10),DD,CH(12:15)
OPEN(6,DELIM='APOSTROPHE')
WRITE(6,*) YY,CH(1:4),MM,CH(6:10),DD,CH(12:15)
TABLE = RESHAPE((/1,2,3,4,5,6/), (/2,3/))
WRITE(6,*) ((TABLE(I,J),I=2,1,-1),J=3,1,-1)
END

```

The output is as follows:

```

$ ./a.out
2002 YEAR 9 MONTH 10 DATE
2002 "YEAR" 9 "MONTH" 10 "DATE"
2002 'YEAR' 9 'MONTH' 10 'DATE'
6 5 4 3 2 1
$

```

7.7.5.2.1 Form of Nondelimited Character Constants Produced by List-Directed Output

In Fortran 66 and Fortran 77, the values are not separated.

In Fortran 95 or later, the values are separated by one blank.

Example:

```
CHARACTER(10)::CH=' '  
WRITE(*,*)'Year',2002  
WRITE(CH,*)'Month',9  
WRITE(*,'(A10)')CH  
END
```

In Fortran 66 and Fortran 77, the output is as follows:

```
$ ./a.out  
Year2002  
Month9  
$
```

In Fortran 95 or later, the output is as follows:

```
$ ./a.out  
Year 2002  
Month 9  
$
```

7.7.5.3 List-Directed Input/Output Version Differences

Output of real number zero

The result of executing a real number type of list output with a value of 0 varies according to the language specification level.

In the Fortran 66, Fortran 77, and Fortran 95 language specifications, output of real number zero is equivalent to the result output when E edit is used.

In the Fortran 2003 or later language specifications, output of real number zero is equivalent to the result output when F edit is used.

Example:

```
REAL :: R = 0  
WRITE(*,*) R
```

When the Fortran program above is executed, in the Fortran 95 language specifications, 0.00000000E+00 is output.

When the Fortran program above is executed, in the Fortran 2003 or later language specifications, 0.00000000 is output.

7.7.6 Internal File Input/Output Statements

Internal file input/output statements treat a character string as a file. See Section "[7.7.2 Sequential Access Input/Output Statements](#)", "[7.7.4 Namelist Input/Output Statements](#)", and "[7.7.5 List-Directed Input/Output Statements](#)" for details.

7.7.7 Nonadvancing Input/Output Statements

Nonadvancing input/output lets you control when the next record will be processed, and a single Fortran record can be read and written as many times as necessary. The statement may position the file at a character position within the current record. You can then use a SIZE=, IOSTAT=, or EOR= specifier to control the Fortran program. The variable specified in the SIZE= specifier becomes defined with the count of the characters transferred by data edit descriptors during execution of the current input statement. If an end of record condition occurs, the control may be processed by an IOSTAT variable or EOR= specifier.

Example: Nonadvancing input statements with a SIZE= and EOR= specifier

```
CHARACTER(LEN=20) NAME  
INTEGER NO  
!  
! 1-20 : NAME  
! 21-28 : NO  
OPEN(10,FORM='FORMATTED')  
DO  
  READ(10,FMT='(A20) ',ADVANCE='NO',EOR=10,SIZE=IS,END=20) NAME
```

```

        IF ( IS < 20 ) CYCLE
        READ(10,FMT=' (I8) ',ADVANCE='NO',SIZE=IS,END=20,IOSTAT=IO) NO
        IF ( IS < 8 ) CYCLE
        GO TO 30
10      CYCLE
30      PRINT *,NAME,' ',NO,' ',IS
        END DO
        GO TO 40
20      PRINT *,'END OF DATA'
40      END

```

7.8 Combining Input/Output Statements

When input/output statements are executed, processing depends on the type of input/output statement executed previously.

The combinations of input/output statements explained here use the same unit number.

Processing according to input/output statement order

Current I/O	Immediately preceding I/O												
	Formatted sequential		Unformatted sequential		Formatted direct		Unformatted direct		Formatted stream		Unformatted stream		
	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	
Formatted sequential READ	o	x (*1)	x	x	x	x	x	x	x	x	x	x	x
Formatted sequential WRITE	o (*2)	o	x	x	x	x	x	x	x	x	x	x	x
Unformatted sequential READ	x	x	o	x (*1)	x	x	x	x	x	x	x	x	x
Unformatted sequential WRITE	x	x	o (*2)	o	x	x	x	x	x	x	x	x	x
Formatted direct access	x	x	x	x	o	o	x (*5)	x (*5)	x	x	x	x	x
Unformatted direct access	x	x	x	x	x (*5)	x (*5)	o	o	x	x	x	x	x
Formatted stream READ	x	x	x	x	x	x	x	x	o	o (*6)	x	x	x
Formatted stream WRITE	x	x	x	x	x	x	x	x	o	o	x	x	x
Unformatted stream READ	x	x	x	x	x	x	x	x	x	x	o	o (*6)	o
Unformatted stream WRITE	x	x	x	x	x	x	x	x	x	x	o	o	o

Current I/O	Immediately preceding I/O											
	Formatted sequential		Unformatted sequential		Formatted direct		Unformatted direct		Formatted stream		Unformatted stream	
	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE
OPEN	o	o	o	o	o	o	o	o	o	o	o	o
CLOSE	o	o	o	o	o	o	o	o	o	o	o	o
FLUSH	-	o	-	o	-	o	-	o	-	o	-	o
WAIT	o	o	o	o	o	o	o	o	o	o	o	o
BACKSPACE	o	o (*3)	o	o (*3)	x	x	x	x	o	o	x	x
REWIND	o	o	o	o	x	x	x	x	o	o	o	o
ENDFILE	o (*4)	o	o (*4)	o	x	x	x	x	o	o	o	o

o: The operation is normal.

x: A diagnostic message is output. Correction processing is performed.

-: No operation is performed.

*1. The READ statement cannot execute, because the Fortran record that follows the one written by the immediately preceding WRITE statement is undefined.

*2. The Fortran record that follows the one written by the current WRITE statement is undefined.

*3. The endfile record is generated before BACKSPACE is executed.

*4. The endfile record is generated immediately after the READ statement reads the Fortran record.

*5. In Fortran 66 and Fortran 77, the operation is normal.

*6. Execution of a stream input statements omitted POS= specifier after the execution of a stream output statements is not allowed.

Note:

Formatted READ and WRITE statements also include list-directed and namelist input/output statements.

Current I/O	Immediately preceding I/O									
	OPEN	CLOSE	FLUSH	WAIT	BACKSPACE	REWIND	END	EOF (*1)	None	
Formatted sequential READ	o	o (*6)	o	o	o	o (*2)	o (*7)	x (*7)	o (*8)	
Formatted sequential WRITE	o	o (*6)	o	o	o	o (*2)	o (*6)	x (*11)	o (*8)	
Unformatted sequential READ	o	o (*6)	o	o	o	o (*2)	o (*7)	x (*7)	o (*8)	
Unformatted sequential WRITE	o	o (*6)	o	o	o	o (*2)	o (*6)	x (*11)	o (*8)	
Formatted direct access	o	x	o	o	x	x	x	x	x (*3)	

Current I/O	Immediately preceding I/O								
	OPEN	CLOSE	FLUSH	WAIT	BACKSPACE	REWIND	END	EOF (*1)	None
Unformatted direct access	o	x	o	o	x	x	x	x	x (*3)
Formatted stream access	o	x	o	o	o	o (*2)	o	o	x (*3)
Unformatted stream access	o	x	o	o	x	o (*2)	o	o	x (*3)
OPEN	o	o	o	o	o	o	o	o	o
CLOSE	o	o	o	o	o	o	o	o	o
FLUSH	-	-	-	-	-	-	-	-	-
WAIT	o	o	o	o	o	o	o	o	o
BACKSPACE	o	o (*6)	o	o	o	-	o (*5)	o (*4)	o (*10)
REWIND	o	o (*9)	o	o	o	-	o	o	-
ENDFILE	o (*5)	o (*6)	o	o	o	o (*5)	- (*7)	- (*9)	o (*5)

o: The operation is normal.

x: A diagnostic message is output. Correction processing is performed.

-: No operation is performed.

*1. EOF means the end-of-file condition occurs.

*2. A combination of formatted and unformatted sequential input/output statements before or after a REWIND statement is not allowed.

*3. OPEN statement shall be executed for old file.

*4. BACKSPACE is executed for the endfile record.

*5. An empty file is created.

*6. Input/output is executed for the endfile record. In Fortran 95 or later, a diagnostic message (jwe0111i-e) is output. In Fortran 66 and Fortran 77, a diagnostic message (jwe0111i-e) is not output.

*7. Input/output is executed for the endfile record.

*8. Input/output is executed for files named 'fort. nn' (nn: unit number.)

*9. Error.

*10. BACKSPACE is executed for the endfile record.

*11. Input/output is executed for the endfile record. In Fortran 95 or later, a diagnostic message (jwe0115i-e) is output. In Fortran 66 and Fortran 77, a diagnostic message (jwe0115i-e) is not output.

Note:

Formatted READ and WRITE statements also include list-directed and namelist input/output statements.

7.8.1 Combinations of Input/Output Statements Not Allowed

Some combinations of input/output statements are not allowed because of the access method or file status.

An input/output statement combination that is not allowed is considered an error. After the error is corrected, execution continues.

7.8.1.1 Formatted Sequential Access Input/Output Statements

If the current statement is a formatted sequential (including list-directed and namelist) data transfer statement, the immediately preceding input/output statement must not be one of the following statements:

- Unformatted sequential access data transfer
- Direct access data transfer
- Stream access data transfer

If the current statement is READ, the immediately preceding input/output statement must not be WRITE, except the input/output is a terminal.

7.8.1.2 Unformatted Sequential Access Input/Output Statements

If the current input/output statement is an unformatted sequential data transfer statement, the immediately preceding input/output statement must not be any of the following statements:

- Formatted sequential data transfer
- List-Directed
- Namelist
- Direct access data transfer
- Stream access data transfer

If the current input/output statement is READ, the immediately preceding input/output statement must not be a WRITE statement.

7.8.1.3 Direct Access Input/Output Statements

If the current input/output statement is a direct access data transfer, the immediately preceding input/output statement must not be any of the following statements:

- Sequential data transfer
- List-Directed data transfer
- Namelist data transfer
- Stream access data transfer
- File positioning statement
- CLOSE

7.8.1.4 Stream Access Input/Output Statements

If the current input/output statement is a stream access data transfer, the immediately preceding input/output statement must not be any of the following statements:

- Sequential data transfer
- Direct access data transfer
- List-Directed data transfer
- Namelist data transfer
- File positioning statement
- CLOSE

7.8.1.5 File Positioning Statements

By combining sequential input/output statements with file positioning statements, you can control file positioning. The file positioning statements are BACKSPACE, ENDFILE, and REWIND.

When a file is connected for sequential access, the initial file position is immediately before the first record in the file unless POSITION is specified in the OPEN statement.

If the current input/output statement is BACKSPACE, which are used for file positioning, the immediately preceding input/output statement must not be either of the following statements:

- Direct access data transfer
- Unformatted stream access data transfer

If the current input/output statement is REWIND, which are used for file positioning, the immediately preceding input/output statement must not be either of the following statements:

- Direct access data transfer

If the current input/output statement is ENDFILE, which are used for file positioning, the immediately preceding input/output statement must not be either of the following statements:

- Direct access data transfer

7.9 File Positioning Statements

By combining sequential input/output statements with file positioning statements, you can control file positioning. The file positioning statements are BACKSPACE, ENDFILE, and REWIND.

- BACKSPACE Statement (see Section "[7.9.1 BACKSPACE Statement](#)")
- ENDFILE Statement (see Section "[7.9.2 ENDFILE Statement](#)")
- REWIND Statement (see Section "[7.9.3 REWIND Statement](#)")

7.9.1 BACKSPACE Statement

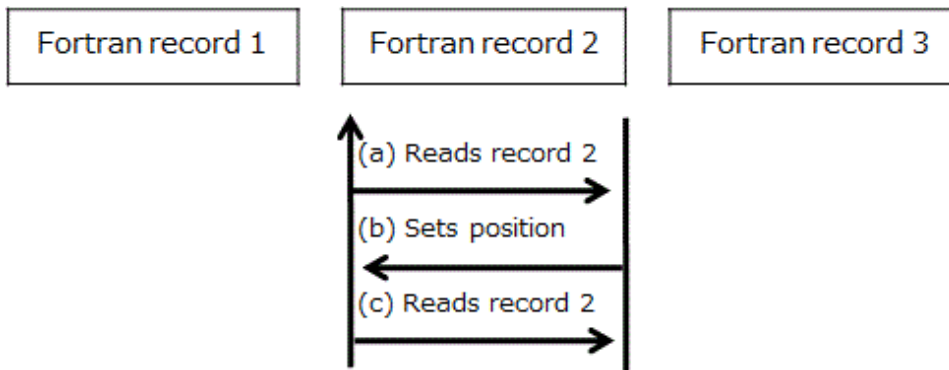
Execution of a BACKSPACE statement causes the file connected to the specified unit to be positioned before the current record if there is a current record, or before the preceding record if there is no current record. If there is no current record and no preceding record, the position of the file is not changed. If the preceding record is an endfile record, the file is positioned before the endfile record. If a BACKSPACE statement causes the implicit writing of an endfile record, the file is positioned before the record that precedes the endfile record.

Example 1: A BACKSPACE statement after a READ statement

Fortran program:

```
READ(1) IDATA ! (a) Reads Fortran record 2
BACKSPACE 1   ! (b) Position is before record 2
READ(1) NDATA ! (c) Reads Fortran record 2
END
```

File position:



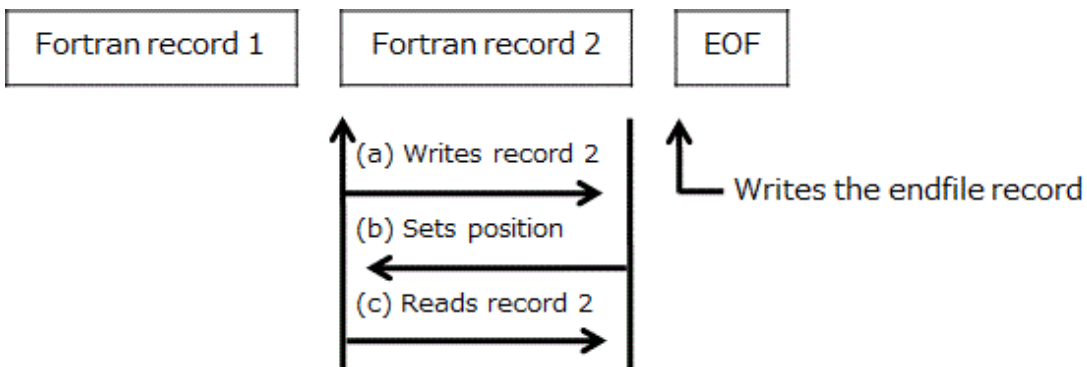
As a result, the same data is read into IDATA and NDATA.

Example 2: A BACKSPACE statement after a WRITE statement

Fortran program:

```
WRITE(1) IDATA      ! (a) Writes Fortran record 2
BACKSPACE 1        ! (b) Position is before record 2
READ(1) NDATA      ! (c) Reads Fortran record 2
END
```

File position:



As a result, the data written by IDATA is read into NDATA.

7.9.2 ENDFILE Statement

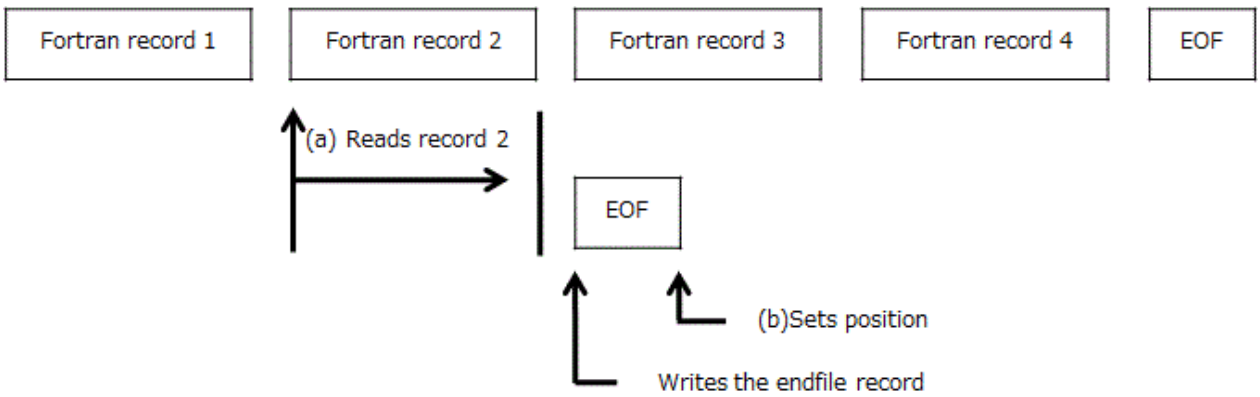
The execution of an ENDFILE statement writes an endfile record as the next record of the file. The file is then positioned after the endfile record, which becomes the last record of the file.

Example 1: An ENDFILE statement after executing a READ statement

Fortran program:

```
READ(1) IDATA      ! (a) Reads Fortran record 2.
ENDFILE 1          ! (b) Writes the endfile record.
```

File position:

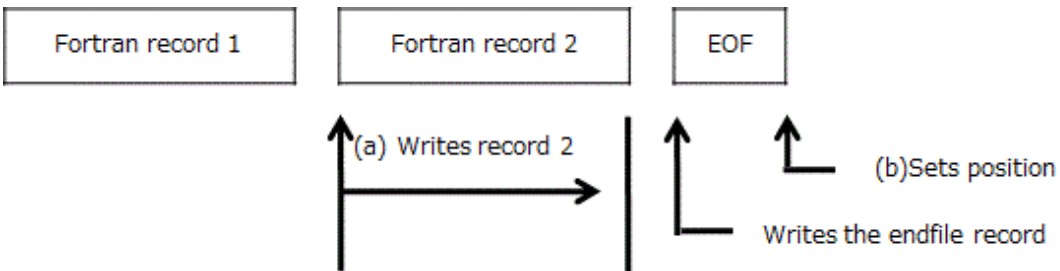


Example 2: An ENDFILE statement after executing a WRITE statement

Fortran program:

```
WRITE(1) IDATA     ! (a) Writes Fortran record 2.
ENDFILE 1         ! (b) Writes the endfile record.
```

File position:

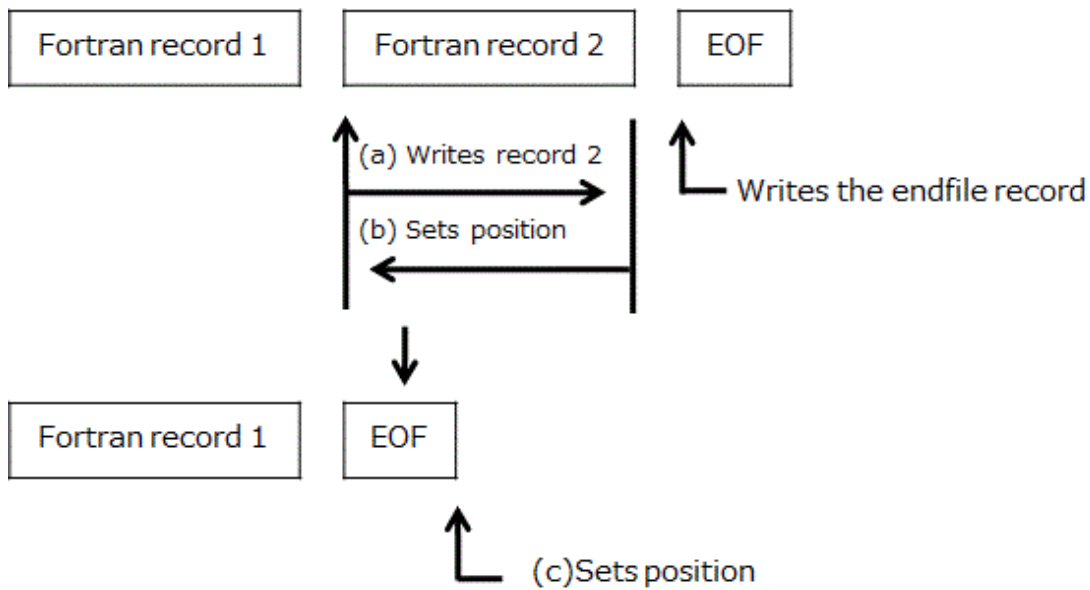


Example 3: An ENDFILE statement after executing a BACKSPACE statement

Fortran program:

```
WRITE(1) IDATA     ! (a) Write Fortran record 2.
BACKSPACE 1       ! (b) Write the endfile record.
ENDFILE 1         ! (c) Erase Fortran record 2.
```

File position:



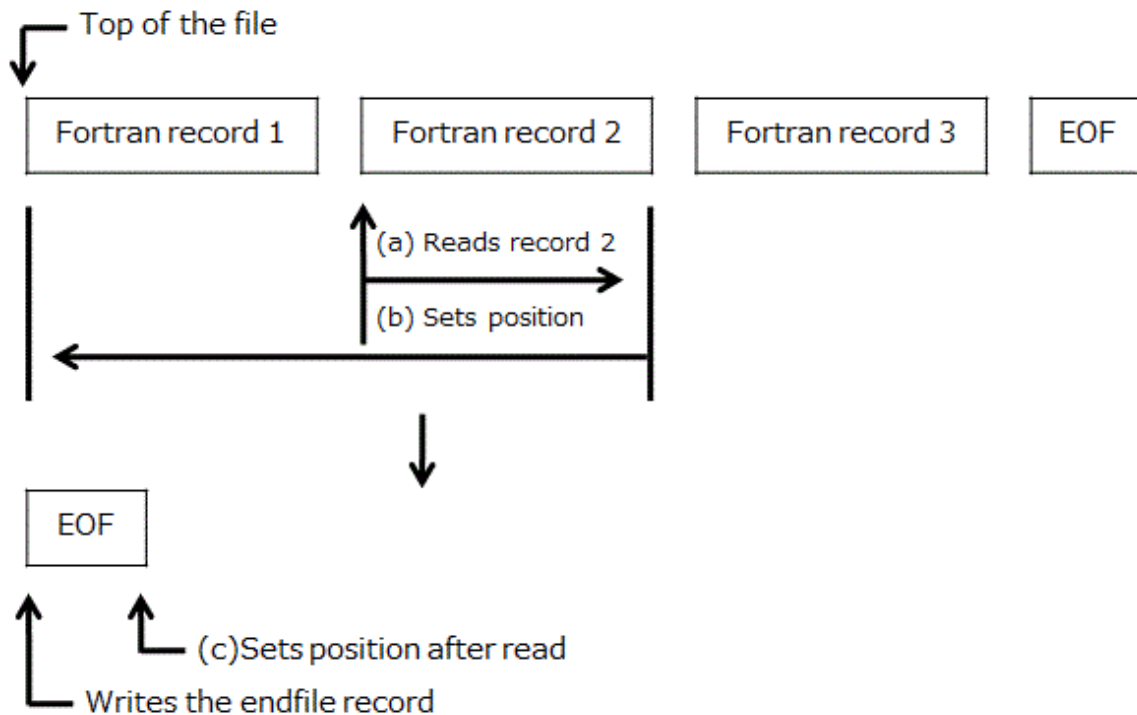
Example 4: An ENDFILE statement after executing a REWIND statement

Fortran Program:

```

READ(1)IDATA  ! (a) Reads Fortran record 2.
REWIND 1      ! (b) The file position is set to the beginning
              ! of the file.
ENDFILE 1     ! (c) Deletes all Fortran records and write the
              ! endfile record.
  
```

File position:



As a result, the file consists of only an endfile record.

7.9.3 REWIND Statement

When the file is connected using sequential or stream access, the file position is set to immediately before the first record in the file.

7.10 FLUSH Statement

Execution of a FLUSH statement causes data written to an external file to be available to other processes, or causes data placed in an external file by means other than Fortran to be available to a READ statement.

Execution of a FLUSH statement for a file that is connected but does not exist has no effect on any file. And when an immediately preceding I/O is not a write statement, it has no effect on any file in the same way.

The UNIT=, IOSTAT= and ERR= specifiers can be specified for the FLUSH statement. See Section "7.7.1 Control Information for Input/Output Data Transfer Statements" for information on these specifiers.

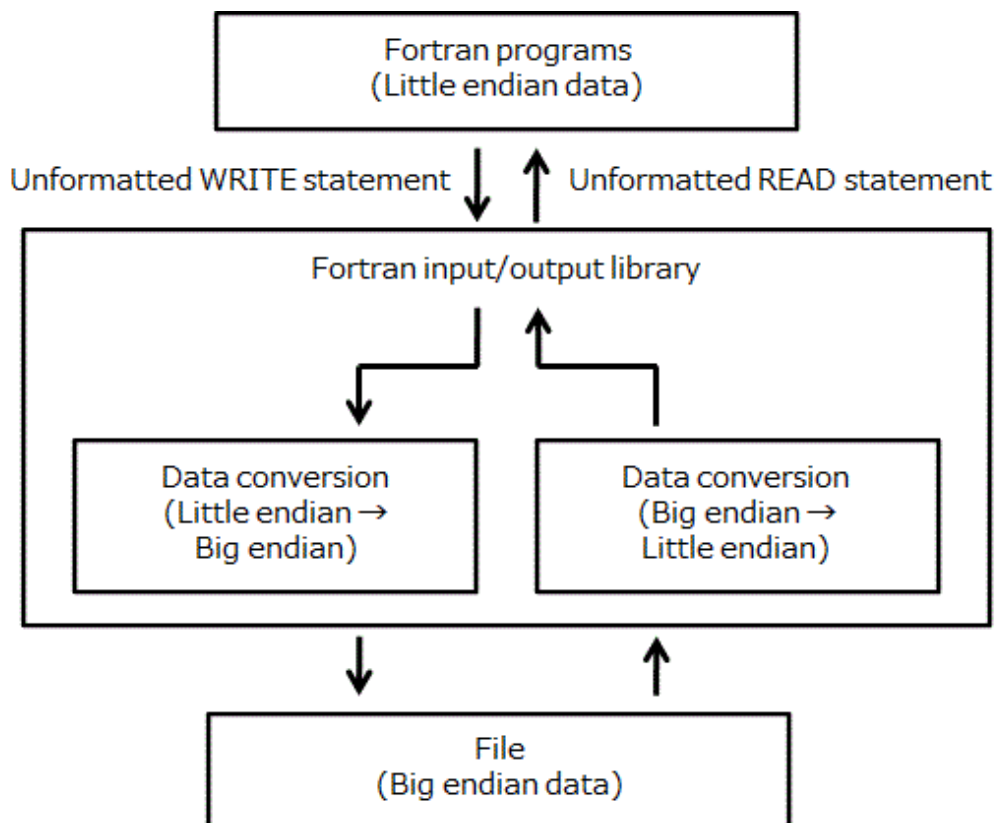
7.11 Conversion between Big Endian Data and Little Endian Data on Input/Output

This section describes the conversion between big endian data and little endian data.

7.11.1 Relationship between Big Endian Data and Input/Output Statements

You may specify the runtime option -T to read and write big endian data. If the runtime option -T is specified, the following operations are executed. See Section "3.3 Runtime Options" for information on the runtime option -T.

The flow of big endian data



The following input/output statements are allowed:

- Unformatted sequential READ/WRITE statement
- Unformatted direct READ/WRITE statement

- Unformatted stream READ/WRITE statement

Files written with variable-length Fortran records by Fujitsu Fortran implementation can be converted.

See Section "[7.3.2 Unformatted Fortran Records](#)" for details on unformatted records in this Fortran system.

The following types of input/output list items are allowed:

- 1-byte integer
- 2-byte integer
- 4-byte integer
- 8-byte integer
- Default real
- Double-precision real
- Quadruple-Precision real
- Default complex
- Double-precision complex
- Quadruple-Precision complex
- 1-byte logical
- 2-byte logical
- 4-byte logical
- 8-byte logical

7.12 Standard Files

7.12.1 Standard Files and Open Modes

The table below shows the relationship between standard files and open modes.

Input/output file	Specification method (terminal open mode if no specification)	Open mode
Standard input	command < file-name	r
	command << eof-sym	r
Standard output	command > file-name	w
	command >> file-name	a
Standard error	command 2> file-name	w
	command 2>> file-name	a

7.12.2 Restrictions on Input/Output Statements for Standard Files

Only sequential access data transfer and non-data-transfer input/output statements may be used for standard input, standard output, and error output files. Standard input files are processed as ACTION='READ'.

Standard output and standard error output files are processed as ACTION='WRITE'.

Executing an OPEN statement with a FILE= specifier for a standard input, standard output, or standard error output unit terminates any existing connection and opens a new file. The unit number of the standard input, standard output, or standard error output file is processed in the same way as a regular file.

7.12.3 Relationship between Input/Output Statements and Seekable and Nonseekable Files

Seekable files are files for which seek operations can be performed. The differences between seekable and nonseekable files are as follows:

Seekable or nonseekable	File type
Seekable file	Regular file
Nonseekable file	Block-type special file
	Character-type special file
	FIFO special file

Note:

File type refers to information obtained from stat or fstat system calls.

The input/output statements and edit descriptors listed here may be executed for seekable files. If they are executed for nonseekable files, a diagnostic message is output, and the statement is ignored.

- REWIND statement
- BACKSPACE statement
- ENDFILE statement (INPUT system)
- Direct access statement
- Stream access statement
- Unformatted sequential statement
- TL and T edit descriptors (only when the same operation as that using the TL edit descriptors is performed)

7.13 Restrictions Related to ACTION= Specifier

Input/output statements with files that have been connected using the ACTION= specifier have the following restrictions.

Input/output statement	ACTION= specifier		
	READ	WRITE	READWRITE
READ	o	x	o
WRITE	x	o	o
PRINT	x	o	o
REWIND	o	x	o
BACKSPACE	o	x	o
OPEN	o	o	o
CLOSE	o	o	o
ENDFILE	--	o	o

o: Normal operation.

x: An error message is generated. Respond accordingly.

--: No operation is performed.

7.14 Number of Significant Digits

The number of significant digits for real type is defined in this system as follows:

Type	Number of significant digits
Default real	9
Double-precision real	16
Quadruple-precision real	35
Default complex	9
Double-precision complex	16
Quadruple-precision complex	35

7.15 I/O Buffer

The table below shows how to change the size of a sequential or stream access I/O buffer. The precedence order is summarized in the below table.

Operators	Size of buffer	Precedence order
BLOCKSIZE= Specifier	Defined Value	High
Environment variable (fuxxbf)	Defined Value	.
Runtime option (-g)	Defined Value	.
Default	8M bytes	Low

7.16 Edit Descriptors

7.16.1 Generalized Integer Editing

In Fortran 66 and Fortran 77, the Gw.d and Gw.dEe edit descriptors follow the rules for the Iw.m edit descriptor when used to specify the input/output of integer data.

In Fortran 95 or later, the Gw.d and Gw.dEe edit descriptors follow the rules for the Iw edit descriptor.

Example:

```
WRITE(*, '(G10.5)')123
```

In Fortran 66 and Fortran 77, the output is ----00123 (- indicates a blank).

In Fortran 95 or later, the output is -----123.

7.16.2 Exponent Character in Formatted Output

In Fortran 66 and Fortran 77, the exponent character is output in lowercase for E, EN, ES, D, G, L, Q, and Z editing. If runtime option -q is specified, the character is output in uppercase.

In Fortran 95 or later, the character is output in uppercase whether runtime option -q is specified or not.

Example:

```
WRITE(*, FMT='(1H E15.5)')6.5432
```

In Fortran 66 and Fortran 77, the output is 0.65432e+01.

In Fortran 95 or later, the output is 0.65432E+01.

7.16.3 Result of G Editing

In Fortran 66 and Fortran 77, a zero value is edited using the E edit descriptor. In Fortran 95 or later, a zero value is edited using the F edit descriptor.

7.16.4 Diagnostic Messages for Character String Edit Descriptor

In Fortran 66 and Fortran 77, the system outputs a level w diagnostic message (jwe0159i-w) for a character string edit descriptor in a format used for input. The character string edit descriptor is replaced by input data.

In Fortran 95 or later, the system outputs a level e diagnostic message (jwe1181i-e). The input list is ignored.

7.16.5 Effect of X Edit Descriptor

If the X edit descriptor (nX) is used for output, the result differs between Fortran 66 and Fortran 77 or later.

In Fortran 77 or later, the record character position is advanced by n characters.

Blanks are not inserted.

In Fortran 66, n blanks are inserted in the record.

This operation differs only if a left-positioning tab is used:

Example:

```
FORMAT (5H FFFF,I4,T2,4X,I2)
```

The Fortran program operates as follows:

```
Fortran 77 or later program:      -FFFFjjii
Fortran 66 program:              -----jjii
```

The two low-order digits of I4 are represented by ii, I2 is represented by jj, and - indicates a blank.

For Fortran 77 or later programs, the Fortran record is first padded with blanks.

7.16.6 L Edit Descriptor

In Fortran 66 or the Fortran 77, if the type of the input item corresponding to the Lw type edit descriptor is not a logical constant, the input item is assigned the value FALSE.

In Fortran 95 or later, a diagnostic message (jwe1202i-w) is output.

Example:

Program a.f:

```
LOGICAL :: L=.TRUE.
OPEN (10,FILE='a.file')
READ (10,FMT='(L1)') L
PRINT *,L
END
```

file a.file:

```
1
```

In Fortran 66 and Fortran 77:

```
$ ./a.out
f
$
```

In Fortran 95 or later:

```
$ ./a.out
jwe1202i-w line 3 Invalid LOGICAL input (unit=10).
error occurs at MAIN__ line 3 loc 0000000000400b96 offset 0000000000000036
MAIN__ at loc 0000000000400b60 called from o.s.
taken to (standard) corrective action, execution continuing.
T
```

```

error summary (Fortran)
error number  error level  error count
  jwe1202i      w           1
total error count = 1
$

```

7.16.7 I, L, F, E, D, G, B, O, and Z Edit Descriptor without w or d Indicators

The I, L, F, E, D, G, B, O, or Z edit descriptor may omit the width (*w*) or the number of decimals (*d*).

Fortran system selects the field width when *w* is omitted.

The table below shows the selected width.

Edit descriptor	Type													
	I1	I2	I4	I8	R4	R8	R16	C8	C16	C32	L1	L2	L4	L8
I	4	6	11	20	11	11	11	11	11	11	4	6	11	20
L	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F	15	15	15	22	15	22	43	15	22	43	15	15	15	22
E	15	15	15	22	15	22	43	15	22	43	15	15	15	22
D	15	15	15	22	15	22	43	15	22	43	15	15	15	22
G	4	6	11	20	15	22	43	15	22	43	2	2	2	2
B	9	17	33	65	33	65	129	33	65	129	9	17	33	65
O	4	7	12	23	12	23	44	12	23	44	4	7	12	23
Z	3	5	9	17	9	17	33	9	17	33	3	5	9	17

I1: 1 byte integer

I2: 2 byte integer

I4: 4 byte integer

I8: 8 byte integer

R4: default real

R8: double-precision real

R16: quadruple-precision real

C8: default complex

C16: double-precision complex

C32: quadruple-precision complex

L1: 1 byte logical

L2: 2 byte logical

L4: 4 byte logical

L8: 8 byte logical

7.16.8 Editing of Special Real-Type Data

When using one of the E, D, Q, G, F, EN, or ES edit descriptors, Inf or NaN are output right-justified within width *w* of the field when E, D, Q, G, F, EN, and ES edit descriptors are specified in the format *Xw.d* in Fortran 2003 or later. In Fortran 66, Fortran 77 and Fortran 95, Inf or NaN are output left-justified.

If the output width has fewer characters than the field width *w*, it will be padded with spaces. -Inf or -NaN is output for a negative value.

Moreover, the sign can be edited with the S/SP/SS edit descriptors.

Character strings (Inf/-Inf, Infinity/-Infinity, NaN/-NaN, NaN()/-NaN()) can be used as the input data of a real number. It is zero or more alphanumeric characters parentheses of NaN(). The positive sign "+" can be omitted.

The table below shows the correspondence between strings in a file and the values set in the variables.

Correspondence between Strings and Values Set in Variables

Strings	Default real	Double-precision	Quadruple-precision
+NaN or +NaN()	0x7fffffff	0x7fffffff ffffffff	0x7fffffffffffffff ffffffff
-NaN or -NaN()	0xffffffff	0xffffffff ffffffff	0xfffffffffffffff ffffffff
+Inf or +Infinity	0x7f800000	0x7ff00000 00000000	0x7ff0000000000000 0000000000000000
-Inf or -Infinity	0xff800000	0xfff00000 00000000	0xfff0000000000000 0000000000000000

7.17 Asynchronous Input/Output Statements

This section describes asynchronous input/output statements.

7.17.1 Execution of Asynchronous Input/Output Statements

An input/output statement that includes an ASYNCHRONOUS= specifier with a YES value is executed as an asynchronous input/output statement when it is used in relation to a file connected via an ASYNCHRONOUS= specifier with a YES value specified in an OPEN Statement.

An asynchronous input/output statement is processed by making data transfer asynchronous for any data that can be transferred asynchronously within the system.

Note, however, that asynchronous data transfer is not enabled for input/output statements other than those for unformatted sequential access or when the following compiler options or runtime options are specified:

- Compiler option -H (see Section "2.2 Compiler Options")
- Compiler option -Eg (see Section "2.2 Compiler Options")
- Runtime option -G (see Section "3.3 Runtime Options")
- Runtime option -C (see Section "3.3 Runtime Options")
- Runtime option -T (see Section "3.3 Runtime Options")

In addition, asynchronous data transfer may not be enabled depending on the number, size, kind, type, or attributes of items that have been specified in an input/output list.

7.17.2 ID= Specifier in Data Transfer Input/Output Statements

A literal value less than 9223372036854775808 more than 0 that uniquely identifies an asynchronous input/output statement executed during the course of a program is assigned to an ID= specifier in a data transfer input/output statement. When a value that has been assigned to an ID= specifier exceeds the allowed range of ID= specifier values, that value is not guaranteed.

7.18 User-Defined Derived-Type Input/Output

7.18.1 User-Defined Derived-Type Input/Output Procedures

- If the parent data transfer statement specifies an internal file, the value of the unit argument in user-defined derived-type input/output procedures has -1.
- If the parent data transfer statement contains a character format specification and the *value-list* of the DT edit descriptor appears in a character format specification, a specifiable value is -2147483648 to 2147483647 in the *value-list*.
- If an input/output error occurs in the user-defined derived-type input/output procedure, termination of the parent data transfer occurs. See Section "8.1.4 Input/Output Error Processing" for the input/output error.
- The child data transfer statement shall not specify ASYNCHRONOUS='YES'.
- The following input/output statements and service routines shall not be executed in user-defined derived-type input/output procedures.
 - OPEN statement
 - CLOSE statement
 - BACKSPACE statement
 - ENDFILE statement
 - REWIND statement
 - WAIT statement
 - FGETC service function
 - FPUTC service function
 - FSEEK service function
 - FSEEKO64 service function
 - FTELL service function
 - FTELLO64 service function
 - GETC service function
 - PERROR service subroutine
 - PUTC service function
- If the parent data transfer statement uses an external unit, FLUSH statement and FLUSH service subroutine are ignored in user-defined derived-type input/output procedures.
- If the parent data transfer statement uses an external unit that is connected to the standard error file and diagnostic messages are output during invoking of user-defined derived-type input/output procedure, the data may not be output in order of writing.
- If the parent data transfer statement and the child data transfer statement access the same internal file, the data output by the child data transfer statement may not be transferred to the internal file.
- If the parent data transfer statement and the child data transfer statement access an internal file and the child data transfer statements are parallelized by OpenMP used in LLVM, the execution may not be completed correctly.

7.18.2 Data Transfer Input/Output List

- If a list item of derived type is polymorphic or has an ultimate component that is allocatable or a pointer, that list item shall be processed by a user-defined derived-type input/output procedure. If a parent data transfer statement contained a format specification, that list item's corresponding edit descriptor shall be a DT edit descriptor.

7.18.3 Namelist Input/Output

- If the parent or the child data transfer statement is namelist input, the namelist group object shall not be an array section in namelist input data.
- If a namelist-group-object is polymorphic or has an ultimate component that is allocatable or a pointer, that object shall be processed by a user-defined derived-type input/output procedure.

7.18.4 INQUIRE Statement

- When the parent input/output statement specified an internal unit, an error condition will occur if an internal unit passed to a user-defined derived-type input/output procedure is used with the INQUIRE statement. Any variable specified in an IOSTAT= specifier will be assigned the value 1124.

Chapter 8 Debugging

This chapter describes the system debugging functions provided for finding and correcting errors in source programs.

8.1 Error Processing

If an error occurs during the execution of a Fortran program, the error monitor controls the following operations:

- Generation of a diagnostic message
- Continuation of execution if allowed by the error type

By using the error monitor, the user can control the following:

- The error count limit for canceling program execution
- The printed message count limit
- Whether to print the trace back map
- Whether to call user-defined error subroutines

If the user does not specify these control operations, the system uses its default operations. For details, see "[Table 8.1 Error control table standard values](#)" in the next section.

The error monitor is invalidated for the error of LLVM OpenMP Library. See "[12.4 Runtime Messages](#)" for the error of LLVM OpenMP Library.

8.1.1 Error Monitor

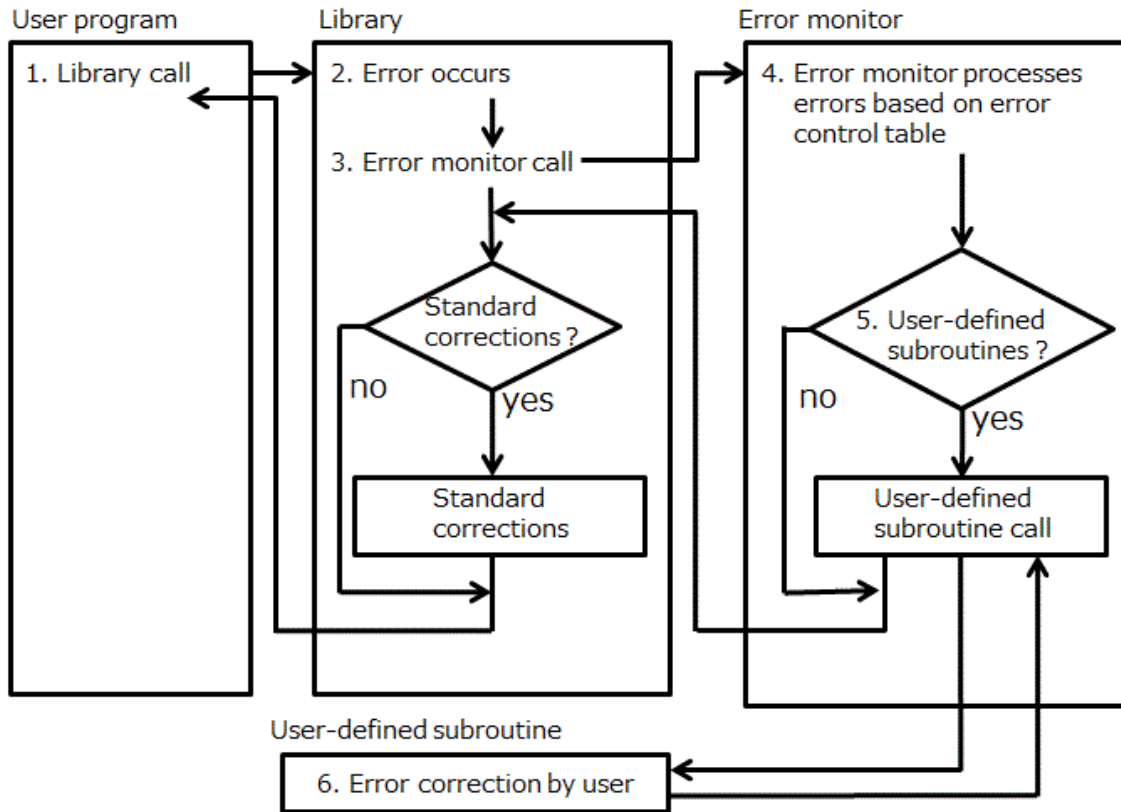
If an error occurs during execution, the error monitor is called. The error monitor processes errors based on the error control table.

The error monitor does the following:

- Checks whether the error count limit has been reached and notifies the user if it has.
- Checks whether the printed message count limit has been reached and prints a diagnostic message if it has not.
- Generates a trace back map if requested. If the Fortran program is linked to a program written in another language, some trace back map information may not be generated.
- Terminates the job if the error count limit has been reached. If the limit has not been reached, corrective action is taken and execution continues.

Corrective action for errors consists of standard corrections provided by the system and the execution of user-defined subroutines.

The figure below shows how the error monitor is called after an error.



1 – 6 : Call flow when an error occurs in libraries

Error monitor operations are based on data in the error control table. The error control table consists of 12-byte error items, as follows:

0	<i>estop</i>	<i>mprint</i>	<i>ecount</i>	<i>inf</i>	(unused)
8	<i>uexit</i>				
16	<i>ecode</i>		(unused)		
24					

The error items are:

- estop* : At byte 0, an error count limit in the range 0 to 255. Zero indicates that unlimited errors are allowed.
- mprint* : At byte 1, a printed message limit in the range 0 to 255. Zero indicates that no messages are printed.
- ecount* : At byte 2, an error count during execution, in the range 0 to 255.
- inf* : At byte 3, an error control item, as follows:
 - Bit 0 : (1) The control character is set to the diagnostic messages. (0) The control character is not set to the diagnostic messages. See Section "8.1.2.3 ERRSET Subroutine".
 - Bit 1 : (1) The user can correct the error item. (0) The user cannot correct the error item.
 - Bit 2 : (1) The error count exceeds 255. (0) The error count does not exceed 255.
 - Bit 3 : (1) The buffer is printed. (0) The buffer is not printed.
 - Bit 4 : Not used
 - Bit 5 : (1) Messages are printed. (0) Message printing is based on the value of *mprint*.
 - Bit 6 : (1) The trace back map is printed. (0) The trace back map is not printed.
 - Bit 7 : Not used

uexit : At byte 4, the address of a user-defined error subroutine. If standard error correction is used, the last bit is set to 1.

ecode : At byte 8, an error level, as follows:

- 0 : i level. Fortran program processing continues after the message is printed.
- 4 : w level. Fortran program processing continues after the error is corrected.
- 8 : e level. The error statement is ignored. Corrective action is taken and execution continues if the error count limit has not been reached. The default error count limit is 10.
- 12 : s level. The error statement is ignored. Corrective action is taken and execution continues if the error count limit has not been reached. The default error count limit is 1.
- 16 : u level. Fortran program processing terminates.

Some error items cannot be edited. For editable items, use the ERRSET subroutine to change the standard values in the error control table. The Error Control Table Standard Values indicates whether error items are editable, and gives the default values for each item in the table.

Table 8.1 Error control table standard values

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
11 to 14	1	1	Not Editable	Not Printed	Not Printed	None	u
17 to 18	1	1	Not Editable	Not Printed	Not Printed	None	u
19	1	1	Not Editable	Not Printed	Printed	None	u
20	1	1	Not Editable	Not Printed	Not Printed	None	u
21 to 22	1	1	Editable	Not Printed	Printed	Yes	s
28	1	1	Editable	Not Printed	Printed	Yes	s
42	10	5	Editable	Not Printed	Printed	Yes	e
55	10	5	Editable	Not Printed	Printed	Yes	e
62	10	5	Editable	Not Printed	Printed	Yes	e
64 to 65	10	5	Editable	Not Printed	Printed	Yes	e
71 to 72	10	5	Editable	Not Printed	Printed	Yes	e
74	10	5	Editable	Not Printed	Printed	Yes	e
80 to 82	10	5	Editable	Not Printed	Printed	Yes	e
83 to 86	Unlimited	5	Editable	Not Printed	Printed	Yes	w
87 to 89	10	5	Editable	Not Printed	Printed	Yes	e
97	Unlimited	5	Editable	Not Printed	Printed	Yes	w
102 to 103	10	5	Editable	Not Printed	Printed	Yes	e
104	Unlimited	5	Editable	Not Printed	Printed	Yes	w
105	10	5	Editable	Not Printed	Printed	Yes	e
109	Unlimited	5	Editable	Not Printed	Printed	Yes	w
111 to 115	10	5	Editable	Not Printed	Printed	Yes	e
120	10	5	Editable	Not Printed	Printed	Yes	e
122	10	5	Editable	Not Printed	Printed	Yes	e
131 to 134	10	5	Editable	Not Printed	Printed	Yes	e
151	10	5	Editable	Not Printed	Printed	Yes	e
152	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
153 to 154	Unlimited	5	Editable	Not Printed	Printed	Yes	w
155 to 157	10	5	Editable	Not Printed	Printed	Yes	e
158 to 159	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
162	Unlimited	5	Editable	Not Printed	Printed	Yes	w
171 to 172	10	5	Editable	Not Printed	Printed	Yes	e
173 to 175	Unlimited	5	Editable	Printed	Printed	Yes	w
176	10	5	Editable	Not Printed	Printed	Yes	e
182 to 183	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
184 to 189	10	5	Editable	Printed	Printed	Yes	e
190	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
202 to 255	10	5	Editable	Not Printed	Printed	Yes	e
256	Unlimited	1	Editable	Not Printed	Printed	Yes	w
257	10	5	Editable	Not Printed	Printed	Yes	s
259 to 282	10	5	Editable	Not Printed	Printed	Yes	e
291	1	1	Not Editable	Not Printed	Not Printed	None	u
292	1	1	Not Editable	Not Printed	Not Printed	None	u
301	10	5	Editable	Not Printed	Printed	Yes	w
304	Unlimited	5	Editable	Not Printed	Printed	Yes	w
306	1	1	Not Editable	Not Printed	Not Printed	None	u
308	1	1	Editable	Not Printed	Printed	Yes	s
311 to 318	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
320	10	5	Editable	Not Printed	Printed	Yes	w
322	10	5	Editable	Not Printed	Printed	Yes	w
323 to 324	Unlimited	5	Editable	Not Printed	Printed	Yes	w
329	1	1	Editable	Not Printed	Printed	Yes	s
330	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1001	1	1	Editable	Not Printed	Printed	Yes	s
1003 to 1008	1	1	Editable	Not Printed	Printed	Yes	s
1017	1	1	Not Editable	Not Printed	Not Printed	None	u
1031	1	1	Editable	Not Printed	Printed	Yes	s
1032	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1033	Unlimited	5	Editable	Not Printed	Printed	Yes	i
1034	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1035	1	1	Not Editable	Not Printed	Printed	None	u
1036	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1038	1	1	Editable	Not Printed	Printed	Yes	s
1039	1	1	Not Editable	Not Printed	Printed	None	u

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
1040 to 1041	1	1	Editable	Not Printed	Printed	Yes	s
1042	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1043	Unlimited	5	Editable	Not Printed	Not Printed	Yes	i
1044 to 1046	1	1	Not Editable	Not Printed	Printed	None	u
1047	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1048	1	1	Not Editable	Not Printed	Not Printed	None	u
1049	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1050	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1051	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1052	1	1	Not Editable	Not Printed	Printed	None	u
1071 to 1072	10	5	Editable	Not Printed	Printed	Yes	e
1073	10	5	Editable	Not Printed	Printed	Yes	e
1111 to 1113	10	5	Editable	Not Printed	Printed	Yes	e
1114 to 1115	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1117 to 1127	10	5	Editable	Not Printed	Printed	Yes	e
1141 to 1143	10	5	Editable	Not Printed	Printed	Yes	e
1146	10	5	Editable	Not Printed	Printed	Yes	e
1147	10	5	Editable	Not Printed	Printed	Yes	e
1148	10	5	Editable	Not Printed	Printed	Yes	e
1149	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1150 to 1151	10	5	Editable	Not Printed	Printed	Yes	e
1152 to 1153	1	1	Editable	Not Printed	Printed	Yes	s
1161	1	1	Editable	Not Printed	Printed	Yes	s
1162 to 1163	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1181 to 1184	10	5	Editable	Not Printed	Printed	Yes	e
1201 to 1203	Unlimited	5	Editable	Printed	Printed	Yes	w
1204	10	5	Editable	Not Printed	Printed	Yes	e
1231	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1232	1	1	Editable	Not Printed	Printed	Yes	s
1301	10	5	Editable	Not Printed	Printed	Yes	e

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
1302	Unlimited	5	Editable	Not Printed	Printed	Yes	i
1303 to 1304	10	5	Editable	Not Printed	Printed	Yes	e
1305 to 1307	1	1	Editable	Not Printed	Printed	Yes	s
1355 to 1370	10	5	Editable	Not Printed	Printed	Yes	e
1371	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1372	10	5	Editable	Not Printed	Printed	Yes	e
1373	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1375 to 1378	1	1	Editable	Not Printed	Printed	Yes	s
1381 to 1387	1	1	Editable	Not Printed	Printed	Yes	s
1390 to 1392	1	1	Editable	Not Printed	Printed	Yes	s
1393	10	5	Editable	Not Printed	Printed	Yes	e
1394 to 1395	1	1	Editable	Not Printed	Printed	Yes	s
1396 to 1414	10	5	Editable	Not Printed	Printed	Yes	e
1415	1	1	Editable	Not Printed	Printed	Yes	s
1416 to 1417	10	5	Editable	Not Printed	Printed	Yes	e
1418 to 1419	1	1	Editable	Not Printed	Printed	Yes	s
1420	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1421	10	5	Editable	Not Printed	Printed	Yes	e
1422 to 1423	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1425 to 1454	10	5	Editable	Not Printed	Printed	Yes	e
1455	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1456 to 1461	10	5	Editable	Not Printed	Printed	Yes	e
1462	1	1	Not Editable	Not Printed	Printed	Yes	s
1463 to 1474	10	5	Editable	Not Printed	Printed	Yes	e
1475	1	1	Not Editable	Not Printed	Printed	Yes	s
1501	1	1	Not Editable	Not Printed	Printed	None	u
1551	10	5	Editable	Not Printed	Printed	Yes	e
1561	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
1563 to 1564	1	1	Not Editable	Not Printed	Not Printed	None	u
1565 to 1566	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1569 to 1571	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1572 to 1575	1	1	Editable	Not Printed	Printed	Yes	s
1576 to 1577	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1578	1	1	Editable	Not Printed	Printed	Yes	s
1579	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1582	Unlimited	5	Editable	Not Printed	Printed	Yes	i
1606	1	1	Not Editable	Not Printed	Printed	None	u
1652	Unlimited	Unlimited	Editable	Not Printed	Not Printed	Yes	w
1653 to 1654	Unlimited	1	Not Editable	Not Printed	Printed	None	w
1655	Unlimited	1	Not Editable	Not Printed	Not Printed	None	i

If an error occurs, the error monitor prints the following message (indicating correction status) after the trace back map:

```
taken to (corrector) corrective action, execution continuing.
```

corrector: { standard | user }

standard: indicates that an error is corrected with standard correction.

user: indicates that an error is corrected with user-defined correction.

8.1.2 Error Subroutines

Several service subroutines provide user control of error processing during execution. These subroutines can be used to optimize error monitor use.

With these subroutines, you can dynamically change items in the error control table, generate a trace back map, and execute user-defined error subroutines.

8.1.2.1 ERRSAV Subroutine

ERRSAV stores the leading 16 bytes of the specified error item in 16 bytes user area.

The ERRSAV service subroutine calling format is as follows:

CALL ERRSAV (*errno* , *darea*)

errno

Default integer scalar. Error number.

darea

16 bytes character scalar. The error item is saved.

If the type of *darea* is not character, this result may be undefined.

8.1.2.2 ERRSTR Subroutine

ERRSTR moves an error item to the error control table element for the specified error number. Thus, when an error occurs, the new error item controls error processing.

The ERRSTR service subroutine calling format is as follows:

CALL ERRSTR (*errno* , *darea*)

errno

Default integer scalar. Error number.

darea

16 bytes character scalar. The error item is set.

If the type of *darea* is not character, this result may be undefined.

Example: ERRSTR and ERRSAV

```
CHARACTER(LEN=16) ERR113 ! (1)ESTOP, (2)MPRINT, (3)ECOUNT, (4)INF
CALL ERRSAV(113,ERR113)
WRITE(ERR113(1:2), '(2a1)')0,0
CALL ERRSTR(113,ERR113)
OPEN(10,FILE='X.DAT',FORM='FORMATTED')
DO I=1,15
  WRITE(10)I
END DO
CLOSE(10,STATUS='DELETE')
END
```

8.1.2.3 ERRSET Subroutine

ERRSET changes the control table data for the error item with the specified error number. The error count limit, message print limit, trace back map printing switch, and error corrective action may be changed.

The ERRSET service subroutine calling format is as follows:

CALL ERRSET (*errno* , *estop* , *mprint* , *trace* , *uexit* , *r*)

errno

Default integer scalar. Error number.

estop

Default integer scalar. Error count limit.

<=0 : Error count limit is the same as before.

>=256 : It indicates unlimited error counts.

mprint

Default integer scalar. Message print limit.

<0 : Message print limit is zero (that is, a no-print message).

=0 : Message print limit is the same as before.

>=256 : It indicates unlimited message prints.

trace

Default integer scalar. One of the following values indicates whether to print a trace back map:

=0 : Do not change.

=1 : Do not print.

=2 : Print.

uexit

Default integer scalar. Error correction or user-defined correction subroutine indicated by one of the following:

=0 : Do not change.

=1 : Perform standard correction.

=other : Execute the user-defined correction.

r

If *errno* is not 132, a four-byte integer expression indicates the largest error number to be changed.

If *errno* is 132, indicates whether a blank character is inserted at the beginning of the correction data with the following values:

=1 : Insert a blank character.

=other : Do not insert a blank character.

Note

If the error count limit for the error item number with an error level of *s* is changed and the program execution continues, the program may not work properly, such as when the program terminates abnormally. If the program does not work properly, specify a value of 0 or less for *estop*.

Example 1:

```
EXTERNAL FIXUP
CALL ERRSET(202,50,-1,0, FIXUP,205)
```

In this example, the error items for error numbers 202 to 205 are changed as follows:

- The error count limit is changed to 50.
- A message will not be printed if an error occurs.
- The print control information for the trace back map is not changed.
- The user-defined subroutine FIXUP is called to process errors.

Example 2:

```
CALL ERRSET(132,0,0,0,0,1)
```

In this example, if the Fortran record specified in the format specification is longer than the record length of the file, the next Fortran record is output as a new record with a leading space added.

8.1.2.4 ERRTRA Subroutine

The ERRTRA service subroutine generates a trace back map up to the program unit currently executing. See Section "4.2 Executable Program Output" for the map format. Execution continues after this subroutine is called.

The ERRTRA service subroutine calling format is as follows:

```
CALL ERRTRA
```

Example: ERRTRA Service Subroutine

```
OPEN(10,FILE='X.DAT')
CALL SUB1
END
SUBROUTINE SUB1
CALL ERRTRA
DO I=1,15
  WRITE(10,*)I
END DO
END
```


8.1.3 Using Error Monitor

By calling the error monitor, the user can control error processing, specify user-defined error subroutines, and use the error service functions. Related topics explain how to use the error service functions.

8.1.3.1 User-Defined Error Subroutines

The ERRSET subroutine may be used to store the address of a user-defined error subroutine in the error control table. If a user-defined error subroutine has been specified, the error monitor calls the error subroutine as follows:

CALL *user*(*ret* , *errno* [, *data1* , ... , *datan*])

user

Name of the user-defined error subroutine.

ret

four-byte integer scalar variable that stores a return code. The return code must be provided by the user-defined error subroutine.

errno

Number of the detected error.

data1 , ... , *datan*

Scalar variable names or array element name list used by the user-defined subroutine.

The arguments passed to the user-defined error subroutine are as follows:

- Input-output error: The arguments in Section "[8.1.4 Input/Output Error Processing](#)" are passed.
- Intrinsic function error: The arguments in Section "[8.1.5 Intrinsic Function Error Processing](#)" are passed.
- Intrinsic subroutine error: The arguments in Section "[8.1.6 Intrinsic Subroutine Error Processing](#)" are passed.
- Interrupt error: The arguments in Section "[8.1.7 Exception Handling Processing](#)" are passed.
- Otherwise, the arguments in Section "[8.1.8 Other Error Processing](#)" are passed.

The following restrictions apply to user-defined error subroutines:

- Restrictions on input/output statements.
If an error occurs, input/output statements cannot be used with the same unit number as the unit with the error. For example, if an error occurs with unit number 10, input/output statements with unit number 10 cannot be used.
- Restrictions on intrinsic functions.
If an error occurs, the intrinsic function that caused the error cannot be used in the user-defined subroutine. For example, if an argument value is incorrect in SQRT, SQRT is disabled in the user-defined subroutine. This restriction also applies to intrinsic functions referenced implicitly when expressions that include exponentiation are used.
- Restrictions on intrinsic subroutines.
If an error occurs, the intrinsic subroutine that caused the error cannot be used in the user-defined subroutine.
- Restrictions on return codes.
If arguments *data1*...*datan* of the user-defined error subroutine are not changed, the user-defined subroutine must set the return code to 0. If the passed arguments are changed, the subroutine must set the return code to 1. Only 0 or 1 may be used as return code values. For all other values, the results are unpredictable.
- Restrictions on the argument type.
For error subroutines written in Fortran, the argument type must match that of the CALL statement.

8.1.3.2 Handling of Incorrect Arguments in Intrinsic Functions

If an intrinsic function argument is incorrect, the argument is assigned to a local variable in the intrinsic function and the error monitor is called. The local variable is passed to the error monitor. In user-defined error subroutines, this local variable must be corrected. Do not correct the argument in the program unit that used the incorrect argument.

8.1.3.3 User Error Processing

The following example shows user error processing. In this example, when the user enters incorrect data (1234567890123), error number 171 is generated. To continue processing, the user corrects the data to 9999 with the user-defined error subroutine FIXUP.

Example: User error processing

Fortran program:

```
EXTERNAL FIXUP
CALL ERRSET(171,0,0,0, FIXUP,0)
READ(5,*)I
WRITE(6,100)I
100 FORMAT(1H, 5X, 'I=',I4)
STOP
END
```

User-defined error correction subroutine:

```
SUBROUTINE FIXUP(IRET,INO,I4CONV)
IF(INO==171)THEN
  I4CONV=9999
  IRET=1
END IF
RETURN
END
```

Input data:

```
1234567890123
```

Result:

```
jwe0171i-e line 3 Integer(kind=1), integer(kind=2) or integer(kind=4) number (1234567890123) is out
of range (unit= 5).
error occurs at MAIN__ line 3 loc 0000000000400bb5 offset 0000000000000055
MAIN__ at loc 0000000000400b60 called from o.s.
taken to (user) corrective action, execution continuing.
I=9999
```

8.1.4 Input/Output Error Processing

If an IOSTAT= specifier, ERR= specifier, END= specifier, or EOR= specifier has been specified, input/output errors are processed based on the specifier without regard to the error count limit. See Section "7.7.1.3 IOSTAT= Specifier", Section "7.7.1.4 Error Branch", Section "7.7.1.5 End-of-File Branch", and Section "7.7.1.6 End-of-Record Branch" for details.

If an error occurs, the file position and input list values being executed at the time of the error are undefined.

The table below lists the standard system and user-defined corrections for errors that occur during input/output statement execution. The following arguments are passed to the user-defined error program:

- Return code
- Error number
- The data listed in the table below

Table 8.2 Error processing for input/output

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
21 to 22	Terminates the program	Not correctable	UNIT	None
28	Terminates the program	Not correctable	UNIT	None

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
42	Ignores the statement	Not correctable	UNIT	None
55	Ignores the statement	Not correctable	UNIT	None
62	Ignores the statement	Not correctable	UNIT	None
64 to 65	Ignores the statement	Not correctable	UNIT	None
71 to 72	Ignores the statement	Not correctable	UNIT	None
74	Ignores the statement	Not correctable	UNIT	None
80	Ignores the statement	Not correctable	UNIT	REC
81	Ignores the statement	Not correctable	None	None
82	Ignores the statement	Not correctable	UNIT	None
83 to 85	Continues processing without setting a specifier value.	Not correctable	None	None
86	Continues processing without setting a specifier value.	Not correctable	UNIT	None
87 to 89	Ignores the statement	Not correctable	UNIT	None
97	Uses DIRECT as the ACCESS= specifier	Not correctable	UNIT	None
102 to 103	Ignores the statement	Not correctable	UNIT	None
104	Ignores a POSITION= specifier and continues processing	Not correctable	UNIT	None
105	Ignores the statement	Not correctable	UNIT	None
109	The maximum value of the type specified by specifier value is set, and processing is continued.	Not correctable	UNIT	None
111 to 115	Ignores the statement	Not correctable	UNIT	None
120	Ignores the statement	Not correctable	UNIT	None
122	Ignores the statement	Not correctable	UNIT	None
131	Ignores subsequent input/output lists	Not correctable	None	None
132	a. For input, ignores data that exceeded the record b. For output, writes the excess data to the next logical record	Not correctable	UNIT	None
133	Ignores subsequent output list items	Not correctable	UNIT	None
134	Ignores subsequent input/output lists	Not correctable	UNIT	None
151	Ignores subsequent input/output lists	Not correctable	UNIT	None
152	Edits according to the edit descriptor	Not correctable	UNIT	None
153	Adds the initial left parentheses	Not correctable	UNIT	None
154	Ignores nests exceeding 30 levels in the format specification	Not correctable	UNIT	None
155	Ignores the edit descriptor and goes by default to the final right parenthesis of the format specification	Not correctable	UNIT	None
156	Ignores the repeat specification and goes by default to the final right parenthesis of the format specification	Not correctable	UNIT	None

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
157	Ignores the incorrect character and goes by default to the final right parenthesis of the format specification	Assign the character of the format specification to FCODE	FCODE	None
158	Adds the right parenthesis and continues format control	Not correctable	UNIT	None
159	Replaces the input data with the character string in the format specification	Not correctable	UNIT	None
162	Ignores the input/output list and continues a format control	Not correctable	UNIT	None
171	<p>a. Uses by default the minimum specifiable value if the one-, two- or four-byte input item is negative</p> <p>b. Uses by default the maximum specifiable value if the one-, two- or four-byte input item is positive</p>	Correct I4CNV	I4CNV	None
172	<p>a. Uses by default the minimum specifiable value if the eight-byte input item is negative</p> <p>b. Uses by default the maximum specifiable value if the eight-byte input item is positive</p>	Correct I8CNV	I8CNV	None
173 to 175	Uses 0 for the incorrect character	Assign the corrected character to IDATA	IDATA	None
176	<p>a. Sets true 0 if the absolute value is less than $2^{**}(-126)*2^{**}(-23)$ (real), $2^{**}(-1022)*2^{**}(-52)$ (double-precision real) or $2^{**}(-16382)*2^{**}(-112)$ (quadruple-precision real)</p> <p>b. Sets $2^{**}127*(1+1.0-2^{**}(-23))$(real), $2^{**}1023*(1+1.0-2^{**}(-52))$ (double-precision real) or $2^{**}16383*(1+1.0- 2^{**}(-112))$ (quadruple-precision real) without changing the sign if the absolute value is greater than $2^{**}127*(1+1.0-2^{**}(-23))$ real), $2^{**}1023*(1+1.0-2^{**}(-52))$ (double-precision real) or $2^{**}16383*(1+1.0-2^{**}(-112))$ (quadruple-precision real)</p>	Correct FCONV	FCONV	None
182 to 183	Executes character editing	Not correctable	UNIT	None
184	Ignores the constant and terminates the statement	Not correctable	UNIT	None
185	Ignores the repeat count or constant and terminates the statement	Not correctable	UNIT	None
186 to 188	Ignores the item name and terminates the statement	Not correctable	UNIT	None
189	Ignores the statement	Not correctable	UNIT	None
190	Ignores the constant that exceeded the element and continues the processing	Not correctable	UNIT	None

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
1073	Ignores the statement	Not correctable	UNIT	None
1111 to 1113	Ignores the statement	Not correctable	UNIT	None
1114	Assigns the maximum value for 4 byte integer to the specifier, and continues the processing	Not correctable	UNIT	None
1115	An invalid specifier in an open statement, and continues the processing	Not correctable	UNIT	None
1117 to 1127	Ignores the statement	Not correctable	UNIT	None
1146	Ignores the statement	Not correctable	UNIT	None
1147	Ignores the statement	Not correctable	UNIT	None
1148	Ignores the statement	Not correctable	UNIT	None
1149	Uses NO as the ASYNCHRONOUS= specifier	Not correctable	UNIT	None
1150 to 1151	Ignores the statement	Not correctable	UNIT	None
1152 to 1153	Terminates the program	Not correctable	UNIT	None
1161	Terminates the program	Not correctable	UNIT	None
1162	The multiple records are written or read, and continues the processing	Not correctable	UNIT	None
1163	Assigns the maximum value for 4 byte integer to the variable, and continues the processing	Not correctable	None	None
1181	Ignores the input data and continues the program	Not correctable	UNIT	None
1182 to 1184	Ignores the statement	Not correctable	UNIT	None
1201	Uses 0 for the incorrect character	Assign the corrected character to IDATA	IDATA	None
1202	Ignores the input data and continues the processing	Not correctable	UNIT	None
1203	Edits with the type of data and continues the processing	Not correctable	UNIT	None
1204	Ignores the statement	Not correctable	UNIT	None

UNIT: For an external file, the unit number (four-byte integer) is specified. For an internal file, -1 is specified.

IDATA: Incorrect character in the input data (one-byte character)

FCODE: Incorrect format code (one-byte character) in the format assigned by the array

REC: Record variable (four-byte integer)

I4CNV: Integer indicating result of format conversion (four-byte integer)

I8CNV: Integer indicating result of format conversion (eight-byte integer)

FCONV: Integer indicating result of format conversion (real)

8.1.5 Intrinsic Function Error Processing

The table below lists the causes of intrinsic function errors and the standard system correction processing in execution.

When -Ncheck_intrfunc option is specified, the error numbers of 202-255, 259-282, 1355-1370, 1396-1414, 1416-1417, 1425-1454, 1456-1461, and 1463-1474 are detected and the intrinsic function error processing is performed.

When -Ncheck_intrfunc option is not specified, the floating point exception occurs instead of the error. The floating point exception is detected that -NRtrap option and environment variable FLIB_EXCEPT=u are specified.

See Section "2.2 Compiler Options" for details of -Ncheck_intrfunc option. See Section "3.8 Using Environment Variables for Execution" for detail of environment variable FLIB_EXCEPT=u.

The following arguments are passed to the user-defined error subroutine:

- Return code
- Error number
- The data listed in the table below

Table 8.3 Error processing for intrinsic function

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
202	SIN(X)	X >=8.23e+05	NaN	X	None
	COS(X)				
203	DSIN(DX)	DX >=3.53d+15	NaN	DX	None
	DCOS(DX)				
204	QSIN(QX)	QX >=2**62*PI	NaN	QX	None
	QCOS(QX)				
205	CSIN(CX)	X1 >=8.23e+05	(NaN,NaN)	X1	None
	CCOS(CX)				
206	CSIN(CX)	X2>=89.415	(SIGN(Inf,SIN(X1)), SIGN(Inf,COS(X1)))	X2	None
		X2<=-89.415	(SIGN(Inf,SIN(X1)), -SIGN(Inf,COS(X1)))		
	CCOS(CX)	X2>=89.415	(SIGN(Inf,COS(X1)), -SIGN(Inf,SIN(X1)))		
		X2<=-89.415	(SIGN(Inf,COS(X1)), SIGN(Inf,SIN(X1)))		
207	CDSIN(CDX)	DX1 >=3.53d+15	(NaN,NaN)	DX1	None
	CDCOS(CDX)				
208	CDSIN(CDX)	DX2>=710.475	(DSIGN(Inf,DSIN(DX1)), DSIGN(Inf,DCOS(DX1)))	DX2	None
		DX2<=-710.475	(DSIGN(Inf,DSIN(DX1)), -DSIGN(Inf,DCOS(DX1)))		
	CDCOS(CDX)	DX2>=710.475	(DSIGN(Inf,DCOS(DX1)), -DSIGN(Inf,DSIN(DX1)))		
		DX2<=-710.475	(DSIGN(Inf,DCOS(DX1)), DSIGN(Inf,DSIN(DX1)))		
209	CQSIN(CQX)	QX1 >=2**62*PI	(NaN,NaN)	QX1	None
	CQCOS(CQX)				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
210	CQSIN(CQX)	$QX2 \geq 11357.125$	(QSIGN(Inf,QSIN(QX1)), QSIGN(Inf,QCOS(QX1)))	QX2	None
		$QX2 \leq -11357.125$	(QSIGN(Inf,QSIN(QX1)), -QSIGN(Inf,QCOS(QX1)))		
	CQCOS(CQX)	$QX2 \geq 11357.125$	(QSIGN(Inf,QCOS(QX1)), -QSIGN(Inf,QSIN(QX1)))		
		$QX2 \leq -11357.125$	(QSIGN(Inf,QCOS(QX1)), QSIGN(Inf,QSIN(QX1)))		
211	TAN(X)	$ X \geq 8.23e+05$	NaN	X	None
	COTAN(X)				
212	TAN(X)	$ X $ is close to singularity (+-PI/2,+3PI/2,...)	Inf	X	None
	COTAN(X)	$ X $ is close to singularity (0,+PI,+2PI,...)			
213	DTAN(DX)	$ DX \geq 3.53d+15$	NaN	DX	None
	DCOTAN(DX)				
214	DTAN(DX)	$ DX $ is close to singularity (+-PI/2,+3PI/2,...)	Inf	DX	None
	DCOTAN(DX)	$ DX $ is close to singularity (0,+PI,+2PI,...)			
215	QTAN(QX)	$ QX \geq 2*62*PI$	NaN	QX	None
	QCOTAN(QX)				
216	QTAN(QX)	$ QX $ is close to singularity (+-PI/2,+3PI/2,...)	Inf	QX	None
	QCOTAN(QX)	$ QX $ is close to singularity (0,+PI,+2PI,...)			
217	ASIN(X)	$ X > 1.0$	NaN	X	None
	ACOS(X)				
218	DASIN(DX)	$ DX > 1.0$	NaN	DX	None
	DACOS(DX)				
219	QASIN(QX)	$ QX > 1.0$	NaN	QX	None
	QACOS(QX)				
220	ATAN(X1,X2)	X1=0.0 and X2=0.0	NaN	X1	X2
	ATAN2(X1,X2)				
221	DATAN2(DX1,DX2)	DX1=0.0 and DX2=0.0	NaN	DX1	DX2

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
222	QATAN2(QX1,QX2)	QX1=0.0 and QX2=0.0	NaN	QX1	QX2
223	SINH(X)	X>=89.415	Inf	X	None
		X<=-89.415	-Inf		
224	COSHH(X)	X >=89.415	Inf	DX	None
		DX>=710.475	Inf		
	DCOSHH(DX)	DX >=710.475	Inf		
225	QSINH(QX)	QX>=11357.125	Inf	QX	None
		QX<=-11357.125	-Inf		
	QCOSHH(QX)	QX >=11357.125	Inf		
226	SQRT(X)	X<0.0	NaN	X	None
227	DSQRT(DX)	DX<0.0	NaN	DX	None
228	QSQRT(QX)	QX<0.0	NaN	QX	None
229	EXP(X)	X>=88.722	Inf	X	None
230	DEXP(DX)	DX>=709.782	Inf	DX	None
231	QEXP(QX)	QX>=11356.5	Inf	QX	None
232	CEXP(CX)	X1>=88.722	(SIGN(Inf,COS(X1)), SIGN(Inf,SIN(X1)))	X1	None
233	CEXP(CX)	X2 >=8.23e+05	(NaN,NaN)	X2	None
234	CDEXP(CDX)	DX1>=709.782	(DSIGN(Inf,DCOS(DX1)), DSIGN(Inf,DSIN(DX1)))	DX1	None
235	CDEXP(CDX)	DX2 >=3.53d+15	(NaN,NaN)	DX2	None
236	CQEXP(CQX)	QX1>=11356.5	(QSIGN(Inf,QCOS(QX1)), QSIGN(Inf,QSIN(QX1)))	QX1	None
237	CQEXP(CQX)	QX2 >=2**62*PI	(NaN,NaN)	QX2	None
238	EXP2(X)	X>=128.0	Inf	X	None
	2.0**X				
239	DEXP2(DX)	DX>=1024.0	Inf	DX	None
	2.0**DX				
240	QEXP2(QX)	QX>=16384.0	Inf	QX	None
	2.0**QX				
241	EXP10(X)	X>=38.531	Inf	X	None
	10.0**X				
242	DEXP10(DX)	DX>=308.254	Inf	DX	None
	10.0**DX				
243	QEXP10(QX)	QX>=4932.0625	Inf	QX	None
	10.0**QX				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
244	ALOG(X)	X=0.0	-Inf	X	None
		X<0.0	NaN		
	ALOG10(X)	X=0.0	-Inf		
		X<0.0	NaN		
	ALOG2(X)	X=0.0	-Inf		
		X<0.0	NaN		
245	DLOG(DX)	DX=0.0	-Inf	DX	None
		DX<0.0	NaN		
	DLOG10(DX)	DX=0.0	-Inf		
		DX<0.0	NaN		
	DLOG2(DX)	DX=0.0	-Inf		
		DX<0.0	NaN		
246	QLOG(QX)	QX=0.0	-Inf	QX	None
		QX<0.0	NaN		
	QLOG10(QX)	QX=0.0	-Inf		
		QX<0.0	NaN		
	QLOG2(QX)	QX=0.0	-Inf		
		QX<0.0	NaN		
247	CLOG(CX)	CX=(0.0,0.0)	(Inf,NaN)	CX	None
248	CDLOG(CDX)	CDX=(0.0,0.0)	(Inf,NaN)	CDX	None
249	CQLOG(CQX)	CQX=(0.0,0.0)	(Inf,NaN)	CQX	None
256	LGT(C1,C2)	An EBCDIC code, not corresponding to an ASCII code was detected in a character string C1 or C2	The default ASCII collation order of incorrect argument, 26(X'1A'), is used.	C3	None
	LGE(C1,C2)				
	LLT(C1,C2)				
	LLE(C1,C2)				
257	IBSET(I,POS)	The value of POS exceeded the range	I	None	
	IBCLR(I,POS)				
	BTEST(I,POS)				
259	IX1**IX2	IX1=0 and IX2<0	0	IX1	IX2
260	JX1**JX2	JX1=0 and JX2<0	0	JX1	JX2
261	X**IX	X=0.0 and IX<0	Inf	X	IX
262	X1**X2	X1=0.0 and X2<0.0	Inf	X1	X2
263	X1**X2	X1<0.0 and X2/=0.0	NaN	X1	X2
264	DX**IX	DX=0.0 and IX<0	Inf	DX	IX
265	DX**JX	DX=0.0 and JX<0	Inf	DX	JX

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
266	DX1**DX2	DX1=0.0 and DX2<0.0	Inf	DX1	DX2
267	DX1**DX2	DX1<0.0 and DX2/=0.0	NaN	DX1	DX2
268	QX**IX	QX=0.0 and IX<0	Inf	QX	IX
269	QX**JX	QX=0.0 and JX<0	Inf	QX	JX
270	QX1**QX2	QX1=0.0 and QX2<0.0	Inf	QX1	QX2
271	QX1**QX2	QX1<0.0 and QX2/=0.0	NaN	QX1	QX2
272	QX1**QX2	QX1**QX2 >=2**16384	Inf	QX1	QX2
273	CX**IX	CX=(0.0,0.0) and IX<0	(NaN,NaN)	CX	IX
274	CDX**IX	CDX=(0.0,0.0) and IX<0	(NaN,NaN)	CDX	IX
275	CDX**JX	CDX=(0.0,0.0) and JX<0	(NaN,NaN)	CDX	JX
276	CQX**IX	CQX=(0.0,0.0) and IX<0	(NaN,NaN)	CQX	IX
277	CQX**JX	CQX=(0.0,0.0) and JX<0	(NaN,NaN)	CQX	JX
278	QX1/QX2	QX1/QX2 >=2**16384	Inf	QX1	QX2
279	QX1/QX2	QX1/QX2 <2**(-16382)	0.0	QX1	QX2
280	QX1/QX2	QX2=0.0	For QX1=0.0 NaN For QX1/=0.0 SIGN(QX1)*Inf	QX1	QX2
281	JX1+JX2	JX1+JX2 >2**63-1	2**63-1	JX1	JX2
	JX1-JX2	JX1-JX2 >2**63-1			
	JX1*JX2	JX1*JX2 >2**63-1			
	JX1**JX2	JX1**JX2 >2**63-1			
282	JX1/JX2	JX2=0	For JX1=0 0 For JX1/=0 2**63-1	QX1	QX2

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1355	TANQ(X)	X is close to singularity (+-1,+3, ...)	Inf	X	None
	COTANQ(X)	X is close to singularity (0,+2,+4, ...)			
1356	DTANQ(DX)	DX is close to singularity (+-1,+3, ...)	Inf	DX	None
	DCOTANQ(DX)	DX is close to singularity (0,+2,+4, ...)			
1357	ASINQ(X)	X >1.0	NaN	X	None
	ACOSQ(X)				
1358	DASINQ(DX)	DX >1.0	NaN	DX	None
	DACOSQ(DX)				
1359	ATAN2Q(X1,X2)	X1=0.0 and X2=0.0	NaN	X1	X2
1360	DATAN2Q(DX1,DX2)	DX1=0.0 and DX2=0.0	NaN	DX1	DX2
1361	SIND(X)	X >=4.72e+07	NaN	X	None
	COSD(X)				
1362	DSIND(DX)	DX >=2.03d+17	NaN	DX	None
	DCOSD(DX)				
1363	TAND(X)	X >=4.72e+07	NaN	X	None
	COTAND(X)				
1364	DTAND(DX)	DX >=2.03d+17	NaN	DX	None
	DCOTAND(DX)				
1365	TAND(X)	X is close to singularity (+90,+270, ...)	Inf	X	None
	COTAND(X)	X is close to singularity (0,+180,+360, ...)			
1366	DTAND(DX)	DX is close to singularity (+90,+270, ...)	Inf	DX	None
	DCOTAND(DX)	DX is close to singularity (0,+180,+360, ...)			
1367	ASIND(X)	X >1.0	NaN	X	None
	ACOSD(X)				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1368	DASIND(DX)	DX >1.0	NaN	DX	None
	DACOSD(DX)				
1369	ATAN2D(X1,X2)	X1=0.0 and X2=0.0	NaN	X1	X2
1370	DATAN2D(DX1,DX2)	DX1=0.0 and DX2=0.0	NaN	DX1	DX2
1371	ISHFTC(I,SHIFT[,SIZE])	SIZE<=0 or SIZE>BIT_SIZE(I)	BIT_SIZE(I) is set to SIZE, and process is continued	None	
		SHIFT >SIZE	Value of SIZE is set to SHIFT, and process is continued		
1372	IBITS(I,POS,LEN)	LEN<0 or LEN>BIT_SIZE(I)-POS	0	None	
		POS<0 or POS>=BIT_SIZE(I)-LEN			
1373	ISHFT(I,SHIFT)	SHIFT >BIT_SIZE(I)	BIT_SIZE(I) is set to SHIFT, and process is continued	None	
1375	MATMUL(MATRIX_A,MATRIX_B)	The size of last dimension of MATRIX_A and the size of first dimension of MATRIX_B is not same	Terminates the program	None	
1376	SPREAD(SOURCE,DIM,NCOPIES)	DIM<=0 or DIM>(The rank of SOURCE)+1	Terminates the program	None	
1377	ALL(MASK[,DIM])	DIM<=0 or DIM>(The rank of MASK)	Terminates the program	None	
	ANY(MASK[,DIM])				
	PARITY(MASK[,DIM])				
	COUNT(MASK[,DIM,KIND])				
	SUM(ARRAY,DIM[,MASK])	DIM<=0 or DIM>(The rank of ARRAY)			
	PRODUCT(ARRAY,DIM[,MASK])				
	MAXVAL(ARRAY,DIM[,MASK])				
	MINVAL(ARRAY,DIM[,MASK])				
	UBOUND(ARRAY[,DIM])				
	LBOUND(ARRAY[,DIM,KIND])				
	SIZE(ARRAY[,DIM])				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	MAXLOC(ARRAY,DIM[,MASK, KIND,BACK]) MINLOC(ARRAY,DIM[,MASK, KIND,BACK]) FINDLOC(ARRAY,VALUE,DI M[,MASK,KIND,BACK]) IALL(ARRAY,DIM[,MASK]) IANY(ARRAY,DIM[,MASK]) IPARITY(ARRAY,DIM[,MASK]) NORM2(X[,DIM])	DIM<=0 or DIM>(The rank of X)			
1378	MAXLOC(ARRAY,DIM[,MASK, KIND,BACK]) or MAXLOC(ARRAY[,MASK,KIN D,BACK]) MINLOC(ARRAY,DIM[,MASK, KIND,BACK]) or MINLOC(ARRAY[,MASK,KIND ,BACK]) SUM(ARRAY,DIM[,MASK]) or SUM(ARRAY[,MASK]) PRODUCT(ARRAY,DIM[,MAS K]) or PRODUCT(ARRAY[,MASK]) MAXVAL(ARRAY,DIM[,MASK])) or MAXVAL(ARRAY[,MASK]) MINVAL(ARRAY,DIM[,MASK]) or MINVAL(ARRAY[,MASK]) PACK(ARRAY,MASK[, VECTO R]) FINDLOC(ARRAY,VALUE,DI M[,MASK,KIND,BACK])	The shape of ARRAY and MASK is different	Terminates the program	None	

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	or FINDLOC(ARRAY,VALUE[,MASK,KIND,BACK]) or IALL(ARRAY,DIM[,MASK]) or IALL(ARRAY[,MASK]) or IANY(ARRAY,DIM[,MASK]) or IANY(ARRAY[,MASK]) or IPARITY(ARRAY,DIM[,MASK]) or IPARITY(ARRAY[,MASK])				
1381	PACK(ARRAY,MASK[,VECTOR])	The elements number of VECTOR is less than total number of TRUE of MASK	Terminates the program	None	
1382	UNPACK(VECTOR,MASK[,FIELD])	The elements number of VECTOR is less than total number of TRUE of MASK	Terminates the program	None	
1383	UNPACK(VECTOR,MASK[,FIELD])	The shape of MASK and FIELD is not same	Terminates the program	None	
1384	CSHIFT(ARRAY,SHIFT[,DIM]) EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	DIM<=0 or DIM>(The rank of ARRAY)	Terminates the program	None	
1385	CSHIFT(ARRAY,SHIFT[,DIM]) EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	The shape of SHIFT is invalid	Terminates the program	None	
1386	EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	The shape of BOUNDARY is invalid	Terminates the program	None	
1387	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The size of SOURCE is less than the product of every element's values of SHAPE	Terminates the program	None	
1390	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The shape of SHAPE and ORDER is different	Terminates the program	None	

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1391	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The value of the elements ORDER must be the combination of the numbers to the size of SHAPE	Terminates the program	None	
1392	REPEAT(String,NCOPIES)	NCOPIES<0	Terminates the program	None	
1393	NEAREST(X,S)	S=0.0	X	None	
1394	SIZE(ARRAY[,DIM])	If ARRAY is the assumed-size array DIM<1 or DIM>=(The rank of ARRAY)	Terminates the program	None	
	UBOUND(ARRAY[,DIM])				
1395	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The type parameter of SOURCE and PAD is different	Terminates the program	None	
	PACK(ARRAY,MASK[,VECTOR])	The type parameter of ARRAY and VECTOR is different			
	UNPACK(VECTOR,MASK[,FIELD])	The type parameter of VECTOR and FIELD is different			
	MERGE(TSOURCE,FSOURCE, MASK)	The type parameter of TSOURCE and FSOURCE is different			
	EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	The type parameter of ARRAY and BOUNDARY is different			
1396	MODULO(A,P)	P=0	For REAL type A Inf For INTEGER type A HUGE(A)	None	
1397	QX1+QX2	QX1+QX2 >=2**16384(overflow)	Inf	QX1	QX2
	QX1-QX2	QX1-QX2 >=2**16384(overflow)			
	QX1*QX2	QX1*QX2 >=2**16384(overflow)			
1398	QX1*QX2	QX1*QX2 <2**(-16382) (under flow)	0.0	QX1	QX2

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1399	QTANQ(QX)	QX is close to singularity (+-1,+3, ...)	Inf	QX	None
	QCOTANQ(QX)	QX is close to singularity (0,+2,+4, ...)			
1400	QASINQ(QX)	QX >1.0	NaN	QX	None
	QACOSQ(QX)				
1401	QATAN2Q(QX1,QX2)	QX1=0.0 and QX2=0.0	NaN	QX1	QX2
1402	QSIND(QX)	QX >=2**62*180	NaN	QX	None
	QCOSD(QX)				
1403	QTAND(QX)	QX >=2**63*90	NaN	QX	None
	QCOTAND(QX)				
1404	QTAND(QX)	QX is close to singularity (+90,+270, ...)	Inf	QX	None
	QCOTAND(QX)	QX is close to singularity (0,+180,+360, ...)			
1405	QASIND(QX)	QX >1.0	NaN	QX	None
	QACOSD(QX)				
1406	QATAN2D(QX1,QX2)	QX1=0.0 and QX2=0.0	NaN	QX1	QX2
1407	CSINQ(CX)	X1 >=5.24e+05	(NaN,NaN)	X1	None
	CCOSQ(CX)				
1408	CSINQ(CX)	X2>=56.92	(SIGN(Inf,SINQ(X1)), SIGN(Inf,COSQ(X1)))	X2	None
		X2<=-56.92	(SIGN(Inf,SINQ(X1)), -SIGN(Inf,COSQ(X1)))		
	CCOSQ(CX)	X2>=56.92	(SIGN(Inf,COSQ(X1)), -SIGN(Inf,SINQ(X1)))		
		X2<=-56.92	(SIGN(Inf,COSQ(X1)), SIGN(Inf,SINQ(X1)))		
1409	CDSINQ(CDX)	DX1 >=2.25d+15	(NaN,NaN)	DX1	None
	CDCOSQ(CDX)				
1410	CDSINQ(CDX)	DX2>=452.30	(DSIGN(Inf,DSINQ(DX1)), DSIGN(Inf,DCOSQ(DX1)))	DX2	None
		DX2<=-452.30	(DSIGN(Inf,DSINQ(DX1)), -DSIGN(Inf,DCOSQ(DX1)))		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	CDCOSQ(CDX)	DX2>=452.30 DX2<=-452.30	(DSIGN(Inf,DCOSQ(DX1)) , DSIGN(Inf,DSINQ(DX1))) (DSIGN(Inf,DCOSQ(DX1)) ,DSIGN(Inf,DSINQ(DX1)))		
1411	CQSINQ(CQX) CQCOSQ(CQX)	QX1 >=2**63	(NaN,NaN)	QX1	None
1412	CQSINQ(CQX) CQCOSQ(CQX)	QX2>=7230.125 QX2<=-7230.125 QX2>=7230.125 QX2<=-7230.125	(QSIGN(Inf,QSINQ(QX1)), QSIGN(Inf,QCOSQ(QX1))) (QSIGN(Inf,QSINQ(QX1)), - QSIGN(Inf,QCOSQ(QX1))) (QSIGN(Inf,QCOSQ(QX1)) , QSIGN(Inf,QSINQ(QX1))) (QSIGN(Inf,QCOSQ(QX1)) ,QSIGN(Inf,QSINQ(QX1)))	QX2	None
1413	X**JX	X=0.0 and JX<0	Inf	X	JX
1414	CX**JX	CX=(0.0,0.0) and JX<0	(NaN,NaN)	CX	JX
1415	SELECTED_REAL_KIND(P,R,R ADIX)	P, R, and RADIX are not specified	Terminates the program	None	
1416	X1**X2	X1**X2 >=2**128	Inf	X1	X2
1417	DX1**DX2	DX1**DX2 >=2**1024	Inf	DX1	DX2
1418	PACK(ARRAY,MASK[,VECTOR])	The elements number of VECTOR must be greater than or equal to the total number of ARRAY if MASK is scalar with the value TRUE	Terminates the program	None	
1419	EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	Do not omit BOUNDARY, when ARRAY is a derived type.	Terminates the program	None	
1420	ICHAR(C [,KIND]) INDEX(STRING,SUBSTRING[,BACK, KIND]) LBOUND(ARRAY,[, DIM, KIND]) LEN(STRING[, KIND])	The function result exceeds the expressible range that kind type parameter is specified by KIND	The kind type parameter of result uses the specified kind value	None	

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	SCAN(String, SET[, BACK, KIND]) SHAPE(SOURCE[, KIND]) SIZE(Array[, DIM, KIND]) UBOUND(Array[, DIM, KIND]) VERIFY(String, STE[, BACK, KIND])				
1421	IEEE_CLASS(X) IEEE_COPY_SIGN(X,Y) IEEE_IS_FINITE(X) IEEE_IS_NORMAL(X) IEEE_IS_NEGATIVE(X) IEEE_LOGB(X) IEEE_NEXT_AFTER(X,Y) IEEE_REM(X,Y) IEEE_RINT(X) IEEE_SCALB(X, I) IEEE_UNORDERD(X,Y) IEEE_VALUE(X,CLASS)	The function used is not supported.	0 or .FALSE.	None	
1422	IEEE_IS_NAN(X) IEEE_VALUE(X,CLASS)	The function used is not supported.	0 or .FALSE.	None	
1425	ACOSH(X)	X<1	NaN	X	None
1426	ACOSH(DX)	DX<1	NaN	DX	None
1427	ACOSH(QX)	QX<1	NaN	QX	None
1428	ATANH(X)	X >1 X=1 X=-1	NaN Inf -Inf	X	None
1429	ATANH(DX)	DX >1 DX=1 DX=-1	NaN Inf -Inf	DX	None
1430	ATANH(QX)	QX >1 QX=1 QX=-1	NaN Inf -Inf	QX	None
1431	TAN(CX)	X1 >=8.23e+05	(NaN,NaN)	X1	None
1432	TAN(CX)	CX is close to singularity ((-PI/2,0),(+3PI/2,0), ...)	For REAL(CX)>=0.0 (Inf,0.0)	CX	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
			For REAL(CX)<0.0 (-Inf,0.0)		
1433	TAN(CDX)	DX1 >=3.53d+15	(NaN,NaN)	DX1	None
1434	TAN(CDX)	CDX is close to singularity ((+PI/2.0),(+3PI/2.0), ...)	For DREAL(CDX)>=0.0 (Inf,0.0) For DREAL(CDX)<0.0 (-Inf,0.0)	CDX	None
1435	TAN(CQX)	QX1 >=2.0**62*PI	(NaN,NaN)	QX1	None
1436	TAN(CQX)	CQX is close to singularity ((+PI/2.0),(+3PI/2.0), ...)	For QREAL(CQX)>=0.0 (Inf,0.0) For QREAL(CQX)<0.0 (-Inf,0.0)	CQX	None
1437	ATAN(CX)	CX=(0.0,+1.0)	(0.0,+Inf)	CX	None
1438	ATAN(CDX)	CDX=(0.0,+1.0)	(0.0,+Inf)	CDX	None
1439	ATAN(CQX)	CQX=(0.0,+1.0)	(0.0,+Inf)	CQX	None
1440	SINH(CX)	X2 >=8.23e+05	(NaN,NaN)	X2	None
	COSH(CX)				
1441	SINH(CX)	X1>=89.4150	(SIGN(Inf,COS(X2)), SIGN(Inf,SIN(X2)))	X1	None
		X1<=-89.4150	(-SIGN(Inf,COS(X2)), SIGN(Inf,SIN(X2)))		
	COSH(CX)	X1>=89.4150	(SIGN(Inf,COS(X2)), SIGN(Inf,SIN(X2)))		
		X1<=-89.4150	(SIGN(Inf,COS(X2)), -SIGN(Inf,SIN(X2)))		
1442	SINH(CDX)	DX2 >=3.53d+15	(NaN,NaN)	DX2	None
	COSH(CDX)				
1443	SINH(CDX)	DX1>=710.475	(DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))	DX1	None
		DX1<=-710.475	(-DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))		
	COSH(CDX)	DX1>=710.475	(DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))		
		DX1<=-710.475	(DSIGN(Inf,DCOS(DX2)), -DSIGN(Inf,DSIN(DX2)))		
1444	SINH(CQX)	QX2 >=2.0**62*PI	(NaN,NaN)	QX2	None
	COSH(CQX)				
1445	SINH(CQX)	QX1>=11357.125	(QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))	QX1	None
		QX1<=-11357.125	(-QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	COSH(QX)	QX1 >= 11357.125 QX1 <= -11357.125	(QSIGN(Inf, QCOS(QX2)), QSIGN(Inf, QSIN(QX2))) (QSIGN(Inf, QCOS(QX2)), -QSIGN(Inf, QSIN(QX2)))		
1446	TANH(CX)	X2 >= 8.23e+05	(NaN, NaN)	X2	None
1447	TANH(CX)	CX is close to singularity ((0, +-PI/2), (0, +-3PI/2), ...)	(NaN, +-Inf)	CX	None
1448	TANH(CDX)	DX2 >= 3.53d+15	(NaN, NaN)	DX2	None
1449	TANH(CDX)	CDX is close to singularity ((0, +-PI/2), (0, +-3PI/2), ...)	(NaN, +-Inf)	CDX	None
1450	TANH(CQX)	QX2 >= 2.0**62*PI	(NaN, NaN)	QX2	None
1451	TANH(CQX)	CQX is close to singularity ((0, +-PI/2), (0, +-3PI/2), ...)	(NaN, +-Inf)	CQX	None
1452	ATANH(CX)	CX = (+-1.0, 0.0)	(+-Inf, 0.0)	CX	None
1453	ATANH(CDX)	CDX = (+-1.0, 0.0)	(+-Inf, 0.0)	CDX	None
1454	ATANH(CQX)	CQX = (+-1.0, 0.0)	(+-Inf, 0.0)	CQX	None
1455	DSHIFTL(I, J, SHIFT)	SHIFT < 0 or SHIFT > BIT_SIZE(I)	0 is set to SHIFT in SHIFT < 0 or BIT_SIZE(I) is set to SHIFT in SHIFT > BIT_SIZE(I), and process is continued	None	
	DSHIFTR(I, J, SHIFT)				
	SHIFTA(I, SHIFT)	I < 0 or I > BIT_SIZE(I)	0 is set to I in I < 0 or BIT_SIZE(I) is set to I in I > BIT_SIZE(I), and process is continued		
	SHIFTL(I, SHIFT)				
	SHIFTR(I, SHIFT)				
	MASKL(I [, KIND])				
	MASKR(I [, KIND])				
1456	BESSEL_J0(X)	X >= 8.23e+05	NaN	X	None
	BESSEL_J1(X)				
	BESSEL_JN(N, X)				
	BESSEL_JN(N1, N2, X)				
1457	BESSEL_J0(DX)	DX >= 3.53d+15	NaN	DX	None
	BESSEL_J1(DX)				
	BESSEL_JN(N, DX)				
	BESSEL_JN(N1, N2, DX)				
1458	BESSEL_J0(QX)	QX >= 2**62*PI	NaN	QX	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	BESSEL_J1(QX)				
	BESSEL_JN(N, QX)				
	BESSEL_JN(N1,N2,QX)				
1459	BESSEL_Y0(X)	X>=8.23e+05 or X<0.0	NaN	X	None
		X=0.0	-Inf		
	BESSEL_Y1(X)	X>=8.23e+05 or X<0.0	NaN		
		X=0.0	-Inf		
	BESSEL_YN(N, X)	X>=8.23e+05 or X<0.0	NaN		
		X=0.0	-Inf		
	BESSEL_YN(N1,N2,X)	X>=8.23e+05 or X<0.0	NaN		
		X=0.0	-Inf		
1460	BESSEL_Y0(DX)	DX>=3.53d+15 or DX<0.0	NaN	DX	None
		DX=0.0	-Inf		
	BESSEL_Y1(DX)	DX>=3.53d+15 or DX<0.0	NaN		
		DX=0.0	-Inf		
	BESSEL_YN(N, DX)	DX>=3.53d+15 or DX<0.0	NaN		
		DX=0.0	-Inf		
	BESSEL_YN(N1,N2,DX)	DX>=3.53d+15 or DX<0.0	NaN		
		DX=0.0	-Inf		
1461	BESSEL_Y0(QX)	QX>=2**62*PI or QX<0.0	NaN	QX	None
		QX=0.0	-Inf		
	BESSEL_Y1(QX)	QX>=2**62*PI or QX<0.0	NaN		
		QX=0.0	-Inf		
	BESSEL_YN(N, QX)	QX>=2**62*PI or QX<0.0	NaN		
		QX=0.0	-Inf		
	BESSEL_YN(N1,N2,QX)	QX>=2**62*PI or NaN	NaN		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
		QX<0.0			
		QX=0.0	-Inf		
1462	BESSEL_JN(N, X)	N<0	NaN	None	
	BESSEL_JN(N, DX)				
	BESSEL_JN(N, QX)				
	BESSEL_YN(N, X)				
	BESSEL_YN(N, DX)				
	BESSEL_YN(N, QX)				
	BESSEL_JN(N1,N2,X)	N1<0 or N2<0	zero-size array		
	BESSEL_JN(N1,N2,DX)				
	BESSEL_JN(N1,N2,QX)				
	BESSEL_YN(N1,N2,X)				
	BESSEL_YN(N1,N2,DX)				
	BESSEL_YN(N1,N2,QX)				
1463	GAMMA(X)	X>=35.03986	Inf	X	None
1464	GAMMA(DX)	DX>=171.6243	Inf	DX	None
	DGAMMA(DX)				
1465	GAMMA(QX)	QX>=1.755q+03	Inf	QX	None
	QGAMMA(QX)				
1466	LOG_GAMMA(X)	X>=0.403711e+37	Inf	X	None
	LGAMMA(X)				
	ALGAMA(X)				
1467	LOG_GAMMA(DX)	DX>=2.55634d+305	Inf	DX	None
	LGAMMA(DX)				
	DLGAMA(DX)				
1468	LOG_GAMMA(QX)	QX>=1.048q+4928	Inf	QX	None
	LGAMMA(QX)				
	QLGAMA(QX)				
1469	GAMMA(X)	X is a negative integer	NaN	X	None
		X=+-0.0	+ -Inf		
	LOG_GAMMA(X)	X is equal to 0.0 or a negative integer	Inf		
	LGAMMA(X)				
	ALGAMA(X)				
1470	GAMMA(DX)	DX is a negative integer	NaN	DX	None
		DX=+-0.0	+ -Inf		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	DGAMMA(DX)	DX is a negative integer	NaN		
		DX=+-0.0	+-Inf		
	LOG_GAMMA(DX)	DX is equal to 0.0 or a negative integer	Inf		
	LGAMMA(DX)				
	DLGAMA(DX)				
1471	GAMMA(QX)	QX is a negative integer	NaN	QX	None
		QX=+-0.0	+-Inf		
	QGAMMA(QX)	QX is a negative integer	NaN		
		QX=+-0.0	+-Inf		
	LOG_GAMMA(QX)	QX is equal to 0.0 or a negative integer	Inf		
	LGAMMA(QX)				
	QLGAMA(QX)				
1472	ERFC_SCALED(X)	$X \leq -9.3824$	Inf	X	None
1473	ERFC_SCALED(DX)	$DX \leq -26.6287$	Inf	DX	None
1474	ERFC_SCALED(QX)	$QX \leq -106.56$	Inf	QX	None

- C1, C2, and C3 are one-byte character data. The value 26('1A') is assigned to C3.
- IX, IX1, and IX2 are four-byte integer data.
- JX, JX1, and JX2 are eight-byte integer data.
- X, X1, and X2 are real data.
- DX, DX1, and DX2 are double-precision real data.
- QX, QX1, and QX2 are quadruple-precision real data.
- CX is complex data. CX indicates (X1,X2).
- CDX is double-precision complex data. CDX indicates (DX1,DX2).
- CQX is quadruple-precision complex data. CQX indicates (QX1,QX2).
- PI is about 3.141592.

Note:

In many case, the arguments passed to the user-defined error subroutines are same as or part of argument of intrinsic procedure referencing, however it is not same area.

8.1.6 Intrinsic Subroutine Error Processing

The table below lists the causes of intrinsic subroutine errors and the standard system correction processing.

The following arguments are passed to the user-defined error subroutine:

- Return code
- Error number
- The data listed in the table bellow

Table 8.4 Error processing for intrinsic subroutine

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine
1301	MVBITS(FROM, FROMPOS, LEN, TO, TOPOS)	LEN < 0 or LEN > BIT_SIZE(FROM)	TO is set to Zero	None
		FROMPOS < 0 or FROMPOS >= BIT_SIZE(FROM) - LEN	TO is set to Zero	None
		TOPOS < 0 or TOPOS >= BIT_SIZE(FROM) - LEN	TO is set to Zero	None
1302	DATE_AND_TIME([DATE, TIME, ZONE, VALES])	The length of DATE is less than 8.	The value to be returned is truncated from the same length as DATE.	None
		The length of TIME is less than 10.	The value to be returned is truncated from the same length as TIME.	None
		The length of ZONE is less than 5.	The value to be returned is truncated from the same length as ZONE.	None
1303	DATE_AND_TIME([DATE, TIME, ZONE, VALES])	Size of VALES is less seven	The values are returned in VALES, and the rest of the values to be returned are discarded.	None
1304	RANDOM_SEED([SIZE, PUT, GET])	Size of PUT is less SIZE	The seed values are returned in PUT, and the rest of the values to be returned are discarded.	None
		Size of GET is less SIZE	The seed values are returned in GET, and the rest of the values to be returned are discarded.	None
1305	RANDOM_SEED([SIZE, PUT, GET])	Two or three arguments are specified.	Terminates the program	None
1306	MOVE_ALLOC([FROM, TO])	The length of argument FROM and TO of the character type is different	Terminates the program	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine
1307	EXECUTE_COMMAND_LINE(COMMAND[,WAIT,EXITSTAT,CMDSTAT,CMDMSG])	An error was detected in a system call.	Terminates the program	None
1422	IEEE_GET_HALTING_MODE(FLAG,HALTING)	The function used is not supported.	Nothing is done.	None
	IEEE_SET_HALTING_MODE(FLAG,HALTING)			
	IEEE_GET_UNDERFLOW_MODE(GRADUAL)			
	IEEE_SET_UNDERFLOW_MODE(GRADUAL)			
	IEEE_VALUE(X,CLASS)			
1423	IEEE_SET_ROUNDING_MODE(ROUND_VALUE)	The function used is not supported.	Nothing is done.	None

8.1.7 Exception Handling Processing

The system has several program interrupts for error processing. These interrupts are such as floating-point exceptions, invalid access exceptions.

The system enables traps corresponding to signals (*), and handles floating-point exceptions (jwe1017i-u). If the compiler option -NRnotrap is enabled, floating-point exceptions are not detected. See Section "2.2 Compiler Options" for more information about the compiler option -NRnotrap.

*: Only signals supported by the Fujitsu CPU A64FX with Arm architecture are targeted. Therefore, enabling the compiler option -NRtrap will not detect unsupported signals.

If runtime option -i is specified, no exception handling is taken. See Section "3.3 Runtime Options" and Section "8.2.2 Debugging Programs for Abend" for details.

The table below shows an exception handling.

Table 8.5 Exception handling processing

Error number	Interrupt cause	Standard correction	User-defined correction	Data passed to the user-defined error subroutine	
				Data1	Data2
17	Terminated abnormally due to runtime option -t	Terminates the program	Not correctable	None	
18	Terminated abnormally due to runtime option -a	Terminates the program	Not correctable	None	
19	Terminated abnormally due to an invalid access.	Terminates the program	Not correctable	None	
20	Detected an error during abnormally termination process	Terminates the program	Not correctable	None	
1017	The result of the floating-point operation exceeds 65504 (half precision real and complex), 3.40282347E+38 (single precision real and complex), 1.797693134862316D+308	Terminates the program	Not correctable	None	

Error number	Interrupt cause	Standard correction	User-defined correction	Data passed to the user-defined error subroutine	
				Data1	Data2
	(double precision real and complex), or 1.189731495357231765085 7593266280070Q+493 (quadruple precision real and complex)				
	The result of the floating-point operation is less than 6.10351562E-05 (half precision real and complex), 1.17549435E-38 (single precision real and complex), 2.225073858507201D-308 (double precision real and complex), or 3.362103143112093506262 6778173217526Q-4932 (quadruple precision real and complex)	Terminates the program	Not correctable	None	
	The divisor in floating-point division is zero.	Terminates the program	Not correctable	None	
	An invalid calculation in floating-point is executed.	Terminates the program	Not correctable	None	

Note:

- When standard correction processing sets a value in the result register, the sign is unchanged. In addition, the maximum values are as follows:

Half precision real : 65504

Single precision real : 3.40282347E+38

Double precision real : 1.797693134862316D+308

Quadruple precision real : 1.1897314953572317650857593266280070Q+4932

- The above interrupts do not occur for quadruple-precision real. See Section "8.1.5 Intrinsic Function Error Processing" for details.

8.1.8 Other Error Processing

The table below lists the error processing performed when the system detects an error other than an input/output, intrinsic function, or program interrupts error. The following arguments are passed to the user-defined error subroutine:

- Return code
- Error number
- The data listed in the table below

Table 8.6 Other error processing

Error number	Standard correction processing	User-defined correction	Data passed to the user-defined error subroutine
301	Continues processing, assuming that first argument is default value	Not correctable	None
304	CALL PRNSET(IX)	Not correctable	IX

Error number	Standard correction processing	User-defined correction	Data passed to the user-defined error subroutine
	Continues processing using IX = 0 for IX < 0, or IX = 15 for IX > 15		
306	Terminates the program	Not correctable	None
308	Terminates the program	Not correctable	None
311 to 318	Continues checking arguments	Not correctable	None
320	Continues processing	Not correctable	None
322 to 324	Continues processing	Not correctable	None
329	Terminates the program	Not correctable	None
330	Continues processing	Not correctable	None
1001	Terminates the program	Not correctable	None
1003 to 1008	Terminates the program	Not correctable	None
1031	Terminates the program	Not correctable	None
1032	The multiple number is defined by CPU number and the program is continued.	Not correctable	None
1033 (*1)	Terminates the program	Not correctable	None
1034	Continues processing	Not correctable	None
1035	Terminates the program	Not correctable	None
1036	Continues processing	Not correctable	None
1038 to 1041	Terminates the program	Not correctable	None
1042	The multiple number is defined by num and the program is continued.	Not correctable	None
1043	Continues processing	Not correctable	None
1044 to 1046	Terminates the program	Not correctable	None
1047	Continues processing	Not correctable	None
1048	Terminates the program	Not correctable	None
1049	Continues processing	Not correctable	None
1050	Continues processing	Not correctable	None
1051	Continues processing	Not correctable	None
1052	Terminates the program	Not correctable	None
1071 to 1072	Continues processing	Not correctable	None
1141 to 1143	Continues processing	Not correctable	None
1231	Continues processing	Not correctable	None
1232	Terminates the program	Not correctable	None

Error number	Standard correction processing	User-defined correction	Data passed to the user-defined error subroutine
1501	Terminates the program	Not correctable	None
1551	Continues processing	Not correctable	None
1561	Continues processing	Not correctable	None
1563 to 1564	Terminates the program	Not correctable	None
1565 to 1566	Continues processing	Not correctable	None
1569	Continues processing	Not correctable	None
1570 to 1571	Continues checking arguments	Not correctable	None
1572 to 1575	Terminates the program	Not correctable	None
1576 to 1577	Continues processing	Not correctable	None
1578	Continues processing	Not correctable	None
1579	Terminates the program	Not correctable	None
1582	Continues processing	Not correctable	None
1606	Terminates the program	Not correctable	None
1652 to 1655	Continues processing	Not correctable	None

IX: Default integer scalar

*1: The diagnostic message (jwe1033i-i) indicates that the Fortran program has been terminated during execution on multiple processors. If this message is output, the results of the Fortran program may be different from the results of terminating the program while running on a single processor by executing a STOP/ERROR STOP statement or an EXIT service subroutine, or by exceeding the error count limit.

8.2 Debugging Functions

This section describes the check functions for debugging Fortran programs and the debug information output when execution ends abnormally.

8.2.1 Debugging Check Functions

The debugging check functions perform debugging during compilation or execution.

The `-Ncheck_global` compiler option is available as check functions to perform debugging during compilation.

During compilation, the compiler option `-Ncheck_global` checks the following:

- The procedure characteristics between external procedure definitions and references
- The procedure characteristics between external procedure definitions and interface body.
- In program units the size of common blocks

During compilation, the compiler option `-Ha` or `-Nquickdbg=argchk` checks the following:

- Array size in procedure in explicit reference specification.

During compilation, the compiler option `-Hs` or `-Nquickdbg=subchk` checks the following:

- The ranges of the declaration and reference for array element, and substring.

During execution, the compile option -H and -Nquickdbg check the following:

Table 8.7 List of runtime check items

	-H option	-Nquickdbg option
Checking Argument Validity (ARGCHK)		
Checking the Number of Arguments	-Ha	-Nquickdbg=argchk
Checking the Argument Types		-
Checking Argument Attributes		-Nquickdbg=argchk
Checking Function Types		-
Checking Argument Sizes		-Nquickdbg=argchk
Checking Explicit Interface		-
Checking Rank of Assumed-Shape Array		-
Checking Subscript and Substring Values (SUBCHK)		
Checking Array and Substring References	-Hs	-Nquickdbg =subchk
Checking Array Section References		-
Checking References for Undefined Data Items (UNDEF)		
Checking for Undefined Data	-Hu	-Nquickdbg =undef -Nquickdbg =undefnan
Checking for an Allocatable Variable		-
Checking for an Optional Argument		-
Shape Conformance Check (SHAPECHK)		
Shape Conformance Check	-He	-
Extended Checking of UNDEF (EXTCHK)		
Checking for Undefined Data of a Module, Submodule, and a Common Block	-Hx	-
Checking for Unassociated Pointer		-
Checking for Stride and Increment Value		-
Checking for DO Variable		-
Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)		
Overlapping Dummy Arguments	-Ho	-
Checking for undefined Assumed-size Array with INTENT(OUT)		-
Checking for undefined variable of SAVE attribute		-
Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)		
Checking Connection of a File to A Unit	-Hf	-
I/O Recursive Call CHECK		-

-: No support

Details of checked items are given in the following sections:

- Check argument validity. (See Section "8.2.1.1 Checking Argument Validity (ARGCHK)")
- Check subscript and substring values. (See Section "8.2.1.2 Checking Subscript and Substring Values (SUBCHK)")

- Check references for undefined data. (See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)")
- Check shape conformance. (See Section "8.2.1.4 Shape Conformance Check (SHAPECHK)")
- Extended check references for undefined data. (e.g. the variable declared in a module, in a submodule, or in a common block).(See Section "8.2.1.5 Extended Checking of UNDEF (EXTCHK)")
- Check the part of overlap is changed when two dummy arguments are overlapped. (See Section "8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)")
- Check the file is connected with two devices or more. (See Section "8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)")
- Check an input/output statement is executed from other input/output statement. (See Section "8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)")

The compiler option -H and -Nquickdbg check have the characteristics shown following:

Table 8.8 Characteristics of runtime check functions

	-H option	-Nquickdbg option
Effect on execution performance	Large	Small
Checked items	Many	Few
Thread parallelism (*1)	Not available	Available

*1: Checks are available/not available during OpenMP or automatic parallel execution.

Compared with the check function of the compiler option -H, the check function of the compiler option -Nquickdbg has less effect on execution performance because the items checked are limited. Debugging is also enabled during OpenMP and automatic parallel execution.

The compiler option -Nquickdbg includes the compiler option -Nquickdbg=inf_detail and -Nquickdbg=inf_simple.

If the compiler option -Nquickdbg=inf_detail is enabled, in addition to the message and line number where the error occurred, information is output such as the variable name, procedure name, and element location in order to identify the cause of the error.

If the compiler option -Nquickdbg=inf_simple is enabled, the error and the line number where it occurred are displayed in the diagnostic message. However, if the compiler option -Nnoline is enabled, the value of line number is not guaranteed.

By limiting the information included in diagnostic messages, the compiler option -Nquickdbg=inf_simple has less impact on execution performance than the compiler option -Nquickdbg=inf_detail check function.

See Section "2.2 Compiler Options" for more information about -H, -Nquickdbg, and -Ncheck_global.

8.2.1.1 Checking Argument Validity (ARGCHK)

When the subprogram, a subroutine or function, is called, the compiler option -Ha or -Nquickdbg=argchk checks the validity of the dummy and actual arguments.

However, checks are not performed in the following cases:

- The actual argument is an actual name that has a POINTER attribute (including cases in which the actual argument is a derived type variable with one or more components that have a POINTER attribute).
- An actual argument is an array section or array expression (other than an array variable).
- A dummy procedure name is used to call a subprogram.
- A derived type actual argument with an ultimate component that is an allocatable variable.
- A procedure pointer or type bound procedure is referred.
- A dummy argument is an assumed type or assumed rank.

8.2.1.1.1 Checking Number of Arguments

The compiler option -Ha or -Nquickdbg=argchk checks whether or not the number of actual arguments matches the number of dummy arguments. The number of actual arguments also includes the alternate return specifier. If the numbers of arguments do not match, a diagnostic message jwe0313i-w is output.

8.2.1.1.2 Checking Argument Types

The compiler option `-Ha` or `-Nquickdbg=argchk` checks whether or not the dummy argument type matches the corresponding actual argument type. If the argument types do not match, a diagnostic message `jwe0315i-w` is output.

Argument types are not checked in the following cases:

- The dummy argument is a procedure name or an asterisk (*).
- The actual argument is an external procedure name or an alternate return specifier (but argument types are checked if the dummy argument and actual arguments are external procedure names that have types).

If the compiler option `-Ha` is set, the following checks are also performed

If the actual argument and dummy argument are derived type variables, the function checks whether or not the types of the components match. If the types do not match, a diagnostic message `jwe1561i-w` is output.

If the argument type is the character type, the function checks whether or not the length of the dummy argument exceeds that of the actual argument.

If the length is exceeded, a diagnostic message `jwe0317i-w` is output.

8.2.1.1.3 Checking Argument Attributes

The compiler option `-Ha` checks whether or not the dummy argument attributes matches the corresponding actual argument attributes. A dummy argument may be associated only with an actual argument as shown in "[Table 8.9 Correspondence for dummy and actual arguments](#)". If a dummy argument is not associated, the diagnostic message `jwe0314i-w` is output. If a dummy argument has `INTENT(IN)`, the actual argument must be defined.

If an actual argument is a constant or an expression that is not a variable, diagnostic message `jwe0318i-w` is output if the corresponding dummy argument is changed in the subprogram.

"[Table 8.9 Correspondence for dummy and actual arguments](#)" lists the correspondence between dummy and actual arguments.

Table 8.9 Correspondence for dummy and actual arguments

Dummy argument	Actual argument
Scalar variable	Scalar variable, substring, array element, constant, result of scalar expression
Array excluding character type	Actual argument excluding character type Whole array, array section, array element, substring of array element, result of array expression
Character type array	Actual argument of character type Scalar variable, substring, constant, result of scalar expression, whole array, array section, array element, substring of array element, result of array expression
Dummy procedure	External procedure, intrinsic procedure, or module procedure
*	Statement label

8.2.1.1.4 Checking Function Types

The compiler option `-Ha` or `-Nquickdbg=argchk` checks whether or not the type of function defined in a function subprogram matches the type of the reference source external procedure. If the function types do not match, a diagnostic message `jwe0311i-w` is generated if the function types do not match.

If a function subprogram is referenced as a subroutine subprogram, or if a subroutine subprogram is referenced as a function subprogram, a diagnostic message `jwe0330i-w` is output.

If the compiler option `-Ha` is set, the following checks are also performed.

If the function type is the character type, the function checks whether or not the lengths are the same. If the function lengths are not the same, a diagnostic message `jwe0312i-w` is output.

8.2.1.1.5 Checking Argument Sizes

For an array dummy argument, the compiler option `-Ha` checks whether or not the size of array dummy argument exceed the corresponding actual argument. If the upper bound of the final dimension of the dummy array is an asterisk (*), or the upper and lower bounds of the dummy array are 1, the size of the dummy array is assumed to be equal to the corresponding actual argument size. Then this check is not performed.

If type of assumed-shape array is character type, character lengths are checked to confirm that the length of the dummy argument and the actual argument match. If the length does not match a diagnostic message `jwe1571i-w` is output.

As listed in "Table 8.10 Array size", the size of the actual argument that corresponds to a dummy argument array is obtained from the actual argument attribute. If the size of the dummy array exceeds the actual argument size, a diagnostic message `jwe0316i-w` or `jwe1577i-w` is output.

Table 8.10 Array size

Actual argument attribute	Actual argument size
Array	Whole array size
Array element	Size from the specified array element up to the end of the array
Substring of array element	Size from the beginning of the specified array element substring to the end of the array
Scalar expression of character type excluding array element and substring of array element	Length for scalar expression of character type

8.2.1.1.6 Checking Explicit Interface

If an explicit reference specification is not present in an external procedure reference that requires an explicit reference specification when the compiler option `-Ha` or `-Nquickdbg=argchk` is set, a diagnostic message `jwe1569i-w` is output.

8.2.1.1.7 Checking Rank of Assumed-Shape Array

If a dummy argument is an assumed-shape array, when the compiler option `-Ha` is set, the rank is compare to actual argument and dummy argument. If the rank is different between actual argument and dummy argument, a diagnostic message `jwe1570i-w` is output.

8.2.1.2 Checking Subscript and Substring Values (SUBCHK)

If the compiler option `-Hs` or `-Nquickdbg=subchk` is set, the validity of the referenced subscript or substring expression is checked against the declared subscript and substring expression.

However, this check is not performed in the following cases:

- The referenced expression has the `POINTER` attribute (including cases in which a structure component has a `POINTER` attribute).
- An assumed-shape array is referenced.
- An array section specified by a vector subscript is referenced.
- An array in a masked array assignment is referenced.
- A named constant is referenced.
- The parent string of substring is a scalar constant.
- Data is referenced by a declaration expression.
- A derived type variable with an ultimate component that is an allocatable variable.
- The subscript range of `io-implicit-do` (but checked if the `-Hs` compiler option is set).
- A polymorphic variable is referenced.
- A variable with complex part designator is referenced.
- A variable of derived type having length type parameter is referenced.

This check is performed during compilation and execution. If an error is detected during compilation, a w level diagnostic message is output. The items checked during execution are explained below.

8.2.1.2.1 Checking Array and Substring References

If the compiler option `-Hs` or `-Nquickdbg=subchk` is set, when substring, array element, or array section is referenced, the array reference check determines whether or not those subscript and substring ranges are within the declared ranges. If it is not within the declared range, a diagnostic message (`-Hs`, `-Nquickdbg=inf_detail:jwe0320i-w`, `-Nquickdbg=inf_simple:jwe1606i-u`) is output.

If the upper and lower bounds of the dummy array are 1, the dummy array is regarded as assumed-size array. If the diagnostic message is output by the compiler option `-Nquickdbg=subchk`, the maximum number of 8 byte integer is output as the upper bound of the last dimension of assumed-size array.

If the compiler option `-Hs` is set, the following checks are also performed

If the upper limit of an explicit shape array is a smaller value than the lower limit, a diagnostic message `jwe1566i-w` is output.

If the compiler option `-Hs` and `-Ha` are set simultaneously, this check whether or not an array element or substring is within the corresponding actual argument range when it corresponds to a dummy array in which the upper limit of the last dimension is an asterisk (*) or in which the upper and lower limit is the constant 1. If it is not within the actual argument range, a diagnostic message (`jwe0322i-w`) is output.

8.2.1.2.2 Checking Array Section References

If the compiler option `-Hs` is set, the following check is performed.

If an array section with a subscript triplet is referenced, the result of adding the first subscript to the extent of the subscript triplet is checked to determine whether the value exceeds the maximum value of an integer or is less than the minimum value of an integer. In either case, diagnostic message `jwe1565i-w` is output.

8.2.1.3 Checking References for Undefined Data Items (UNDEF)

If the compiler option `-Hu`, `-Nquickdbg=undef`, or `-Nquickdbg=undefnan` is set, the function checks whether or not referenced data is defined.

However, this check is not performed in the following cases:

- An actual name that has a `POINTER` attribute is referenced (including cases in which a structure component has a `POINTER` attribute, but checked if the compiler option `-Nquickdbg=undef` or `-Nquickdbg=undefnan` is set).
- A structure component has an `ALLOCATABLE` attribute.
- An assumed-shape array is referenced (but checked if the compiler option `-Nquickdbg=undef` or `-Nquickdbg=undefnan` is set).
- An array section specified by a vector subscript is referenced.
- Data in a masked array assignment is referenced.
- A derived type variable that has an allocatable array to an ultimate component is referenced.
- A variable of derived type having length type parameter is referenced.
- A polymorphic variable is referenced.
- A variable with complex part designator is referenced.
- An array allocatable to an ultimate component by a structure component is referenced (but checked if the compiler option `-Nquickdbg=undef` or `-Nquickdbg=undefnan` is set).
- A structure component is referenced (but checked if the compiler option `-Hu` or `-Nquickdbg=undef` is set).
- The entities that share a storage sequence have different types (but checked if the compiler option `-Hu` or `-Nquickdbg=undef` is set).
- The subscript range of `io-implicit-do` (but checked if the compiler option `-Hu` is set).

If diagnostic messages `jwe0320i-w`, `jwe0322i-w`, `jwe1565i-w`, and `jwe1566i-w` are detected by the `SUBCHK` function checks when the compiler option `-Hu` and `-Hs` are set simultaneously, the `UNDEF` function is disabled.

Undefined data checks are explained below.

8.2.1.3.1 Checking for Undefined Data

If the compiler option `-Hu` or `-Nquickdbg=undef` is set, when module or submodule declaration part or variable not included in common block is referenced, the values is checked to be defined for that data. When array variable is referenced, check is performed for each array

element. If derived type variable is referenced when the compiler option `-Hu` is set, check is performed for each of the structure variable components.

If values is not defined, a diagnostic message (`-Hu,-Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u`) is output.

If a function subprogram ends when the compiler option `-Hu` is set, the values in the function results is checked. If it is not defined in the function results, a diagnostic message (`-Hu, -Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u`) is output. The line number at that time becomes the END statement of the function subprogram.

If the compiler option `-Nquickdbg=undef` is set, when a pointer variable appears in the PRIVATE clause of OpenMP is referenced, check is performed for the reference of the pointer variable. If the pointer variable is not associated, `jwe1572i-s` is output.

If data value has the values below, a diagnostic message (`-Hu, -Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u`) is output, but, as the Fortran program is correct, no correction is required. To change the data values below, specify the `FLIB_UNDEF_VALUE` environment variable. See Section "3.8 Using Environment Variables for Execution" for detail.

One-byte integer	: -117
Two-byte integer	: -29813
Four-byte integer	: -1953789045
Eight-byte integer	: -8391460049216894069
Half precision real	: -2.301931e-04
Single precision real	: -5.37508134e-32
Double precision real	: -4.696323204354320d-253
Quadruple precision real	: -9.0818487627532284154072898964213742q-4043
Half precision complex	: (-2.301931e-04,-2.301931e-04)
Single precision complex	: (-5.37508134e-32,-5.37508134e-32)
Double precision complex	: (-4.696323204354320d-253,-4.696323204354320d-253)
Quadruple precision complex	: (-9.0818487627532284154072898964213742q-4043, -90818487627532284154072898964213742q-4043)
Character	: Z'8B'

8.2.1.3.2 Checking for Undefined Data Using Invalid Operation Exception

If the compiler option `-Nquickdbg=undefnan` is set, when module or submodule declaration part or variable not included in common block is referenced, the value is checked to be defined for that data. When array variable is referenced, check is performed for each array element. If value is not defined, a diagnostic message (`-Hu,-Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u`) is output.

Output diagnostic message is different according to a variable and a type of function results.

If integer type and character type checks show that values are not defined, a diagnostic message (`-Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u`) is output.

If data has the values below, a diagnostic message (`-Hu, -Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u`) is output, but, as the Fortran program is correct, no correction is required. To change the data values below, specify the environment variable `FLIB_UNDEF_VALUE`. See Section "3.8 Using Environment Variables for Execution" for detail.

One-byte integer	: -117
Two-byte integer	: -29813
Four-byte integer	: -1953789045
Eight-byte integer	: -8391460049216894069
Character	: Z'8B'

The real type and complex type variable have SNaN as the initial value. If values are not defined, the real type and complex type checks detect an invalid operation exception (`jwe0292i-u`).

However, the invalid operation exception may not be detected in the following case:

- The real type and complex type variables do not appear in the operands of the command that issues invalid operation exceptions. (The CPU architecture determines which instruction issues invalid operation exception. For example, in an architecture where the reference instruction does not issue exception, undefined variable is detected during referencing when the compiler option `-Nquickdbg=undef` is set, but undefined variable is not detected during referencing when the compiler option `-Nquickdbg=undefnan` is set.)
- The invalid operation exception is masked.

In addition undefined data reference, the invalid operation exception is detected when the execution result is SNaN.

8.2.1.3.3 Checking for an Allocatable Variable

If the compiler option `-Hu` is set, the variable checked to already allocated by `ALLOCATE` statement when an `ALLOCATABLE` variable is referenced; if not, diagnostic message `jwe0324i-w` is generated.

8.2.1.3.4 Checking for an Optional Argument

If the compiler option `-Hu` is set, the argument checked to be present when an optional argument is referenced; if not, diagnostic message `jwe1575i-s` is generated.

8.2.1.4 Shape Conformance Check (SHAPECHK)

When an array expression or array assignment statement is executed, the compiler option `-He` checks shape conformance.

Diagnostic message `jwe0329i-w` is generated if a mismatch of shape conformance occurs during execution.

8.2.1.5 Extended Checking of UNDEF (EXTCHK)

The `-Hx` option checks to see if the variable declared in a module or submodule, or in a common block is defined when the variable is referenced, to see if the pointer is associated when a pointer is referenced, and to see if the incrementation parameter of the `DO` construct, stride of the `FORALL`, incrementation parameter of array constructor implied `do` control and stride of subscript-triplet is not 0.

Diagnostic message `jwe0323i-w` is generated for the variable declared in a module or submodule, or in a common block that is not defined. Note that the message is generated if the item is defined with a value listed in Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)", even though the Fortran program may be correct.

A Pointer is referenced, the pointer checked to be associated with target; if not, diagnostic message `jwe1572i-s` is generated. However, this check facility is not performed for a pointer of derived type and a pointer of component.

If the incrementation parameter of the `DO` construct, stride of the `FORALL`, incrementation parameter of array constructor implied `do` control and stride of subscript-triplet is zero, diagnostic message `jwe1573i-s` is generated.

In the range of a `DO` construct, if an actual argument is a `do`-variable, the definition of `do`-variable is checked. Diagnostic message `jwe1574i-s` is generated if the `do`-variable is defined.

The execution of a `RETURN` or an `END` statement within a subprogram causes all local variables in that scope unit without the `SAVE` attribute becomes undefined when the `-Hx` option is present.

When the variables have a `BIND` attribute, the undefined value is not checked.

If the `-Hx` option is specified, the `-Hu` option is in effect, see Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)".

Note the following when the `-Hx` option using:

- All program units that have a definition or initialization for common block object shall be compiled with `-Hx` option.
- All program units that uses the ancestor modules and descendant submodules shall be compiled with `-Hx` option, if the submodule is compiled with `-Hx` option.

Example: Invalid specification of `-Hx`

File: a.f90

```
BLOCK DATA INIT
COMMON /CMN/ J
DATA J/200/
END BLOCK DATA
```

File: b.f90

```
PROGRAM MAIN
COMMON /CMN/ J
PRINT *,J
END PROGRAM
```

Command:

```
$ frtpr a.f90 -c
$ frtpr b.f90 a.o -Hx
```

The -Hx options shall be specified for compiling a.f90.

- All program units that uses the module shall be compiled with -Hx option, if the module is compiled with -Hx option.

Example: Invalid specification of -Hx

File: a.f90

```
MODULE MOD
INTEGER :: J
END MODULE
```

File: b.f90

```
PROGRAM MAIN
USE MOD
J = 200
CALL SUB()
END PROGRAM
```

File : c.f90

```
SUBROUTINE SUB()
USE MOD
PRINT *,J
END SUBROUTINE
```

Command:

```
$ frtpr a.f90 -c -Hx
$ frtpr b.f90 -c
$ frtpr c.f90 a.o b.o -Hx
```

The -Hx options shall be specified for compiling b.f90.

8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)

When compiler option -Ho is specified, the following check is done.

- Two dummy arguments are overlapped and the part of overlap is changed.
- When an assumed-size array with INTENT(OUT) attribute is referenced during execution, checks to see if the variable is defined.
- When a variable with SAVE attribute is referenced during execution, checks to see if the variable is defined.

8.2.1.6.1 Overlapping Dummy Arguments

Do not define or undefine the value of the part overlapping executing the procedure when two dummy arguments are overlapping in the same procedure. When compiler option -Ho is specified, the function checks whether the value is defined or undefined while executing. Diagnostic message jwe1576i-w is generated if a mismatch occurs during execution. The validity is not checked if any of the following conditions is true.

The validity is not checked if:

- The dummy argument is defined in a masked array assignment.
- The dummy argument is defined in FORALL assignment statement.

- The dummy argument as DO variable is defined.
- The dummy argument is defined with array section of vector subscript.
- The dummy argument is defined and the corresponding actual argument has no continuation element.
- The dummy argument is defined in reference of procedure.
- When the dummy argument which has TARGET attribute or POINTER attribute is overlapping the object of the other dummy argument which has neither TARGET attribute nor POINTER attribute, and the dummy argument is defined.
- When the dummy argument of assumed shape array is overlapping the object of the other dummy argument, and the dummy argument is defined.

8.2.1.6.2 Undefined Assumed-Size Array with INTENT(OUT)

When compiler option -Ho is specified, the function checks whether assumed-size array with INTENT(OUT) attribute is defined while executing. Diagnostic message jwe0323i-w is generated if a mismatch occurs during execution. If compiler option -Hs also is specified, the function does not check the data if diagnostic message jwe0320i-w, jwe0322i-w, jwe1565i-w, or jwe1566i-w is generated.

The validity is not checked if:

- A pointer variable is referenced.
- The referenced expression is an array section with a vector subscript
- The referenced expression is in a masked array assignment.
- The referenced expression is a derived type variable with an ultimate component that is an allocatable variable.
- The referenced expression is an allocatable variable as an ultimate component by a structure component.

8.2.1.6.3 Undefined Variable of SAVE Attribute

When compiler option -Ho is specified, the function checks whether variable with SAVE attribute is defined while executing. Diagnostic message jwe0323i-w is generated if a mismatch occurs during execution. If compiler option -Hs also is specified, -Hu option does not check the data if diagnostic message jwe0320i-w, jwe0322i-w, jwe1565i-w, or jwe1566i-w is generated.

The validity is not checked if:

- The referenced expression has a structure variable one of whose structure components has the POINTER attribute.
- The referenced expression is an array section with a vector subscript.
- The referenced expression is in a masked array assignment.
- The referenced expression is a derived type variable with an ultimate component that is an allocatable variable.
- The referenced expression is an allocatable variable as an ultimate component by a structure component.

8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)

When compiler option -Hf is specified, the following check is done.

A file is connected with two devices or more at the same time in the input/output statement.

An input/output statement is called by the function while executing other input/output statement.

8.2.1.7.1 Checking Connection of a File to a Unit

In the input/output statement, do not connect the one file with two devices or more at the same time. When compiler option -Hf is specified, the function checks whether the one file is connected with two devices or more at the same time while executing. Diagnostic message jwe1231i-w is generated if a mismatch occurs during execution.

The validity is not checked if:

- The file is specified by the file names other than the real name when file system that the file is not local is used.
- As for this file, specification different from the real name is used when the file is defined and the alias is defined by the "alias" command.

- "../" is included in the path of the file.

8.2.1.7.2 I/O Recursive Call CHECK

Do not recurrently execute the input/output statement. When compiler option -Hf is specified, the function checks whether the input/output statement is executed recurrently while executing. Diagnostic message jwe1232i-s is generated if a mismatch occurs during execution.

8.2.2 Debugging Programs for Abend

If an abnormal termination event (hereafter called an abend) occurs during execution of a Fortran program, the system generates information to help the user determine the cause of the abend.

8.2.2.1 Causes of Abend

"Table 8.11 Detected errors and corresponding signal codes" lists the errors that the system detects and the corresponding signal codes.

The system enables traps corresponding to signals (*), and handles floating-point exceptions (jwe1017i-u). If the compiler option -NRnotrap is enabled, floating-point exceptions are not detected. See Section "2.2 Compiler Options" for more information about the compiler option -NRnotrap.

*: Only signals supported by the Fujitsu CPU A64FX with Arm architecture are targeted. Therefore, enabling the compiler option -NRtrap will not detect unsupported signals.

If the -i runtime option is specified, the system does not check for these errors.

Table 8.11 Detected errors and corresponding signal codes

Signal number	Meaning of signal number	Signal code		Meaning of signal code
SIGILL(04) *	Incorrect instruction executed	1	ILL_ILLOPC	Illegal opcode
		2	ILL_ILLOPN	Illegal operand
		3	ILL_ILLADR	Illegal addressing mode
		4	ILL_ILLTRP	Illegal trap
		5	ILL_PRVOPC	Illegal privileged opcode
		6	ILL_PRVREG	Illegal privileged register
		7	ILL_COPROC	Coprocessor error
		8	ILL_BADSTK	Internal stack error
SIGFPE(08)	Floating-point exception (*1)	3	FPE__FLTDIV	Floating-point divide zero
		4	FPE__FLTTOVF	Floating-point overflow
		5	FPE__FLTUND	Floating-point underflow (*2)
		7	FPE__FLTINV	Invalid floating-point operation
		-	-	Floating-point exception
SIGBUS(10) *	Storage protection exception	1	BUS_ADRALN	Invalid address alignment
		2	BUS_ADRERR	Non-existent physical address
		3	BUS_OBJERR	Object-specific hardware error
SIGSEGV(11) *	Segmentation exception	1	SEGV_MAPERR	Address not mapped to object
		2	SEGV_ACCERR	Invalid permissions for mapping object
SIGXCPU(30) *	CPU time interrupt	-	-	-

*1: Note that FPE_INTDIV (the integer divide by zero) is not detected in the Fujitsu CPU A64FX with Arm architecture. See Section "A.2.3 Integer Division Exception when Divisor Is Zero" for details.

*2: Interrupt detected when the -NRtrap compiler option and the environment variable FLIB_EXCEPT=u (exponent underflow) were specified.

Note: For signal numbers marked with an asterisk, a core file is created when the -i runtime option is specified to disable system error detection.

8.2.2.2 Information Generated when an Abend Occurs

Depending on the error type, information is written to the standard error file as explained in the following sections. Information that cannot be written to the standard error file is sent to the terminal.

8.2.2.2.1 Information Generated for a General Abend

The information generated for a general abend is follows. This information is generated when SIGILL, SIGBUS, or SIGSEGV is detected. A trace back map is printed after this information.

```
jwe0019i-u The program was terminated abnormally with signal number SSSSSSS.  
      signal identifier = NNNNNNNNNNN, (Detailed information.)
```

SSSSSSS: SIGILL, SIGBUS, or SIGSEGV

NNNNNNNNNN: Signal code for the abend cause

(Detailed information.): Detailed information of Signal code NNNNNNNNNNN

When it is strong possibility that the cause of SIGBUS or SIGSEGV is stack-overflow, add the following information.

```
The cause of this exception may be stack-overflow
```

8.2.2.2.2 Information of SIGXCPU

The output when SIGXCPU is detected is follows.

```
jwe0017i-u The program was terminated with signal number SIGXCPU.
```

8.2.2.2.3 Information Generated by RunTime Option -a

The information generated by runtime option -a when SIGIOT is detected is follows. The program is forced to abend without deleting the temporary files being used by the program and a core file is created.

The cause of this exception may be stack-overflow.

```
jwe0018i-u The program was terminated abnormally due to runtime option -a with signal number SIGIOT.
```

8.2.2.2.4 Information when an Abend Occurs Again during Abend Processing

The output when an abend occurs again during abend processing is follows.

```
jwe0020i-u An error was detected during an abnormal termination process.
```

8.2.3 Hook Function

This section describes the hook function which calls a user-defined subroutine by specified location in a Fortran program, or by regular time interval. Program operation can be checked via a user-defined subroutine to output trace information and specific variable values.

8.2.3.1 User-Defined Subroutine Format

The user-defined subroutine name is fixed (USER_DEFINED_PROC).

Define a user-defined subroutine in the format shown below.

Format:

```

SUBROUTINE USER_DEFINED_PROC(FLAG, NAME, LINE, THREAD)
INTEGER(KIND=4), INTENT(IN) :: FLAG, LINE, THREAD
CHARACTER(LEN=*) , INTENT(IN) :: NAME

```

Arguments:

- FLAG** : Indicate the calling source for the user-defined subroutine.
- = 0 : Program entry
 - = 1 : Program exit
 - = 2 : Procedure entry
 - = 3 : Procedure exit
 - = 4 : Parallel region (OpenMP or automatic parallelization) entry
 - = 5 : Parallel region (OpenMP or automatic parallelization) exit
 - = 6 : Regular time interval
 - = 7~99 : Reserved for system
 - = 100~ : Can be used by the user
- NAME** : Indicate the calling source procedure name.
NAME can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.
- LINE** : Indicate the calling source line number.
LINE can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.
- THREAD** : With thread parallelization using OpenMP or automatic parallelization, indicate the number of the thread that calls the user-defined subroutine.
THREAD can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.

8.2.3.2 Notes on Hook Function

The following notes apply when the hook function is used:

- The procedure name "USER_DEFINED_PROC" must not be used for any other purpose.
- Operation is not guaranteed if the user-defined subroutine is not defined.
- Values must not be set in the user-defined subroutine arguments.
- If the -Nnoline compile option is set, the LINE argument value is not guaranteed.
- In thread parallelized programs, the user-defined subroutine may be called from multiple threads.
- Operation is not guaranteed if the user-defined subroutine is compiled with the -CcI4I8 option set.
- Operation is not guaranteed if the user-defined subroutine is compiled with the -AU option set.
- In procedures that include an ENTRY statement, the procedure name set in the NAME argument may be the entry name of the ENTRY statement.
- The procedure exit becomes the END statement line number.
- The characteristics of a user-defined subroutine procedure cannot be changed.
- The entry and exit of user-defined subroutine except the calling from any location in the program must not be calling source for the user-defined subroutine.
- Operation is not guaranteed if the STOP/ERROR STOP statement or EXIT/SETRCDD service subroutine is executed directly or indirectly in the user-defined subroutine.
- The OpenMP specifications which are available in the user-definition subroutine are the following OpenMP service subroutines.
 - OMP_GET_ACTIVE_LEVEL
 - OMP_GET_ANCESTOR_THREAD_NUM

- OMP_GET_DYNAMIC
- OMP_GET_LEVEL
- OMP_GET_MAX_ACTIVE_LEVELS
- OMP_GET_MAX_THREADS
- OMP_GET_NESTED
- OMP_GET_NUM_PROCS
- OMP_GET_NUM_THREADS
- OMP_GET_PROC_BIND
- OMP_GET_SCHEDULE
- OMP_GET_THREAD_LIMIT
- OMP_GET_THREAD_NUM
- OMP_GET_WTICK
- OMP_GET_WTIME
- OMP_IN_FINAL
- OMP_IN_PARALLEL
- OMP_SET_DYNAMIC
- OMP_SET_MAX_ACTIVE_LEVELS
- OMP_SET_NESTED
- OMP_SET_NUM_THREADS
- OMP_SET_SCHEDULE

8.2.3.3 Calling a User-Defined Subroutine from a Specified Location

When -Nhook_func option is specified when the object program and executable program are created, user-defined subroutine can be called from the following locations:

- Program entry and exit
- Procedure entry and exit

If the -Kopenmp or -Kparallel option is enabled, user-defined subroutine can also be called from the following locations:

- Parallel region (OpenMP or automatic parallelization) entry and exit

The following internal procedure name is passed to NAME argument in OpenMP, automatic parallelization entry and exit.

- OpenMP "_OMP_IdNumber_"
- automatic parallelization "_OMP_IdNumber_"

When the -Nhook_func is set, operation is as follows for Fortran, and C linked programs:

- If user-defined subroutine/function is defined within the Fortran program and the C program respectively, the user-defined subroutine defined in the Fortran program is called from the Fortran program, and the user-defined function defined in the C program is called from the C program.
- If user-defined subroutine/function is defined within only the Fortran program or only the C program, the defined user-defined subroutine/function is called from both the Fortran program and the C program.

An example using the hook function with location specified is shown below.

User-defined subroutine program example:

File: hook.f95

```

WRITE(*,*) 'HELLO'
CALL SUB
END

SUBROUTINE SUB
WRITE(*,*) 'SUB'
END

SUBROUTINE USER_DEFINED_PROC(FLAG,NAME,LINE,THREAD)
INTEGER(KIND=4), INTENT(IN):: FLAG,LINE,THREAD
CHARACTER(LEN=*), INTENT(IN):: NAME
SELECT CASE(FLAG)
  CASE(0)
    WRITE(*,*) 'PROGRAM START'
  CASE(1)
    WRITE(*,*) 'PROGRAM END'
  CASE(2)
    WRITE(*,*) 'PROC START: ',NAME,' LINE: ',LINE
  CASE(3)
    WRITE(*,*) 'PROC END: ',NAME,' LINE: ',LINE
END SELECT
END

```

Executable program creation:

```
$ frt -Nhook_func hook.f95
```

Execution results:

```

$ ./a.out
PROGRAM START
HELLO
PROC START: SUB LINE: 5
SUB
PROC END: SUB LINE: 7
PROGRAM END
$

```

8.2.3.3.1 Notes on Calling from a Specified Location

The following notes apply when the `-Nhook_func` option is set:

- If a procedure with few computations is called repeatedly, calling a user-defined subroutine from a procedure entry/exit may affect execution performance.
- If a parallel region with few computations is executed repeatedly, calling a user-defined subroutine from a parallel region entry/exit may affect execution performance.
- User-defined subroutine is not called from procedures called from input-output lists.

8.2.3.4 Calling a User-Defined Function at Regular Time Interval

When the `-Nhook_time` option is set when an executable program is created, the user-defined function is called by regular user time interval.

The calling interval can be specified using the `FLIB_HOOK_TIME` environment variable. If it is not specified, the user-defined function is called once every minute. See Section "3.8 Using Environment Variables for Execution", for information about the `FLIB_HOOK_TIME` environment variable.

See the "C User's Guide" or the "C++ User's Guide", for information about the user-defined function.

8.2.3.4.1 Notes on Calling by Regular Time Interval

The following notes apply when the `-Nhook_time` option is set.

- The environment variable `FLIB_HOOK_TIME` specification must not be changed while program execution is in progress.

- Operation is not guaranteed if a value outside the range 0 to 2147483647 is specified in the environment variable FLIB_HOOK_TIME.
- If the calling interval is short, the user-defined function is called at the regular time intervals but execution performance may be affected.
- The user-defined function must be defined in C or C++ program.

This function calls the user-defined function from asynchronous signals. Therefore, the following restrictions apply to calling the user-defined function at regular time interval:

- Functions excluding async-signal-safe functions cannot be used in the user-defined function.
- When the -Nhook_time option is set, signal processing routines corresponding to SIGVTALRM must not be defined.
- When the -Nhook_time option is set, ALARM(3) service function must not be used.
- The profiler must not be used for the executable program created with the -Nhook_time option.

8.2.3.5 Calling from Any Location in the Program

A user-defined subroutine can be called from any location in a program by specifying arguments (FLAG:100-, NAME, LINE, THREAD).

Chapter 9 Optimization Functions

This chapter explains functions to use optimization functions effectively, as well as some points to note.

9.1 Overview of Optimization

Optimization aims to generate object modules (instructions and data areas) that allow the program to execute as quickly as possible.

Some optimization functions can significantly increase the size of the object program. Therefore, the amount of virtual storage space can also increase.

Therefore, it can be selected whether to use optimization function by options.

See Section "2.2 Compiler Options" for more information about optimizations parameters.

9.1.1 Standard Optimization

When an optimization level between level 1 (-O1 option) and level 3 (-O3 option) is specified, standard optimization is performed.

To determine whether each optimization was applied, specify the -Koptmsg=2 option.

9.1.1.1 Elimination of Common Expression

If two expressions that give equal calculation results (common expressions) are present, the result of the former expression can be used in the latter expression without calculation.

An example of elimination of common expressions is given below.

Example:

[Original source-code]

```
...
A=X*Y+C
...
B=X*Y+D
```

->

[Optimized pseudo-code]

```
T=X*Y
A=T+C
...
B=T+D
```

The X*Y portion of the expression on the right hand side of the assignment is a common expression. The code is changed so that the second calculation is eliminated and the result of the first calculation T is used instead. T is a variable generated by the compiler.

9.1.1.2 Movement of Invariant Expressions

Expressions with values that do not change within a loop (invariant expressions) are moved outside the loop.

An example of the movement of invariant expressions is given below.

Example:

[Original source-code]

```
DO I=1,N
  Y=A(J)*2
  X=X+Y*Z
ENDDO
```

->

[Optimized pseudo-code]

```
Y=A(J)*2
T=Y*Z
DO I=1,N
  X=X+T
ENDDO
```

The entire statement Y=A(J)*2 and the Y*Z portion of the expression are moved outside the loop. T is a variable generated by the compiler.

The objects that are usually moved by this optimization are statements or parts of statements that are always executed within the loop.

However, if the -Kpreex option is specified, invariant expressions in statements that are selectively executed within the loop depending on a decision statement are also moved.

To differentiate this optimization from the normal movement of invariant expressions, it is called advance evaluation of invariant expressions. This optimization can further reduce execution time.

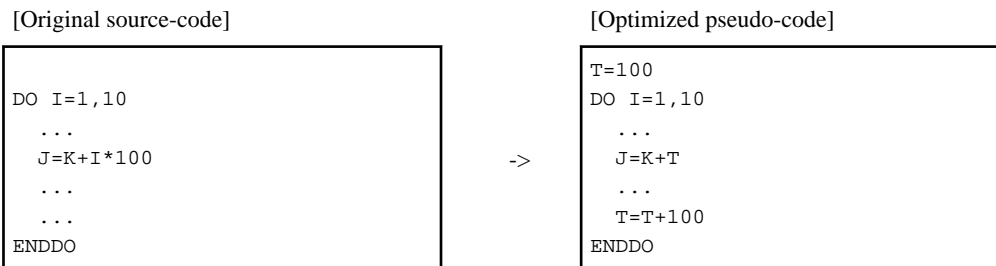
However, side effects may occur due to the movement. For details of the side effects, see Section "9.1.2.3 Optimization by Modifying Evaluation Methods".

9.1.1.3 Reducing Strength of Operators

The "strength" of operators describes the relative amount of execution time required for operation. A "strong" operator requires a large amount of execution time. Typically, addition and subtraction have the same strength. Multiplication is stronger than addition or subtraction and division is even stronger than multiplication. In addition, type conversion between the integer and float types is stronger than addition or subtraction but weaker than multiplication.

An example of reducing strength is given below.

Example:



Optimization is performed on the operation $I*100$ on the derivative variable I , so the operation $I*100$ within the loop is converted to the addition $T=T+100$. t is a variable generated by the compiler.

9.1.1.4 Loop Unrolling

Loop unrolling is a type of optimization in which all of the statements executed within a loop are expanded n times within the loop (where n is called the "multiplicity"), and instead, the number of iterations of the loop is reduced to $1/n$. However, if the `-Ksimd[=level]` option is set and the loop has been optimized using SIMD Extended instructions, the executable statement is unrolled "SIMD Width" $*n$, and the loop iteration count is reduced to "SIMD Width" $*n$.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

The multiplicity n is an optimal value determined by the compiler depending on the number of iterations of the loop, the number and type of statements executed within the loop and the data types used. In addition, the multiplicity can be specified in the source code with the optimization control line (OCL). For details on this function, see Section "9.10 Using Optimization Control line (OCL)".

Since the number of iterations is reduced and the executed statements that are expanded are optimized as a group within the loop, a faster object is obtained. However, since the statements executed within the loops are expanded, the size of the object module increases.

Loop unrolling works before and after SIMD.

- To apply SIMD or the software pipelining on the outer loop, the inner loop is fully unrolled before SIMD when the iteration count of the inner loop is small. This is called full unrolling before SIMD.
- The inner loop is unrolled or fully unrolled to promote the optimization such as common equations after SIMD.

Loop unrolling is performed in optimization level 2 (-O2 option) or higher.

Loop unrolling works before and after SIMD when `-Kunroll[=n]` option or the UNROLL specifier is the effective loop.

Loop unrolling can be suppressed by specifying `-Knounroll` option or the NOUNROLL specifier.

You can control the behavior of full unrolling before SIMD by specifying the FULLUNROLL_PRE_SIMD or NOFULLUNROLL_PRE_SIMD specifier. If the FULLUNROLL_PRE_SIMD or NOFULLUNROLL_PRE_SIMD specifier is specified for the loop that has the `-Kunroll[=n]` option, `-Knounroll` option, UNROLL specifier, or NOUNROLL specifier enabled, the FULLUNROLL_PRE_SIMD or NOFULLUNROLL_PRE_SIMD specifier takes precedence in full unrolling before SIMD.

To output line numbers and expansion counts of optimized DO loops, specify `-Koptmsg=2` option.

Statements that include array assignments (array expressions) are converted to DO loops, and will be subject to loop unrolling. However, the messages for -Koptmsg=2 are not output.

9.1.1.5 Loop Blocking

Loop blocking is the optimization to subdivide the access for array by multiplexing the loop in block size. As a result, the localization of data access is improved, and then efficient use of the cache is promoted.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Loop blocking is performed in optimization level 2 (-O2 option). It can be suppressed by specifying the -Kloop_noblocking option.

The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

An example of loop blocking is given below.

Example:

[Original source-code]

```
DO K=1,NK
  DO J=1,NJ
    DO I=1,NI
      A(I,J)=A(I,J)+B(I,K)*C(K,J)
    ENDDO
  ENDDO
ENDDO
```

[Optimized pseudo-code]

```
DO KK=1,NK,MK+1
  DO JJ=1,NJ,MJ+1
    DO II=1,NI,MI+1
      DO K=KK,MIN(NK,II+MK)
        DO J=JJ,MIN(NJ,JJ+MJ)
          DO I=II,MIN(NI,II+MI)
            A(I,J)=A(I,J)+B(I,K)*C(K,J)
          ENDDO
        ENDDO
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

->

9.1.1.6 Software Pipelining

Software pipelining schedules instructions in a loop to execute as parallel as possible.

Software pipelining can be suppressed by specifying the compile-time option -Knoswp.

Software pipelining performs an instruction scheduling to arrange instructions of a particular iteration in a loop with instructions of the following iterations, and reshapes the loop. Thus software pipelining requires enough loop iteration count. Other optimizations affect the required iteration count because it depends on instructions in the loop. If the original loop is short, software pipelining may be less effective because instruction scheduling is performed moderately to keep required iteration count small.

When software pipelining is applied to a loop whose iteration count is variable, as shown in the following example, the compiler inserts a branch instruction to choose the software-pipelined loop or the original loop in case the iteration count is short. It is decided at run time whether the software-pipelined loop is chosen. The size of the object module will increase.

Example:

[Original source-code]

```
SUBROUTINE SUB(N)
INTEGER(KIND=4) :: N
(The original loop: the iteration count is N.)
RETURN
END
```

[Optimized pseudo-code]

```
SUBROUTINE SUB(N)
INTEGER(KIND=4) :: N
IF (Is N enough?) THEN
  (The software-pipelined loop)
ELSE
  (The original loop: the iteration count is N.)
```

```

ENDIF
RETURN
END

```

If the compile-time option `-Koptmsg=2` is specified, optimization messages about results of software pipelining are output with the line numbers of loops. When software pipelining is applied to a loop, optimization messages `jwd8204o-i` and `jwd8205o-i` are output at once. When software pipelining is not applied to a loop, one of optimization messages from `jwd8662o-i` to `jwd8674o-i` is output. Optimization messages about software pipelining are not output when the `-Knoswp` option is specified, or when a loop disappears because of other optimizations before software pipelining, such as full unrolling.

The optimization message `jwd8205o-i` shows the minimum iteration count required to choose the software-pipelined loop. Make sure to consider following optimizations applied to the same loop because the target loop of software pipelining is the innermost loop; when loop collapse, loop interchange, thread parallelization, or inline expansion is applied, the iteration count of the target loop of software pipelining is changed as shown in the following table. If this is the case, the value shown in the optimization message `jwd8205o-i` should be compared with the iteration count in the table.

Optimization applied to the same loop	Message number	The iteration count of the target loop to which software pipelining is applied
Loop collapse	<code>jwd8330o-i</code>	Product of the iteration counts of the collapsed loops.
Loop interchange	<code>jwd8211o-i</code>	The iteration count of the innermost loop after loop interchange.
Thread parallelization	<code>jwd5001p-i</code> , etc.	Quotient of the iteration count divided by the number of threads.
Inline expansion	<code>jwd8101o-i</code>	The iteration count of the innermost loop after inline expansion.

As for the iteration count shown by optimization message `jwd8205o-i`, the value is output with taking account of the iteration counts executed at the same time by SIMD. Note that when the `-Ksimd_reg_size=agnostic` option is effective, the value is output as if the SVE vector register size is 128 bits is output, because the SVE vector register size is unknown at compilation.

9.1.1.7 SIMD

9.1.1.7.1 Normal SIMD

The compiler may apply optimizations using SIMD extensions. Using SIMD extensions, multiple operations of the same kind are executed at once.

The optimizations using SIMD extensions are suppressed by specifying compiler option `-Knosimd`. If compiler option `-Koptmsg=2` is specified, messages listing the line numbers where optimized loops are placed are output.

9.1.1.7.2 SIMD Extensions for Loop Containing IF Construct

When `-Ksimd={ 2 | auto }` option is specified, SIMD extensions can be applied to the loop that contains IF construct. The conditional execution instructions are only store instruction and move instruction in CPU architecture, and other instructions are executed speculatively (statements in the IF construct are executed even if the condition of the IF construct is false). Therefore, when the true rate of condition of the IF construct is high, the performance may be improved.

However, side effects, such as exceptions at execution, may occur in the execution results because instructions are executed speculatively.

9.1.1.7.3 List Vector Conversion

When the IF construct in loop contains many instructions and the true rate of the IF construct is low, the performance may be improved by applying list vector conversion.

The list vector conversion generates the following two loops:

1. The loop to save the value of loop control variable into a new array when the condition of IF construct is true.
2. The loop to execute the statement of the original IF construct, and its loop iteration count is equal to the number of the new array element.

In the case of 2, that loop becomes list access method, and becomes the target of SIMD extensions and software pipelining.

Example of list vector conversion is shown in the following. The list vector conversion is applied by specifying the optimization control line. See Section "9.10.4 Optimization Control Specifiers" for information on "!OCL SIMD_LISTV[({ALL | THEN | ELSE})]".

Example:

[Original source-code]

```
DO I=1, N
!OCL SIMD_LISTV
  IF (M(I)) THEN
    A(I)=B(I)+C(I)
  ENDIF
ENDDO
```

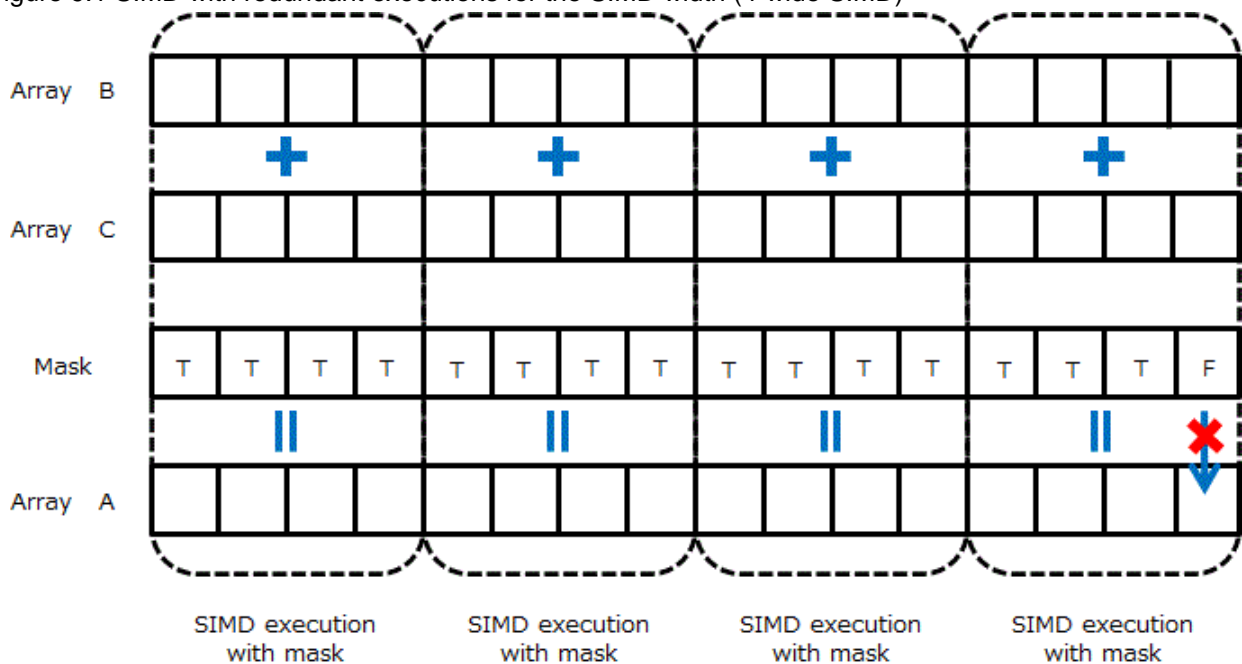
[Optimized pseudo-code]

```
J=1
DO I=1, N
  IF (M(I)) THEN
    IDX(J)=I
    J=J+1
  ENDIF
ENDDO
!OCL NORECURRENCE
DO K=1, J-1
  I=IDX(K)
  A(I)=B(I)+C(I)
ENDDO
```

9.1.1.7.4 SIMD with Redundant Executions for the SIMD Width

When the loop iteration count is not a multiple of the SIMD width, this function generates loops which use SIMD extensions over the whole iterations by using SIMD instructions with masks. (Refer to the example and figures below.) When SIMD is performed, this function is applied by default. The following figure is the image of this function.

Figure 9.1 SIMD with redundant executions for the SIMD width (4-wide SIMD)



[Explanation]

The whole iterations are executed by using SIMD instructions with masks.

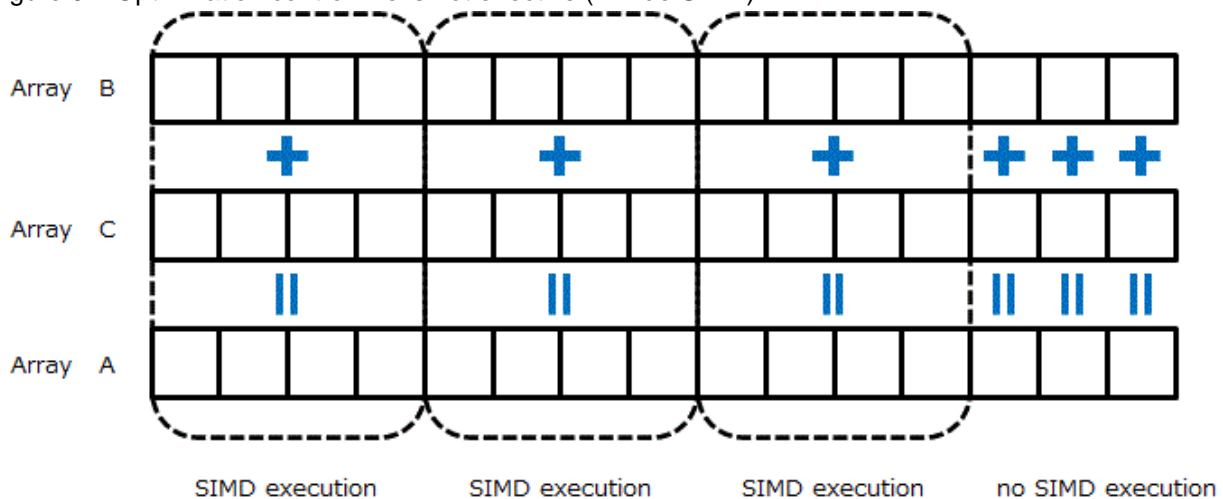
The loops whose remainder dividing the loop iteration count by the SIMD width tend to be 1 or 2 may not obtain the effect of optimization. In order to suppress this function, specify the `SIMD_NOREDUNDANT_VL` specifier in the optimization control line.

When the function is not applied to the source program in the example, SIMD instructions are used as shown in "Figure 9.2 Optimization control line is not effective (4-wide SIMD)". The remaining iterations, which correspond the remainder after dividing the loop iteration count by the SIMD width, do not use SIMD instructions.

Example: Optimization control line (OCL) is specified

```
!OCL SIMD_NOREDUNDANT_VL
DO I=1,M
  A(I)=B(I)+C(I)
END DO
```

Figure 9.2 Optimization control line is not effective (4-wide SIMD)



[Explanation]

When the loop control variable I is from 13 to 15, SIMD instructions are not used.

This function can be suppressed by specifying `-Kextract_stride_store` option.

9.1.1.7.5 Intrinsic Functions that SIMD Extensions can be Applied to

The following shows the intrinsic functions that SIMD extensions can be applied to.

Numeric Functions

`int`, `int4`, `jfix`, `int1`, `int2`, `real`, `double`, `cmplx`, `dcmplx`, `aint`, `anint`, `nint`, `i2nint`, `abs`, `mod`, `sign`, `dim`, `dprod`, `max`, `min`, `aimag`, `conjg`, `ceiling`, `floor`, `merge`, `logical`, `is_iostat_end`, `is_iostat_eor`

Mathematical Intrinsic Functions

`sqrt`, `cbprt`, `exp`, `exp10`, `exp2`, `log`, `log10`, `log2`, `sin`, `sind`, `sinq`, `cos`, `cosd`, `cosq`, `tan`, `tand`, `tanq`, `cotan`, `cotand`, `cotanq`, `asin`, `asind`, `asinq`, `acos`, `acosd`, `acosq`, `atan`, `atand`, `atanq`, `atan2`, `atan2d`, `atan2q`, `sinh`, `cosh`, `tanh`, `asinh`, `acosh`, `atanh`, `erf`, `erfc`, `gamma`, `log_gamma`, `lgamma`, `hypot`

Bit Manipulation Intrinsic Functions

`not`, `iand`, `and`, `ior`, `or`, `ieor`, `xor`, `ishft`, `ishftc`, `lshift`, `shiftr`, `rshiftr`, `shiftr`, `lshiftr`, `shiftr`, `ishl`, `ibset`, `ibclr`, `btest`, `izext`, `izext2`, `jzext`, `jzext2`, `jzext4`, `ble`, `blt`, `bge`, `bgt`, `poppar`

SIMD extensions may not be applicable for some functions or some parameter types.

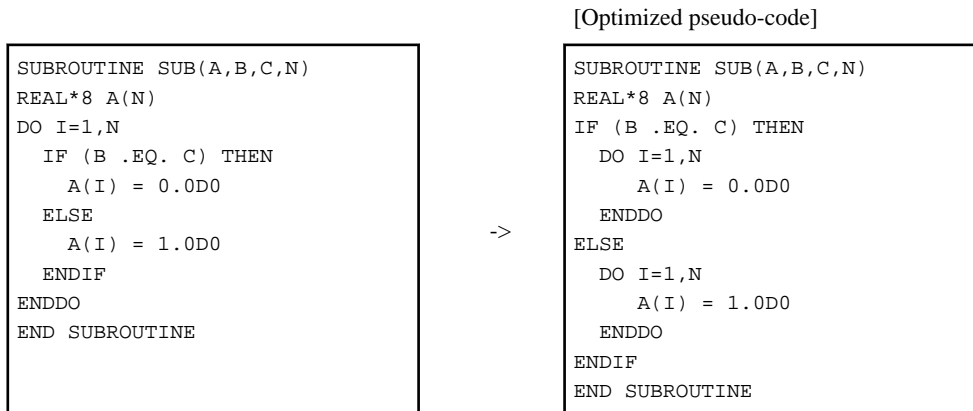
See also the description of the `-Kilfunc` and `-Kmfnc` options in the section "2.2 Compiler Options" for information on intrinsic functions that SIMD extensions can be applied to.

9.1.1.8 Loop Unswitching

Loop unswitching is an optimization to modify loop by putting out the IF construct out of the loop and creating a loop in each "TRUE" and "FALSE" blocks of the IF construct in order to promote some optimizations such as instruction scheduling if the condition of the IF construct is invariant in a loop.

The loop unswitching is enabled by specifying the -O3 or -Kparallel option, and is disabled by specifying the -O2 option or less, and invalidating the -Kparallel option.

The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.



9.1.2 Extended Optimization

When one of the extended function options is specified regardless of optimization, extended optimization is performed in addition to the standard optimization.

This section describes the functions comprising extended optimization.

Depending on the extended optimization function, the size of the object modules (instructions and data areas) generated by the compiler may be greatly enlarged or side effects may occur in the execution results.

9.1.2.1 Inline Expansion

Applying inline expansion, user-defined procedures are inlined to referenced points. This function is enabled by specifying *-xinl_arg*. For details about the *-xinl_arg* option, see Section "2.2.3 Description of Compiler Options".

The following expression shows how much size of object modules in program units increases when inline expansion is performed on user-defined procedures.

$$(\text{Size of instructions} + \text{Size of data sections}) * \text{Expansion count}$$

The size of instructions is determined by the kind and number of executable statements.

The size of data sections is the total size of variables, arrays, and constants, excluding common blocks, dummy arguments, allocatable variables, and elements with host associations.

Even if a user-defined procedure is inlined to a program unit more than once, the size of data sections increases only once.

Even if a user-defined external or module procedure is inlined to all referenced points, the procedure is not deleted completely; output into the object file. This is because the procedure may be referenced from other program units. Therefore, internal procedures which are inlined to all referenced points are deleted completely; not output into object file.

If compile-time option -Koptmsg=2 is specified, messages are output listing the expanded places and procedures.

9.1.2.2 Advance evaluation of Invariant Expressions

When the compiler option -Kpreex is specified, the optimizations which move invariant expressions evaluated in advance are applied. As a result, execution may abort because an instruction which should not be executed based on the logic of the program is executed.

However, this will not affect the precision of calculation results.

The effects of moving the execution location may appear upon referencing an intrinsic function or array element, and upon dividing.

The following example shows the effect of moving the execution location:

Example1: Using intrinsic functions

<pre> DIMENSION A(100) ... DO I=1,100 IF(X.GT.0.0) THEN A(I)=ALOG(X) END IF ENDDO ... </pre>	->	<pre> DIMENSION A(100) ... T=ALOG(X) DO I=1,100 IF(X.GT.0.0) THEN A(I)=T END IF ENDDO ... </pre>
-------------------------------------------------------------------------------------------------------	----	----------------------------------------------------------------------------------------------------------

[Description]

In the program on the left, there are no side effects because the function "alog" is called while the argument is greater than 0.0. In the program on the right, optimization (advance evaluation of invariant expressions) moves the function "alog(x)" out of the loop, and it is therefore always executed. Consequently, an error occurs when 'x' is less or equal than 0.0, and error message indicating that the argument of "alog" is incorrect is output.

Example2: Using array elements

<pre> SUBROUTINE SUB(A,B,N) DIMENSION A(N),B(N) DO I=1,N IF(J.LE.N) THEN A(I)=B(J)*F END IF ENDDO ... </pre>	->	<pre> SUBROUTINE SUB(A,B,N) DIMENSION A(N),B(N) T=B(J)*F DO I=1,N IF(J.LE.N) THEN A(I)=T END IF ENDDO ... </pre>
-----------------------------------------------------------------------------------------------------------------------	----	---------------------------------------------------------------------------------------------------------------------------

[Description]

In the program on the left, the reference to array "b" will not exceed the boundary, because array element "b(j)" is referenced only when "j" is less than "n". In the program on the right, however, optimization (advance evaluation of invariant expressions) has moved the array "b(j)" out of the loop, and it is therefore always referenced regardless the value of "j". Consequently, "b(j)" may be outside of the array boundary. If an array element outside the program boundary is referenced, the program will be aborted with an error message indicating that a memory protection exception (read) has occurred.

Example3: Division

<pre> SUBROUTINE SUB(A,N,F) DIMENSION A(N) DO I=1,N IF(F.NE.0.0) THEN A(I)=B/F ELSE A(I)=0.0 END IF ENDDO ... </pre>	->	<pre> SUBROUTINE SUB(A,N,F) DIMENSION A(N) T=B/F DO I=1,N IF(F.NE.0.0) THEN A(I)=T ELSE A(I)=0.0 END IF ENDDO ... </pre>
-------------------------------------------------------------------------------------------------------------------------------------	----	-----------------------------------------------------------------------------------------------------------------------------------------

[Description]

In the program on the left, division is performed only when the variable "f" is not 0.0, so a division exception will not occur. In the program on the right, however, optimization (advance evaluation of invariant expressions) has moved the division "b/f" out of the loop,

and therefore it is always executed regardless the value of "f". This means that the program may be aborted with a division exception error, if the division is performed while the value of "f" is 0.0.

Specifying -NRtrap option at the same time may cause exceptions that would not normally occur.

9.1.2.3 Optimization by Modifying Evaluation Methods

If the compiler option -Keval is specified, the optimization is performed that modifies how operations are evaluated for object programs, which can cause side effects in the calculation results. Operation exceptions, such as exponential overflow, may also occur. Changing division to multiplication is performed on division that uses floating point type variables.

The following example shows the effects of changing division to multiplication:

Example: Changing division-to-Multiplication operator

<pre>DIMENSION A(100),B(100) ... DO I=1,100 A(I)=B(I)/C ENDDO ...</pre>	->	<pre>DIMENSION A(100),B(100) ... T=1.0/C DO I=1,100 A(I)=B(I)*T ENDDO ...</pre>
----------------------------------------------------------------------------	----	------------------------------------------------------------------------------------

[Description]

The variable "c" is not changed in the loop. With optimization (changing division to multiplication), the reciprocal of variable "c" is calculated outside the loop, and the division within the loop is changed into a multiplication of the reciprocal. In other words, a single division operation is calculated from a division and a multiplication. If the result of "t=1.0/c" overflows the precision of the type of variable "c", the lower digit is rounded off, introducing a different result to the one obtained when optimization is not performed. Further, depending on the value of the variable "c", an exponential overflow exception may occur when calculating the reciprocal outside the loop.

9.1.2.4 Loop Striping

Loop striping superimposes an iteration of a loop on next several iterations of the loop. (The number of superimposed loop is called 'striped length'.) It can reduce overheads to iterate the loop, and it can promote software pipelining.

Example:

```
DO I=1,N
  A(I) = B(I) + C(I)
ENDDO
```

If striped length is equal to 4, the -Kstriping=4 option is specified, instructions within a loop are expanded as shown in the following.

Example:

```
DO I=1,N,4
  TMP_B1 = B(I)
  TMP_B2 = B(I+1)
  TMP_B3 = B(I+2)
  TMP_B4 = B(I+3)
  TMP_C1 = C(I)
  TMP_C2 = C(I+1)
  TMP_C3 = C(I+2)
  TMP_C4 = C(I+3)
  TMP_A1 = TMP_B1 + TMP_C1
  TMP_A2 = TMP_B2 + TMP_C2
  TMP_A3 = TMP_B3 + TMP_C3
  TMP_A4 = TMP_B4 + TMP_C4
  A(I) = TMP_A1
  A(I+1) = TMP_A2
  A(I+2) = TMP_A3
```

```
A(I+3) = TMP_A4
ENDDO
```

Loop striping unrolls statements in a loop as with loop unrolling, so the sizes of object modules increase. It also increases compilation time and memory requirement.

Note that performance may deteriorate by applying this optimization because the number of used registers increases.

9.1.2.5 zfill

The zfill optimization speeds up write operations for array data by using an instruction that allocates space on the cache for writing (DC ZVA) without loading data from the memory. The zfill is applied to array data to be written in a loop.

However, the zfill is not applied to array data which is referred in the same loop, accessed non-sequentially, or stored in IF constructs. Further, if the zfill optimization is applied, prefetch instructions to the second level cache are not generated.

The zfill optimization works on the data N blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and N is an integer value between 1 and 100. If a value is not specified for N , the compiler will automatically determine a value.

The following optimizations are suppressed because the loop is modified that data is always stored to the cache lines allocated by the zfill optimization:

- Loop unrolling
- Loop striping

Performance may also be reduced under the following conditions:

- The program is not affected by memory bandwidth bottleneck.
- When the loop iteration count is too few.
- When the block size is explicitly specified with the `-Kzfill= N` option and the memory size where the data is written in the loop is smaller than N blocks.

The zfill optimization is suppressed by specifying the `-Knozfill` option. Do not specify the `-Kzfill` option if the execution performance is likely to decrease as a result. It is usually desirable to control zfill at the loop unit. It is therefore recommended to specify the ZFILL optimization control specifier rather than specifying an option that has an effect on the entire program.

9.1.2.6 Loop Versioning

The loop versioning generates two loops to which optimization is applied and is not applied. The object program created by the compiler judges the data dependency of array at execution time and selects either loop.

The optimization, such as SIMD, software pipelining, automatic parallelization, is promoted by loop versioning.

The size of the object module and compile time may increase because the loop versioning generates two loops. The execution performance may decrease because the processing for judgement is overhead.

The loop versioning is applied only to the innermost loop.

Moreover, the message to show where optimized by the loop versioning can be output by specifying the `-Koptmsg=2` option.

The following example shows the loop versioning.

Example:

```
DO I=1,N
  A(I) = A(I+M) + B(I)
ENDDO
```

[Optimized pseudo-code]

```
IF ((1 .GT. N+M) .OR. (N .LT. 1+M)) THEN
!OCL NORECURRENCE
  DO I=1,N
    A(I) = A(I+M) + B(I)
  ENDDO
ELSE
```

```

DO I=1,N
  A(I) = A(I+M) + B(I)
ENDDO
ENDIF

```

When the `-Kloop_versioning` option is specified, the compiler generates two loops with IF-constructs for judging the values of variable "N" and "M" so that the data dependency of the array "A" can be analyzed at execution time.

If the array "A" judges no data dependency, SIMD may be promoted for the loop.

9.1.2.7 CLONE Optimization

CLONE optimization generates conditional branches of variables for loops in order to promote other optimizations, such as full unrolling. CLONE optimization is applied by specifying the optimization control line. The calculation result is not guaranteed if the optimization control line CLONE is used incorrectly. See Section "9.10.4 Optimization Control Specifiers" and "9.18 Incorrectly Specified Optimization Indicators" for more information about CLONE.

Example:

```

!OCL CLONE(N==10)
DO I=1,N
  A(I) = I
ENDDO

```

[Optimized pseudo-code]

```

IF (N==10) THEN
  DO I=1,10
    A(I) = I
  ENDDO
ELSE
  DO I=1,N
    A(I) = I
  ENDDO
ENDIF

```

9.1.2.8 Link Time Optimization

The following describes link time optimization.

9.1.2.8.1 Overview of Link Time Optimization

If the program consists of two or more source files, link time optimization is performed between the files at linking. Link time optimization can be controlled with the `-Klto` option. To apply link time optimization, `-Klto` option should be specified at both compilation and linking. Note that link time optimization is not applied when not `frtpx` command but `ld` command is used at linking.

Example:

- When the compilation and the linking are specified at a time:

```

$ frtpx -Kfast -Klto sample1.f90 sample2.f90 sample3.f90

```

- When the compilation and the linking are specified at different times:

```

$ frtpx -Kfast -Klto -c sample1.f90
$ frtpx -Kfast -Klto -c sample2.f90
$ frtpx -Kfast -Klto -c sample3.f90
$ frtpx -Kfast -Klto sample1.o sample2.o sample3.o

```

Link time optimization is performed for the object files which are compiled with the `-Klto` option. And, only files with an ".o" extension are subject to link time optimization. Therefore, the library such as `.so` or `.a` is excluded from link time optimization.

The size of the compiled object files with the `-Klto` option is increased by the additional information for optimization. However, the additional information for optimization is not included in the executable.

When the `-Klto` option is specified, the compiler outputs an intermediate file to the temporary directory by both of the compilation and the linking. This file is deleted at the end of compilation. The default temporary directory is `/tmp`. The temporary directory can be changed by setting the environment variable `TMPDIR`.

Link time optimization for Fortran cannot be applied to mixed language programming.

9.1.2.8.2 Notes on Link Time Optimization

The following notes apply when the link time optimization is used

- Link time optimization is a function which optimizes the program again before linking by using the information added to the object files. Therefore, the memory or time which is needed to generate the executable may increase significantly.
- The following functions may be affected when link time optimization is applied.

- Information collection function at compilation

The optimization which is actually applied may be different from the following information at compilation:

- Compilation information which is output by the compiler option `-Nlst=p` or `-Nlst=t` (Optimization information)
- Optimization message which is displayed by the compiler option `-Koptmsg`.

See Section "[4.1 Compilation Output](#)" for compilation output of Fortran program.

- Program checks

The runtime check to the following Fortran program may not be executed:

- Validity of arguments and result of procedure checks (`-Nquickdbg=argchk`, `-Ha`)
- Subscript range checks (`-Nquickdbg=subchk`, `-Hs`)
- Undefined data reference checks (`-Nquickdbg=undef`, `-Nquickdbg=undefnan`, `-Hu`)

See Section "[8.2.1 Debugging Check Functions](#)" for debugging function of Fortran program.

- Profiler

The following information which the Profiler outputs may not be guaranteed:

- Call graph information.

The function names in call graph information may include the function name which is generated internally by link time optimization.

- Source code information.

The each line cost may not be displayed correctly.

See the "[Profiler User's Guide](#)" for the Profiler.

- Runtime information output function

The line number may not match the line number of the source program in information which runtime information output function outputs.

See the "[Appendix H Runtime Information Output Function](#)" for runtime information.

- Trace back map

The line number of the statement where error occurred may not be correct.

See Section "[4.2.2 Trace Back Map](#)" for trace back map.

9.1.2.9 Unroll-and-Jam

Unroll-and-jam is the optimization to unroll an outer loop of a nested loop n times and jam the unrolled statements into the inner loop as loop fusion. Unroll-and-jam promotes removing common expression and improves the execution performance. The increase of data stream and change of the order of data access may decrease the cache efficiency and the execution performance. This optimization is not applied to innermost loop.

Do not specify the `-Kunroll_and_jam[=M]` option if the execution performance is likely to decrease as a result. It is usually desirable to control unroll-and-jam at the loop unit. It is therefore recommended to specify the `UNROLL_AND_JAM` or `UNROLL_AND_JAM_FORCE` optimization control specifier rather than specifying an option that has an effect on the entire program. See Section "9.10.4 Optimization Control Specifiers" about `UNROLL_AND_JAM` and `UNROLL_AND_JAM_FORCE`.

To output line numbers and expansion counts of optimized loops, specify `-Koptmsg=2` option.

Example:

<p>[Original source-code]</p> <pre>!OCL UNROLL_AND_JAM_FORCE(2) DO J=1, 128 DO I=1, 128 A(I, J) = B(I, J) + B(I, J+1) ... END DO END DO</pre>	->	<p>[Optimized pseudo-code]</p> <pre>DO J=1, 128, 2 DO I=1, 128 A(I, J) = B(I, J) + B(I, J+1) A(I, J+1) = B(I, J+1) + B(I, J+2) ... END DO END DO</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------	----	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.1.2.10 Tree-Height-Reduction Optimization

The tree-height-reduction optimization increases the instruction-level parallelism by changing the order of calculations in a loop to keep the calculation tree as short as possible.

The calculation tree is an expression of arithmetic operations in a form of tree structure in the order of operations. Leaf nodes of the tree structure hold values and other nodes hold operators. Nodes whose operator has higher priority are placed lower in the following example.

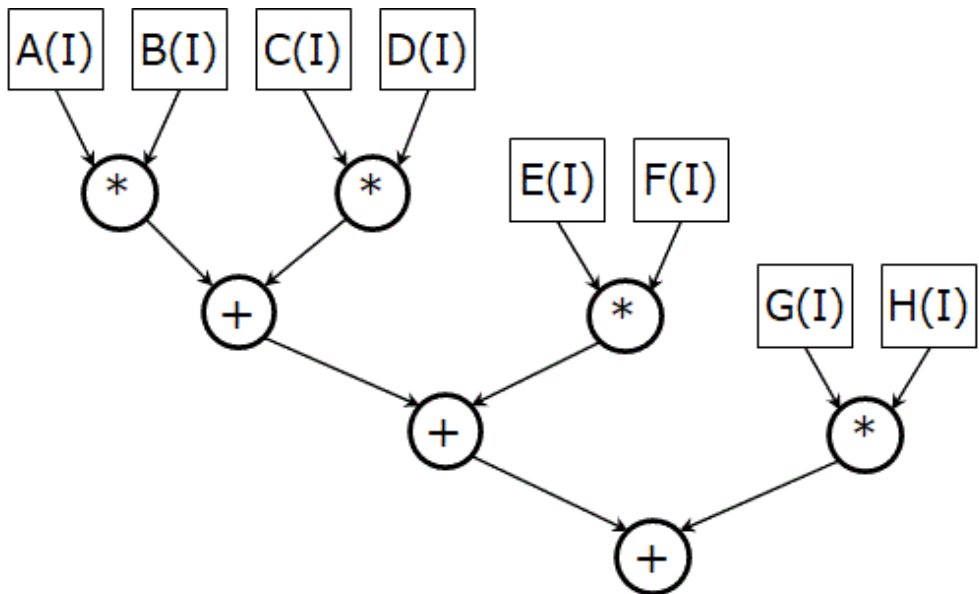
This optimization is applied to floating-point operations when the `-Keval_concurrent` option is set. In this case, side effects (calculation errors) may occur in the execution results. See "9.20 Side Effect of Optimizations for Floating-Point Operation" for the side effects of optimizations.

Examples of optimization is shown below.

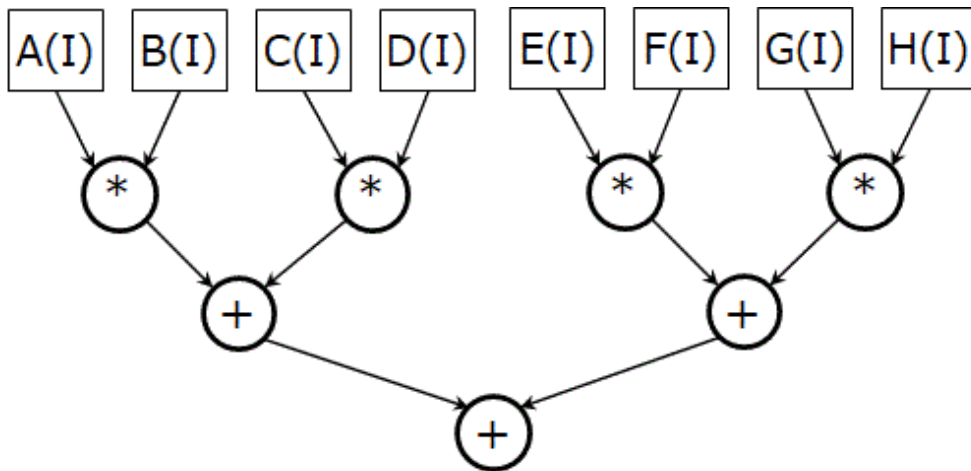
Example:

```
DO I=1,N
  X(i) = A(i) * B(i) + C(i) * D(i) + E(i) * F(i) + G(i) * H(i)
ENDDO
```

- The calculation tree when `-Keval_noconcurrent` option or optimization control specifier `eval_noconcurrent` is set :



- The calculation tree when -Keval_concurrent option or optimization control specifier eval_concurrent is set :



9.1.2.11 Loop Fission

Loop fission is a function to fission a loop into two or more small loops for aiming at the following objectives:

- Promotion of the software pipelining
- Improvement of the cache memory efficiency
- Resolution of register shortage

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Loop fission is performed on loops when the -Kloop_fission and -Kocl option are set and the optimization control specifier LOOP_FISSION_TARGET is specified. It can be suppressed by specifying the -Kloop_nofission option. The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

An example of loop fission is given below.

Example:

[Original source-code]

```

!OCL LOOP_FISSION_TARGET
DO I=1,N
  E(I) = A1(I)*B1(I) + A2(I)*B2(I) + A3(I)*B3(I) + A4(I)*B4(I) + A5(I)*B5(I)
  F(I) = C1(I)*D1(I) + C2(I)*D2(I) + C3(I)*D3(I) + C4(I)*D4(I) + C5(I)*D5(I)
ENDDO
  
```

[Optimized pseudo-code]

```

DO I=1,N
  E(I) = A1(I)*B1(I) + A2(I)*B2(I) + A3(I)*B3(I) + A4(I)*B4(I) + A5(I)*B5(I)
ENDDO

DO I=1,N
  F(I) = C1(I)*D1(I) + C2(I)*D2(I) + C3(I)*D3(I) + C4(I)*D4(I) + C5(I)*D5(I)
ENDDO
  
```

9.1.2.11.1 Strip-Mining

Strip-mining is an optimization to fragment the loop by small iteration. Applied by this optimization for loops after automatic loop fission, improvement of the cache memory efficiency is expected for the data accessed between fissioned loops.

Strip-mining is performed when the -Kloop_fission_stripmining option is specified. The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

Examples of automatic loop fission and strip-mining with specifying the -Kloop_fission_stripmining=256 are shown as follows.

Example:

[Source program]

```
REAL A(N),B(N),P(N),Q
!OCL LOOP_FISSION_TARGET
DO I=1,N
  Q = A(I) + B(I)
  ...
  P(I) = P(I) + Q
  ...
ENDDO
```

[Automatic loop fission pseudo-code]

```
REAL A(N),B(N),P(N),Q
REAL TEMPARRAY_Q(N)
DO I=1,N
  TEMPARRAY_Q(I) = A(I) + B(I)
  ...
ENDDO
DO I=1,N
  P(I) = P(I) + TEMPARRAY_Q(I)
  ...
ENDDO
```

The temporary array TEMPARRAY_Q is generated by the compiler with loop fission. The array is required to hand over data between the fissioned loops in the source code. The number of array elements is the same value as the number of loop iteration.

[Automatic loop fission and strip-mining pseudo-code]

```
REAL A(N),B(N),P(N),Q
REAL TEMPARRAY_Q(256)
DO II=1,N,256
  DO I=II,MIN(N,II+255)
    TEMPARRAY_Q(I) = A(I) + B(I)
    ...
  ENDDO
  DO I=II,MIN(N,II+255)
    P(I) = P(I) + TEMPARRAY_Q(I)
    ...
  ENDDO
ENDDO
```

A loop is generated outside of the fissioned loop. The original loop iteration is fragmented by the strip length 256. The number of elements of temporary array TEMPARRAY_Q is 256.

9.2 Notes of Wrong Erroneous Program

When optimization is performed on the following erroneous programs, execution results may differ with the optimization level. If results differ, check the following and correct the programs.

- Variables containing undefined values are referenced
- Variables containing values with unacceptable formats are referenced
- Array elements are referenced using index values that exceed array declarations
- There are violations of Fortran syntax

9.3 Effects of Environmental Change on Optimization

As optimization functions operate based on the information from the source program that is converted in the compiler, differences in the version of the compiler, the application of patches, and changes in the system environment may yield different results, even if the same source program and compiler options are being used.

This may cause the following symptoms:

- Information in compile messages and compile log files regarding optimization and automatic parallelization show different compilation results to previous ones.
- The results have some difference within the margin of error.

To know the difference of optimization results, compare compile messages or compile log file to before.

9.4 Effects of Changing Execution Order

The locations or numbers of the interruption on debugging may differ, because the execution order may be changed by removing a common expression from a loop or moving an invariant expression.

Note that the interruption caused by calculating constant values will occur only at runtime, because the calculations are not performed at compilation.

9.5 Branching Target of Assigned GO TO Statements

The label-lists of assigned GO TO statements must be accurate.

Assigned GO TO statements must not branch to a label that is not in the label-list.

Optimization assumes that each branch is directed to one of the labels in the label-list. Therefore, if there is branching to a label that is not in the list, unexpected results may occur.

If the label-list is omitted, the compiler analyzes the program and creates all possible statement labels as branch targets. Therefore, explicitly defining branching labels will result in better optimization.

9.6 Effect of Dummy Arguments

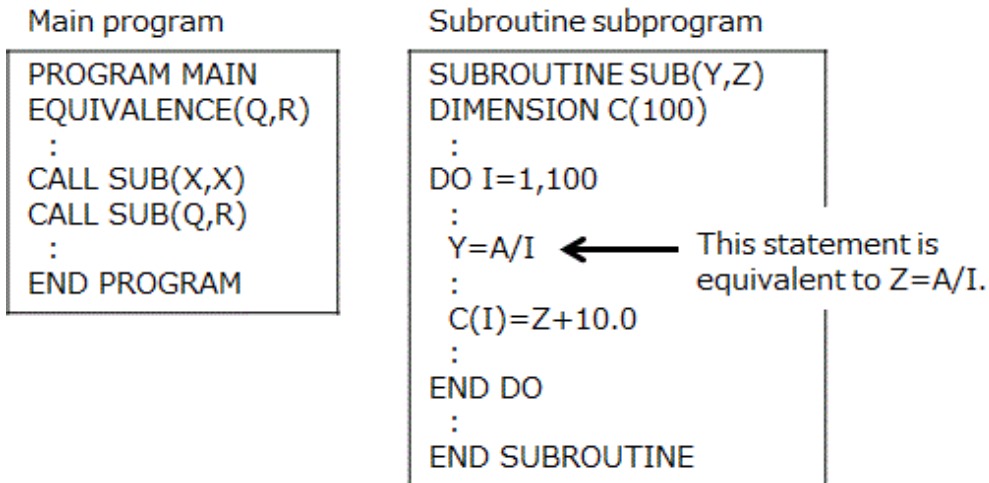
Note that using incorrect syntax for dummy arguments will cause unexpected results.

Associate with an actual argument

Each dummy argument in a procedure is optimized as a separate variable. Therefore, updating a dummy variable may also update an actual argument and introduce an incorrect result. Even if the variables have different actual arguments, the result may be incorrect if the memory is shared. According to the syntax, it is prohibited to redefine a value for a dummy argument both directly and indirectly in a subprogram.

The following example shows the effects of associating a dummy argument with an actual argument.

Example: Effect of connection with the same actual argument



[Description]

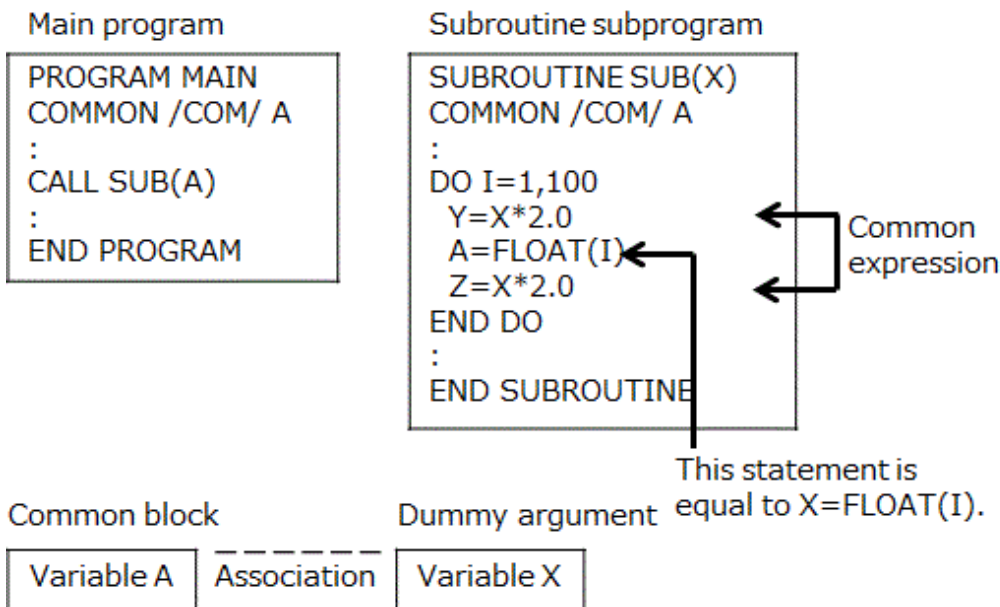
Dummy arguments "Y" and "Z" are optimized as different variables. Therefore, the optimization assumes that the value of "Z" in the DO loop is unchanged, and moves "Z+10.0" to outside the loop (moving an invariant expression). As a result, the results of the "Y=A/I" operation are not seen in the results of the "Z+10.0" operation. Further, even if the value of "Z" does change in the DO loop (Z+10.0 is not moved), the results of the "Y=A/I" operation are not seen in the results of the Z+10.0 operation if the variables "Y" and "Z" are allocated to a register in the DO loop. This is because different registers are allocated to variable "Y" and variable "Z".

Linkage with data in a common block

Dummy arguments and variables in a common block are optimized under the assumption that they are not associated (that is, the user has not made grammar errors). For this reason, associating dummy arguments with variables in a common block will introduce an error.

The following example shows the effects of associating a dummy argument with data in a common block.

Example 1: Effect of linking data in a common block

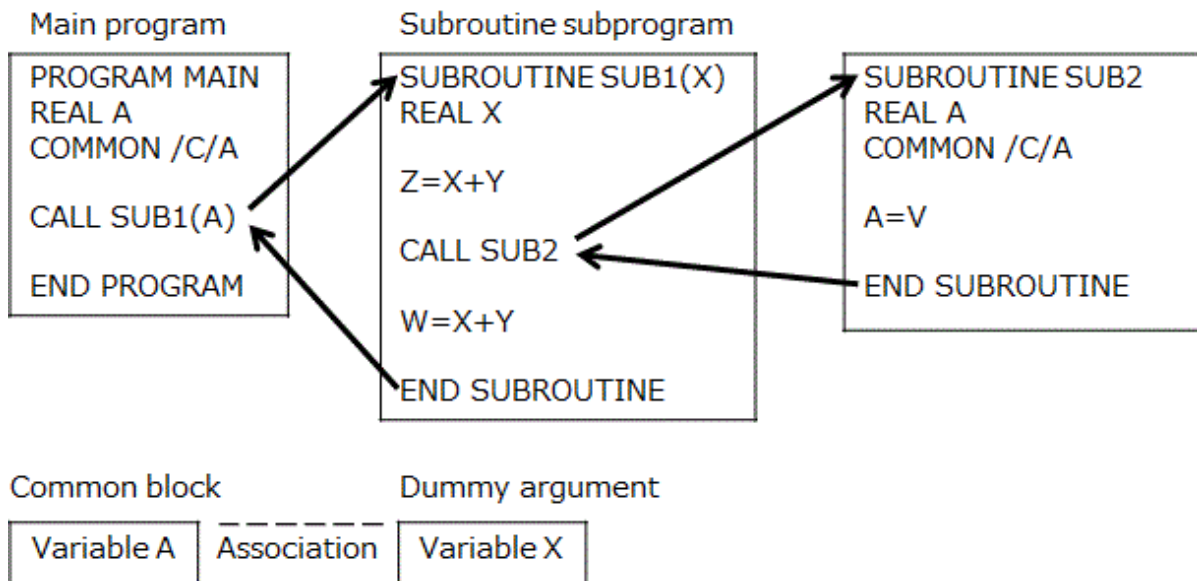


[Description]

The dummy-data "X" and the variable "A" in the common block are optimized as independent variables. For this reason, the optimization assumes that the "X*2.0" in the DO loop is a common expression, and replaces "Z=X*2.0" with Z=Y" (removing common expression). As a result, the value "Z" will not include the result of the expression "A=FLOAT(I)". The syntax prohibits assigning a

value by associating both a dummy argument and a common block with one instance at the same time. Only the dummy arguments or the common block can be associated.

Example 2: Effect of associating a dummy argument with data in a common block



[Description]

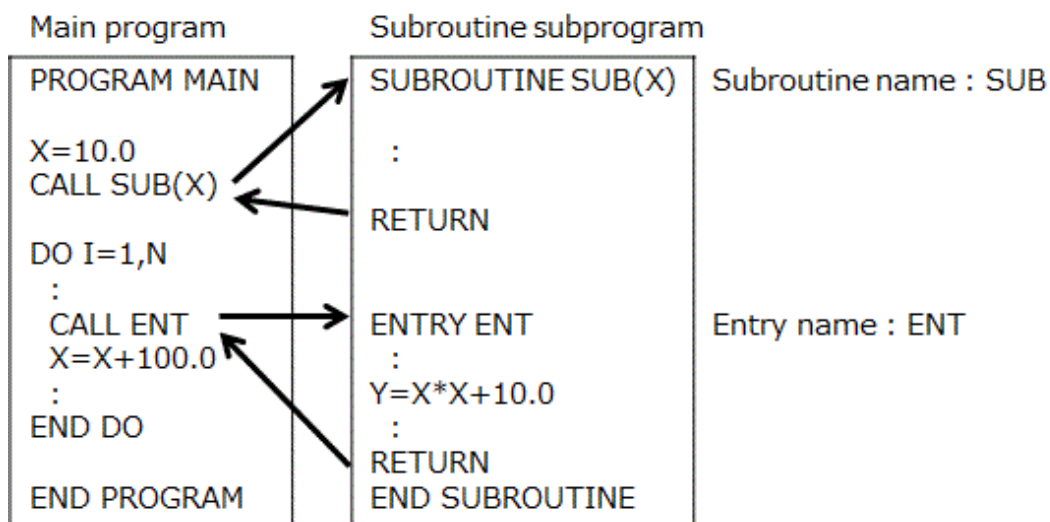
The value of the instance "A" in the common block is changed by "SUB2" called by "SUB1". At the same time, dummy argument "X" associated with common block object "A" is also changed. However, the compiler cannot identify the changes and the expression "X +Y" before and after calling "SUB2" is subject to optimization (removing common expression). As a result, the "W=X+Y" after calling "SUB2" is replaced with "W=Z", and the "W" will no longer include the change in "X" made by "SUB2".

Association using an ENTRY statement

Suppose a subprogram that uses the ENTRY statement is called using an entry name, function name, or subroutine name, and then the dummy arguments and the corresponding actual arguments are associated. Referencing the variable may introduce a problem if the subroutine is called using a different entry name, function name, or subroutine name, without instantiating the dummy argument as an actual argument. This is based on the assumption that the address of the actual argument is passed to the subprogram.

The following example shows the effect of the association using an entry name.

Example: Effect of association using a subentry



Explanation:

To be corrected, the call must be changed to

```
CALL ENT(X)
```

and the entry must be changed to

```
ENTRY ENT(X)
```

9.7 Restrictions on Inline Expansion

As there are various restrictions, such as correspondence between actual and dummy variables, user-defined procedures selected for inline expansion are not always inline expanded. The procedures that can be inline expanded are external procedures, internal procedures, and module procedures.

The following describes the restrictions on inline expansion.

9.7.1 Restrictions on Called User-Defined Procedures

a. Excessive number of instructions

If the total number of instructions for a user-defined procedure exceeds the allowed maximum, the user-defined procedure is not inline expanded.

The value of the allowed maximum can be changed with the parameter of the compile-time option -x. The default maximum value is determined by the compiler.

b. If the size of an array exceeds the allowed limit

The user-defined procedure will not be inline expanded when the size of the array exceeds the maximum, excluding the common block of the user-defined procedure, dummy argument, and assigned variable (that is, the arrays that are allocated as static and local within the user-defined procedure).

The value of the allowed maximum can be changed using the parameter of the compile-time option -x. The default maximum value is infinite.

c. Including a statement that restricts inline expansion

Procedures that include the following statements will not be inline expanded:

- ALLOCATE statement
- DEALLOCATE statement
- ASSIGN statement
- Assigned GO TO statement
- NAMELIST statement
- Variable group input-output statement
- RETURN statement

If a user-defined procedure contains RETURN statement that returns an integer expression, it is not inline expanded.

- NULLIFY statement

d. When a procedure is subject to restrictions on inline expansion, the following procedures will not be inline expanded:

- Procedures with a VOLATILE attribute
- Functions result that is an array, polymorphic, or of derived type with polymorphic component
- Procedures whose result has a VOLATILE attribute
- Module procedures with a host module that has a PRIVATE variable.
- Module procedures defined by an ENTRY statement
- Procedures that has a procedure language binding specifier

- Procedures that have an ASYNCHRONOUS attribute
 - Function result has a deferred length type parameter.
 - Function result has a character type and the FUNCTION statement or ENTRY statement has a procedure language binding specifier.
 - Procedures that have a variable with the SAVE attribute
 - When there is a SAVE statement that activates a variable, excluding the entirety of common blocks or elements of a common block (that is, variables that are local to the user-defined procedure), the user-defined procedure is not inline expanded.
 - Procedures that have an allocatable recursive component
 - Separate module procedure
 - Procedures that have a pointer initial value except a reference to the NULL intrinsic function
 - Procedures that have a definition of an internal procedure
 - Module procedures which host module has an allocatable variable when -xaccept=module_allocatable option is not specified.
 - Procedures using associated an allocatable variable when -xaccept=module_allocatable option is not specified.
- e. When changing a value of a variable or an array that has an initial value.
- A user-defined procedure is not inline expanded if a variable of the user-defined procedure has an initial value defined by a DATA statement or type declaration statement, and the value may be changed directly or indirectly.
 - A module procedure is not inline expanded if a variable or array has an initial value defined by a DATA statement or type declaration statement in its host module, and the value may be changed directly or indirectly.
- f. An ENTRY statement is included
- A subprogram that includes the following ENTRY statements is not inline expanded.
- The dummy parameters, defined with the function name or subroutine name, are different from ones defined on the entry name.
- g. The -Knoalias=s option is specified
- A procedure is not inline expanded if the procedure of either of the following conditions:
1. The calling and called procedures appear in different program units. And, the following either appears in the called procedure:
 - Derived type variable with pointer attribute.
 - Derived type variable contains a pointer component.
 2. The calling and called procedures appear in different program units. And, the called procedure appears in module program unit. And, the following either appears in declaration part of the called module procedure:
 - Derived type variable with pointer attribute.
 - Derived type variable contains a pointer component.
 3. The calling and called procedures appear in same program units. And, the dummy argument contained in the called procedure has a variable with a pointer attribute in the derived type variable.

9.7.2 Restrictions on Relationship between Calling and Called Program Units

- a. When the actual and dummy arguments are mismatched in the following ways:
- The number of actual and dummy arguments differs.
 - An actual argument and a dummy argument differ in attribute, type, or kind.
 - Only either an actual argument or a dummy argument is a procedure.

- b. When the relationship between actual and dummy arguments is one of the following:
- A dummy argument is a dummy procedure name, and it corresponds to an actual argument which is an intrinsic function or a procedure pointer
 - A dummy argument is *
 - An actual argument is a statement label
 - There is an array expression in the actual argument
 - An actual argument is an array name with the POINTER attribute
 - An actual argument is a %VAL or the intrinsic function VAL
 - An actual argument is an object of type having allocatable component and it is enclosed in parentheses
 - An actual argument is structure constructor of type which has allocatable component
 - A dummy argument has the OPTIONAL attribute
 - A dummy argument has the POINTER attribute
 - A dummy argument has the ALLOCATABLE attribute
 - A dummy argument has the VOLATILE attribute
 - A dummy argument or an actual argument is an assumed-shape array
 - A dummy argument has the VALUE attribute
 - A dummy argument has the ASYNCHRONOUS attribute
 - A dummy argument or an actual argument is polymorphic
 - The component of derived type for a dummy argument or an actual argument is polymorphic
 - A dummy argument is of parameterized derived type having length type parameter
 - A dummy argument has an assumed type or assumed rank
- c. When there is an error in the way user-defined external procedures are referenced
- When an external function subprogram is referenced as an external subroutine subprogram
 - When an external subroutine subprogram is referenced as an external function subprogram
 - When an external function subprogram type is referenced with a different type
- d. The user-defined external procedure and module procedure corresponds to the following criteria:
- A procedure specified as a local name by specifying a tentative name.

9.7.3 Restrictions on User-Defined External Procedure Names

When there is a user-defined external procedure with the same name, a program error occurs and inline expansion is not applied:

- There is an external subroutine subprogram with the same name
- There is an external function subprogram with the same name
- There is a block data program unit with the same name
- There are two or more block data program units without names

When the compiler option `-Koptmsg=2` is specified, a message is output indicating whether inline expansion is applied.

If inline expansion is not applied, a message is output indicating the reason.

9.7.4 Restrictions on Option

When `-H` option is specified, inline expansion is not applied.

9.7.5 Restrictions on Elemental Procedures

- a. Including a statement that restricts inline expansion

Elemental procedures that include the following statements will not be inline expanded:

- DATA statement
- FORMAT statement
- COMMON statement
- EQUIVALENCE statement

- b. When an elemental procedure is subject to restrictions on inline expansion, the following elemental procedures will not be inline expanded:

- The elemental procedures are subroutine
- Function result has a character type
- The procedures which call the elemental procedure appear in different program units
- The optimization control lines appear in program units
- There is an input/output format specifier except wildcard character "*"

9.8 Effects of Cray Pointer Variables

A pointer-based variable may be considered to share the memory of the variable with the address returned by the intrinsic function LOC. However, if the pointer contains the address of a common block object, a structure member, or a binding object, it is assumed that the address is shared with other data that belongs to the same common block object, structure, or binding object type.

In addition, assigning an address to a pointer-based variable without using the intrinsic function LOC may introduce an unexpected result.

Extra attention is required when a pointer-based variable is passed as a dummy argument, because the corresponding pointer is not considered to be sharing memory with the following variables:

- Other dummy arguments
- If other dummy arguments are pointer-based variables, the corresponding pointers
- Variables belonging to common blocks

9.9 Effects of Compiler Option -Kcommonpad[=N]

If the compiler option -Kcommonpad is specified during a separate compilation, the compiler option -Kcommonpad must also be specified for source files that include common blocks with the same names. (Refer to example 1)

When compiling with the compiler option -Kcommonpad=N specified for multiple files, the value for N must be the same.

In addition, the program may not run properly if the compiler option -Kcommonpad is specified when the element of a common block with the same name is changed and used. (Refer to example 2)

If a source file which includes a common block is compiled with the -Kcommonpad[=N] option, any other source file which includes the same common block must be compiled with the -Kcommonpad[=N] option, and value of N must be identical when the source files are compiled separately. (Refer to example 1.)

If common blocks with the same name have different elements, the execution result is unpredictable when the -Kcommonpad[=N] option is specified. (Refer to example 2.)

Example 1:

a.f

```
COMMON/CMN/A,B
INTEGER,DIMENSION(10) ::A=1,B=2
...
```

b.f

```
COMMON/CMN/A,B
INTEGER,DIMENSION(10) ::A,B
PRINT *,B
...
```

Both a.f and b.f include the common block named cmn, so the compile-time option `-Kcommonpad` must be specified for both a.f and b.f.

Example 2:

```
SUBROUTINE SUB_A
COMMON/CMN/A,B
REAL(4),DIMENSION(10) ::A=1.0,B=2.0
...
END SUBROUTINE SUB_A
SUBROUTINE SUB_B
COMMON /CMN/X
COMPLEX(4),DIMENSION(10) :: X
PRINT *,X
...
END SUBROUTINE SUB_B
```

As the element of the common block cmn is different in subroutines sub_a and sub_b, operation will be incorrect if the compile-time option `-Kcommonpad` is specified.

9.10 Using Optimization Control line (OCL)

By providing relevant optimization information in the source program, the effectiveness of optimization can be increased. Specifying OCLs in the source program can improve optimizations.

9.10.1 Notation Rules for Optimization Control Lines

Optimization control lines are the following either according to the source form.

- Free form

The optimization control lines consist of lines beginning with the five characters `!"OCL "` (the fifth character is a space). Blank characters can appear before the `!"OCL "`.

- Fixed form

The optimization control lines consist of lines beginning with the five characters `!"OCL "` (the fifth character is a space) in the columns 1 through 5.

Lines that begin with `!"OCL'` are usually treated as comment lines, but when the compiler option `-Kocl` is specified, they are used as optimization control lines and the operands become effective.

The notation rules are as follows:

```
!"OCL i[,i]...
```

i: Optimization control specifier. One or more blank spaces are necessary between `!"OCL"` and `"i"`.

9.10.2 Description Position of Optimization Control Line

The position where the optimization control line can be inserted depends on the type of optimization indicator.

Optimization control lines are specified in program units: DO loop units, statement units, or array assignment statement units except an optimization control line of specifier `FIXED`. Program units, DO loop units, statement units, and array assignment statement units are defined as follows:

- Program unit

The top of each program unit.

- DO Loop unit

Immediately in front of DO statements, however, blank lines, comment lines, and other optimization control lines (that can be specified within DO-loop unit) can be specified between optimization control lines and the DO statement.

- Statement units

Immediately in front of statements.

- Array assignment statement units

Immediately in front of array assignment statements.

An optimization control line of specifier FIXED can be specified in declaration part of program unit.

9.10.3 Wild Card Specification

In the operand of the following optimization control specifiers, a wild card may be specified for a variable name or a procedure name:

- NORECURRENCE
- NOVREC

See "[12.2.6.5 Wild Card Specification](#)" for more information about Wild Card.

9.10.4 Optimization Control Specifiers

Table below lists optimization control specifiers that can be included in optimization control lines.

Table 9.1 List of optimization control specifiers that can be included in optimization control lines

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
ARRAY_DECLARATION_OPT	It specifies to perform the optimization assuming that the array subscript does not exceed the range of the array declaration.	*	*	-	*
NOARRAY_DECLARATION_OPT	It specifies to perform the optimization without assuming that the array subscript does not exceed the range of the array declaration.	*	*	-	*
ARRAY_FUSION[,opt] END_ARRAY_FUSION	Fusion of the array assignment statement within the range is promoted. The specified optimization specifier designates to all the array assignment statement within the range.	-	-	-	*
ARRAY_MERGE (array1,array2[,array3]...) or ARRAY_MERGE (base_array.array1[,array2]...)	It specifies to merge the array or the array of few dimensions to the array of many dimensions. <i>base_array, array1, array2, ...</i> are names of arrays	*	-	-	-
ARRAY_SUBSCRIPT (array1[,array2]...)	It specifies to perform dimension shift of array. <i>array1, array2 ...</i> are names of arrays.	*	-	-	-
ASSUME ({SHORTLOOP NOSHORTLOOP MEMORY_BANDWIDTH	It specifies to control optimization considering features of the program.	*	-	-	-

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
NOMEMORY_BANDWIDTH TIME_SAVING_COMPILATION NOTIME_SAVING_COMPILATION})					
CLONE(<i>var</i> == <i>nI</i> [, <i>n2</i>]...)	<p>It directs to generate conditional branches along to the arguments and generate loop copies and put them for the conditional blocks assuming that the <i>var</i> is invariant in the loops.</p> <p>The conditional expressions are equals-to expressions which operands are <i>var</i> and <i>nI</i>[,<i>n2</i>]... of specified arguments.</p> <p>Variable <i>var</i> is a name of integer variable.</p> <p>Decimal from -9223372036854775808 to 9223372036854775807 or integer named constants can be specified as constant value (<i>nI</i>[,<i>n2</i>]...).</p>	-	*	-	*
EVAL	Instructs the system to perform optimization that changes the expression evaluation sequence.	*	*	-	*
NOEVAL	Cancel the EVAL.	*	*	-	*
EVAL_CONCURRENT	Gives the priority to the instruction-level parallelism in the tree-height-reduction optimization.	*	*	-	*
EVAL_NOCONCURREN	Suppresses the instruction-level parallelism and give priority to utilizing FMA instructions in the tree-height-reduction optimization.	*	*	-	*
EXTRACT_STRIDE_STORE	It directs to use scalar instructions for stride access store in the target loop.	*	*	-	*
NOEXTRACT_STRIDE_STORE	It directs to use SIMD instructions for stride access store in the target loop.	*	*	-	*
FISSION_POINT[(<i>n</i>)]	It specifies to instruct to apply loop fission optimization.	-	-	*	-
FIXED <i>array1</i> (<i>shape-spec-list</i>) [, <i>array2</i> (<i>shape-spec-list</i>)] ...	<p>It specifies either of the following association or allocation status for pointer or allocatable array.</p> <ul style="list-style-type: none"> - no associated or allocated - Lower and upper bounds of the array are associated or allocated with the <i>shape-spec-list</i>. <p>It specifies in declaration part of program unit.</p>	*	*	*	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
FP_CONTRACT	Instructs the system to use M&A instruction.	*	*	-	*
NOFP_CONTRACT	Cancel the FP_CONTRACT.	*	*	-	*
FP_RELAXED	The FP_RELAXED specifier instructs to apply optimizations which use reciprocal approximation instructions.	*	*	-	*
NOFP_RELAXED	The NOFP_RELAXED specifier instructs to suppress optimizations which use reciprocal approximation instructions.	*	*	-	*
FULLUNROLL_PRE_SIMD(<i>n</i>)	The FULLUNROLL_PRE_SIMD specifier instructs to perform the prioritize promotion of the full unrolling before SIMD. <i>n</i> is the upper limit of iteration count for a target loop, an integer value from 2 to 100.	-	*	-	*
NOFULLUNROLL_PRE_SIMD	The NOFULLUNROLL_PRE_SIMD specifier instructs to suppress the full unrolling before SIMD.	-	*	-	*
ILFUNC	The ILFUNC specifier instructs to apply inline expansion to intrinsic functions and operations.	*	*	-	*
NOILFUNC	The NOILFUNC specifier instructs to not to apply inline expansion to intrinsic functions and operations.	*	*	-	*
ITERATIONS(max= <i>n1</i>) ITERATIONS(avg= <i>n2</i>) ITERATIONS(min= <i>n3</i>)	Performs optimization assuming the maximum iteration count is max= <i>n1</i> , the average of the iteration count is avg= <i>n2</i> , and the minimum iteration count is min= <i>n3</i> . Decimal from 0 to 2147483647 can be specified as <i>n1</i> , <i>n2</i> and <i>n3</i> . One or more max, avg, and min specifiers can be specified in random order. When specifying max, avg and min at the same time, use comma to separate them like "ITERATIONS(max= <i>n1</i> ,avg= <i>n2</i> ,min= <i>n3</i>)".	*	*	-	-
LOOP_BLOCKING(<i>n</i>)	Designates blocking. Decimal from 2 to 10000 can be specified as block size <i>n</i> .	*	*	-	*
LOOP_NOBLOCKING	Cancel the LOOP_BLOCKING.	*	*	-	*
LOOP_FISSION_STRIPMININ G[({ <i>n</i> <i>c-level</i> })]	The LOOP_FISSION_STRIPMININING specifier instructs to perform the strip-mining optimization for fissioned loops. <i>n</i> is a number from 2 to 100000000. <i>c-level</i> is "L1" or "L2".	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
LOOP_NOFISSION_STRIPMINING	Cancels the LOOP_FISSION_STRIPMINING.	*	*	-	*
LOOP_FISSION_TARGET[({ CL LS })]	Performs loop fission optimization.	-	*	*	-
LOOP_FISSION_THRESHOLD(<i>n</i>)	Designates the threshold to decide the granularity of loops after loop fission. <i>n</i> is a number from 1 to 100.	*	*	-	*
LOOP_INTERCHANGE(<i>var1, var2, var3...</i>)	Designates that the sequence of a nested DO loop is changed according to the specified order. <i>var1, var2, var3...</i> are induction variables.	-	*	-	-
LOOP_NOINTERCHANGE	The LOOP_NOINTERCHANGE specifier instructs to suppress loop interchange optimizations.	-	*	-	-
LOOP_NOFUSION	This directs that fusion of the specified loop and the adjacent loops are not performed.	*	*	-	-
LOOP_PART_SIMD	Performs optimization which divides loops and partially uses SIMD extensions.	*	*	-	*
LOOP_NOPART_SIMD	Suppresses optimization which divides loops and partially uses SIMD extensions.	*	*	-	*
LOOP_PERFECT_NEST	Performs optimization which fissions the imperfectly nested loop into the perfectly nested loops.	*	*	-	*
LOOP_NOPERFECT_NEST	Suppress optimization which fissions the imperfectly nested loop into the perfectly nested loops.	*	*	-	*
LOOP_VERSIONING	Designates to perform optimization by the loop versioning.	*	*	-	-
LOOP_NOVERSIONING	Cancels the LOOP_VERSIONING.	*	*	-	-
MFUNC[<i>(level)</i>]	Designates to change the function to a multi-operation function. Decimal 1, 2 or 3 can be specified as <i>level</i> .	*	*	-	*
NOMFUNC	Designates canceling to change the function to a multi-operation function.	*	*	-	*
NOALIAS	Designates that pointer variables and other variable do not share memory area.	*	*	-	*
NOARRAYPAD(<i>array1</i>)	Designates canceling to execute padding with array element. <i>array1</i> is the name of array.	*	-	-	-
NORECURRENCE[<i>(array1[, array2,...])</i>]	Designates that array section references do not extend into loops of plural iterations.	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
NOVREC[(array1,array2,...)]	Designates that arrays of recurrence operation do not exist in loop which can use SIMD instructions. <i>array1, array2, ...</i> are names of arrays.	*	*	-	*
PREEX	Instructs the system to perform optimization by evaluating invariant first.	*	*	-	*
NOPREEX	Cancels the PREEX.	*	*	-	*
PREFETCH	Instructs the system to perform the automatic PREFETCH function of the compiler.	*	*	-	-
NOPREFETCH	Cancels the PREFETCH.	*	*	-	-
PREFETCH_CACHE_LEVEL (<i>c-level</i>)	Designates cache-level to prefetch of data. 1, 2 or all can be specified as <i>c-level</i> .	*	*	-	*
PREFETCH_CONDITIONAL	Instructs the system to generate the prefetch instruction to array data used in blocks in IF constructs and CASE constructs.	*	*	-	*
PREFETCH_NOCONDITIONAL	Cancels the PREFETCH_CONDITIONAL.	*	*	-	*
PREFETCH_INDIRECT	Instructs the system to generate the prefetch instruction to array data accessed indirectly in loops.	*	*	-	*
PREFETCH_NOINDIRECT	Cancels the PREFETCH_INDIRECT	*	*	-	*
PREFETCH_INFER	Instructs the system to be prefetch by distance is guessed from existing information while be omitted of address calculation for prefetch.	*	*	-	*
PREFETCH_NOINFER	Cancels the PREFETCH_INFER.	*	*	-	*
PREFETCH_ITERATION(<i>n</i>)	Instructs the system to be target of a prefetch instruction would be data that is referred to <i>n</i> iterations of a loop. This targets prefetch instructions only for the first level cache. Decimal from 1 to 10000 can be specified as <i>n</i> . If the optimization using SIMD extensions is applied to a loop, specifies the loop iteration count after using SIMD extensions for <i>n</i> .	*	*	-	*
PREFETCH_ITERATION_L2(<i>n</i>)	Instructs the system to be target of a prefetch instruction would be data that is defined or is referred to <i>n</i> iterations of a loop. This target of prefetch instruction while is prefetch only second cache.	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
	Decimal from 1 to 10000 can be specified as <i>n</i> . If the optimization using SIMD extensions is applied to a loop, specifies the loop iteration count after using SIMD extensions for <i>n</i> .				
PREFETCH_LINE(<i>n</i>)	Instructs the system to target data after <i>n</i> cache line(s) for generating prefetch instruction to the first level cache. Decimal from 1 to 100 can be specified as <i>n</i> .	*	*	-	*
PREFETCH_LINE_L2(<i>n</i>)	Instructs the system to target data after <i>n</i> cache line(s) for generating prefetch instruction to the second level cache. Decimal from 1 to 100 can be specified as <i>n</i> .	*	*	-	*
PREFETCH_READ (<i>name</i> [,level={ 1 2 }] [,strong={ 0 1 }])	Instructs the system to generate the prefetch instruction to the referred data. <i>name</i> is the array element or array section. level specifies the level of cache to prefetch, and strong specifies to use strong prefetch or not.	-	-	*	-
PREFETCH_WRITE (<i>name</i> [,level={ 1 2 }] [,strong={ 0 1 }])	Instructs the system to generate the prefetch instruction to the defined data. <i>name</i> is the array element or array section. level specifies the level of cache to prefetch, and strong specifies to use strong prefetch or not.	-	-	*	-
PREFETCH_SEQUENTIAL[({AUTO SOFT})]	This directs that prefetch instructions are generated for array data that is accessed sequentially.	*	*	-	*
PREFETCH_NOSEQUENTIAL	This directs that prefetch instructions are not generated for array data that is accessed sequentially.	*	*	-	*
PREFETCH_STRIDE [({SOFT HARD_AUTO HARD_ALWAYS})]	Prefetch instructions are generated for array data that is accessed with a stride larger than the cache line size used in a loop.	*	*	-	*
PREFETCH_NOSTRIDE	Prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop.	*	*	-	*
PREFETCH_STRONG	This directs that the prefetch instruction for the first level cache is to be the strong prefetch.	*	*	-	*
PREFETCH_NOSTRONG	This directs that the prefetch instruction for the first level cache will not be strong prefetch.	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
PREFETCH_STRONG_L2	This directs that the prefetch instructions for the second level cache are to be strong prefetch.	*	*	-	*
PREFETCH_NOSTRONG_L2	This directs that the prefetch instructions for the second level cache are not to be strong prefetch.	*	*	-	*
PRELOAD	This directs to perform speculative execution of load instructions.	*	*	-	*
NOPRELOAD	This directs to suppress speculative execution of load instructions.	*	*	-	*
SCACHE_ISOLATE_WAY(L2= <i>n</i> [,L1= <i>n</i> 2]) END_SCACHE_ISOLATE_WAY	It directs to specify the maximum number of ways of first level cache and second level cache in sector 1 of the cache.	*	-	*	-
SCACHE_ISOLATE_ASSIGN(<i>array1</i> , <i>array2</i> ...) END_SCACHE_ISOLATE_ASSIGN	It directs to specify an array stored in sector 1 of the cache.	*	-	*	-
SIMD [({ALIGNED UNALIGNED})]	The SIMD specifier instructs to use SIMD instructions.	*	*	-	*
NOSIMD	The NOSIMD specifier instructs not to use SIMD instructions.	*	*	-	*
SIMD_LISTV [({ALL THEN ELSE})]	This directs that the list vector conversion is performed to the execution statements in IF construct.	-	-	*	-
SIMD_NOREDUNDANT_VL	This directs that SIMD with redundant executions for the SIMD width is not applied.	*	*	-	*
SIMD_USE_MULTIPLE_STRUCTURES	This directs to use the SVE Load Multiple Structures instructions and SVE Store Multiple Structures instructions.	*	*	-	*
SIMD_NOUSE_MULTIPLE_STRUCTURES	This directs not to use the SVE Load Multiple Structures instructions and SVE Store Multiple Structures instructions.	*	*	-	*
STRIPING[<i>(n)</i>]	This directs that the striping optimization is performed. The striped length (number of expansions) can be specified for <i>n</i> . <i>n</i> indicates the stripe-length (number of expansions), an integer value from 2 to 100.	*	*	-	*
NOSTRIPING	This directs that the loop striping optimization is suppressed.	*	*	-	*
SWP	Designates software pipelining.	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
SWP_FREG_RATE(<i>n</i>) SWP_IREG_RATE(<i>n</i>) SWP_PREG_RATE(<i>n</i>)	Changes the condition of the register number of software pipelining. <i>n</i> indicates the rate (percentage) of the register number that can be used by software pipelining. <i>n</i> is a number from 1 to 1000.	*	*	-	*
SWP_POLICY ({AUTO SMALL LARGE})	Specifies a policy to select an instruction scheduling algorithm used in software pipelining.	*	*	-	*
SWP_WEAK	Adjusts software pipelining to make overlapping of the instructions less.	*	*	-	*
NOSWP	Cancels the SWP.	*	*	-	*
UNROLL[(<i>n</i>)]	Designates unrolling a corresponding DO loop. <i>n</i> is an iteration count for unrolling, an integer value from 2 to 100.	-	*	-	*
UNROLL('full')	Designates full unrolling.	-	*	-	*
NOUNROLL	Cancels the UNROLL.	-	*	-	*
UNROLL_AND_JAM[(<i>n</i>)]	Performs unroll-and-jam to the loop when the optimization is expected to be effective. <i>n</i> is an iteration count for unrolling, an integer value from 2 to 100.	*	*	-	-
UNROLL_AND_JAM_FORCE[(<i>n</i>)]	Performs unroll-and-jam. <i>n</i> is an iteration count for unrolling, an integer value from 2 to 100.	-	*	-	-
NOUNROLL_AND_JAM	Suppresses unroll-and-jam.	*	*	-	-
UNSWITCHING	The UNSWITCHING specifier instructs loop unswitching to IF construct.	-	-	*	-
ZFILL[(<i>n</i>)]	Directs that the zfill optimization to be performed. The optional parameter <i>n</i> is an integer between 1 and 100 that specifies the number of blocks the DC ZVA instruction writes.	-	*	-	*
NOZFILL	Directs that the zfill optimization not to be performed.	-	*	-	*

P: Program unit

D: DO loop unit

S: Statement unit

A: Array assignment statement unit

*: The optimization control specifier can be specified in the optimization control lines.

-: The optimization control specifier cannot be specified in the optimization control lines.

1. !OCL ARRAY_DECLARATION_OPT

The ARRAY_DECLARATION_OPT specifier instructs to perform the optimization assuming that the array subscript does not exceed the range of the array declaration.

The following shows an example:

Example:

```
REAL*8 A(8)
!OCL ARRAY_DECLARATION_OPT
DO I=1,N
  A(I) = 0
ENDDO
```

The optimization is performed assuming that the operation range of the subscript is equal to or less than the SIMD length.

2. !OCL NOARRAY_DECLARATION_OPT

The NOARRAY_DECLARATION_OPT specifier instructs to perform the optimization without assuming that the array subscript does not exceed the range of the array declaration.

The following shows an example:

Example:

```
REAL*8 A(8)
!OCL NOARRAY_DECLARATION_OPT
DO I=1,N
  A(I) = 0
ENDDO
```

Performs optimization with the operating range of the subscript unknown.

3. !OCL ARRAY_FUSION[*opt*] !OCL END_ARRAY_FUSION

When ARRAY_FUSION specifier is put at the beginning of the target range and END_ARRAY_FUSION specifier is put at the end of the range, array fusion is promoted in the range.

Not all array assignment statements within the range are fused even if the optimization control lines are used. Further, *opt* may be specified in the optimization control specifier, and this will be effective in all array assignment statements within the range.

The following shows an example:

Example:

```
REAL, DIMENSION(10) :: A,B,C
...
!OCL ARRAY_FUSION      *
  A=B+ ...             *Useful range
  C=C+ ...             *
!OCL END_ARRAY_FUSION *
...
```

4. !OCL ARRAY_MERGE(*array1,array2[,array3]...*)

The ARRAY_MERGE specifier instructs to merge arrays. The arrays to be merged are explicit shape arrays.

See Section "[9.15.2 Restrictions of Merging Local Array Variables](#)" for information on restrictions.

This function will not be applied within specification expressions nor to the constant expressions of declaratives.

The following shows an example:

Example 1:

```
!OCL ARRAY_MERGE(A,B,C)
INTEGER, DIMENSION(101,102,103)::A,B,C
CALL SUB(A)
```

```

PRINT *,B(101,102,103)
A=C
A(101,102,103)=C(1,2,3)

```

When the `-Kocl` option is specified, the program above is considered as the following program.

```

INTEGER,DIMENSION(3,101,102,103)::TMP
! A, in 1 element of the 1st dimension, merge : TMP (1,101,102,103)
! B, in 2 element of the 1st dimension, merge : TMP (2,101,102,103)
! C, in 3 element of the 1st dimension, merge : TMP (3,101,102,103)
CALL SUB(TMP(1,:,:,)) ! An array name is changed to the array
                        ! element which has shape.
...
PRINT *,TMP(2,101,102,103)
TMP(1,:,:,) = TMP(3,:,:,)
TMP(1,101,102,103)=TMP(3,1,2,3)

```

Note:

TMP is a variable generated by the compiler.

Example 2:

```

!OCL ARRAY_MERGE(A,B)
PROGRAM MAIN
INTEGER,DIMENSION(100,101,102) :: A,B
...
CALL SUB(A)
END
SUBROUTINE SUB(C)
INTEGER,DIMENSION(100) :: C
...
END

```

When this feature is in effect, the arguments for the procedure SUB specified in MAIN may not be associated with the parameter C(100) in SUB, because a new element is appended in a dimension of A.

5. !OCL ARRAY_MERGE(*base_array*:*array1*[,*array2*]...)

This merges arrays to the *base_array*. This will affect intrinsic functions, for which the keyword arguments DIM and MASK can be specified.

The arrays are merged into the final dimension of the base variable. The target arrays are explicit shape arrays with bounds that are constant expressions (local arrays and common block objects).

See Section "[9.15.2 Restrictions of Merging Local Array Variables](#)" for information on restrictions (except for common-block-object items). This function will not be applied within specification expressions or the constant expressions of declaratives.

The following shows an example:

Example 1:

```

!OCL ARRAY_MERGE(A:B,C)
PROGRAM MAIN
INTEGER,DIMENSION(10,11,12,2) :: A
INTEGER,DIMENSION(10,11,12) :: B,C
...
A=1
B=1
C(1,2,3)=1

```

The following shows how arrays are merged:

```

PROGRAM MAIN
INTEGER,DIMENSION(10,11,12,4) :: A
A=1

```

```
A(:, :, :, 3)=1
A(1, 2, 3, 4)=1
```

Example 2:

```
!OCL ARRAY_MERGE(A:B,C)
  INTEGER, DIMENSION(1, 2, 3, 4) :: A, X
  INTEGER, DIMENSION(1, 2, 3) :: B, C
  ...
  X=A ! An error occurs during compilation.
  A=RESHAPE((/(i,i=1,24)/), (/1, 2, 3, 4/))
  !The calculation result is not guaranteed because of expression
  ! shape difference from the right side to the left.
  PRINT *, UBOUND(A, 4)
  ! 6 is output.
```

6. !OCL ARRAY_SUBSCRIPT(*array1*[,*array2*]...)

The ARRAY_SUBSCRIPT specifier instructs to perform dimension shift of arrays. The target array variables are allocatable arrays and explicit shape arrays with bounds that are constant expressions (local arrays, common block objects and dummy arguments).

See Section "9.14.2 Restrictions of Target Variable of Dimension Shift of Array Declaration" for information on restrictions (except the restriction for combination substance). This function will not be applied within specification expressions or the constant expressions of declaratives.

The following shows an example:

Example 1:

```
!OCL ARRAY_SUBSCRIPT(A)
PROGRAM MAIN
  INTEGER, DIMENSION(101, 102, 103, 5) :: A
  CALL SUB(A)
  ...
  PRINT *, A(1, 2, 3, 4)
```

If this type of program is compiled with the -Kocl option specified, the compiler will treat it as one of the following types of program:

```
PROGRAM MAIN
  INTEGER, DIMENSION(5, 101, 102, 103) :: A
  CALL SUB(A)
  ...
  PRINT *, A(4, 1, 2, 3)
```

Example 2:

```
!OCL ARRAY_SUBSCRIPT(A)
PROGRAM MAIN
  INTEGER, DIMENSION(100, 101, 102, 3) :: A
  ...
  CALL SUB(A)
END
SUBROUTINE SUB(B)
  INTEGER, DIMENSION(100) :: B
  ...
END
```

In this example, the argument A(3,100,101,102) for SUB specified in MAIN will be merged with the parameter B(100) defined in SUB. However, the result may be incorrect because the dimension of the array is not identical.

Example 3:

```
!OCL ARRAY_SUBSCRIPT(A)
PROGRAM MAIN
  INTEGER, DIMENSION(1, 2, 3, 4) :: A, B
```

```

...
B=A ! An error occurs during compilation.
A=RESHAPE((/(i,i=1,24)/),(/1,2,3,4/))
!The calculation result is not guaranteed because of expression shape
! difference from the right side to the left.
PRINT *,UBOUND(A,4)
!3 is output.

```

7. !OCL ASSUME({SHORTLOOP | NOSHORTLOOP | MEMORY_BANDWIDTH | NOMEMORY_BANDWIDTH | TIME_SAVING_COMPILATION | NOTIME_SAVING_COMPILATION})

The assume specifier instructs whether or not to control optimization considering features of the program. Multiple specifiers can be instructed at the same time.

!OCL ASSUME(SHORTLOOP)

Optimization is performed assuming that iteration counts of the innermost loops in the program are small. Loop optimizations such as SIMD extensions, automatic parallelization, loop unrolling and software prefetch may be controlled or invalidated.

!OCL ASSUME(NOSHORTLOOP)

Optimization is performed assuming that iteration counts of innermost loops in the program are not small.

!OCL ASSUME(MEMORY_BANDWIDTH)

Optimization is performed assuming that innermost loops in the program has a memory bandwidth bottleneck. Loop optimizations such as SIMD extensions, automatic parallelization and loop unrolling may be controlled or invalidated, and zfill optimization may be controlled or promoted.

!OCL ASSUME(NOMEMORY_BANDWIDTH)

Optimization is performed assuming that innermost loop in the program does not have a bottleneck in the memory bandwidth.

!OCL ASSUME(TIME_SAVING_COMPILATION)

Optimization is controlled to decrease complication time of the program.

!OCL ASSUME(NOTIME_SAVING_COMPILATION)

Optimization is performed with giving priority to the speed of the executable program rather than decrease of the compilation time.

The following shows examples:

Example:

```

!OCL ASSUME(SHORTLOOP)
!OCL ASSUME(MEMORY_BANDWIDTH)
!OCL ASSUME(NOTIME_SAVING_COMPILATION)
SUBROUTINE SUB(A,B,C,D,N)
  INTEGER :: I,N,A(:),B(:),C(:),D(:)
  DO I = 1,N
    A(I) = B(I) + B(I)
  ENDDO
END SUBROUTINE

```

The optimization is controlled considering the specified features in the target loop.

8. !OCL CLONE(*var*==*nI*[,*n2*]...)

The CLONE specifier instructs to generate conditional branches along to the arguments and generate loop copies and put them for the conditional blocks assuming that the *var* is invariant in the loops. The order of the generated branches is the order of the corresponding arguments. This specifier promotes other optimizations, such as full unrolling. The execution result is not guaranteed in case that the value of variable *var* changes in the loop. For more details, refer to "[9.18 Incorrectly Specified Optimization Indicators](#)".

The size of the object module and compile time may increase because this specifier makes copies of loops. This specifier is effective only when the -O3 option is set.

Variable *var* is a name of integer variable whose kind value is 1, 2, 4, or 8.

You cannot specify following integer variable.

- Structure component
- Array, Array elements, or Substrings
- Pointer variables
- Allocatable variables
- Threadprivate variables

And, you cannot specify integer variable with following attributes.

- ALLOCATABLE
- CHANGEENTRY
- CONTIGUOUS
- DIMENSION
- EXTERNAL
- INTRINSIC
- PARAMETER
- POINTER

Decimal from -9223372036854775808 to 9223372036854775807 or integer named constants can be specified as constant values ($n1[,n2]...$). Do not specify any kind type parameter with an underscore '_' regardless of values.

You can use comma to enumerate constant values and colon to specify a range. The following two specifications are equivalent.

```
!OCL CLONE(var==1,3:5,7)
```

```
!OCL CLONE(var==1,3,4,5,7)
```

You can specify up to 20 values at once. When more than 20 values are specified, excess values are ignored. The following specification is limited in the number of values because it specifies 30 values.

```
!OCL CLONE(var==11:40)
```

The above specification is treated as the same as the following specification because excess values are ignored.

```
!OCL CLONE(var==11:30)
```

The following shows examples:

Example 1:

```
SUBROUTINE SUB(A,N)
...
!OCL CLONE(N==10,20)
DO I=1,N
    A(I) = I
ENDDO
...
END SUBROUTINE
```

->

```
SUBROUTINE SUB(A,N)
...
IF (N==10) THEN
DO I=1,10
    A(I) = I
ENDDO
ELSE IF (N==20) THEN
DO I=1,20
    A(I) = I
ENDDO
ELSE
DO I=1,N
    A(I) = I
ENDDO
ENDIF
...
END SUBROUTINE
```

Loop is copied with IF constructs in the order of the specification.

Example 2:

<pre> SUBROUTINE SUB(A,N,M) ... !OCL CLONE(N==10) !OCL CLONE(M==20) DO I=1,N A(I) = M ENDDO ... END SUBROUTINE </pre>	->	<pre> SUBROUTINE SUB(A,N,M) ... IF (N==10) THEN DO I=1,10 A(I) = M ENDDO ELSE IF (M==20) THEN DO I=1,N A(I) = 20 ENDDO ELSE DO I=1,N A(I) = M ENDDO ENDIF ... END SUBROUTINE </pre>
-------------------------------------------------------------------------------------------------------------------------------	----	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Loop is copied with IF constructs in the order of the multiple specifications.

Example 3:

<pre> SUBROUTINE SUB(A,N,M) ... !OCL CLONE(M==10) DO J=1,M !OCL CLONE(N==20) DO I=1,N A(I,J) = 0 ENDDO ENDDO ... END SUBROUTINE </pre>	->	<pre> SUBROUTINE SUB(A,N,M) ... IF(M==10) THEN DO J=1,10 IF (N==20) THEN DO I=1,20 A(I,J) = 0 ENDDO ELSE DO I=1,N A(I,J) = 0 ENDDO ENDIF ENDDO ELSE DO J=1,M IF (N==20) THEN DO I=1,20 A(I,J) = 0 ENDDO ELSE DO I=1,N A(I,J) = 0 ENDDO ENDIF ENDDO ENDIF ... END SUBROUTINE </pre>
------------------------------------------------------------------------------------------------------------------------------------------------	----	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

CLONE specifiers for nested loops generate nested IF constructs.

9. !OCL EVAL

The EVAL specifier instructs to perform the optimization that changes the operation evaluation type. The following shows an example:

Example:

<pre>!OCL EVAL DO I=1,N A(I)=A(I)+B(I)+C(I)+D(I) ENDDO</pre>	->	<pre>DO I=1,N A(I)=(A(I)+B(I))+(C(I)+ D(I)) ENDDO</pre>
----------------------------------------------------------------	----	-----------------------------------------------------------

Optimization that changes the operation evaluation type in the target loop is performed.

10. !OCL NOEVAL

The NOEVAL specifier instructs to suppress the optimization that changes the operation evaluation type.

The following shows an example:

Example:

```
!OCL NOEVAL
DO I=1,N
  A(I) = A(I) + B(I) + C(I) + D(I)
ENDDO
```

Optimization that changes the operation evaluation type in the target loop is not performed.

11. !OCL EVAL_CONCURRENT

The EVAL_CONCURRENT specifier instructs to give priority to the instruction-level parallelism in the tree-height-reduction optimization.

For more details, refer to "[9.1.2.10 Tree-Height-Reduction Optimization](#)".

The following shows an example:

Example:

```
!OCL EVAL_CONCURRENT
DO I=1,N
  X(I) = A(I) * B(I) + C(I) * D(I) + E(I) * F(I) + G(I) * H(I)
ENDDO
```

12. !OCL EVAL_NOCONCURRENT

The EVAL_NOCONCURRENT specifier instructs to suppress the instruction-level parallelism and specifies to give priority to utilizing FMA instructions in the tree-height-reduction optimization.

For more details, refer to "[9.1.2.10 Tree-Height-Reduction Optimization](#)".

The following shows an example:

Example:

```
!OCL EVAL_NOCONCURRENT
DO I=1,N
  X(I) = A(I) * B(I) + C(I) * D(I) + E(I) * F(I) + G(I) * H(I)
ENDDO
```

13. !OCL EXTRACT_STRIDE_STORE

The EXTRACT_STRIDE_STORE specifier instructs to use scalar instructions for stride access store in the target loop.

The following shows an example:

Example:

```
!OCL EXTRACT_STRIDE_STORE
DO I=1,N,2
  A(I) = B(I) + C(I)
ENDDO
```

Performs optimization using scalar instructions for store of array A that can be converted to SIMD instruction.

14. !OCL NOEXTRACT_STRIDE_STORE

The NOEXTRACT_STRIDE_STORE specifier instructs to use SIMD instructions for stride access store in the target loop.

The following shows an example:

Example:

```
!OCL NOEXTRACT_STRIDE_STORE
DO I=1,N,2
  A(I) = B(I) + C(I)
ENDDO
```

Performs optimization using SIMD instructions for array A.

15. !OCL FISSION_POINT[(n)]

The FISSION_POINT(n) specifier instructs to apply loop fission optimization.

The multiloop which is (n-1)th-parent from innermost loop is distributed.

Decimal from 1 to 6 can be specified to n. If a value was not specified for n, the compiler distributes innermost loop.

The FISSION_POINT[(n)] specifier is effective only when it is placed in the innermost loop.

An example is shown in the following.

Example:

<pre>DO I=2,N DO J=2,N A(I,J) = A(I-1,J-1) !OCL FISSION_POINT(1) A(I,J) = A(I,J) + A(I-1,J) ENDDO ENDDO</pre>	->	<pre>DO I=2,N DO J=2,N A(I,J) = A(I-1,J-1) ENDDO DO J=2,N A(I,J) = A(I,J) + A(I-1,J) ENDDO ENDDO</pre>
---------------------------------------------------------------------------------------------------------------------------	----	------------------------------------------------------------------------------------------------------------------

Note

If loop fission is performed on a loop that cannot be executed because of the program logic, some side effects, such as exceptions on runtime, can happen.

For instance, the compiler generates some temporary arrays for passing data between the split loops. In this case, the necessary instructions may be executed speculatively.

In the following example, the load instructions of the array element IDX(M, M) is speculatively executed. If N is less than or equal to 0 and M used in array element IDX(M, M) is greater than the declared size, an exception may be thrown at runtime.

Example:

<pre>DO I=1,N DO J=1, IDX(M,M) X = A(J) + B(J) !OCL FISSION_POINT(1) C(J) = X ENDDO ENDDO</pre>	->	<pre>ALLOCATE(TMP(IDX(M,M))) ! speculative execution DO I=1,N DO J=1,IDX(M,M) TMP(J) = A(J) + B(J) ENDDO DO J=1,IDX(M,M) C(J) = TMP(J) ENDDO ENDDO DEALLOCATE(TMP)</pre>
-------------------------------------------------------------------------------------------------------------	----	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

16. !OCL FIXED array1 (shape-spec-list)[, array2 (shape-spec-list)] ...

The FIXED specifier instructs either of the following association or allocation status for pointer or allocatable array.

- no associated or allocated

- Lower and upper bounds of the array are associated or allocated with the *shape-spec-list*.

The *array* must be a name of pointer or allocatable array.

The *shape-spec* is the following syntax:

```
[allocate-lower-bound : ] allocate-upper-bound
```

A constant expression or * can be specified in the *allocate-lower-bound* and *allocate-upper-bound*.

If the *array* is associated or allocated, the *array* must not re-associate or re-allocate.

When the *allocate-lower-bound* or *allocate-upper-bound* is specified with *, It is specified that the bound the *array* is not specified with a constant.

If the *allocate-lower-bound* is omitted, the default value is 1.

The optimization control line of FIXED specifier specifies in declaration part of program unit.

The optimization control line must appear after declaration of the pointer or allocatable array.

If a named constant appears in the *allocate-lower-bound* or *allocate-upper-bound*, the optimization control line must appear after declaration of the named constant.

The optimization control line of FIXED specifier must not specify with any other optimization control specifier.

If a pointer array is specified, the pointer array must be contiguous sequence of storage.

The pointer or allocatable array must not specify by host association.

Example:

```
MODULE MOD
INTEGER, POINTER :: P( :, : )
!OCL FIXED P(2:3,4:*)
END
SUBROUTINE S(K)
USE MOD
ALLOCATE( P(2:3,4:K) )
P( :, : )=1
:
END
```

Because a constant value is not possible to specify for the last upper bound of pointer array p, an asterisk(*) is specified in the last upper bound of the FIXED specifier.

17. !OCL FP_CONTRACT

The FP_CONTRACT specifier instructs that Floating-Point Multiply-Add/Subtract instructions are to be used. Note that use of these instructions may introduce errors on the same level as rounding errors.

Example:

```
!OCL FP_CONTRACT
DO I=1,N
  A(I) = A(I) + B(I) * C(I)
ENDDO
```

18. !OCL NOFP_CONTRACT

The NOFP_CONTRACT specifier instructs that Floating-Point Multiply-Add/Subtract floating point operation instructions are not to be used.

Example:

```
!OCL NOFP_CONTRACT
DO I=1,N
  A(I) = A(I) + B(I) * C(I)
ENDDO
```

19. !OCL FP_RELAXED

The `FP_RELAXED` specifier instructs to perform high-speed reciprocal approximation operations for division or `SQRT` functions. This applies to single-precision and double-precision floating point calculations. This optimization may introduce rounding errors, compared to when the standard division or `SQRT` instructions are used.

When the `-NRtrap` option and either of the `-Knosimd` option or the `-KNOSVE` option are effective, the optimization to convert a `SQRT` function into reciprocal approximation instructions is suppressed. Therefore, the execution performance may decrease as compared with when the `-NRnotrap` option is effective. Note that the reciprocal approximation instructions generated by this specifier may be evaluated in advance even if the `-Knopreex` option is enabled. And a floating-point exception may occur when the `-Knopreex` option, `-NRtrap` option, and this specifier are valid.

Example:

```
!OCL FP_RELAXED
DO I=1,N
  A(I) = SQRT(B(I) / C(I))
ENDDO
```

20. !OCL NOFP_RELAXED

The `NOFP_RELAXED` specifier instructs to perform operations using division and `SQRT` without using reciprocal approximation instructions, for single-precision and double-precision floating point calculations.

An example is shown in the following.

Example:

```
!OCL NOFP_RELAXED
DO I=1,N
  A(I) = SQRT(B(I) / C(I))
ENDDO
```

21. !OCL FULLUNROLL_PRE_SIMD[*n*]

The `FULLUNROLL_PRE_SIMD` specifier instructs to perform the prioritize promotion of the full unrolling before `SIMD`. *n* is the upper limit of iteration count for a target loop. *n* is an integer value from 2 to 100. If a value was not specified for *n*, the compiler will automatically determine the value.

Note that only the `DO`-loop or array expressions immediately after the optimization control line is subject to optimization control. Optimization is not performed if the iteration count is unknown.

The following shows an example:

Example:

```
DO I=1,N
!OCL FULLUNROLL_PRE_SIMD
  DO J=1,16
    A(J,I) = B(I,J) + C(I,J)
  ENDDO
ENDDO
```

Full unrolling before `SIMD` is applied for an inner loop.

22. !OCL NOFULLUNROLL_PRE_SIMD

The `NOFULLUNROLL_PRE_SIMD` specifier instructs to suppress the full unrolling before `SIMD`.

Note that only the `DO`-loop or array expressions immediately after the optimization control line is subject to optimization control.

The following shows an example:

Example:

```
DO I=1,N
!OCL NOFULLUNROLL_PRE_SIMD
  DO J=1,2
    A(J,I) = B(I,J) + C(I,J)
```

```
ENDDO
ENDDO
```

Full unrolling before SIMD is not applied for an inner loop.

23. !OCL ILFUNC

The ILFUNC specifier instructs to apply inline expansion to intrinsic functions and operations. The targets are inline expanded unconditionally, which may result in performance degradation.

When inline expansion is applied, side effects may occur in the execution result. Also, this optimization may result in generating a floating-point exception when -NRtrap is not valid.

The following shows an example:

Example:

```
!OCL ILFUNC
DO I=1,N
  A(I) = SIN(B(I))
ENDDO
```

24. !OCL NOILFUNC

The NOILFUNC specifier instructs not to apply inline expansion to intrinsic functions and operations.

The following shows an example:

Example:

```
!OCL NOILFUNC
DO I=1,N
  A(I) = SIN(B(I))
ENDDO
```

25. !OCL ITERATIONS(max=*n1*)

!OCL ITERATIONS(avg=*n2*)

!OCL ITERATIONS(min=*n3*)

These specifiers instruct to optimize assuming the maximum loop iteration count is max=*n1*, the average loop iteration count is avg=*n2*, and the minimum loop iteration count is min=*n3*. Decimal from 0 to 2147483647 can be specified as *n1*, *n2*, and *n3*. One or more max, avg, and min specifiers can be specified in random order. When specifying max, avg and min at the same time, use comma to separate them like "ITERATIONS(max=*n1*,avg=*n2*,min=*n3*)". Note that values *n1*, *n2* and *n3* specified as max, avg, and min are required to be $n1 \geq n2 \geq n3$.

These specifiers are effective when the loop iteration count is unknown at compilation.

The ITERATIONS(max=*n1*) specifier instruct to optimize assuming the maximum loop iteration count is *n1*. If the actual loop iteration count is greater than *n1*, the execution result is not guaranteed. See section "[9.18.4 Example of ITERATIONS Specifier](#)" for more details.

The ITERATIONS(avg=*n2*) specifier instruct to optimize assuming the average loop iteration count is *n2*. The execution result is guaranteed even if the actual average iteration loop count is different from *n2* because ITERATIONS(avg=*n2*) is used as an optimization hint.

The ITERATIONS(min=*n3*) specifier instruct to optimize assuming the minimum loop iteration count is *n3*. If the actual loop iteration count is less than *n3*, the execution result is not guaranteed. See section "[9.18.4 Example of ITERATIONS Specifier](#)" for more details.

These specifiers can be specified in the program unit or DO loop unit, but it is recommended to specify in DO loop unit because the iteration counts of loops in the program may not be the same.

Examples are shown in the following.

Example 1: avg

```
!OCL ITERATIONS(avg=32)
DO I=1, M
```

```
A(I) = B(I) + C(I)
ENDDO
```

Example 2: min and avg

```
!OCL ITERATIONS(min=1,avg=8)
DO I=1, M
  A(I) = B(I) + C(I)
ENDDO
```

Example 3: max, avg, and min

```
!OCL ITERATIONS(max=128,avg=16,min=16)
DO I=1, M
  A(I) = B(I) + C(I)
ENDDO
```

26. !OCL LOOP_BLOCKING(*n*)

The LOOP_BLOCKING specifier instructs to perform the blocking optimization. Specify the block size when blocking in *n*.

The following shows an example:

Example:

<pre>!OCL LOOP_BLOCKING(80) DO J=1,N DO I=1,N A(I,J) = A(I,J) + B(J,I) ENDDO ENDDO</pre>	->	<pre>DO JJ=1,N,80 DO II=1,N,80 DO J=JJ,MIN(N,JJ+79) DO I=II,MIN(N,II+79) A(I,J) = A(I,J) + B(J,I) ENDDO ENDDO ENDDO ENDDO</pre>
--------------------------------------------------------------------------------------------------	----	-----------------------------------------------------------------------------------------------------------------------------------------------------------------

Loop blocking is performed with a block size of 80.

27. !OCL LOOP_NOBLOCKING

The LOOP_NOBLOCKING specifier instructs that loop blocking is suppressed.

The following shows an example:

Example:

```
!OCL LOOP_NOBLOCKING
DO J=1,N
  DO I=1,N
    A(I, J) = A(I, J) + B(I, J)
  ENDDO
ENDDO
```

28. !OCL LOOP_FFISSION_STRIPMINING[({ *n* | *c-level* })]

The LOOP_FFISSION_STRIPMINING specifier instructs to perform strip-mining optimization at the loop fission. There are two ways to specify the strip length: by specifying the length (*n*) directly or by specifying the cache level (*c-level*) to adjust the strip length to the size of level 1 cache or level 2 cache. *n* is a number from 2 to 100000000 as the strip length. Specify "L1" or "L2" for *c-level*. When "L1" is specified, the strip length is adjusted to the size of level 1 cache for the cache memory efficiency. When "L2" is specified, the strip length is adjusted to the size of level 2 cache as well. If the strip length is omitted, the compiler chooses the strip length automatically. This specifier is effective when the optimization control specifier LOOP_FFISSION_TARGET is specified in the optimization control line, and the -Kloop_fission option and the -O2 option or higher are set.

For more details, refer to "[9.1.2.11.1 Strip-Mining](#)".

An example is shown in the following.

Example:

```

REAL A(N),B(N),P(N),Q
!OCL LOOP_FISSION_TARGET
!OCL
LOOP_FISSION_STRIPMINING(256)
DO I=1,N
  Q = A(I) + B(I)
  ...
  P(I) = P(I) + Q
  ...
ENDDO

```

->

```

REAL A(N),B(N),P(N),Q
REAL TEMPARRAY_Q(256)
DO II=1,N,256
  DO I=II,MIN(N,II+255)
    TEMPARRAY_Q(I-II) = A(I) +
B(I)
    ...
  ENDDO
  DO I= II,MIN(N,II+255)
    P(I) = P(I) + TEMPARRAY_Q(I-
II)
    ...
  ENDDO
ENDDO

```

A loop is generated outside of the fissioned loop. The original loop iteration is fragmented by the strip length 256. The temporary array is generated by the compiler with loop fission. The number of elements of temporary array TEMPARRAY_Q is 256.

29. !OCL LOOP_NOFISSION_STRIPMINING

The LOOP_NOFISSION_STRIPMINING specifier instructs to suppress the strip-mining optimization for fissioned loops.

An example is shown in the following.

Example:

```

REAL A(N),B(N),P(N),Q
!OCL LOOP_FISSION_TARGET
!OCL LOOP_NOFISSION_STRIPMINING
DO I=1,N
  Q = A(I) + B(I)
  ...
  P(I) = P(I) + Q
  ...
ENDDO

```

30. !OCL LOOP_FISSION_TARGET[({CL | LS})]

The LOOP_FISSION_TARGET specifier instructs to perform loop fission for target loop. The CL or LS is an indicator for specifying the algorithm for loop fission. This specifier is effective when the -Kloop_fission option is set. If neither CL or LS is specified, CL is set by default.

For more details, refer to "[9.1.2.11 Loop Fission](#)".

!OCL LOOP_FISSION_TARGET(CL)

The LOOP_FISSION_TARGET(CL) specifier instructs to perform loop fission by the clustering algorithm for target loop. This optimization performs the loop fission to prioritize reduction of work array for temporary data transfer by the loop fission. This optimization may increase the translation time.

An example is shown in the following.

Example:

```

!OCL LOOP_FISSION_TARGET(CL)
DO I=1,N
  S1 = A(I) + B(I)
  S2 = C(I) + D(I)
  ...
  P(I) = S1 + Q(I)
  P(I) = S2 + Y(I)
  ...
ENDDO

```

Fission the loop to prioritize reduction of work array for temporary data transfer by the loop fission.

!OCL LOOP_FISSION_TARGET(LS)

The `LOOP_FISSION_TARGET(LS)` specifier instructs to perform loop fission by the local search algorithm for target loop. This optimization performs the loop fission to prioritize promotion of the software pipelining. This optimization may increase the translation time longer than the clustering algorithm.

An example is shown in the following.

Example:

```
!OCL LOOP_FISSION_TARGET(LS)
DO I=1,N
  S1 = A(I) + B(I)
  S2 = C(I) + D(I)
  ...
  P(I) = S1 + Q(I)
  X(I) = S2 + Y(I)
  ...
ENDDO
```

Fission the loop to prioritize promotion of the software pipelining.

31. !OCL LOOP_FISSION_THRESHOLD(*n*)

The `LOOP_FISSION_THRESHOLD` specifier directs the threshold to decide the granularity of loops after loop fission. *n* is a number from 1 to 100. This specifier is effective when the `-O2` option or higher is set and the `-Kloop_fission` option and the optimization control specifier `LOOP_FISSION_TARGET` are set.

Examples are shown in the following. In the example 2 which specifying a small value for the `LOOP_FISSION_THRESHOLD` specifier, the compiler fissions the loop into small granularity compared with example 1 and the number of fissioned loops increase.

Example 1:

```
!OCL LOOP_FISSION_TARGET
!OCL LOOP_FISSION_THRESHOLD(50)
DO I=1,N
  A1(I) = A1(I) + B1(I)
  ...
  A2(I) = A2(I) + B2(I)
  ...
  A3(I) = A3(I) + B3(I)
  ...
  A4(I) = A4(I) + B4(I)
  ...
ENDDO
```

->

```
DO I=1,N
  A1(I) = A1(I) + B1(I)
  ...
  A2(I) = A2(I) + B2(I)
  ...
ENDDO
DO I=1,N
  A3(I) = A3(I) + B3(I)
  ...
  A4(I) = A4(I) + B4(I)
  ...
ENDDO
```

Example 2:

```
!OCL LOOP_FISSION_TARGET
!OCL LOOP_FISSION_THRESHOLD(20)
DO I=1,N
  A1(I) = A1(I) + B1(I)
  ...
  A2(I) = A2(I) + B2(I)
  ...
  A3(I) = A3(I) + B3(I)
  ...
  A4(I) = A4(I) + B4(I)
  ...
ENDDO
```

->

```
DO I=1,N
  A1(I) = A1(I) + B1(I)
  ...
ENDDO
DO I=1,N
  A2(I) = A2(I) + B2(I)
  ...
ENDDO
DO I=1,N
  A3(I) = A3(I) + B3(I)
  ...
ENDDO
DO I=1,N
  A4(I) = A4(I) + B4(I)
  ...
ENDDO
```


32. !OCL LOOP_INTERCHANGE(*var1, var2, var3*...)

The LOOP_NOINTERCHANGE specifier instructs not to apply loop interchange. This allows DO loops to be interchanged in a specified order. However, loop interchange is not performed when interchanging is not possible because it may introduce different result.

Example:

<pre>!OCL LOOP_INTERCHANGE(I, J) DO I=1, M DO J=1, N A(I, J) = A(I, J) + B(J, I) ENDDO ENDDO</pre>	->	<pre>DO J=1, N DO I=1, M A(I, J) = A(I, J) + B(J, I) ENDDO ENDDO</pre>
------------------------------------------------------------------------------------------------------------	----	--------------------------------------------------------------------------------

When LOOP_INTERCHANGE is not specified, parallelization occurs using DO-variable I, but with LOOP_INTERCHANGE, loops are interchanged and parallelization occurs using DO-variable J. When the variable N is sufficiently larger than the variable M, the parallelization becomes more effective.

33. !OCL LOOP_NOINTERCHANGE

The LOOP_NOINTERCHANGE specifier instructs that nested DO loops interchange is not performed.

Example:

```
!OCL LOOP_NOINTERCHANGE
DO I=1, M
  DO J=1, N
    A(I, J) = A(I, J) + B(J, I)
  ENDDO
ENDDO
```

34. !OCL LOOP_NOFUSION

The LOOP_NOFUSION specifier instructs that fusion of the specified loop and the adjacent loops are not performed. An example is shown in the following.

Example:

```
!OCL LOOP_NOFUSION
DO I=1, N
  statement
ENDDO
DO J=1, N
  statement
ENDDO
DO K=1, N
  statement
ENDDO
```

Fusion of the loop I and J are not performed. Fusion of the loop J and K are performed.

35. !OCL LOOP_PART_SIMD

The LOOP_PART_SIMD specifier instructs to perform optimization which divides loops and partially uses SIMD extensions. An example is shown in the following.

Example:

```
!OCL LOOP_PART_SIMD
DO I=2, N
  A(I) = A(I)-B(I)+LOG(C(I)) ! SIMD extensions can be applied to
                             ! this statement
  D(I) = D(I-1)+A(I)         ! SIMD extensions cannot be applied to
                             ! this statement
ENDDO
```

[Optimized pseudo-code]

```
!OCL LOOP_PART_SIMD
DO I=2,N
  A(I) = A(I)-B(I)+LOG(C(I))  ! SIMD execution
ENDDO
DO II=2,N
  D(II) = D(II-1)+A(II)      ! no SIMD execution
ENDDO
```

The loop is divided and SIMD extensions are applied to the part of the divided loop.

36. !OCL LOOP_NOPART_SIMD

The LOOP_NOPART_SIMD specifier instructs to suppress optimization which divides loops and partially uses SIMD extensions. An example is shown in the following.

Example:

```
!OCL LOOP_NOPART_SIMD
DO I=2,N
  A(I) = A(I)-B(I)+LOG(C(I))
  D(I) = D(I-1)+A(I)
ENDDO
```

Neither loop dividing nor using SIMD extensions is applied to the loop.

37. !OCL LOOP_PERFECT_NEST

The LOOP_PERFECT_NEST specifier instructs to perform optimization which fissions the imperfectly nested loop into the perfectly nested loops.

An example is shown in the following.

Example:

```
!OCL LOOP_PERFECT_NEST
DO J=1,N
  A(J) = B(J)+1
  DO I=1,N
    C(J,I) = D(J,I)+A(J)
  ENDDO
ENDDO
```

->

```
DO J=1,N
  A(J) = B(J)+1
ENDDO
DO J=1,N
  DO I=1,N
    C(J,I) = D(J,I)+A(J)
  ENDDO
ENDDO
```

Fission the imperfectly nested loop I and J into the perfectly nested loops.

38. !OCL LOOP_NOPERFECT_NEST

The LOOP_NOPERFECT_NEST specifier instructs to suppress optimization which fissions the imperfectly nested loop into the perfectly nested loops.

An example is shown in the following.

Example:

```
!OCL LOOP_NOPERFECT_NEST
DO J=1,N
  A(J) = B(J)+1
  DO I=1,N
    C(J,I) = D(J,I)+A(J)
  ENDDO
ENDDO
```

Do not optimize the imperfectly nested loop I and J into the perfectly nested loops.

39. !OCL LOOP_VERSIONING

The LOOP_VERSIONING specifier instructs to perform optimization by the loop versioning.

The loop versioning is applied to only an innermost loop, and also the loop contains only one array with unknown data dependency.

The following shows an example:

Example:

```
DO I=1,N
!OCL LOOP_VERSIONING
  DO J=1,N
    A(J) = A(J+M) + B(J,I)
  ENDDO
ENDDO
```

The data dependency of array "A" is unknown at compile time because values of variable "N" and "M" are unknown at compile time.

When LOOP_VERSIONING is specified, the compiler generates two loops with IF-constructs for judging the values of variable "N" and "M" so that the data dependency of the array "A" can be analyzed at execution time.

If the array "A" judges no data dependency, SIMD may be promoted for the loop.

40. !OCL LOOP_NOVERSIONING

The LOOP_NOVERSIONING specifier instructs that loop versioning is suppressed.

The LOOP_NOVERSIONING specifier is effective only for the innermost loop.

The following shows an example:

Example:

```
DO I=1,N
!OCL LOOP_NOVERSIONING
  DO J=1,N
    A(J) = A(J+M) + B(J,I)
  ENDDO
ENDDO
```

41. !OCL MFUNC[(level)]

The MFUNC specifier instructs to perform optimization by converting intrinsic functions within DO loops, array assignment statement intrinsic functions, and operations within loops to multi-operation functions. Specify 1, 2, or 3 for level. Refer to the compiler option -Kmfunc for more information on level.

An example is shown in the following.

Example: When specified for loop

```
REAL(KIND=8), DIMENSION(10) :: A,B,C,D,E,F,G
!OCL MFUNC
DO I=1,10
  A(I) = LOG(B(I))
  E(I) = LOG(F(I))
ENDDO
D = LOG(C) + LOG(G)
...
```

LOG(B(I)) and LOG(F(I)) are converted into multi-operation functions.

42. !OCL NOMFUNC

The NOMFUNC specifier instructs that optimization using multi-operation functions is not performed.

An example is shown in the following.

Example:

```
REAL(KIND=8), DIMENSION(10) :: A,B,C,D,E,F,G
!OCL NOMFUNC
DO I=1,10
  A(I) = LOG(B(I))
  E(I) = LOG(F(I))
ENDDO
D = LOG(C) + LOG(G)
...
```

LOG(B(I)) and LOG(F(I)) are not converted into multi-operation functions.

43. !OCL NOALIAS

The NOALIAS specifier informs the compiler that no pointer variables share an address.

The compiler can identify that the pointer variable will not share memory area with other pointers during the compilation, although the content of the pointer is determined only at runtime. This promotes optimizations on the pointer variable. Note that this specifier may not promote the optimization when the association status of the pointer variable is changed in the loop.

An example is shown in the following.

Example:

```
REAL, DIMENSION(100), TARGET :: X
REAL, DIMENSION(:), POINTER :: A,B
A=>X(1:10)
B=>X(11:20)
!OCL NOALIAS
DO I=1,10
  B(I) = A(I) + 1.0
ENDDO
END
```

Note:

If there are two or more pointer variables referring the same memory area in a loop with the NOALIAS specifier, the execution result may be incorrect.

44. !OCL NOARRAYPAD(*arrayI*)

The NOARRAYPAD specifier instructs that padding is not performed for the specified array elements. This is valid when the compiler option

-Karraypad_const[=*N*] or -Karraypad_expr=*N* is set.

An example is shown in the following.

Example:

```
!OCL NOARRAYPAD(A)
PROGRAM MAIN
  INTEGER, DIMENSION(100,100) :: A
  COMMON /X/A
  PRINT *,UBOUND(A)
  PRINT *,SIZE(A,1),SIZE(A,2)
  CALL SUB
END PROGRAM
!OCL NOARRAYPAD(A)
SUBROUTINE SUB
  INTEGER, DIMENSION(100,100) :: A
  COMMON /X/A
  PRINT *,UBOUND(A)
  PRINT *,SIZE(A,1),SIZE(A,2)
END SUBROUTINE
```

Padding will not be performed to the array "A".

45. !OCL NORECURRENCE[(*array1* [, *array2*]...)]

The NORECURRENCE specifier instructs that each element of the arrays which are specified by this OCL is defined and referenced within the limits of iteration.

This enables some optimizations to be performed for the DO loop because the order of definitions and references become certain.

For example, the following optimizations are applied:

- Loop Slicing(Automatic Parallelization)
- SIMD
- Software Pipelining

Here, '*array*' is an array, for which loop slicing can be performed. Wildcards can be specified in '*array*'. When no array name is specified, this specifier is applied to all of the arrays within the target range. See Section "12.2.6.5 Wild Card Specification" for information on specifying wildcards.

In "Example: Code without NORECURRENCE specifier", the compiler cannot determine whether loop slicing can be performed because the index of array-A is another array element L(I). For this reason, loop slicing is not performed for this DO loop.

Example: Code without NORECURRENCE specifier

```
REAL, DIMENSION(10000) :: A, B
INTEGER, DIMENSION(10000) :: L
DO I=1, 10000
    A(L(I)) = A(L(I)) + B(I)
ENDDO
END
```

If it is certain that loop slicing will not affect array-A, specify NORECURRENCE as in "Example: Usage of NORECURRENCE specifier" so that DO loop is sliced.

Example: Usage of NORECURRENCE specifier

```
REAL, DIMENSION(10000) :: A, B
INTEGER, DIMENSION(10000) :: L
!OCL NORECURRENCE(A)
p DO I=1, 10000
p     A(L(I)) = A(L(I)) + B(I)
p ENDDO
END
```

Note:

Usage of this specifier is the programmer's responsibility.

When an element of the arrays which are specified by the OCL is defined or referenced in plural iterations, undesirable optimizations may be applied. And the results will be unpredictable.

46. !OCL NOVREC[(*array1* [, *array2*]...)]

The NOVREC specifier is used to prescribe that arrays of recurrence operation do not exist in loop. The arrays in loop are used SIMD instructions. But, the arrays in loop are not used SIMD instructions by kind of operation or loop structure.

The pointer array cannot be specified in *array*.

Example 1:

```
REAL A(20), B(20)
!OCL NOVREC
DO I=2, 10
    A(I) = A(I+N) + 1
    B(I) = B(I+M) + 2
ENDDO
```

The NOVREC specifier is used to prescribe that all arrays in loop is not recurrence operation. The operation of arrays A and B uses SIMD instructions.

Example 2:

```
REAL A(20),B(20)
!OCL NOVREC(A)
DO I=2,10
  A(I) = A(I+N) + 1
  B(I) = B(I) + 2
ENDDO
```

The NOVREC specifier is used to prescribe that array A whose data dependency is unknown is not recurrence operation. The operation of arrays A and B uses SIMD instructions.

Refer to Section "9.18.1 Example of NOVREC Specifier" for invalid usages of NOVREC specifier

47. !OCL PREEX

The PREEX specifier instructs that optimization via advance evaluation of invariant expressions is performed. An example is shown in the following.

Example:

```
!OCL PREEX
DO I=1,N
  IF (M(I)) THEN
    A(I) = A(I) / B(K)
  ENDIF
ENDDO
```

->

```
T = 1 / B(K)
DO I=1,N
  IF (M(I)) THEN
    A(I) = A(I) * T
  ENDIF
ENDDO
```

Advance evaluation of invariant expressions is performed.

48. !OCL NOPREEX

The NOPREEX specifier instructs to suppress optimizations which evaluate invariant expressions in advance. Note that a floating-point exception may occur when the -Kfp_relaxed option, -NRtrap option, and this specifier are valid. An example is shown in the following.

Example:

```
!OCL NOPREEX
DO I=1,N
  IF (M(I)) THEN
    A(I) = A(I) / B(K)
  ENDIF
ENDDO
```

Advance evaluation of invariant expressions is not performed.

49. !OCL PREFETCH

```
!OCL NOPREFETCH
!OCL PREFETCH_CACHE_LEVEL(c-level)
!OCL PREFETCH_INFER
!OCL PREFETCH_NOINFER
!OCL PREFETCH_ITERATION(n)
!OCL PREFETCH_ITERATION_L2(n)
```

This instructs to perform the automatic prefetch function of the compiler. The compiler automatically determines the most appropriate position to insert and generates the prefetch instructions.

50. !OCL PREFETCH_CONDITIONAL

The PREFETCH_CONDITIONAL specifier instructs to generate the prefetch instructions to array data used in blocks in IF constructs and CASE constructs.

An example is shown in the following.

Example:

```
!OCL PREFETCH_CONDITIONAL
DO I=1,N
  IF(X(I) == 1.0) THEN
    A(I) = B(I)
  ENDF
ENDDO
```

The PREFETCH_CONDITIONAL specifier generates the prefetch instructions to array data used in blocks in IF constructs.

51. !OCL PREFETCH_NOCONDITIONAL

The PREFETCH_NOCONDITIONAL specifier instructs not to generate the prefetch instructions to array data used in blocks in IF constructs and CASE constructs.

An example is shown in the following.

Example:

```
!OCL PREFETCH_NOCONDITIONAL
DO I=1,N
  IF(X(I) == 1.0) THEN
    A(I) = B(I)
  ENDF
ENDDO
```

The PREFETCH_NOCONDITIONAL specifier does not generate the prefetch instructions to array data used in blocks in IF constructs.

52. !OCL PREFETCH_INDIRECT

The PREFETCH_INDIRECT specifier instructs to generate the prefetch instruction to array data accessed indirectly (list access) in loops.

An example is shown in the following.

Example:

```
!OCL PREFETCH_INDIRECT
DO I=1,N
  A(INDX(I)) = B(INDX(I))
ENDDO
```

The PREFETCH_INDIRECT specifier generates the prefetch instruction to array data indirectly accessed (list access) in loops.

53. !OCL PREFETCH_NOINDIRECT

The PREFETCH_NOINDIRECT specifier instructs not to generate the prefetch instruction to array data accessed indirectly (list access) in loops.

An example is shown in the following.

Example:

```
!OCL PREFETCH_NOINDIRECT
DO I=1,N
  A(INDX(I)) = B(INDX(I))
ENDDO
```

The PREFETCH_NOINDIRECT specifier does not generate the prefetch instruction to array data indirectly accessed (list access) in loops.

54. !OCL PREFETCH_LINE(*n*)

The PREFETCH_LINE specifier instructs to target data after *n* cache line(s) for generating prefetch instruction to the first level cache. Decimal from 1 to 100 can be specified as *n*.

An example is shown in the following.

Example:

```
!OCL PREFETCH_LINE(20)
DO I=1,N
  A(I) = B(I)
ENDDO
```

The PREFETCH_LINE specifier targets data after 20 cache lines for generating prefetch instructions to the first level cache.

55. !OCL PREFETCH_LINE_L2(*n*)

The PREFETCH_LINE_L2 specifier instructs to target data after *n* cache lines(s) for generating prefetch instruction to the second level cache. Decimal from 1 to 100 can be specified as *n*.

An example is shown in the following.

Example:

```
!OCL PREFETCH_LINE_L2(20)
DO I=1,N
  A(I) = B(I)
ENDDO
```

The PREFETCH_LINE_L2 specifier targets data after 20 cache lines for generating prefetch instructions to the second level cache.

56. !OCL PREFETCH_READ (*name*[,level={ 1 | 2 }] [,strong={ 0 | 1 }])

The PREFETCH_READ specifier instructs to generate prefetch instructions for referenced data.

In *name*, specify the data (array element or array section) referenced in the program. Expressions can also be specified in subscripts. When the array section is specified for *name*, specify the array section by using subscript-triplet without stride. level={ 1 | 2 } directs which cache level is to have data prefetched. level=1 is default.

Specify whether to strong prefetch by specifying strong={ 0 | 1 }. If strong=0 is specified, strong prefetch is not performed. If strong=1 is specified, strong prefetch is performed. The strong=1 is default.

For data that is referenced and defined, use PREFETCH_WRITE.

57. !OCL PREFETCH_WRITE (*name*[,level={ 1 | 2 }] [,strong={ 0 | 1 }])

The PREFETCH_WRITE specifier instructs to generate prefetch instructions for defined data.

In *name*, specify the data (array element or array section) defined in the program. Expressions can also be specified in subscripts. If specify the array section for *name*, specify the array section by using subscript-triplet without stride. level={ 1 | 2 } directs which cache level is to have data prefetched. The level=1 is default.

Specify whether to strong prefetch by specifying strong={ 0 | 1 }. If strong=0 is specified, strong prefetch is not performed. If strong=1 is specified, strong prefetch is performed. The strong=1 is default.

The following shows an example:

Example 1: To generate prefetch instruction for A(INDX(I+16))

```
DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)))
  T = T + A(INDX(I))
ENDDO
```

Example 2: To generate prefetch instruction to the second level cache for A(INDX(I+16))

```
DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)),level=2)
  T = T + A(INDX(I))
ENDDO
```

Moreover, when the automatic PREFETCH function of the compiler and it wants to control it exclusively because of the cache use efficiency, it is possible to achieve it by combining OCL as follows.

Example 3: To generate prefetch instruction without using the strong prefetch for A(INDX(I+16))

```
DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)),strong=0)
T = T + A(INDX(I))
ENDDO
```

Example 4: To generate prefetch instruction for INDX(I+32) and A(INDX(I+16))

```
DO I = 1,N
!OCL PREFETCH_READ(INDX(I+32))
!OCL PREFETCH_READ(A(INDX(I+16)))
T = T + A(INDX(I))
ENDDO
```

When controlling the use of the automatic prefetch provided by the compiler to improve cache utilization, use multiple OCL statements as shown below.

Example 5: Specify combination of OCL

```
!OCL NOPREFETCH
DO I = 1,N,4
!OCL PREFETCH_READ(A(I+16))
!OCL PREFETCH_READ(C(I+16))
!OCL PREFETCH_WRITE(B(I+16))
B(I) = A(I) + C(I)
B(I+1) = A(I+1) + C(I+1)
B(I+2) = A(I+2) + C(I+2)
B(I+3) = A(I+3) + C(I+3)
ENDDO
```

The name can also specify the array section, as in Example 6.

The PREFETCH_READ specifier with array section as an argument is expanded into a loop that calls PREFETCH_READ with an array element as an argument.

If the array elements to be prefetched are sequential, the PREFETCH_READ is not performed for every iteration and the cache-line-conscious prefetch is output. Example 6 is an example of a usage that can be expected to improve performance if the following conditions are met:

- The thread parallelism specification of OpenMP is specified for the parent loop including the PREFETCH_READ specifier and the succeeding loop of DO variable I (so that the PREFETCH_READ specifier is expected to work in the same thread as the data referenced by the DO variable I).
- In the loop of the DO variable I referring to the data specified the PREFETCH_READ specifier, the number of instructions is large and the number of iterations is small.
- The data specified for the PREFETCH_READ specifier is used in the next iterations of the loop of the DO variable I.

Example 6: Specify the array section

```
!$OMP PARALLEL DO
DO J=1,N
:
!OCL PREFETCH_READ(A(1:IDX(J),J+1),level=2,strong=1)
:
DO I=1,IDX(J) ! loop contains many instructions
:
... = A(I,J) + ...
:
ENDDO
:
ENDDO
!$OMP END PARALLEL DO
```

Note the following specifications for using the array section:

- It cannot be used with the automatic parallelization (the `-Kparallel` or `-Kparallel_strong` option). Use with OpenMP (`-Kopenmp`).
- If the `PREFETCH_READ` or `PREFETCH_WRITE` specifier with array section as an argument is described between thread parallelism specification such as `!$OMP PARALLEL DO` and its target loop, the warning message is outputted and the specifier is invalid. The OpenMP specifications subject to the warning are as follows:
 - `DO`, `PARALLEL DO`, `DO SIMD`, `PARALLEL DO SIMD`, `WORKSHARE`, `PARALLEL WORKSHARE`, `TASKLOOP`, `TASKLOOP SIMD`
- The `PREFETCH_READ` or `PREFETCH_WRITE` specifier with array section as an argument is expanded into loops. Therefore, optimizations that have an impact on performance may stop. For example, by inserting the specifier in the middle of a perfectly nested loop, optimization such as collapsing do not work.

58. `!OCL PREFETCH_SEQUENTIAL[({AUTO | SOFT})]`

The `PREFETCH_SEQUENTIAL` specifier instructs that prefetch instructions are generated for array data that is accessed sequentially. If neither `AUTO` nor `SOFT` is specified, `AUTO` is set by default.

`!OCL PREFETCH_SEQUENTIAL(AUTO)`

The `PREFETCH_SEQUENTIAL(AUTO)` instructs that the compiler automatically selects whether to use hardware-prefetch or to generate prefetch instructions for array data that is accessed sequentially in the loop.

The following shows an example:

Example:

```
!OCL PREFETCH_SEQUENTIAL(AUTO)
DO I=1,N
  A(I) = B(I)
ENDDO
```

Hardware prefetch is used for the target loop rather than generating prefetch instructions.

`!OCL PREFETCH_SEQUENTIAL(SOFT)`

The `PREFETCH_SEQUENTIAL(SOFT)` specifier instructs that the compiler does not use hardware-prefetch, but rather generates prefetch instructions for array data that is accessed sequentially in the loop.

The following shows an example:

Example:

```
!OCL PREFETCH_SEQUENTIAL(SOFT)
DO I=1,N
  A(I) = B(I)
ENDDO
```

Prefetch instructions are generated for the target loop rather than using hardware-prefetch.

59. `!OCL PREFETCH_NOSEQUENTIAL`

The `PREFETCH_NOSEQUENTIAL` specifier instructs that prefetch instructions are not generated for array data that is accessed sequentially.

The following shows an example:

Example:

```
!OCL PREFETCH_NOSEQUENTIAL
DO I=1,N
  A(I) = B(I)
ENDDO
```

Prefetch instructions are not generated for the target loop.

60. !OCL PREFETCH_STRIDE[({SOFT | HARD_AUTO | HARD_ALWAYS})]

Prefetch instructions are generated for array data that is accessed with a stride larger than the cache line size used in a loop. If the argument is omitted, SOFT is set by default.

!OCL PREFETCH_STRIDE(SOFT)

Prefetch instructions are generated for array data that is accessed with a stride larger than the cache line size used in a loop.

The following shows an example:

Example:

```
!OCL PREFETCH_STRIDE(SOFT)
DO I=1,N,K
  A(I) = B(I)
ENDDO
```

Prefetch instructions are generated for the target loop.

!OCL PREFETCH_STRIDE(HARD_AUTO)

Prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop. The `-Kprefetch_stride=hard_auto` option specifies to perform the prefetch using the hardware stride prefetcher. This specifier sets the stride prefetcher to prefetch only data that is not on cache.

The following shows an example:

Example:

```
!OCL PREFETCH_STRIDE(HARD_AUTO)
DO I=1,N,K
  A(I) = B(I)
ENDDO
```

Hardware stride prefetcher is used for the target loop rather than generating prefetch instructions. Stride prefetcher is set to prefetch only data that is not on cache.

!OCL PREFETCH_STRIDE(HARD_ALWAYS)

Prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop. The `-Kprefetch_stride=hard_always` option specifies to perform the prefetch using the hardware stride prefetcher. In contrast with the `-Kprefetch_stride=hard_auto` option, this option sets the stride prefetcher to always prefetch.

The following shows an example:

Example:

```
!OCL PREFETCH_STRIDE(HARD_ALWAYS)
DO I=1,N,K
  A(I) = B(I)
ENDDO
```

Hardware stride prefetcher is used for the target loop rather than using prefetch instructions. Stride prefetcher is set to always prefetch.

!OCL PREFETCH_NOSTRIDE

Prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop.

The following shows an example:

Example:

```
!OCL PREFETCH_NOSTRIDE
DO I=1,N,K
  A(I) = B(I)
ENDDO
```

Prefetch instructions are not generated for the target loop.

61. !OCL PREFETCH_STRONG

The PREFETCH_STRONG specifier instructs that the prefetch instruction for the first level cache is to be the strong prefetch.

The following shows an example:

Example:

```
!OCL PREFETCH_STRONG
DO I=1,N
  A(I) = B(I)
ENDDO
```

Prefetch instruction for the first level cache generated for the target loop will be strong prefetch.

62. !OCL PREFETCH_NOSTRONG

The PREFETCH_NOSTRONG specifier instructs that the prefetch instruction generated for the first level cache will not be strong prefetch.

The following shows an example:

Example:

```
!OCL PREFETCH_NOSTRONG
DO I=1,N
  A(I) = B(I)
ENDDO
```

The prefetch instruction for the first level cache generated for the target loop will not be strong prefetch.

63. !OCL PREFETCH_STRONG_L2

The PREFETCH_STRONG_L2 specifier instructs that the prefetch instructions generated for the second level cache are to be strong prefetch.

The following shows an example:

Example:

```
!OCL PREFETCH_STRONG_L2
DO I=1,N
  A(I) = B(I)
ENDDO
```

The prefetch instruction for the second level cache generated for the target loop will be strong prefetch.

64. !OCL PREFETCH_NOSTRONG_L2

The PREFETCH_NOSTRONG_L2 specifier instructs that the prefetch instructions generated for the second level cache are not to be strong prefetch.

The following shows an example:

Example:

```
!OCL PREFETCH_NOSTRONG_L2
DO I=1,N
  A(I) = B(I)
ENDDO
```

The prefetch instruction for the second level cache generated for the target loop will not be strong prefetch.

65. !OCL PRELOAD

The PRELOAD specifier instructs to perform speculative execution of load instructions.

Example:

```
!OCL PRELOAD
DO I=1,N
```

```

    IF (M(I) > 0) THEN
      A(I) = B(I) + C(I)
    ENDIF
  ENDDO

```

The optimization for speculative execution of load instructions is performed.

66. !OCL NOPRELOAD

The NOPRELOAD specifier instructs to suppress speculative execution of load instructions.

Example:

```

!OCL NOPRELOAD
DO I=1,N
  IF (M(I) > 0) THEN
    A(I) = B(I) + C(I)
  ENDIF
ENDDO

```

The optimization for speculative execution of load instructions is not performed.

67. !OCL SCACHE_ISOLATE_WAY(L2=*n1* [,L1=*n2*]) !OCL END_SCACHE_ISOLATE_WAY

See Section "9.17.2.1 Software Control with Optimization Control Lines" for details.

68. !OCL SCACHE_ISOLATE_ASSIGN(*array1* [,*array2*...]) !OCL END_SCACHE_ISOLATE_ASSIGN

See Section "9.17.2.1 Software Control with Optimization Control Lines" for details.

69. !OCL SIMD[({ALIGNED | UNALIGNED})]

The SIMD specifier instructs to perform SIMD optimization. It depends on the type of operation and the loop structure that SIMD optimization actually is performed or not.

The following shows an example:

Example:

```

REAL(KIND=8), DIMENSION(10) :: A, B
!OCL SIMD
DO I=1,10
  A(I) = A(I) + B(I)
ENDDO

```

SIMD optimization is performed.

!OCL SIMD ({ALIGNED | UNALIGNED})

This is equivalent to !OCL SIMD.

The parameters "ALIGNED" and "UNALIGNED" are deprecated for this product and left for backward compatibility with previous products only.

70. !OCL NOSIMD

The NOSIMD specifier instructs that SIMD optimization is not performed.

Example:

```

REAL(KIND=8), DIMENSION(10) :: A, B
!OCL NOSIMD
DO I=1,10
  A(I) = A(I) + B(I)
ENDDO

```

SIMD optimization is not performed.

71. !OCL SIMD_LISTV[({ALL | THEN | ELSE})]

The SIMD_LISTV specifier instructs to perform list vector conversion to the execution statements of the specified IF construct. ALL, THEN or ELSE is specifier for controlling the range of list vector conversion optimization. When the -O2 option or higher is set, and the -Ksimd=2 option or the same effect by the SIMD specifier is performed, this specifiers is effective. If neither ALL, THEN, nor ELSE is specified, ALL is set by default.

See Section "[9.1.1.7.3 List Vector Conversion](#)" for the list vector conversion.

!OCL SIMD_LISTV(THEN)

The list vector conversion is performed to the statements in the subsequent block of the specified IF THEN statement.

The following shows an example:

Example:

```
!OCL SIMD
  DO I=1,N
!OCL SIMD_LISTV(THEN)
  IF (A(I).EQ.0.0) THEN
    A(I) = A(I)+B(I)
  ELSE
    A(I) = A(I)+C(I)
  ENDIF
ENDDO
```

!OCL SIMD_LISTV(ELSE)

The list vector conversion is performed to the statements in the subsequent block of the specified ELSE statement.

The following shows an example:

Example:

```
!OCL SIMD
  DO I=1,N
!OCL SIMD_LISTV(ELSE)
  IF (A(I).EQ.0.0) THEN
    A(I) = A(I)+B(I)
  ELSE
    A(I) = A(I)+C(I)
  ENDIF
ENDDO
```

!OCL SIMD_LISTV(ALL)

The list vector conversion is performed to the statements in the blocks of the specified IF THEN statement and ELSE statement.

The following shows an example:

Example:

```
!OCL SIMD
  DO I=1,N
!OCL SIMD_LISTV(ALL)
  IF (A(I).EQ.0.0) THEN
    A(I) = A(I)+B(I)
  ELSE
    A(I) = A(I)+C(I)
  ENDIF
ENDDO
```

72. !OCL SIMD_NOREDUNDANT_VL

The SIMD_NOREDUNDANT_VL specifier instructs to invalidate SIMD with redundant executions for the SIMD width. For details about SIMD with redundant executions for the SIMD width, see section "[9.1.1.7.4 SIMD with Redundant Executions for the SIMD Width](#)".

The following shows an example. This specifier is used to suppress the application to specific loops SIMD with redundant executions for the SIMD width.

Example:

```

SUBROUTINE SUB(A,B,C)
  ...
!OCL SIMD_NOREDUNDANT_VL
  DO I=1,M
    A(I)= B(I)+C(I)
  END DO
  ...
END SUBROUTINE

```

73. !OCL SIMD_USE_MULTIPLE_STRUCTURES

The SIMD_USE_MULTIPLE_STRUCTURES specifier instructs to use the SVE Load Multiple Structures instructions and the Store Multiple Structures instructions when using SIMD extensions.

This specifier is effective when the -Ksimd[={ 1|2|auto}] option or the simd specifier is effective, and the -KSVE option is set.

The following shows an example:

Example:

```

COMPLEX*16 A(N), B(N), C(N)
REAL*8 Y(N), X(4,N)
...
!OCL SIMD_USE_MULTIPLE_STRUCTURES
DO I=1,N
  A(I) = B(I) * C(I)
ENDDO
...
!OCL SIMD_USE_MULTIPLE_STRUCTURES
DO I=1,N
  Y(I)=X(1,I)+X(2,I)+X(3,I)+X(4,I)
ENDDO

```

The SVE Load Multiple Structures instruction and the Store Multiple Structures instruction are used in this loop.

74. !OCL SIMD_NOUSE_MULTIPLE_STRUCTURES

The SIMD_NOUSE_MULTIPLE_STRUCTURES specifier instructs not to use the SVE Load Multiple Structures instructions and the Store Multiple Structures instructions when using SIMD extensions.

The following shows an example:

Example:

```

COMPLEX*16 A(N), B(N), C(N)
REAL*8 Y(N), X(4,N)
...
!OCL SIMD_NOUSE_MULTIPLE_STRUCTURES
DO I=1,N
  A(I) = B(I) * C(I)
ENDDO
...
!OCL SIMD_NOUSE_MULTIPLE_STRUCTURES
DO I=1,N
  Y(I)=X(1,I)+X(2,I)+X(3,I)+X(4,I)
ENDDO

```

The SVE Load Multiple Structures instruction and the Store Multiple Structures instruction are not used in this loop.

75. !OCL STRIPING[(n)]

The STRIPING specifier instructs that the striping optimization is performed. The striped length (number of expansions) can be specified for *n*.

n is an integer value from 2 to 100. If a value was not specified for n , 2 is set by default. When the iteration count of a loop in the source program is known, the expansion number determined by the compiler will be used if a number that exceeds the iteration count is specified for n .

Example 1:

```
!OCL STRIPING
  DO I=1,N
    statement
  ENDDO
```

This specifies that loop striping is to be performed on the statement using the striping length determined by the compiler.

Example 2:

```
!OCL STRIPING(4)
  DO I=1,N
    statement
  ENDDO
```

This specifies that loop striping is to be performed on the statement using the striping length 4.

Example 3:

```
!OCL STRIPING(8)
  DO I=1,4
    statement
  ENDDO
```

The specified stripe length (8) exceeds the loop iteration count (4), so the striped length automatically determined by the compiler will be used.

Example 4:

```
!OCL STRIPING(4)
  SUBROUTINE SUB
    ...
    ...
  DO I=1,N
    statement
  ENDDO
  DO I=1,N
    statement
  ENDDO
```

If this is specified at the start of the program unit, all the DO loops in the program are targeted.

76. !OCL NOSTRIPING

The NOSTRIPING specifier instructs that the loop striping optimization is suppressed.

Example:

```
!OCL NOSTRIPING
  DO I=1,N
    statement
  ENDDO
```

Loop striping is not performed.

77. !OCL SWP

The SWP specifier instructs that software pipelining is performed. This specifier (or indicator) is valid only when the -O2 option or above is in effect at compilation.

The following shows an example:

Example:

```
!OCL SWP
  DO I=1,N
    statement
  ENDDO
```

78. !OCL SWP_FREG_RATE (*n*)
!OCL SWP_IREG_RATE (*n*)
!OCL SWP_PREG_RATE (*n*)

Specifies the rate (percentage) about the following registers that can be used by software pipelining.

- Floating-point register and SVE vector register
- Integer register
- SVE predicate register

When specifying to SWP_FREG_RATE specifier, the rate about floating-point register and SVE vector register is adjusted.

When specifying to SWP_IREG_RATE specifier, the rate about integer register is adjusted.

When specifying to SWP_PREG_RATE specifier, the rate about SVE predicate register is adjusted.

The number from 1 to 1000 following specifiers instructs the rate (percentage) of the number of registers which is assumed to be usable by the software pipelining. This specifier is effective only if the -O2 option or higher is set.

An example is shown in the following.

Example:

```
!OCL SWP_FREG_RATE(120)
!OCL SWP_IREG_RATE(150)
!OCL SWP_PREG_RATE(80)
  DO I=1,N
    . . .
  ENDDO
```

Perform the optimization of the software pipelining which adjusts the condition about the number of each kind of register for the target loop.

79. !OCL SWP_POLICY({AUTO|SMALL|LARGE})

The SWP_POLICY specifier instructs a policy to select an instruction scheduling algorithm used in software pipelining.

SWP_POLICY(AUTO)

The compiler automatically selects a fit algorithm for each loop.

SWP_POLICY(SMALL)

An algorithm fit for a small loop, such as a loop with low register pressure, is used.

SWP_POLICY(LARGE)

An algorithm fit for a large loop, such as a loop with high register pressure, is used.

80. !OCL SWP_WEAK

The SWP_WEAK specifier instructs the compiler to adjust software pipelining to make smaller overlapping of the instructions. This specifier is effective only if the -O2 option or higher is set.

An example is shown in the following.

Example:

```
!OCL SWP_WEAK
  DO I=1,N
    . . .
  ENDDO
```

Perform the optimization of the software pipelining which adjust software pipelining for the target loop, and make smaller overlapping of the instructions.

81. !OCL NOSWP

The NOSWP instructs that software pipelining is suppressed.

The following shows an example:

Example:

```
!OCL NOSWP
DO I=1,N
    statement
ENDDO
```

82. !OCL UNROLL[*(n)*]

The UNROLL specifier instructs the number of unrolls for loop unrolling at *n*. This is described immediately before a DO construct with a DO variable. *n* is an integer value from 2 to 100. If a value was not specified for *n*, the compiler will automatically determine the value.

Further, if the iteration count for a loop is known, the known number of iterations is given priority, even if a number is specified in *n* that exceeds the iteration count.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control.

When the optimization control line has not been specified, the compiler will determine the loop unrolling expansion count based on the iteration count, number and type of instructions in the loop, and the data type being used.

The following shows an example:

Example 1:

```
SUBROUTINE SUB
...
!OCL UNROLL(8)
DO I=1,N
    statement
ENDDO
...
END SUBROUTINE
```

This specifies that the statement is to be unrolled 8 times.

Example 2:

```
SUBROUTINE SUB
...
!OCL UNROLL(80)
DO I=1,50
    statement
ENDDO
...
END SUBROUTINE
```

The specified expansion count (80) exceeds the loop iteration count (50), so the expansion count will be deemed to be 50.

83. !OCL UNROLL('full')

If 'full' is specified as the loop expansion count, instructions will be expanded as many times as the iteration count in the source program. Optimization is not performed if the iteration count is unknown.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control.

The following shows an example:

Example 1:

```
SUBROUTINE SUB
...
!OCL UNROLL('full')
  DO I=1,10
    statement
  ENDDO
...
END SUBROUTINE
```

This specifies that the statement is to be unrolled 10 times, the same as the loop iteration count.

Example 2:

```
SUBROUTINE SUB
...
!OCL UNROLL('full')
  DO K=1,2
!OCL UNROLL('full')
  DO J=1,2
!OCL UNROLL('full')
    DO I=1,2
      statement
    ENDDO
  ENDDO
ENDDO
...
END SUBROUTINE
```

This specifies that the statement is to be unrolled 2*2*2 times.

84. !OCL NOUNROLL

The NOUNROLL specifier instructs that loop unrolling optimization is suppressed.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control.

The following shows an example:

Example:

```
SUBROUTINE SUB
...
!OCL NOUNROLL
  DO I=1,N
    statement
  ENDDO
...
END SUBROUTINE
```

Loop unrolling is not performed.

85. !OCL UNROLL_AND_JAM[(n)]

The UNROLL_AND_JAM specifier instructs the compiler to perform the optimization of loop unrolling for the corresponding loop.

The number from 2 to 100 following the specifier instructs the upper bound on the number of loops to be unrolled. If the number of the iteration is omitted, the compiler automatically determines a suitable value. However, the optimization is suppressed in the following case:

- Assuming that the optimization is not effective.
- Assuming that there is a data dependency over iterations of the loops.

Note that the UNROLL_AND_JAM specifier targets only the loop immediately after the optimization control line. If the loop is the innermost, the unroll-and-jam is not applied.

This specifier is effective only if the -O2 option or higher is set.

The following shows examples:

Example 1:

<pre>!OCL UNROLL_AND_JAM(2) DO J=1, 128 DO I=1, 128 A(I, J) = B(I, J) + B(I, J +1) ... END DO END DO</pre>	->	<pre>DO J=1, 128, 2 DO I=1, 128 A(I, J) = B(I, J) + B(I, J+1) A(I, J+1) = B(I, J+1) + B(I, J +2) ... END DO END DO</pre>
------------------------------------------------------------------------------------------------------------------------	----	------------------------------------------------------------------------------------------------------------------------------------------

Unroll-and-jam is applied to the loop whose control variable is J.

Example 2:

```
DO J=1, 128
!OCL UNROLL_AND_JAM(2)
  DO I=1, 128
    A(I, J) = B(I, J) + B(I, J+1)
  END DO
END DO
```

Unroll-and-jam is not applied because the specified loop is the innermost.

Example 3:

```
!OCL UNROLL_AND_JAM(2)
DO J=1, 128
  DO I=1, 127
    A(I, J) = A(I+1, J-1) + B(I, J)
  END DO
END DO
```

Unroll-and-jam is not applied because an array "A" has a data dependency over iterations of the loops.

86. !OCL UNROLL_AND_JAM_FORCE[(n)]

The UNROLL_AND_JAM_FORCE specifier instructs that the unroll-and-jam optimization is performed to the corresponding loop.

The execution result is not guaranteed if the specifier is used incorrectly. See Section "9.18.3 Example of UNROLL_AND_JAM_FORCE Specifier".

The number of unrolls is specified as *n*. *n* is an integer value from 2 to 100. If a value was not specified for *n*, the compiler will automatically determine the value.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control. If the loop is the innermost, the unroll-and-jam is not applied. This specifier is effective only if the -O2 option or higher is set.

The following shows examples:

Example 1:

<pre>!OCL UNROLL_AND_JAM_FORCE(2) DO J=1, 128 DO I=1, 128 A(I, J) = B(I, J) + B(I, J+1) ... END DO END DO</pre>	->	<pre>DO J=1, 128, 2 DO I=1, 128 A(I, J) = B(I, J) + B(I, J+1) A(I, J+1) = B(I, J+1) + B(I, J+2) ... END DO END DO</pre>
-----------------------------------------------------------------------------------------------------------------------------	----	-----------------------------------------------------------------------------------------------------------------------------------------

Unroll-and-jam is applied to the loop whose do variable is J.

Example 2:

```
DO J=1, 128
!OCL UNROLL_AND_JAM_FORCE(2)
DO I=1, 128
  A(I, J) = B(I, J) + B(I, J+1)
END DO
END DO
```

Unroll-and-jam is not applied because the specified loop is the innermost.

87. !OCL NOUNROLL_AND_JAM

The nounroll_and_jam specifier instructs the compiler not to apply unroll-and-jam.

An example is shown in the following.

Example:

```
!OCL NOUNROLL_AND_JAM
DO J=1, 128
DO I=1, 128
  A(I, J) = B(I, J) + B(I, J+1)
END DO
END DO
```

Unroll-and-jam is not applied.

88. !OCL UNSWITCHING

The UNSWITCHING specifier instructs loop unswitching to IF construct. This specifier must be described immediately before IF construct which is invariant in a loop. This specifier is not effective in the case of specified other than the above.

The following shows an example:

Example 1:

```
SUBROUTINE SUB(A,B,C,X,N)
REAL(KIND=8), DIMENSION(N) :: A,B,C
INTEGER(KIND=8) :: X
...
DO I=1,N
!OCL UNSWITCHING
  IF (X == 0) THEN
    A(I) = B(I)
  ELSE
    A(I) = C(I)
  ENDIF
ENDDO
END SUBROUTINE
```

Loop unswitching to IF construct is performed.

Note that IF construct without this specifier is not the target of loop unswitching. The following shows an example:

Example 2:

```
SUBROUTINE SUB(A,B,C,D,X,N)
REAL(KIND=8), DIMENSION(N) :: A,B,C,D
INTEGER(KIND=8) :: X
...
DO I=1,N
  IF (X == 0) THEN
    A(I) = B(I)
!OCL UNSWITCHING
  ELSE IF (X == 1) THEN
    A(I) = C(I)
  ELSE
```

```

        A(I) = D(I)
    ENDIF
ENDDO
END SUBROUTINE

```

Only IF construct with this specifier is the target of loop unswitching.

The memory consumption and the compilation time may increase drastically when loops to which loop unswitching is applied include a lot of execution statements.

89. !OCL ZFILL[(*n*)]

The zfill optimization speeds up write operations for array data that is only written in a loop, by using an instruction that allocates space on the cache for writing without loading data from the memory. The zfill optimization works on the data *n* blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and *n* is an integer value between 1 and 100. If a value is not specified for *n*, the compiler will automatically determine a value.

For details about zfill, see "[9.1.2.5 zfill](#)".

The following shows an example:

Example 1:

```

SUBROUTINE SUB(A,N)
...
!OCL ZFILL
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE

```

This specifies to improve the cache utilization for the data on certain blocks ahead, and the number is determined by the compiler.

Example 2:

```

SUBROUTINE SUB(A,N)
...
!OCL ZFILL(1)
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE

```

This specifies to improve the cache utilization for the data on 1 block ahead.

90. !OCL NOZFILL

The NOZFILL specifier instructs not to apply the zfill optimization.

The following shows an example:

Example:

```

SUBROUTINE SUB(A,N)
...
!OCL NOZFILL
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE

```

This specifies that the zfill optimization will not be performed.

9.11 Effects of Allocating on Stack

The following shows the effects of allocating stack using the compiler option `-Kautoobjstack`, `-Kauto`, `-Kthreadsafe`, or `-Ktemparraystack`. See Section "2.2 Compiler Options" for information on these options.

If the maximum stack area size is smaller than the memory required by the program, stack area may not be allocated, and the program may terminate abnormally. In this case, increase the maximum stack area size, or disable the above compiler options.

To view and change the maximum, use the `limit` command on (csh) and the `ulimit` command on (sh). In addition, when executing as a batch request using Job Operation Software, the maximum size of the corresponding stack area of the batch queue (Per-process stack size limit) must be increased.

For thread parallelized programs, specify the required thread stack size with the environment variable `OMP_STACKSIZE` or `THREAD_STACK_SIZE`. See section "J.2.2 Execution Process (Automatic Parallelization)" or "J.3.2 Execution Process (OpenMP Parallelization)" for details.

Caution should be taken for specifying the compiler option `-Knoauto` to an OpenMP program. See Section "2.2.3 Description of Compiler Options" for information on the compiler option `-Knoauto`.

The following examples show the effects of specifying each of the options:

Example 1: When the compiler options `-Kautoobjstack` and `-Kauto` are specified

```
$ cat a.f90
SUBROUTINE SUB(N)
REAL(KIND=8),DIMENSION(N) :: A
A = 1.
PRINT*, A(N)
END SUBROUTINE
CALL SUB(2000000)
END
$ ulimit -s
8192
$ ./a.out
Segmentation Fault(core dumped)
```

Program terminated abnormally due to an insufficient stack area size.

To avoid an abnormal termination, do one of the following:

- Increase the upper limit of stack size and execute.
- Compile specifying the compiler options `-Knoauto` and/or `-Knoautoobjstack`.

```
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ ./a.out
1.0000000000000000
```

The program ended normally because the stack size was increased.

Example 2: When the compiler options `-Ktemparraystack` and `-Kauto` are specified

```
$ cat c.f90
SUBROUTINE SUB(A, N1, N2, M1, M2)
REAL(KIND=8),DIMENSION(M2) :: A
A(N1:N2) = A(M1:M2) + 1.
PRINT*, A(1)
END SUBROUTINE
REAL(KIND=8),DIMENSION(2000000) :: A
A = 0.
CALL SUB(A,1,1999999,2,2000000)
END
$ ulimit -s
8192
```

```
$ ./a.out
Segmentation Fault(core dumped)
```

Program terminated abnormally due to an insufficient stack area size.

To avoid an abnormal termination, do one of the following:

- Increase the upper limit of stack size and execute.
- Compile specifying the compiler options `-Knoauto` and/or `-Knotemparraystack`.

```
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ ./a.out
1.0000000000000000
```

The program ended normally because the stack size was increased.

9.12 Effects of Applying Optimization to Change Shape of Array

Increasing the maximum array size using the compiler option `"-Karraypad_const[=N]"` or `"-Karraypad_expr=N"` will affect the result of the functions `UBOUND` and `SIZE`. Further, if the `"SUBCHK"` functionality is used, it will be as if the maximum size has been specified in the source program.

When using the compiler options `"-Karraypad_const[=N]"` and `"-Karraypad_expr=N"`, the same options must be specified when compiling all the program units that make up the executable program.

This feature is applied to each individual array, without considering storage association and argument associations that are introduced by the `EQUIVALENCE` statement and `COMMON` statement. For this reason, the correct result may not be obtained by using this function so caution should be used.

Example 1: `-Karraypad_expr=N`

```
PROGRAM MAIN
REAL, DIMENSION(1024*1024) :: A
CALL SUB(A,1024,1024)
:
SUBROUTINE SUB(X,M,N)
REAL, DIMENSION(M,N) :: X
:
```

If this type of program is compiled with the `"-Karraypad_expr=1"` option, then it will be compiled as one of the following types of programs.

```
PROGRAM MAIN
REAL, DIMENSION(1024*1024+1) :: A
CALL SUB(A,1024,1024)
:
SUBROUTINE SUB(X,M,N)
REAL, DIMENSION(M+1,N) :: X
:
```

If this feature is not used, `A(1025)` in `MAIN` is associated with `X(1,2)` in `SUB`. When it is used, `A(1025)` is associated with `X(1025,1)` and will derive a different result.

Example 2: `-Karraypad_const[=N]`

```
INTEGER A(3,3), B(3,3)
A(:,1)=B(1,:)
A(:,2)=RESHAPE((/1,2,3/), (/3/))
```

If the `"-Karraypad_const=1"` option is specified on compile, the array shape on the right and left of the assignment will differ, and the results may be incorrect.

9.13 Effects of Generating Prefetch on Program Performance

If the `-Kprefetch_sequential`, `-Kprefetch_stride`, `-Kprefetch_indirect`, or `-Kprefetch_conditional` option is valid, it may decrease in performance to generate prefetch instructions depending on the cache-efficiency of the loop, the presence or absence of branch instructions, and the complexity of subscripts.

9.14 Dimension Shift of Array Declaration

This section explains dimension shift of array declarations.

9.14.1 Effects of Dimension Shift of Array Declaration

The dimension position of arrays is shifted by specifying the compiler option `-Karray_subscript`. This will affect intrinsic functions, for which the keyword arguments `DIM` and `MASK` can be specified.

When using the compiler options `-Karray_subscript`, the same options must be specified when compiling all the program units that make up the executable program. The target array variables are allocatable arrays and explicit shape arrays with bounds that are constant expressions (local arrays, common block objects, and dummy arguments). This function will not be applied within specification expressions or the constant expressions of declaratives. This feature is applied to each individual array, without considering argument association. For this reason, the correct result may not be obtained by using this function so caution should be used.

The following shows an example:

Example 1:

```
INTEGER, DIMENSION(101,102,103,5) :: A
CALL SUB(A)
:
PRINT *, A(1,2,3,4)
```

If this type of program is compiled with the `"-Karray_subscript"` option, then it will be compiled as one of the following types of programs:

```
INTEGER, DIMENSION(5,101,102,103) :: A
CALL SUB(A)
:
PRINT *, A(4,1,2,3)
```

Example 2:

```
PROGRAM MAIN
INTEGER, DIMENSION(100,101,102,3) :: A
:
CALL SUB(A)
END
SUBROUTINE SUB(B)
INTEGER, DIMENSION(100) :: B
:
END
```

In this sample, the argument `A(3,100,101,102)` for `SUB` specified in `MAIN` will be merged with the parameter `B(100)` defined in `SUB`. However, the result may be incorrect because the dimensions of the arrays are not identical.

Example 3:

```
INTEGER, DIMENSION(1,2,3,4) :: A
:
A=RESHAPE(((i,i=1,24)), (/1,2,3,4/))

! The right-hand shape of the substitution expression is different
! from the left-hand one, so the execution result may not be correct.
PRINT *, UBOUND(A,4)
! 3 is output.
```

9.14.2 Restrictions of Target Variable of Dimension Shift of Array Declaration

Arrays that meet any of the following conditions cannot be the target of dimension shift of array declarations:

- Derived type, character type, or polymorphic
- Literal with name
- An array with a SAVE attribute (implicit SAVE attribute in declaration part of module is excluded) or TARGET attribute
- Pointer variable
- An array with an initial value
- Equivalence-object
- Namelist-group-object
- Automatic array
- An array with a PUBLIC attribute declared in the module specification part
- Address holding variable
- Function result
- Declaration in BLOCK

9.14.3 Restriction for Ancestor Module of Submodule

Ancestor module of submodule cannot be specified when dimension shift of array is enabled by a compilation option `-Karray_subscript` or an optimization control line.

9.15 Merging Array Variables

This section explains merging of array declarations.

9.15.1 Effects of Merging Array Variables

Multiple arrays are merged into one array by specifying the compiler option `-Karray_merge`.

The target array variables are explicit-shape arrays or automatic object arrays with bounds that are constant expressions, and automatically allocated arrays (Note). This function will not be applied within specification expressions or the constant expressions of declaratives. The compiler automatically adds the element of the first dimension and merges the elements. The merging order is undefined. This may cause a compile error or may introduce an incorrect result, because the array names will be replaced with the merged array names. This feature is applied to each individual array, without considering lower bound of declaration and argument association. For this reason, using this function may cause incorrect results.

(Note)

If the bound expression of automatic object array is the following condition, the automatic object array is not restricted.

1. Bound expression of declaration contain exactly one variable, or
2. Bound expression of declaration contain is constant expression.

The following shows an example:

Example 1:

```
INTEGER, DIMENSION(101,102,103)::A,B,C
CALL SUB(A)
:
PRINT *,B(101,102,103)
A=C
A(101,102,103)=C(1,2,3)
```

If this type of program is compiled with the `-Karray_merge` option specified, then it will be compiled as one of the following types of programs:

```

INTEGER,DIMENSION(3,101,102,103)::TMP
  ! A, in 1 element of the 1st dimension, merge : TMP (1,101,102,103)
  ! B, in 2 element of the 1st dimension, merge : TMP (2,101,102,103)
  ! C, in 3 element of the 1st dimension, merge : TMP (3,101,102,103)
CALL SUB(TMP(1,::,:))
:
PRINT *,TMP(2,101,102,103)
TMP(1,::,:)=TMP(3,::,:))
TMP(1,101,102,103)=TMP(3,1,2,3)

```

Note:

TMP is a variable generated by the compiler.

Example 2:

```

PROGRAM MAIN
INTEGER,DIMENSION(100,101,102) :: A
:
CALL SUB(A(1,1,1))
END
SUBROUTINE SUB(C)
INTEGER,DIMENSION(100) :: C
:
END

```

When this feature is used, the "A", which is an argument of SUB on MAIN, may not be associated with C(100), which is a parameter of SUB, because an element is appended in the first dimension of "A".

The compiler option `-Karray_merge_common` facilitates to merge the named common block object array.

The target array variables are explicit shape arrays within named common blocks with bounds that are constant expressions. In this case, a common block name is created. It is the following form:

```
_0_ <common block name>
```

All of the common-block-objects within the same named common block must be arrays with the same shape. This function will not be applied within specification expressions or the constant expressions of declaratives. The compiler automatically adds the element of the first dimension and merges the elements. The merging order is undefined. This may cause a compile error or may introduce an incorrect result, because the array name will be replaced with the merged array names. This feature is applied to each individual array, without considering argument association. For this reason, using this function may cause incorrect results.

The following shows an example:

Example 1:

```

INTEGER,DIMENSION(101,102,103)::A,B,C
COMMON /COM/ A,B,C
CALL SUB(A)
:
PRINT *,B(101,102,103)

```

If this type of program is compiled with the `-Karray_merge_common` option, then it will be compiled as one of the following types of programs:

```

INTEGER,DIMENSION(3,101,102,103)::TMP
COMMON /_0_COM/ TMP
CALL SUB(TMP(1,::,:))
PRINT *,TMP(2,101,102,103)

```

Note:

TMP is a variable generated by the compiler.

9.15.2 Restrictions of Merging Local Array Variables

Arrays that meet any of the following conditions cannot be the target of local array variable merge:

- Derived types or character types
- Literal with name
- An array with a SAVE attribute or TARGET attribute
- Pointer variable
- An array with an initial value
- Equivalence-object
- Namelist-group-object
- Declaration within the module or submodule
- Address holding variable
- Function result
- Common block objects
- Dummy arguments
- Specified in THREADPRIVATE statement
- Automatic array

If the bound expression of automatic object array is the following condition, the automatic object array is not restricted.

1. Bound expression of declaration contains exactly one variable, or
2. Bound expression of declaration is constant expression.

- Declaration in BLOCK

9.15.3 Restrictions of Target Variables on Common Block Object Array Merge

Arrays that meet any of the following conditions cannot be the target of common-block-object array merge:

- Derived types or character types
- An array with a TARGET attribute
- An array with an initial value
- Equivalence-object
- Namelist-group-object
- An array within program units that include OpenMP statements

9.16 Multi-Operation Function

9.16.1 Calling of Multi-Operation Function

Multi-operation functions are functions that improve performance by using intrinsic functions for calculations and operations on multiple arguments called at one time. By specifying compiler option `-Kmfunc` or optimization control line `MFUNC`, the compiler analyzes loops and replaces them with multi-operation functions if possible.

If there are complex loops, and the compiler cannot analyze it, modify the program to call multi-operation function directly.

The following is required:

- The argument n which specifies the size must be an 8-byte integer.
- The external procedure names of the multi-operation functions are the external procedure names for the C language. For this reason, when called from a Fortran program, specifies to change how to process external procedure name by \$pragma specifier.
- The argument check may be omitted in the multi-operation function for high-speed. Therefore, the user program may be terminated abnormally when the special value (NaN and Inf, etc.) defined by IEEE 754 is input.

Table 9.2 lists the multi-operation functions that can be called directly from user programs.

Table 9.2 Multi-operation functions that can be called directly from user programs

function/operation	type argument	calling format	content of calculation
ACOS	real(4)	v_acos(x,n,y)	y(i) = acos(x(i))
	real(8)	v_dacos(x,n,y)	y(i) = dacos(x(i))
ASIN	real(4)	v_asin(x,n,y)	y(i) = asin(x(i))
	real(8)	v_dasin(x,n,y)	y(i) = dasin(x(i))
ATAN	real(4)	v_atan(x,n,y)	y(i) = atan(x(i))
	real(8)	v_datan(x,n,y)	y(i) = datan(x(i))
ATAN2	real(4)	v_atan2(x1,x2,n,y)	y(i) = atan2(x1(i),x2(i))
	real(8)	v_datan2(x1,x2,n,y)	y(i)=datan2(x1(i),x2(i))
ERF	real(4)	v_erf(x,n,y)	y(i) = erf(x(i))
	real(8)	v_derf(x,n,y)	y(i) = derf(x(i))
ERFC	real(4)	v_erfc(x,n,y)	y(i) = erfc(x(i))
	real(8)	v_derfc(x,n,y)	y(i) = derfc(x(i))
EXP	real(4)	v_exp(x,n,y)	y(i) = exp(x(i))
	real(8)	v_dexp(x,n,y)	y(i) = dexp(x(i))
	complex(8)	v_cdexp(x,n,y)	y(i) = cdexp(x(i))
EXP10	real(4)	v_exp10(x,n,y)	y(i) = exp10(x(i))
	real(8)	v_dexp10(x,n,y)	y(i) = dexp10(x(i))
LOG	real(4)	v_alog(x,n,y)	y(i) = alog(x(i))
	real(8)	v_dlog(x,n,y)	y(i) = dlog(x(i))
LOG10	real(4)	v_log10(x,n,y)	y(i) = log10(x(i))
	real(8)	v_dlog10(x,n,y)	y(i) = dlog10(x(i))
SIN	real(4)	v_sin(x,n,y)	y(i) = sin(x(i))
	real(8)	v_dsin(x,n,y)	y(i) = dsin(x(i))
COS	real(4)	v_cos(x,n,y)	y(i) = cos(x(i))
	real(8)	v_dcos(x,n,y)	y(i) = dcos(x(i))
SIN & COS	real(4)	v_scn(x,n,y,z)	y(i) = sin(x(i))
			z(i) = cos(x(i))
	real(8)	v_dscn(x,n,y,z)	y(i) = dsin(x(i))
			z(i) = dcos(x(i))
exponentiation	real(4)	v_arxr(x1,x2,n,y)	y(i) = x1(i)**x2(i)
	real(4)	v_arxr1(x,a,n,y)	y(i) = x(i)**a
	real(8)	v_adxd(x1,x2,n,y)	y(i) = x1(i)**x2(i)
	real(8)	v_adxd1(x,a,n,y)	y(i) = x(i)**a

Note:

When calling multi-operation functions directly from user programs, the memory used for the argument to return the results of the operations must be separate from the memory used for the other arguments. If the areas overlap, the result may be incorrect.

The following shows the example of calling multi-operation functions.

Example 1:

```
INTEGER,PARAMETER :: N=1000
REAL(KIND=8) :: A
REAL(KIND=8),DIMENSION(N) :: X,Y,Z
:
A = 0.2D0
DO I=1,N
  Y(I) = DEXP(X(I))
  Z(I) = X(I)**A
ENDDO
```

Invocation of multi-operation function

```
INTEGER(KIND=8),PARAMETER :: N=1000
REAL(KIND=8) :: A
REAL(KIND=8),DIMENSION(N) :: X,Y,Z
EXTERNAL V_DEXP !$PRAGMA C(V_DEXP)
EXTERNAL V_ADXD1 !$PRAGMA C(V_ADXD1)
:
A = 0.2D0
CALL V_DEXP(X,N,Y)
CALL V_ADXD1(X,A,N,Z)
```

Example 2:

If a function is called within an IF construct, multi-operation functions can be used by storing only the elements that need calculation in arrays.

```
INTEGER,PARAMETER :: N=1000
REAL(KIND=8) :: X,Y,Z
REAL(KIND=8),EXTERNAL :: FUNC
:
DO I=1,N
  X = ..
  IF (X.GT.A) THEN
    Y = Y + SIN(X)
    Z = Z + COS(X)
  END IF
ENDDO
```

Invocation of multi-operation function

```
INTEGER(KIND=8),PARAMETER :: N=1000
INTEGER(KIND=8) :: J
REAL(KIND=8) :: X,Y,Z
REAL(KIND=8),DIMENSION(N) :: WX,WY,WZ
EXTERNAL V_DSCN !$PRAGMA C(V_DSCN)
:
J = 0
DO I=1,N
  X = ..
  IF (X.GT.A) THEN
    J = J + 1
    WX(J) = X
  END IF
ENDDO
CALL V_DSCN(WX,J,WY,WZ)
```

```

DO I=1,J
  Y = Y + WY(I)
  Z = Z + WZ(I)
ENDDO

```

9.16.2 Effects of Compiler Option -Kmfunc=3

If the compile option -Kmfunc=3 is specified, the program may terminate abnormally as a side effect of changing loops that include IF constructs into multi-operation functions. The following shows an example of the effects of -Kmfunc=3:

Example: Loops that include IF constructs

```

SUBROUTINE SUB(A,B)
DIMENSION A(1000),B(1000)
.
.
DO I=1,2000
  IF (I.LE.1000) THEN
    A(I) = COS(B(I))
  ENDIF
ENDDO

```

[Description]

In the above example, the value of the subscript of array B never exceeds the declared range because the intrinsic function COS, of which the actual argument is the array element, is called if the condition I.LE.1000 holds.

By applying the multi-operation function optimization, the DO loop references an array element for each iteration of the loop, and uses it as an argument of multi-operation function regardless the result of IF-construct. Access may therefore exceed the array bounds. This will cause an abnormal termination of the program, with a message informing of a memory protection exception (READ).

To prevent this, do not specify the compiler option -Kmfunc=3.

9.17 Software Control of Sector Cache

9.17.1 Using Sector Cache

Sector cache is a mechanism to prevent reusable data on the cache from being driven out by non-reusable data. In particular, Sector cache is effective for parallel execution when the shared L2 cache is accessed by multiple cores. By using Sector cache, reusable data is separated from non-reusable data on the cache. There are two methods to control Sector cache with software; environment variables and optimization control lines. The environment variables can specify the maximum number of ways for each sector, while the OCL can specify that of sector 1, both of which are detailed below.

Performance will not be improved if the maximum number of ways is specified without consideration for the size of the array that is to be protected from being driven out. The cache utilization will decline, and may cause a reduction in speed. Further, even if software control of the Sector cache is not used, cache control with LRU (Least Recently Used) still works. Thus speed may not improve even if this is specified. One effective way to improve the performance of Sector caches is to determine the reusable array size and specify the number of ways for sector 1 to store the array, after ensuring L2 cache miss has occurred by confirming the PA information. Note that in order to use Sector caches, the compile-time option -Khpctag must be set.



Information

The maximum number of cache way for each sector

In the A64FX processor, the maximum number of the first level cache is 4, and that of the second level cache is 16.

Note that the assistant core always uses two ways for the second level cache.

9.17.2 How to Control Sector Cache by Software

There are two methods to control Sector caches with software in this processor, as follows:

- Optimization control lines
- Environment variables and optimization control lines

9.17.2.1 Software Control with Optimization Control Lines

```
!OCL SCACHE_ISOLATE_WAY(L2=n1 [,L1=n2])
and
!OCL END_SCACHE_ISOLATE_WAY
```

The SCACHE_ISOLATE_WAY specifier directs the maximum number of ways of sector 1 of the caches.

The argument of the SCACHE_ISOLATE_WAY specifier directs the maximum number of ways of sector 1 of the caches. The L2=*n1* directs the maximum number of ways of sector 1 in the second level cache and the L1=*n2* directs the maximum number of ways of sector 1 in the first level cache. Specifying the L1=*n2* is optional. In this case, this specifier controls only the maximum number of ways of sector 1 of the second level cache.

Note that the assistant core always uses two ways for the second level cache. With that in mind, the arguments *n1* and *n2* must be as follows:

- 0 <= *n1* <= "the maximum number of ways in the second level cache - 2"
- 0 <= *n2* <= "the maximum number of ways in the first level cache"

To direct in function, the SCACHE_ISOLATE_WAY specifier must be used in procedure line. In this case, the SCACHE_ISOLATE_WAY specifier is valid in the scope of the function. To direct a part of function, the range must be indicated by the SCACHE_ISOLATE_WAY and END_SCACHE_ISOLATE_WAY specifiers in statement lines. Note that the range must not be nested. However, combined use of procedure line and statement line is accepted.

```
!OCL SCACHE_ISOLATE_ASSIGN(array1 [,array2...])
and
!OCL END_SCACHE_ISOLATE_ASSIGN
```

The SCACHE_ISOLATE_ASSIGN specifier directs to specify an array stored in sector 1 of the caches.

Note that it becomes effective only when the array of numeric type or logical type is specified.

To direct in function, the SCACHE_ISOLATE_ASSIGN specifier must be used in procedure line. In this case, the SCACHE_ISOLATE_ASSIGN specifier is valid in the scope of the function.

To direct a part of function, the range must be indicated by the SCACHE_ISOLATE_ASSIGN and END_SCACHE_ISOLATE_ASSIGN specifiers in statement lines.

Note that the range must not be nested.

However, combined use of procedure line and statement line is accepted.

The following shows an example:



Example

Software Control with Optimization Control Lines

```
SUBROUTINE OCL_SECTOR(A,B,N,N2)
REAL(KIND=8),DIMENSION(N) :: A
REAL(KIND=8),DIMENSION(N,N2) :: B
! Reuse the array A that can fit in 10 ways in the L2 cache
!OCL SCACHE_ISOLATE_WAY(L2=10)
!OCL SCACHE_ISOLATE_ASSIGN(A)
DO J=1,N2
!$OMP PARALLEL DO
  DO I=1,N
    A(I)=A(I)+B(I,J)
  ENDDO
!$OMP END PARALLEL DO
ENDDO
!OCL END_SCACHE_ISOLATE_ASSIGN
```



```
!OCL END_SCACHE_ISOLATE_WAY
END SUBROUTINE
```

9.17.2.2 Software Control with Environment Variables and Optimization Control Lines

The initial values for the maximum number of ways for each sector can also be specified with the following environment variable. The maximum number of ways for the object program at startup can be specified with environment variables.

FLIB_SCCR_CNTL

This environment variable specifies to use Sector cache. The values that can be set in the environment variable are shown below, along with their meanings.

Value	Description
TRUE	Use the Sector cache. This value is set by default.
FALSE	Do not use Sector cache.

FLIB_L1_SCCR_CNTL

The Sector cache for the second level cache is not available when multiple processes run on a NUMA node.

This environment variable specifies whether to use the Sector cache for the first level cache when the Sector cache for the second level cache is unavailable.

FLIB_L1_SCCR_CNRTL is effective only when FLIB_SCCR_CNTL is assigned to the value TRUE.

The values that can be set in the environment variable are shown below, along with their meanings.

Value	Description
TRUE	If the Sector cache for the second level cache is not available, the Sector cache for the first level cache is used. The value is set by default.
FALSE	If the Sector cache for the second level cache is not available, the Sector cache for the first level cache is not used. And the following error message is output, the Sector cache control function is disabled, and program execution continues. <pre>jwe1047i-w A sector cache couldn't be used.</pre>

FLIB_L2_SECTOR_NWAYS_INIT

This environment variable specifies the initial maximum number of cache ways for each sector of the second level cache. FLIB_L2_SECTOR_NWAYS_INIT is effective only when the value of FLIB_SCCR_CNTL is TRUE.

The values for the first sector (sector 0) *n0* and for the second sector (sector 1) *n1* should be specified as follows:

```
n0 , n1
```

The possible values are as follows:

- $0 \leq n0 \leq$ "the maximum number of ways in the second level cache - 2"
- $0 \leq n1 \leq$ "the maximum number of ways in the second level cache - 2"

The two ways are reserved for the assistant cores.

It is recommended to satisfy the following condition to avoid conflicts.

- $n0 + n1 =$ "the maximum number of ways in the second level cache - 2"

Example

Software Control with Environment Variables and Optimization Control Rows

As shown in the following example of software control of Sector cache, by setting the value "2,10" to the environment variable FLIB_L2_SECTOR_NWAYS_INIT before execution starts, the maximum number of ways is set to 10 and the array data is cached on the sector 1 cache.

1. Specify the environment variable, before starting the program (bash).

```
FLIB_SCCR_CNTL=TRUE
export FLIB_SCCR_CNTL
FLIB_L2_SECTOR_NWAYS_INIT=2,10
export FLIB_L2_SECTOR_NWAYS_INIT
```

2. Specify an array stored in sector 1 of the cache.

```
SUBROUTINE OCL_SECTOR(A,B,N,N2)
REAL(KIND=8),DIMENSION(N) :: A
REAL(KIND=8),DIMENSION(N,N2) :: B
! Assign sector 1 to the array A
!OCL SCACHE_ISOLATE_ASSIGN(A)
DO J=1,N2
!$OMP DO PARALLEL
DO I=1,N
A(I)=A(I)+B(I,J)
ENDDO
!$OMP ENDDO
ENDDO
!OCL END_SCACHE_ISOLATE_WAY
END SUBROUTINE
```

9.17.2.3 Behavior when an Invalid Value Is Specified

If a value specified for the SCACHE_ISOLATE_WAY specifier or the environment variable FLIB_L2_SECTOR_NWAYS_INIT exceeds the upper limit, it is assumed that the upper limit is specified. If a value less than 0 is specified, it has no effect. If a value outside the range of two-byte signed integer is specified, the behavior is undefined.

9.18 Incorrectly Specified Optimization Indicators

When optimization control lines are incorrectly specified, the execution result may not be incorrect.

9.18.1 Example of NOVREC Specifier

When array A is not recurrence data and NOVREC specifier is effective, compiler uses SIMD instructions. When array A is recurrence data and NOVREC specifier is effective, compiler uses SIMD instructions. But the execution result is unpredictable.

Example:

```
!OCL NOVREC
DO I=1,100
A(I)=A(I+M)+...
ENDDO
```

9.18.2 Example of CLONE Specifier

In the copied loops under the generated branches, the specified variables are treated as invariants in the loops. Thus, the execution result is not guaranteed in case that the value of the variables changes in the loop.

Example 1:

```
INTEGER, DIMENSION(32)::A
INTEGER, TARGET::N
INTEGER, POINTER::M
M=>N
!OCL CLONE(N==10)
DO I=1, 32
    M=5
    A(I) = N
ENDDO
```

Do not specify a variable which is changed in the loop because the execution result is not guaranteed.

Example 2:

```
INTEGER::N
!OCL CLONE(N==10)
DO I=1, 32
    CALL SUB(N)
ENDDO
```

Do not specify a variable which may be changed in the loop because the execution result is not guaranteed.

9.18.3 Example of UNROLL_AND_JAM_FORCE Specifier

The UNROLL_AND_JAM_FORCE specifier instructs that unroll-and-jam is performed assuming that there is no data dependency over iterations of the loops. The result of executions is not guaranteed if the specifier is used incorrectly.

Example:

```
!OCL UNROLL_AND_JAM_FORCE(2)
DO J=2, 128
    DO I=1, 127
        A(I, J) = A(I+1, J-1) + B(I, J)
        ...
    END DO
END DO
```

Do not use the UNROLL_AND_JAM_FORCE specifier because there exists data dependency of array A over iterations of the loops.

9.18.4 Example of ITERATIONS Specifier

The ITERATIONS(max= $n1$, min= $n3$) specifier instructs to perform optimization assuming the maximize loop iteration count is $n1$, and the minimum loop iteration count is $n3$. This is effective when the loop iteration count is unknown at compilation. However, if the actual loop iteration count is greater than $n1$, or less than $n3$, the execution result is not guaranteed.

Example 1:

```
!OCL ITERATIONS(max=100,min=1)
DO I=1,M ! The actual maximum count is 1000 or the minimum count is 0.
    A(I) = A(I) + B(I)
END DO
```

When the actual maximum iteration loop count is greater than 100 or the minimum iteration loop count is 0, the execution result is not guaranteed

9.19 Attentions for Mixed Language Programming

The following describes the notes when programming with different languages in this processor.

9.19.1 Address Parameters Received in C language

If an external C program stores the addresses of the parameters, the result may be incorrect because such operation is unsupported and the optimization is performed without considering it. (Refer to example 1)

In this case, to obtain the correct result, pass the address explicitly using arguments rather than saving the parameter address in the function. (Refer to example 2)

Alternatively, pass the parameter address by specifying "%VAL(LOC(argument))" as the argument of the function caller in the Fortran program in Example 1. (Refer to Example 3)

Example 1: Incorrect saving of pointer

```
int fun_(int *flag, int *p)
{
    static int *save;
    if (*flag == 0) {
        save = p; /* Saving address dummy argument. */
        return 0;
    } else {
        return *save;
    }
}
```

```
INTEGER(KIND=4)::I,J
INTEGER(KIND=4),EXTERNAL::FUN
J = 0
I = FUN(0, J)
J = 1
I = FUN(1, 0)
PRINT *, I, J
END
```

Example 2: Format for not saving pointer

```
int fun_(int *flag, int *p)
{
    if (*flag == 0) {
        return 0;
    } else {
        return *p;
    }
}
```

```
INTEGER(KIND=4)::I,J
INTEGER(KIND=4),EXTERNAL::FUN
J = 0
I = FUN(0, J)
J = 1
I = FUN(1, J)
PRINT *, I, J
END
```

Example 3: Format for explicit passing of address

```
INTEGER(KIND=4)::I,J
INTEGER(KIND=4),EXTERNAL::FUN
J = 0
I = FUN(0, %VAL(LOC(J)))
J = 1
I = FUN(1, 0)
PRINT *, I, J
END
```

9.19.2 Pointers Received in the C language

If a pointer address is returned as the function result or COMMON when the pointer is received in the C program, the result may be incorrect because such an operation is unsupported and the optimization is performed without considering it. (Refer to example 4)

In this case, using the intrinsic function LOC to store the address in the pointer will produce a correct result. (Refer to example 5)

Example 4: Incorrect return of pointer

```
int *my_addr_(int *p)
{
    return p;
}
```

```
INTEGER(KIND=4) I,IP
POINTER(PTR,IP)
INTEGER(KIND=8) MY_ADDR
I = 2
PTR = MY_ADDR(I)
IP = 1
PRINT *, I
END
```

Example 5: Correct pointer usage

```
INTEGER(KIND=4) I,IP
POINTER(PTR,IP)
I = 2
PTR = LOC(I)
IP = 1
PRINT *, I
END
```


9.20 Side Effect of Optimizations for Floating-Point Operation


Optimizations for floating-point operation might cause the side effect. This section explains the side effect (mainly, computation error).

This system basically creates objects which comply with IEEE 754 arithmetic. Note that the numerical operations may not comply with IEEE 754 arithmetic due to the optimizations with calculation errors in "Table 9.3 Side effect of optimization for floating-point operation".

See Section "2.2 Compiler Options" for compiler options. See section "9.10.4 Optimization Control Specifiers" and "12.2.6.4 Optimization Control Specifiers for Automatic Parallelization" for optimization control specifiers.

Table 9.3 Side effect of optimization for floating-point operation

Compiler Option	Optimization Control Specifier	Side Effect
-Keval	EVAL	<p>Calculation errors may be induced when optimization that modifies how operations are evaluated for object programs.</p> <p> Example</p> <hr style="border-top: 1px dotted #0000FF;"/> <div style="border: 1px solid black; padding: 5px; width: fit-content;"> $x*y + x*z \rightarrow x*(y+z)$ </div> <hr style="border-top: 1px dotted #0000FF;"/> <p>When the -Keval option is set, the following options are valid. See the description about the side effect of each option.</p> <ul style="list-style-type: none"> - -Kfsimple option - -Kreduction option (When -Kparallel option is valid)

Compiler Option	Optimization Control Specifier	Side Effect
		- <code>-Ksimd_reduction_product</code> option (When <code>-Ksimd[={ 1 2 auto }]</code> option is valid)
<code>-Kfsimple</code>	(None)	<p>Simplification of floating point operation on source programs is performed. Therefore, the numerical operations do not comply with IEEE 754 arithmetic, as in the example below.</p> <p> Example</p> <hr style="border-top: 1px dotted blue;"/> <div style="border: 1px solid black; padding: 2px; width: fit-content;"> <code>x*0.0 -> 0.0</code> </div> <p>Operations such as "x*0.0" will be simplified to "0.0".</p> <hr style="border-top: 1px dotted blue;"/>
<code>-Kfp_contract</code>	<code>FP_CONTRACT</code>	Rounding errors may be induced when optimization using Floating-Point Multiply-Add/Subtract floating point operation instructions is performed on source programs.
<code>-Kfp_relaxed</code>	<code>FP_RELAXED</code>	<p>Side effects may occur because reciprocal approximation operation instructions are used on single-precision or double-precision floating point division or SQRT functions.</p> <p>The side effects that may occur are:</p> <ul style="list-style-type: none"> - Rounding errors. - Replacing denormalized numbers found in arguments or the return value with zero regardless of set of <code>-Kfz</code> option. - Replacing negative zeroes found in arguments or the return value with positive zeroes. - Behaviors not conforming to IEEE 754 when NaN, positive or negative Inf, numbers which are close to maximal normalized number or numbers which are close to minimal normalized number are found in arguments or the return value.
<code>-Kfz</code>	(None)	Flush-to-zero mode is used. If a result or a source operand is a denormalized number, flush-to-zero mode replaces it with zero with the same sign.
<code>-Kilfunc</code>	(None)	<p>Side effects similar to the ones caused by <code>-Kfp_relaxed</code> may occur when using inline expansion for intrinsic functions and operations, because reciprocal approximation instructions and trigonometric instructions, etc. are used. Plus, side effects shown below may occur:</p> <ul style="list-style-type: none"> - If the argument of the intrinsic function corresponds to the "Cause of error" described in Section "8.1.5 Intrinsic Function Error Processing", error messages are not output and execution results are not guaranteed. - Use of reciprocal approximation instructions or Floating-Point Multiply-Add/Subtract floating point operation instructions regardless of set of <code>-Knofp_contract</code> and/or <code>-Knofp_relaxed</code> options or their setting order. <p>Also, <code>atan2</code> is computed according to Fortran 95 language specification when the <code>-X03</code> or <code>-X08</code> option is set.</p>
<code>-Kmfunc[={ 1 2 3 }]</code>	<code>MFUNC[<i>(level)</i>]</code>	Side effects similar to the ones caused by <code>-Kfp_relaxed</code> and/or <code>-Kilfunc</code> may occur because different algorithms, reciprocal approximation instructions, trigonometric instructions, etc. are used internally when the intrinsic function or the operation is converted to the multi-operation function.

Compiler Option	Optimization Control Specifier	Side Effect
		Also, atan2 is computed according to Fortran 95 language specification when the -X03 or -X08 option is set. See Section "9.16 Multi-Operation Function".
-Kreduction	REDUCTION	The automatic parallelization loop reduction is performed. Therefore, calculation errors may be induced when the optimization that modifies how operations are evaluated for object programs.
-Ksimd_reduction_product	(None)	SIMD extensions are used to the reduction operation of product. Therefore, calculation errors may be induced when the optimization that modifies how operations are evaluated for object programs.
-Kfast	(None)	-Kfast option induces the -Keval option, -Kilfunc option, and so on.
-Kvisimpact	(None)	-Kvisimpact option induces the -Kfast option.

Also, If the -O1 option or higher is set, this system performs optimizations that execute floating-point arithmetic instructions that may not be executed based on the logic of the program. This may result in generating extra floating-point exceptions at execution time. In this case, the following phenomena may occur.

- If a program uses the intrinsic module IEEE_EXCEPTIONS and accesses a floating-point status flag by it, the flag which should not be set based on the logic of the program is set.

This phenomenon may be avoided by setting the option -NRtrap.

Information

To avoid side effect for floating-point operations error at parallelization (automatic parallelization/OpenMP)

Specify the following options to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads. When these options are set, the execution performance may decrease.

- -Kparallel_fp_precision option (When the -Kparallel option or -Kopenmp option is valid.)
- -Kopenmp_ordered_reduction option (When the -Kopenmp option is valid.)

See Section "2.2 Compiler Options" for compiler options.

9.21 Notes on Specified SVE Vector Register Size

-Ksimd_reg_size={ 128 | 256 | 512 } specifies the bitwise SVE vector register size. When this option is specified, the optimization is performed considering the vector register size as a fixed value at compilation. Therefore, the generated executable program works normally on the CPU architecture which has the same size of the SVE vector register as the specified size.

When the program is executed on the CPU whose implemented vector register size is different from the vector register size specified by -Ksimd_reg_size={ 128 | 256 | 512 } option, an abnormal end occurs. And, the result of executions is not guaranteed.

It is required to change the effective size of the vector register by the system call prctl(2) or in other ways when the vector register size specified by the compiler option -Ksimd_reg_size={ 128 | 256 | 512 } is smaller than that of the CPU.

When -Ksimd_reg_size=agnostic is effective, the executable program does not depend on the SVE vector register size.

See also section "A.2.2 Changing SVE Vector Register Size" for notes on changing the SVE vector register size.

Chapter 10 Fortran Modules and Submodules

A module reference is a USE statement specifying the module name, a SUBMODULE statement specifying the module name, or a SUBMODULE statement specifying the submodule name.

This chapter describes items related to compiling input files that have modules, submodule, modules references and intrinsic modules references.

10.1 Compilation of Modules and Module References

Note the following:

The object file of a module is created in the current directory. Its filename is the source filename with the suffix replaced by the suffix .o.

When a module is compiled, an information file is created by the compiler. The name of the file is the module name with the suffix .mod appended. This information file is needed when compiling a program that contains a USE statement for the module. The .mod information file is created according to the following rules:

- If the compiler option -M is specified, it is created in the directory specified in the -M option.
- If the compiler option -M is not specified, it is created in the directory from which frtpx is being executed.

When a program that contains a USE statement is compiled, the corresponding information file is searched for in the following order:

- A directory named in the -M option (if specified).
- The directory from which frtpx is being executed.
- A directory named in the -I option (if specified).

When linking a program containing a module reference, you must specify the object file of the corresponding module.

Example1:

The module and the module reference:

! file name : a.f90

```
MODULE MOD
  INTEGER :: VALUE=1
END MODULE

PROGRAM MAIN
  USE MOD
  PRINT *, VALUE
END PROGRAM
```

Compile command:

```
$frtpx a.f90
```

Example2:

The module and the module reference appear in different files:

! file name : a.f90

```
MODULE MOD
  INTEGER :: VALUE=1
END MODULE
```

! file name : b.f90

```
PROGRAM MAIN
  USE MOD
  PRINT *, VALUE
END PROGRAM
```


Compile command:

```
$ frtpx a.f90 -c
$ frtpx b.f90 a.o
```

or

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx a.o b.o
```

Example3:

The module and two references to the module reference appear in three different files:

! file name : a.f90

```
MODULE MOD
  INTEGER::VALUE=1
END MODULE
```

! file name : b.f90

```
PROGRAM MAIN
  USE MOD
  VALUE=VALUE+1
  CALL EXTERNAL_SUB
END PROGRAM
```

! file name : c.f90

```
SUBROUTINE EXTERNAL_SUB
  USE MOD
  PRINT *,VALUE
END SUBROUTINE
```

Compile command:

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx c.f90 a.o b.o
```

or

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx c.f90 -c
$ frtpx a.o b.o c.o
```

Example4:

The module and the module reference appear in different files using the compiler options -I, and -M:

! file name : a.f90

```
MODULE MOD1
  INTEGER::VALUE=1
END MODULE
```

! file name : b.f90

```
MODULE MOD2
  USE MOD1
  INTEGER::VALUE2=2
CONTAINS
SUBROUTINE MOD2_SUB
```

```
VALUE=VALUE2+1
END SUBROUTINE
END MODULE
```

Compile command:

```
$ frtpr a.f90 -c
```

Note:

The information file mod1.mod for module mod1 is generated in the current directory.

```
$ frtpr a.f90 -c -Mdirectory
```

Note:

The information file mod1.mod for module mod1 is generated in the directory specified by the -M option argument.

```
$ frtpr b.f90 -c -Idirectory
```

Note:

1. The information file mod1.mod for module mod1 is searched for in the directory specified by the -I option argument or in the current directory.
2. The information file mod2.mod for module mod2 is generated in the current directory.

```
$ frtpr b.f90 -c -Iinput_directory -Moutput_directory
```

Note:

1. The information file mod1.mod for module mod1 is searched for in the directory specified in the -I option argument, the directory specified by the -M option argument, or in the current directory.
2. The information file mod2.mod for module mod2 is generated in the directory specified by the -M option argument.

10.2 Recompiling a Program that Contains a Reference to a Module Information that Has Changed

In either of the following cases, a program unit that contains a module reference must be recompiled.

- An entity with the PUBLIC attribute in the module specification part is changed
- The characteristics of a module procedure with the PUBLIC attribute are changed

If an entity in a module or submodule specification part is changed, its descendant submodules must be recompiled.

10.3 Restrictions on Modules

When the following compiler option is specified for compile a module or submodule, you must specify the same option for compile the other program that including the module reference.

- -AA option
- -AU option
- -Kcommonpad[=N] option
- -Karray_subscript option

When a module is compiled with compiler option -X9, -X03, or -X08, the other program including the module reference also has to be compiled with compiler option -X9, -X03, or -X08.

When a module is compiled with compiler option -X6, the other program including the module reference also has to be compiled with compiler option -X6.

When a module is compiled with compiler option `-X7`, the other program including the module reference also has to be compiled with compiler option `-X7`.

A module cannot be specified as an ancestor module of a submodule if the module is compiled with `-Karray_subscript` or is specified to shift dimension position of arrays by optimization control specifier.

10.4 Submodule

Note the following:

The object file of a submodule is created in the current directory. Its filename is the source filename with the suffix replaced by the suffix `.o`.

When a submodule is compiled, an information file is created by the compiler. The name of the file is "ancestor_module_name.submodule_name.smod". This information file is needed when compiling a program that is a descendant submodule of the submodule. The `.smod` information file is created according to the following rules:

- If the compiler option `-M` is specified, it is created in the directory specified in the `-M` option.
- If the compiler option `-M` is not specified, it is created in the directory from which `frtpx` is being executed.

When the programming unit of descendant submodule of that submodule is compiled, the corresponding information file is searched for in the following order:

- A directory named in the `-M` option (if specified).
- The directory from which `frtpx` is being executed.
- A directory named in the `-I` option (if specified).

When linking a program containing a module reference, you must specify the object file of the corresponding submodule.

Example 1:

The submodule and the module reference:

```
! file name: a.f90
```

10.5 Intrinsic Modules

The intrinsic modules are provided in the processor. There are standard intrinsic module and non-standard intrinsic modules in the intrinsic modules.

The standard intrinsic modules are the following modules:

- Fortran environment module (`ISO_FORTRAN_ENV`)
- IEEE module (`IEEE_EXCEPTIONS`, `IEEE_ARITHMETIC` and `IEEE_FEATURES`)
- Module for interoperable with C program (`ISO_C_BINDING`)

The non-standard intrinsic module are the following modules:

- `SERVICE_ROUTINES` for service routines. See "[6.6.6 Service Routines](#)" for service routines.
- `OMP_LIB`, and `OMP_LIB_KINDS` for OpenMP
- `MPI`, names starting with `MPI_`, names starting with `PMPI_`, and names starting with `OMPI_` for MPI
- `FAST_DD`, `PLASMA`, `PLASMA_S`, `PLASMA_C`, `PLASMA_Z`, `PLASMA_D`, `PLASMA_ZC`, and `PLASMA_DS` for mathematical library

Note the following when the intrinsic module is referenced.

- When the compiler option `-AU` is specified, the language entities of intrinsic modules must be lowercase letter.

10.5.1 COMPILER_OPTIONS Intrinsic Module Function

Compiler options information on the function result is the same as compiler options information with compiler option -Q or -Nlst. Consider 1200 characters or more for character strings of the function result on programming.

10.5.2 COMPILER_VERSION Intrinsic Module Function

Version information on the function result is the same as information on the Fortran compiler when compiler option -V is specified. Consider 90 characters or more for character strings of the function result on programming.

Chapter 11 Mixed Language Programming

This chapter describes the specification and notes when linking a Fortran program and a C/C++ program.

Note

The description of C program on this chapter is applied C++ program. In linking with the C++ program, see Section "Notes on Linking with Different Languages and Trad/Clang Modes" of the "C++ User's Guide" in addition to this chapter.

11.1 Compile Commands and Required Options when Linking

In this system, there are two types of compile commands for the Fortran, C, and C++ languages, a cross compiler and a native compiler. In addition, the compiler commands are used depending on whether the program uses the MPI library as shown in the table below.

The following description uses the compile command of the cross compiler (No use of MPI libraries). When using the MPI library or native compiler, replace the compile commands in the table below.

Programming Language	MPI Library	Cross Compiler	Native Compiler
Fortran	Not Use	frtpx	firt
	Use	mpifrtpx	mpifirt
C	Not Use	fccpx	fcc
	Use	mpifccpx	mpifcc
C++	Not Use	FCCpx	FCC
	Use	mpiFCCpx	mpiFCC

Object programs can be linked in a combination of Fortran, C Trad Mode, C Clang Mode, C++ Trad Mode, and C++ Clang Mode. For more information on Trad Mode and Clang Mode, see the "C++ User's Guide" and "C++ User's Guide".

Depending on the combination of object programs, some compilation commands can not use at linking. For object programs combinations, and compilation commands/required options at linking, see "[Table 11.1 Object programs combinations, compilation commands, and required options](#)". For more information on the compile commands and options, see the user's guide for each language.

Table 11.1 Object programs combinations, compilation commands, and required options

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for STL	Use libstdc++ for STL		
*	*	-	-	-	-	Not Use	frtpx or fccpx --linkfortran
						Use	frtpx -Ncoarray or fccpx --linkcoarray
*	-	*	-	-	-	Not Use	frtpx or fccpx -Nclang --linkfortran
						Use	frtpx -Ncoarray or fccpx -Nclang --linkcoarray

Object to Link						COARRAY	Compile Commands/Required Options	
Fortran	C trad	C clang	C++ trad	C++ clang				
				Use libc++ for STL	Use libstdc++ for STL			
*	-	-	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran	
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray	
*	-	-	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++	
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray	
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran	
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray	
-	*	*	-	-	-	-	fccpx -Nclang	
-	*	-	*	-	-	-	-	FCCpx
-	*	-	-	*	-	-	-	FCCpx -Nclang
				-	*			FCCpx -Nclang -stdlib=libstdc++
-	-	*	*	-	-	-	-	FCCpx
-	-	*	-	*	-	-	-	FCCpx -Nclang
				-	*			FCCpx -Nclang -stdlib=libstdc++
-	-	-	*	*	-	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*			Can not to be linked. (Refer to "Note 2")
*	*	*	-	-	-	Not Use	frtpx or fccpx -Nclang --linkfortran	
						Use	frtpx -Ncoarray or fccpx -Nclang --linkcoarray	
*	*	-	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran	
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray	

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for STL	Use libstdc++ for STL		
*	*	-	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
*	-	*	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray
*	-	*	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
*	-	-	*	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for STL	Use libstdc++ for STL		
-	*	*	*	-	-	-	FCCpx
-	*	*	-	*	-	-	FCCpx -Nclang -stdlib=libc++
				-	*		FCCpx -Nclang -stdlib=libstdc++
-	*	-	*	*	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
-	-	*	*	*	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
*	*	*	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray
*	*	*	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
*	*	-	*	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	Can not to be linked. (Refer to "Note 2")
*	-	*	*	*	-	Not Use	frtpx --linkstl=libc++ or

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for STL	Use libstdc++ for STL		
							FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	
	*	*	*	*	-		FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
				*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
*	*	*	*			Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	

11.1.1 Linking C++ Trad Mode with C++ Clang Mode

Whether object programs generated in C++ Trad Mode and object programs generated in C++ Clang Mode can be linked depends on STL used at compilation.

C++ Trad Mode	C++ Clang Mode	Compile Commands/Required Options
(libc++ use only)	Use libc++ for STL	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
	Use libstdc++ for STL	Can not to be linked. (Refer to "Note 2")

Note 1

When all of the following conditions are met, the operation may become indefinite and can not be guaranteed.

1. An object generated in Clang Mode and an object generated in Trad Mode have an interface between functions.
2. An argument or return value of the function in condition 1 is a class type.

Note 2

The default STL that C++ Clang Mode uses is libstdc++.

Specify libc++ instead of the default libstdc++ for STL used in Clang Mode.

11.1.2 Linking C++ Clang Mode with C++ Clang Mode

Whether object programs generated in C++ Trad Mode and object programs generated in C++ Clang Mode can be linked depends on STL used at compilation.

STL used in compilation		Compile Commands/Required Options
libc++	libc++	FCCpx -Nclang -stdlib=libc++
libc++	libstdc++	Can not to be linked.
libstdc++	libstdc++	FCCpx -Nclang -stdlib=libstdc++ (Note that the -stdlib = libstdc++ option can be omitted because it is the default.)

11.1.3 Linking MPI Object Programs and Use Profiler

Specify the following option at linkage:

- Use mpifrtpx command : -lfjprofmpi option
- Use mpifccpx command or mpiFCCpx command : -lfjprofmpif option

11.1.4 Objects Generated in Clang Mode with -flto Option

Objects generated with the "-Nclang -flto" options are in the format used internally by LLVM (different format from normal objects) and can only be linked in Clang Mode.

When including an object generated with the "-Nclang -flto" options at mixed language programming, specify the "-Nclang -flto" options at linking.

11.2 Features for Mixed Language Programming

This system has Fortran standard and FUJITSU Fortran extension specifications for mixed language programming.

The following Fortran standard specifications for mixed language programming are offered.

- Intrinsic module ISO_C_BINDING
- BIND statement, BIND attribute specifier, Language binding specifier, Procedure language binding specifier
- VALUE statement, VALUE attribute specifier
- Assumed type
- Assumed rank
- ISO_Fortran_binding.h

The following FUJITSU Fortran extension specifications for mixed language programming are offered.

- VAL intrinsic function, %val(a) specifies that an actual argument
- CHANGEENTRY statement
- \$pragma specifier

The following features are supported by compiler option.

- The compiler options -ml and -mldefault to change how to make external procedure. See Section "[11.3 Rules for Processing Fortran Procedure Names](#)".
- The compiler option -Az to change how to call the value of character type for the argument. See Section "[11.13 Notes of Compiler Option -Az](#)".

11.2.1 Passing Arguments by Value

By default, actual arguments of external procedure reference are called by address. See Section "11.7 Passing and Getting Arguments". In the following cases, the actual argument is passed by value.

- The intrinsic function VAL(a) or %val(a) for actual argument is specified.
- The VALUE attribute is specified in a procedure interface.

11.2.1.1 Passing Arguments by Value in Procedure Interface

When the VALUE attribute is specified in procedure interface, the argument is passed by value.

See Section "11.2.2 Getting Arguments by Value" for information about the actual argument that can be associated.

Example: the VALUE attribute in procedure interface

```
INTERFACE
  SUBROUTINE CSUB(ARG1,ARG2)  BIND(C)
  USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT
  INTEGER(C_INT), VALUE::ARG1
  INTEGER(C_INT)           ::ARG2
  END SUBROUTINE CSUB
END INTERFACE
CALL CSUB(2,3)              ! The first argument is passed by value,
                           !the second argument is called by address.
END
```

11.2.1.2 Passing Arguments by Value in Intrinsic Function

When the compiler option -Nobsfun is specified, the function VAL(a) and %val(a) are intrinsic functions. The intrinsic function VAL(a) and %val(a) specifier can specify only for actual argument of external procedure reference. The actual argument passed by value must be scalar expression of type one-byte integer, two-byte integer, four-byte integer, eight-byte integer, one-byte logical, two-byte logical, four-byte logical, eight-byte logical, half-precision real, single-precision real or double-precision real.

Example: intrinsic function VAL()

```
CALL CSUB(VAL(2),3)        ! first argument passed by value,
                           ! second argument passed by address
END
```

11.2.2 Getting Arguments by Value

By default, dummy arguments of external procedure are got by address. See Section "11.7 Passing and Getting Arguments". The VALUE attribute is specified for a dummy argument of an external procedure, the Fortran program gets the dummy arguments by value. The VALUE attribute can specify by the VALUE statement or attribute specifier in a type declaration statement.

When the dummy argument has the Quadruple-precision complex, the character type of procedure without procedure language binding specifier, the derived type whose size is 16 bytes or less, the derived type who has POINTER attribute in component, the derived type who has ALLOCATABLE attribute in component or OPTIONAL attribute, the address of the variable is always received.

Example: VALUE statement

```
SUBROUTINE CSUB(I,K)
VALUE :: I                ! first argument passed by value,
                           ! second argument passed by address
RRINT *, I,K
END
```

11.2.3 CHANGEENTRY Statement

The CHANGEENTRY statement changes rules for processing Fortran procedure names combine with -ml compiler option. See Section "11.3 Rules for Processing Fortran Procedure Names" for details.

The external procedure name must not have the procedure language binding specifier.

Example: CHANGEENTRY statement

```
CHANGEENTRY :: CSUB
CALL CSUB()
END
```

11.2.4 \$pragma Specifier

\$pragma specifier has the following form and can be specified like a comment.

```
!$pragma c (prc)
```

prc must be an external procedure name. The \$pragma specifier specifies that the procedure *prc* is coded by C program and changes rules for processing procedure names. See Section "11.3 Rules for Processing Fortran Procedure Names".

The external procedure name of \$pragma specifier must not have the procedure language binding specifier.

Example: \$pragma specifier

```
EXTERNAL CSUB          !$pragma C(CSUB)
:
CALL CSUB              ! CSUB is treated as a calling procedure in C program.
END
```

11.2.5 Intrinsic Module ISO_C_BINDING

The following specifications are offered by intrinsic module ISO_C_BINDING.

- Named constants for interoperable with C program
- Types for interoperable with C program
- Intrinsic procedures for interoperable with C program

11.2.5.1 Named Constants by Intrinsic Module ISO_C_BINDING

The following named constants are offered by intrinsic module ISO_C_BINDING.

- Named constants for kind type parameter
 - Named constants for names of C program character
 - Named constants for null pointer
1. The value of named constants by intrinsic module ISO_C_BINDING and C program types correspond in "[Table 11.2 Types of C program and, types and kind type parameters of Fortran correspond](#)". If data is passed between Fortran and C programs, type of C program and, type and kind type parameter of Fortran must correspond. If data type is character, the value of type length parameter must be 1.

Table 11.2 Types of C program and, types and kind type parameters of Fortran correspond

Named constants	Value	C program types	Type specifiers of Fortran
C_SHORT	2	short int	INTEGER
C_INT	4	int	INTEGER
C_LONG	8	long int	INTEGER
C_LONG_LONG	8	long long int	INTEGER
C_SIGNED_CHAR	1	signed char / unsigned char	INTEGER
C_SIZE_T	8	size_t	INTEGER
C_INT8_T	1	int8_t	INTEGER

Named constants	Value	C program types	Type specifiers of Fortran
C_INT16_T	2	int16_t	INTEGER
C_INT32_T	4	int32_t	INTEGER
C_INT64_T	8	int64_t	INTEGER
C_INT_LEAST8_T	1	int_least8_t	INTEGER
C_INT_LEAST16_T	2	int_least16_t	INTEGER
C_INT_LEAST32_T	4	int_least32_t	INTEGER
C_INT_LEAST64_T	8	int_least64_t	INTEGER
C_INT_FAST8_T	1	int_fast8_t	INTEGER
C_INT_FAST16_T	8	int_fast16_t	INTEGER
C_INT_FAST32_T	8	int_fast32_t	INTEGER
C_INT_FAST64_T	8	int_fast64_t	INTEGER
C_INTMAX_T	8	intmax_t	INTEGER
C_INTPTR_T	8	intptr_t	INTEGER
C_FLOAT_16	2	_Float16	REAL
C_FLOAT	4	float	REAL
C_DOUBLE	8	double	REAL
C_LONG_DOUBLE	16	long double	REAL
C_FLOAT_16_COMPLEX	2	_Float16_Complex	COMPLEX
C_FLOAT_COMPLEX	4	float_Complex	COMPLEX
C_DOUBLE_COMPLEX	8	double_Complex	COMPLEX
C_LONG_DOUBLE_COMPLEX	16	long double_Complex	COMPLEX
C_BOOL	1	_Bool	LOGICAL
C_CHAR	1	char	CHARACTER

Example: Using of named constant of kind type parameter by intrinsic module ISO_C_BINDING
 USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT

```

INTEGER(C_INT)::J,K,L
INTERFACE
  SUBROUTINE SUB(J,K,L) BIND(C)
    USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
    INTEGER(C_INT)::J,K,L
  END SUBROUTINE
END INTERFACE
J = 10
K = 20
L = 30
CALL SUB(J,K,L)
END

```

C program:

```

#include <stdio.h>
void sub(j,k,l)
int *j,*k,*l;
{
printf ("J:%d K:%d L:%d \n",*j,*k,*l) ;
}

```

2. Named constants of C program character by intrinsic module ISO_C_BINDING, and the meaning are follows.

Named constants	C program characters
C_NULL_CHAR	null character
C_ALERT	alert
C_BACKSPACE	backspace
C_FORM_FEED	form feeds
C_NEW_LINE	new line
C_CARRIAGE_RETURN	carriage return
C_HORIZONTAL_TAB	horizontal tab
C_VERTICAL_TAB	horizontal tab

The type of named constants is character type and length type parameter is 1.

Example: Using of named constant of C program character by intrinsic module ISO_C_BINDING

```
USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_NULL_CHAR
CHARACTER (LEN=5) :: ST
ST='ABCD' // C_NULL_CHAR ! Append null character
:
END
```

3. Named constants of null pointer by intrinsic module ISO_C_BINDING, the meaning and type are follows.

Named constants	Meaning	Type
C_NULL_PTR	NULL	C_PTR
C_NULL_FUNPTR	null pointer to function	C_FUNPTR

Example: Using of named constant of null pointer by intrinsic module ISO_C_BINDING

```
USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_PTR, C_NULL_PTR
INTERFACE
  SUBROUTINE SUB(PTR) BIND(C)
    USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_PTR
    TYPE(C_PTR), VALUE :: PTR
  END SUBROUTINE
END INTERFACE
TYPE(C_PTR) :: PTR
PTR=C_NULL_PTR ! Set NULL
CALL SUB(PTR)
END
```

11.2.5.2 Types by Intrinsic Module ISO_C_BINDING

C_PTR type and C_FUNPTR type are offered by intrinsic module ISO_C_BINDING.

C_PTR type is a pointer type of C program. C_FUNPTR type is a function pointer type of C program. C_PTR and C_FUNPTR are derived type with private component.

11.2.5.3 Intrinsic Procedures by Intrinsic Module ISO_C_BINDING

Intrinsic functions C_LOC, C_FUNLOC, C_ASSOCIATED, C_SIZEOF and, Intrinsic subroutines C_F_POINTER, C_F_PROCPOINTER are offered by intrinsic module ISO_C_BINDING.

C_LOC gets pointer (C_PTR type) of C program corresponding to the argument.

C_FUNLOC gets function pointer (C_FUNPTR type) of C program corresponding to the argument.

C_ASSOCIATED compares pointer or function pointer of C program corresponding to the argument.

C_F_POINTER assigns data pointer of C program to pointer of Fortran.

C_F_PROCPOINTER assigns function pointer of C program to procedure pointer of Fortran.

C_SIZEOF gets size of the argument.

See online manual intrinsic(3) for details of these intrinsic procedures.

Example: Using intrinsic procedure by intrinsic module ISO_C_BINDING

```
USE, INTRINSIC::ISO_C_BINDING, ONLY:C_PTR,C_INT,C_LOC
INTERFACE
  SUBROUTINE SUB(PTR) BIND(C)
    USE, INTRINSIC::ISO_C_BINDING, ONLY:C_PTR
    TYPE(C_PTR), VALUE::PTR
  END SUBROUTINE
END INTERFACE
TYPE(C_PTR)::PTR
INTEGER(C_INT), TARGET::TARGET
TARGET=2
PTR=C_LOC(TARGET) ! Set C program pointer of variable TARGET
CALL SUB(PTR)
END
```

C program:

```
#include <stdio.h>
void sub(k)
int *k;
{
  printf ("K=%d \n",*k) ;
}
```

11.2.6 BIND Statement, BIND Attribute Specifier, Language Binding Specifier and Procedure Language Binding Specifier

When a language binding specifier is specified, data can interoperate between Fortran and C program.

When a procedure language binding is specified, procedure can interoperate between Fortran and C program.

The language binding specifier and procedure language binding specifier are following syntax.

```
BIND ( C [, NAME = scalar character constant expression] )
```

External name of language entity is decided by language binding specifier or procedure language binding specifier. The external name by the specifier is called a binding label.

If the language binding specifier or procedure language binding specifier is specified, the external name is the follows. However, value of NAME specifier is discarding leading and trailing blanks.

- If NAME specifier is not specified or length of NAME specifier value is zero, the binding label is the same as the name of the entity using lower case letters.
- If NAME specifier value has nonzero length, the binding label has the value. The case of letters in the binding label is significant.

If the language binding specifier of a BIND statement or attribute specifier in type declaration statement is specified, a variable or common block linkage interoperates with an external variable of C program.

If the language binding specifier is specified, the data entity has BIND attribute.

A procedure language binding specifier can specify in SUBROUTINE statement, FUNCTION statement or ENTRY statement, and the procedure interoperates with C program. A procedure language binding specifier must not be specified in the SUBROUTINE statement or FUNCTION statement of an internal procedure.

Example: Specify language binding specifier and procedure language binding specifier

```
MODULE MOD
  INTEGER, BIND(C, NAME='gvar01') :: VAR
  INTEGER, BIND(C
                ) :: GVAR02
END MODULE

USE MOD
USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_INT
INTERFACE
  SUBROUTINE SUB() BIND(C, NAME='Sub_proc')
  END SUBROUTINE
END INTERFACE
INTEGER(C_INT) :: K1, K2
COMMON /COM/ K1, K2
BIND(C, NAME='gvar03') :: /COM/
VAR=1
GVAR02=2
K1=3
K2=4
CALL SUB
END
```

C program

```
#include <stdio.h>
struct {int k1; int k2;} gvar03;
int gvar01;
int gvar02;

void Sub_proc(void)
{
  printf ("gvar01=%d \n", gvar01) ;
  printf ("gvar02=%d \n", gvar02) ;
  printf ("gvar03.k1=%d \n", gvar03.k1) ;
  printf ("gvar03.k2=%d \n", gvar03.k2) ;
}
```

11.2.6.1 BIND Statement

A BIND statement specify BIND attribute, and the statement has the following form.

```
Language binding specifier[ :: ] binding-entity-list
```

The binding-entity is a variable name or /common-block-name/. If a variable name is specified, the BIND statement must appear in declaration part of module.

If a NAME specifier specify in language binding specifier, then binding-entity-list must contain exactly one binding-entity.

Example: Interoperates with C program by BIND statement

```
MODULE MOD
  USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_INT
  INTEGER(C_INT) :: N
  INTEGER(C_INT) :: K1, K2
  COMMON /COM/ K1, K2
  BIND(C) :: N, /COM/
END MODULE

USE MOD
INTERFACE
  SUBROUTINE SUB() BIND(C)
  END SUBROUTINE
END INTERFACE
```



```
N=1
K1=2
K2=3
CALL SUB
END
```

C program

```
#include <stdio.h>
struct {int k1; int k2;} com;
int n;

void sub(void)
{
printf ("n=%d \n",n) ;
printf ("com.k1=%d \n",com.k1) ;
printf ("com.k2=%d \n",com.k2) ;
}
```

11.2.6.2 BIND Attribute by Attribute Specifier of Type Declaration Statement

A language binding specifier is specified in attribute specifier of type declaration statement, BIND attribute is specified.

A language binding specifier of type declaration statement can specify in declaration part of module. If a NAME specifier specify in language binding specifier, then variable list must contain exactly one variable.

Example: Specify language binding specifier of type declaration statement.

```
MODULE MOD
  INTEGER,BIND(C,NAME='global01')::VAR
  INTEGER,BIND(C
                ):: GLOBAL02,GLOBAL03
END MODULE

USE MOD
INTERFACE
  SUBROUTINE SUB() BIND(C)
  END SUBROUTINE
END INTERFACE
VAR=1
GLOBAL02=2
GLOBAL03=3
CALL SUB
END
```

C program

```
#include <stdio.h>
int global01;
int global02;
int global03;

void sub(void)
{
printf ("global01=%d \n",global01) ;
printf ("global02=%d \n",global02) ;
printf ("global03=%d \n",global03) ;
}
```

11.2.6.3 Procedure Language Binding Specifier

A procedure language binding specifier specify following dummy argument list in SUBROUTINE statement, FUNCTION statement or ENTRY statement. When the procedure language binding is specified and the procedure does not have dummy argument list, parentheses enclose the dummy argument are not omissible.

Example: Specify procedure language binding specifier

```
SUBROUTINE SUB(K) BIND(C)
  USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT
  INTEGER(C_INT)::K
  PRINT *, 'K=', K
END SUBROUTINE
FUNCTION IFUN(K) BIND(C, NAME='fun') RESULT(R)
  USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT
  INTEGER(C_INT)::R, K
  R=K
END FUNCTION
```

C program

```
#include <stdio.h>
void sub(int *);
int fun(int *);
int foo(void)
{
  int n, k;
  k=2;
  sub(&k);
  n=fun(&k);
  printf (" n= %d \n", n);
  return 0;
}
```

11.2.7 VALUE Statement, VALUE Attribute Specifier

A VALUE statement or VALUE attribute specifier of type declaration statement is specified, the dummy argument has VALUE attribute. If program calls procedure of VALUE attribute dummy argument, the procedure must have an explicit interface. When a dummy argument has VALUE attribute, the dummy argument is passed by value without follows:

- A character type of procedure without procedure language binding
- A component has POINTER attribute in derived type
- A component has ALLOCATABLE attribute in derived type
- OPTIONAL attribute

Example: Specify a VALUE attribute and VALUE statement

```
SUBROUTINE SUB(K) BIND(C)
  USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT
  INTEGER(C_INT), VALUE::K
  PRINT *, 'K=', K
END SUBROUTINE
!
USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT
INTERFACE
  SUBROUTINE FOO(K1, K2) BIND(C)
    USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT
    INTEGER(C_INT), VALUE::K1
    INTEGER(C_INT)      : :K2
    VALUE::K2
  END SUBROUTINE
END INTERFACE
INTEGER(C_INT)::N1, N2
N1=1
N2=2
CALL FOO(N1, N2)
END
```

C program

```
void sub(int);
int foo(int k1,int k2)
{
sub(k1) ;
sub(k2) ;
return 0 ;
}
```

11.2.8 Interoperability of Derive Type and C Language Structure Type

In case Fortran derived type is interoperable with structure type of C language, the following conditions should be met:

- Fortran derived type must have BIND Attribute.
- Fortran derived type must not be sequence type.
- Fortran derived type must not have Type parameter.
- Fortran derived type must not have EXTENDS attribute.
- Fortran derived type must not have type bound procedure part.
- Each component of Fortran derived type must be interoperable type as well as type parameter.
- Each component of Fortran derived type must not be pointer or allocatable.
- Fortran derived type must have same number of components as that of C language structure type.
- Each component of Fortran derived type must have component type which supports structure type of C language, interoperable type and type parameter.

11.2.9 Assumed Type

An assumed type is declared with a TYPE(*) type specifier and it is an unlimited polymorphic entity with dummy argument.

If a dummy argument is an assumed type without assumed rank, the formal parameter of C program is a pointer to void.

Example: Using an assumed type without assumed rank

Fortran program:

```
use iso_c_binding,only:c_int, c_long, c_float
interface
  subroutine c_sub(d,n) bind(c)
  import:: c_int
  type(*):: d
  integer(c_int),value:: n
  end subroutine
end interface
integer(c_int) :: a=10
integer(c_long):: b=11
real (c_float)::c=12.0
call c_sub(a,1)
call c_sub(b,2)
call c_sub(c,3)
end
```

C program:

```
#include<stdio.h>
#include<stdlib.h>
void c_sub(void *p, int n)
{
  switch (n) {
    case 1:
```

```

    printf (" Value : %d \n", *(int *)p ) ;
    break ;
case 2:
    printf (" Value : %ld \n", *(long int *)p ) ;
    break ;
case 3:
    printf (" Value : %f \n", *(float *)p ) ;
}
return ;

```

Execution result:

```

Value : 10
Value : 11
Value : 12.000000

```

11.2.10 Assumed Rank

An assumed rank is a dummy data object that assumes the rank, shape, and size of its effective actual argument.

If a dummy argument is an assumed rank, the formal parameter of C program is a pointer to CFI_cdesc_t. See "[11.2.11 ISO_Fortran_binding.h](#)" for CFI_cdesc_t.

Example1: Using an assumed rank with contiguous array

Fortran program:

```

use iso_c_binding,only:c_int
interface
  subroutine c_sub(d) bind(c)
    type(*):: d(..)
  end subroutine
end interface
integer(c_int):: a(2,3)=reshape([1,2,3,4,5,6],[2,3])
call c_sub(a(:,2)) ! Contiguous
end

```

C program:

```

#include"ISO_Fortran_binding.h"
#include<stdio.h>
#include<stdlib.h>
void c_sub(CFI_cdesc_t *cfi_ptr)
{
  int d,d1,d2 ;
  char *p ;

  printf ("Element length : %ld \n", cfi_ptr->elem_len) ;
  switch (cfi_ptr->type) {
    case CFI_type_int:
      printf (" Type : int \n") ;
      break ;
    default:
      printf (" Type : Other \n") ;
  }
  switch (cfi_ptr->attribute) {
    case CFI_attribute_other:
      printf (" Attribute : other \n") ;
  }
  printf (" Rank : %ld \n", cfi_ptr->rank) ;
  if (cfi_ptr->rank != 0) {
    for (d=0; d < cfi_ptr->rank ; ++d) {
      printf (" Dimension : %d \n", d+1) ;
      printf (" Lower bound : %ld \n", cfi_ptr->dim[d].lower_bound);
    }
  }
}

```

```

        printf ("   Element num  : %ld \n", cfi_ptr->dim[d].extent);
        printf ("   Memory stride: %ld \n", cfi_ptr->dim[d].sm);
    }
}

if (cfi_ptr->rank == 0) {
    p = cfi_ptr->base_addr ;
    printf (" Value           : %d \n", *(int *)p) ;
} else {
    if (cfi_ptr->rank == 2) {
        printf (" Values           : ") ;
        for (d2=0; d2 < cfi_ptr->dim[1].extent ; ++d2) {
            for (d1=0; d1 < cfi_ptr->dim[0].extent ; ++d1) {
                p = cfi_ptr->base_addr ;
                p = p + (d1*cfi_ptr->dim[0].sm) + (d2*cfi_ptr->dim[1].sm) ;
                printf ("%d ", *(int *)p) ;
            }
        }
    }
    printf ("\n");
}
return ;
}

```

Execution result:

```

Element length  : 4
Type            : int
Attribute       : other
Rank           : 2
Dimension      : 1
  Lower bound  : 0
  Element num  : 2
  Memory stride: 4
Dimension      : 2
  Lower bound  : 0
  Element num  : 2
  Memory stride: 8
Values         : 1 2 3 4

```

Example2: Using an assumed rank with non-contiguous array

Fortran program:

```

use iso_c_binding, only: c_int
interface
  subroutine c_sub(d) bind(c)
    type(*) :: d(..)
  end subroutine
end interface
integer(c_int) :: a(2,3)=reshape([1,2,3,4,5,6],[2,3])
call c_sub(a(:,:,2)) ! Non contiguous
end

```

C program:

See "[Example1: Using an assumed rank with contiguous array](#)" for the C program.

Execution result:

```

Element length  : 4
Type            : int
Attribute       : other
Rank           : 2
Dimension      : 1

```

```

Lower bound : 0
Element num : 2
Memory stride: 4
Dimension   : 2
Lower bound : 0
Element num : 2
Memory stride: 16
Values      : 1 2 5 6

```

Example3: Using an assumed rank with scalar

Fortran program:

```

use iso_c_binding, only: c_int
interface
  subroutine c_sub(d) bind(c)
    type(*) :: d(..)
  end subroutine
end interface
integer(c_int) :: a(2,3)=reshape([1,2,3,4,5,6],[2,3])
call c_sub(a(2,1)) ! Scalar
end

```

C program:

See "[Example1: Using an assumed rank with contiguous array](#)" for the C program.

Execution result:

```

Element length : 4
Type           : int
Attribute      : other
Rank           : 0
Value          : 2

```

11.2.11 ISO_Fortran_binding.h

ISO_Fortran_binding.h exists in install directory that contains headers provided by this system.

The file is a header file of C program and it is used with assumed rank for mixed language programming. See "[11.2.10 Assumed Rank](#)" for assumed rank.

C program that includes the ISO_Fortran_binding.h must not use any names starting with CFI_ that are not defined in the header.

The following specifications are offered by ISO_Fortran_binding.h.

- CFI_cdesc_t structure type
- CFI_dim_t structure type
- Macros and typedefs

This system does not offer C functions for ISO_Fortran_binding.h.

11.2.11.1 CFI_cdesc_t Structure Type

CFI_cdesc_t is a typedef name for a C structure. The following members exist:

1. base_addr

An address of object.

2. elem_len

If the object is a scalar, the value is size in bytes. If the object an array, the value is size of an element in bytes.

3. version

The value is equal to the value of CFI_VERSION in ISO_Fortran_binding.h.

4. rank

The value is equal to rank of array. If the object is a scalar, the value is zero.

5. type

The value is the value for type code. See "11.2.11.3 Type and Macro Names in ISO_Fortran_binding.h" for types and the macro name.

6. attribute

The value is the value of attribute code. The attribute is only "nonallocatable nonpointer" in this system.

7. dim

The dim is declared with array. The number of elements in the dim array is equal to the rank of the object. The dim has a CFI_dim_t structure type.

11.2.11.2 CFI_dim_t Structure Type

CFI_dim_t is a typedef name for a C structure. It is used to represent lower bound, extent, and memory stride. The following members exist:

1. lower_bound

The value is equal to the value of the lower bound for the each dimension. The value is zero.

2. extend

The value is equal to the number of elements in the each dimension or -1 for the final dimension of an assumed-size array.

3. sm

The value is equal to the memory stride for an each dimension. The value is equal to between the addresses of successive elements.

11.2.11.3 Type and Macro Names in ISO_Fortran_binding.h

CFI_CDESC_T is a function-like macro that takes one argument. The argument is the rank of the C descriptor. The argument must be an integer constant expression with a value that is greater than or equal to zero and less than or equal to 30.

CFI_index_t is a typedef name for the lower bound, the elements number, or the memory stride.

CFI_MAX_RANK macro has value of 30. The value is a max value of array rank in this system.

CFI_VERSION macro has value of 1. The value is a version number of ISO_Fortran_binding.h in this system.

CFI_attribute_other macro is specified an attribute with nonallocatable and nonpointer. This system does not support a pointer and allocatable attribute in ISO_Fortran_binding.h.

CFI_type_t is a typedef name for type code. See "Table 11.3 ISO_Fortran_binding.h macro names and corresponding types" for the types corresponding to macro names.

Table 11.3 ISO_Fortran_binding.h macro names and corresponding types

Macro names	C program types
CFI_type_signed_char	signed char
CFI_type_short	short int
CFI_type_int	int
CFI_type_long	long int
CFI_type_long_long	long long int
CFI_type_size_t	size_t
CFI_type_int8_t	int8_t
CFI_type_int16_t	int16_t
CFI_type_int32_t	int32_t
CFI_type_int64_t	int64_t
CFI_type_int_least8_t	int_least8_t

Macro names	C program types
CFI_type_int_least16_t	int_least16_t
CFI_type_int_least32_t	int_least32_t
CFI_type_int_least64_t	int_least64_t
CFI_type_int_fast8_t	int_fast8_t
CFI_type_int_fast16_t	int_fast16_t
CFI_type_int_fast32_t	int_fast32_t
CFI_type_int_fast64_t	int_fast64_t
CFI_type_intmax_t	intmax_t
CFI_type_intptr_t	intptr_t
CFI_type_ptrdiff_t	ptrdiff_t
CFI_type_Float16	_Float16
CFI_type_float	float
CFI_type_double	double
CFI_type_long_double	long double
CFI_type_Float16_Complex	_Float16_Complex
CFI_type_float_Complex	float_Complex
CFI_type_double_Complex	double_Complex
CFI_type_long_double_Complex	long double_Complex
CFI_type_Bool	_Bool
CFI_type_char	char
CFI_type_cptr	void *
CFI_type_struct	Interoperable C structure
CFI_type_other	Not otherwise specified

11.3 Rules for Processing Fortran Procedure Names

In the object, the Fortran procedure name processed from source name. If a Fortran procedure name is called from C program, the name must conform to the Fortran name rules.

The following table lists the rules for processing Fortran procedure names.

Table 11.4 Rules for processing Fortran procedure names

Procedure name		Processing name
Main program name		MAIN__
External procedure name		external-procedure-name_
Block data program unit	The BLOCK DATA program with a name	block-data-program-unit-name_
	The BLOCK DATA program without a name	_BLKDT__
Startup routine		main

Notes:

1. All procedure names are uniformly lowercase if compiler option -AU is not specified.
2. The startup routine is the routine to activate the Fortran system.

If compiler option -AU is specified, the processing names are same spell in source program.

The processing name of the procedure with procedure language binding specifier is the binding label.

Rules for Processing Fortran Procedure Names can be changed by compiler option `-mldefault`.

When the compiler option `-mldefault` is specified, the procedures with procedure language binding specifier, the procedures specified in `CHANGEENTRY` statement and the external procedures name specified in `$pragma` specifier are not effective.

The following table lists the rules for processing Fortran external procedure names when the `-mldefault` option is specified.

Table 11.5 Rules for processing Fortran external procedure names when the `-mldefault` option is specified

Compiler option <code>-mldefault</code>	Processing name
<code>-mldefault=cdecl</code>	external-procedure-name
<code>-mldefault=frt</code>	external-procedure-name_
Not specified	

Compiler option `-ml` changes rules for processing Fortran external procedure names of external procedure name statement. The following table lists rules for processing Fortran external procedure names.

Table 11.6 Rules for processing Fortran external procedure names specified in `CHANGEENTRY` statement

Compiler option <code>-ml</code>	Processing name
<code>-ml=cdecl</code>	external-procedure-name
<code>-ml=frt</code>	external-procedure-name_
Not specified	in accordance with specifying <code>-mldefault</code> option

The following table lists rules for processing Fortran external procedure names specified in `$pragma` specifier.

Table 11.7 Rules for processing Fortran external procedure names in `$pragma` specifier

<code>\$pragma</code> specifier	Processing name
<code>c (prc)</code>	external-procedure-name

11.4 Correspondence of Type with C

If data is passed between Fortran and other programs, data attributes must correspond. See Section "[5.1 Internal Data Representation](#)" for Fortran data types.

When data is passed between Fortran and C programs, it is necessary to make the type correspond.

See "[Table 11.2 Types of C program and, types and kind type parameters of Fortran correspond](#)" for the type that can be shared when the `BIND` attribute is specified.

The following table lists data type that Fortran and C when `BIND` attribute is not specified correspond.

Table 11.8 Data type that A and B correspond

Data Type	Fortran	C
One-byte logical	LOGICAL(1) L1	unsigned char L1 ;
Two-byte logical	LOGICAL(2) L2	short int L2 ;
Four-byte logical	LOGICAL(4) L4	int L4 ;
Eight-byte logical	LOGICAL(8) L8	long long int L8 ;
One-byte integer	INTEGER(1) I1	signed char I1 ;
Two-byte integer	INTEGER(2) I2	short int I2 ;
Four-byte integer	INTEGER(4) I4	int I4 ;
Eight-byte integer	INTEGER(8) I8	long long int I8 ;
Half-precision real	REAL(2) R2	_Float16 R2;

Data Type	Fortran	C
Single-precision real	REAL(4) R4	float R4 ;
Double-precision real	REAL(8) R8	double R8 ;
Quadruple-precision real	REAL(16) R16	long double R16 ;
Half-precision complex	COMPLEX(2) C4	_Float16 _Complex C4
Single-precision complex	COMPLEX(4) C8	float _Complex C8;;
Double-precision complex	COMPLEX(8) C16	double _Complex C16;
Quadruple-precision complex	COMPLEX(16) C32	long double _Complex C32;
Character	CHARACTER(10) S	char S [10] ;
Derived type	TYPE TAG INTEGER I4 REAL(8) R8 END TYPE TAG TYPE(TAG)::D	struct tag { int i4; double r8; } d;

11.5 Calling Procedures

If a Fortran procedure name is called from a C program or C function name is called from a Fortran program, the processing name must confirm. See Section "11.3 Rules for Processing Fortran Procedure Names" for processing Fortran procedure names.

The example of the calling procedure is shown as follows.

Example 1: Calling C function from Fortran program with the procedure language binding specifier.

Fortran program:

```
INTERFACE
  SUBROUTINE SUB() BIND(C)
  END SUBROUTINE
END INTERFACE
CALL SUB( )
END
```

C program:

```
#include <stdio.h>
void sub( )
{
  printf ("%s\n", "*** sub ***");
}
```

The external name of Fortran calling procedure SUB is the binding label sub. Therefore, function name of C is sub.

Example 2: Calling Fortran procedure with the procedure language binding specifier from C program

C program:

```
void sub( );
int c_( )
{
  sub( );
  return 0;
}
```

Fortran program:

```
SUBROUTINE SUB() BIND(C)
PRINT *, '*** SUB ***'
END
```

The external name of Fortran subroutine SUB is the binding label sub. Therefore, subroutine SUB is able to interoperate.

Example 3: Calling C function from Fortran program without the procedure language binding specifier

Fortran program:

```
EXTERNAL SUB
CALL SUB( )
END
```

C program:

```
#include <stdio.h>
void sub_( )
{
printf ("%s\n", "*** sub ***");
}
```

In Fortran program, calling SUB are processed "sub_", so C function name must be "sub_".

Example 4: Calling Fortran procedure without the procedure language binding specifier from C program

C program:

```
void sub( );
int c_( )
{
sub( );
return 0;
}
```

Fortran program:

```
SUBROUTINE SUB( )
CHANGEENTRY :: SUB
PRINT *, '*** SUB ***'
END
```

In Fortran program, specify procedure name in CHANGEENTRY statement and specify compiler option -ml=cdecl to confirm processing procedure names.

11.5.1 Passing Control First to a C Program

If the program to which control is first passed is in C, the function name must be MAIN__. Do not use main. However, if the compiler option -mlcmain=main is specified, main can be used for the function name.

Example 1: Passing control first to a C program:

C program: a.c

```
void sub_( );
int MAIN__( )
{
sub_( );
return 0;
}
```

Fortran program: b.f95

```
SUBROUTINE SUB( )
PRINT *, 'SUB'
END
```

Compile, link, and execute:

```

$ fccpx -c a.c
$ frtpx a.o b.f95
$ ./a.out
SUB
$

```

Example 2: Passing control first to a C program (function name is main)

C program: a.c

```

void sub_( );
int main( )
{
sub_( );
return 0;
}

```

Fortran program: b.f95

```

SUBROUTINE SUB( )
PRINT *, 'SUB'
END

```

Compile, link, and execute:

```

$ fccpx -c a.c
$ frtpx -mlcmain=main a.o b.f95
$ ./a.out
SUB
$

```

11.5.2 Calling Standard C Libraries

When a standard C library is called from the Fortran program, there are two methods.

- The method of the specification of the procedure language binding specifier
- The method of no specification of the procedure language binding specifier

11.5.2.1 Method of Specification of Procedure Language Binding Specifier

A standard library of C can be called from the Fortran program.

Example: Calling a standard C library with the procedure language binding specifier.

```

USE, INTRINSIC::ISO_C_BINDING, ONLY:C_NULL_CHAR
INTERFACE
  FUNCTION STRCMP(STR1,STR2) BIND(C)
    USE, INTRINSIC::ISO_C_BINDING, ONLY:C_INT,C_CHAR
    INTEGER(C_INT)::STRCMP
    CHARACTER(KIND=C_CHAR), DIMENSION(*)::STR1,STR2
  END FUNCTION
END INTERFACE
CHARACTER(LEN=5)::ST1,ST2
ST1='ABCD'//C_NULL_CHAR
ST2='ABCD'//C_NULL_CHAR
IF (STRCMP(ST1,ST2)==0)PRINT *, "same string"
END

```

In this example, the actual argument of the character type scalar corresponds to the dummy argument of the character type array.

Compile and execute:

```

$ frtpx a.f95
$ ./a.out

```

```
same string
$
```

11.5.2.2 Method of No Specification of Procedure Language Binding Specifier

To calling standard C libraries from Fortran program directory, you must do following:

- Specify the standard C library name in CHANGEENTRY statement
- Specify compiler option -ml=cdecl

See Section "[11.3 Rules for Processing Fortran Procedure Names](#)", and be careful to passing argument. See Section "[11.7 Passing and Getting Arguments](#)" for details.

Example: Calling a standard C library without the procedure language binding specifier.

Fortran program: a.f95

```
INTEGER :: STRCMP
CHANGEENTRY :: STRCMP
CHARACTER(LEN=5) :: ST1,ST2
ST1 = 'ABCD\0' ; ST2 = 'ABCD\0' ! Compiler option -Ae need to be specified.
IF (STRCMP(ST1,ST2)==0) PRINT *, " same string "
END
```

Compile and execute:

```
$ frtpx a.f95 -ml=cdecl -Ae
$ ./a.out
  same string
$
```

11.6 Passing and Receiving Function Values

This section describes passing and receiving function values between Fortran and C program without the procedure language binding specifier. See "[Table 11.2 Types of C program and, types and kind type parameters of Fortran correspond](#)" for the type that can be shared when the procedure language binding is specified.

11.6.1 Passing Fortran Function Values to C without Procedure Language Binding Specifier

The following table lists the methods for calling Fortran procedures from C programs for each type of function result that a Fortran program returns.

Fortran function value	How C accepts values	Example of values returned by C
INTEGER(1)	signed char	res = sub_();
INTEGER(2)	short int	
INTEGER(4)	int	
INTEGER(8)	long long int	
REAL(2)	_Float16	
REAL(4)	float	
REAL(8)	double	
REAL(16)	long double	
LOGICAL(1)	unsigned char	
LOGICAL(2)	short int	
LOGICAL(4)	long int	

Fortran function value	How C accepts values	Example of values returned by C
LOGICAL(8)	long long int	
COMPLEX(2)	structure	
COMPLEX(4)		
COMPLEX(8)		
COMPLEX(16)		
CHARACTER	void	sub_(&res, len); (*1)
Derived type	structure	res = sub_();

sub_: Called Fortran procedure name

res: Area for the function value returned. See Section "11.4 Correspondence of Type with C" about the corresponding type

len: Character length of the function type. Then, Fortran receives character length as a value of eight byte integer type.

*1) The area of the return value of the procedure without the procedure language binding specifier is used the area of the actual argument. In this case, the address of the return value is put in the first argument and the length of the return value is put in the second argument. And the actual argument is put after the second arguments.

Example 1: Acceptance of an integer function value:

C program:

```
#include <stdio.h>
int ifun_();
int main( )
{
    int i;
    i = ifun_();
    printf("%d\n", i);
    return 0;
}
```

Fortran program:

```
FUNCTION IFUN()
INTEGER IFUN
IFUN = 100
END
```

Example 2: Acceptance of a double-precision real function value:

C program:

```
#include <stdio.h>
double r8fun_() ;
int main( )
{
    double r8;
    r8 = r8fun_() ;
    printf("%f\n", r8);
    return 0;
}
```

Fortran program:

```
FUNCTION R8FUN()
REAL(8) R8FUN
R8FUN= 12.2D+0
END
```

Example 3: Acceptance of a character function value:

C program:

```
#include <stdio.h>
void cfun_(char *, long long int);
int main( )
{
    char cha [11];
    cha[10]=0x00;
    cfun_(cha,10);
    printf("%s\n",cha);
    return 0;
}
```

Fortran program:

```
FUNCTION CFUN()
CHARACTER(LEN=*) CFUN
CFUN = '1234567890'
END
```

To accept the value of character function CFUN, set the address of the return area to the first argument, and set the length of function to the second argument. Then, Fortran receives length of character function CFUN as a value of eight byte integer type.

Example 4: Acceptance of a derived type function value:

C program:

```
#include <stdio.h>
struct tag
{ int i;
  double fr8; };
struct tag dfun_( );
int main( )
{
    struct tag d;
    d = dfun_( );
    printf("%d %f\n",d.i, d.fr8);
    return 0;
}
```

Fortran program:

```
FUNCTION DFUN()
TYPE TAG
SEQUENCE
INTEGER(4) I
REAL(8) FR8
END TYPE
TYPE (TAG) ::DFUN
DFUN%I = 10
DFUN%FR8 = 1.2D+0
END
```

11.6.2 Receiving Function Values from C without Procedure Language Binding Specifier

The following table lists the methods for calling C functions from Fortran programs without the procedure language binding Specifier for each type of function result that a C program returns.

Specifier for each type of function result that a C program returns.

C function value	How Fortran accepts value	Example of values returned by Fortran
void	Not accepted	CALL SUB()
signed char	INTEGER(1)	RES = SUB()
short int	INTEGER(2)	
int	INTEGER(4) or LOGICAL(4)	
long long int	INTEGER(8)	
_Float16	REAL(2)	
float	REAL(4)	
double	REAL(8)	
long double	REAL(16)	
structure	Derived type	

SUB: Calling C function processed in sub_. See Section "[11.3 Rules for Processing Fortran Procedure Names](#)".

RES: Area for the function value returned.

Example 1: Acceptance of an integer function value:

Fortran program:

```
INTEGER(4) I,IFUN
I = IFUN()
PRINT *,I
END
```

C program:

```
int ifun_()
{
    return 100;
}
```

Example 2: Acceptance of a derived type function value:

Fortran program:

```
TYPE TAG
    INTEGER(4)
    REAL(8) FR8
END TYPE
TYPE (TAG) ::DFUN,D
D = DFUN()
PRINT *,D%I,D%FR8
END
```

C program:

```
struct tag
{int i;
 double fr8; };
struct tag dfun_()
{
    struct tag d ;
    d.i=100 ;
    d.fr8=1.2 ;
    return(d) ;
}
```


11.7 Passing and Getting Arguments

The passing and receiving arguments between procedures are the address in the default. If arguments are passed and received from Fortran program, see Section "11.2.1 Passing Arguments by Value".

If a character type argument is passed, the argument length as a value of eight byte integer type is also passed. The actual argument that shows the argument length exists only by the number of arguments of the character type be after all the specified arguments.

However, the argument length is not passed when the argument is the procedure with the procedure language binding specifier or VALUE attribute. The following sections explain how arguments pass data without the procedure language binding specifier.

11.7.1 Passing Arguments from Fortran with Procedure Language Binding Specifier to C

Because Fortran passes actual arguments by address, the address is accepted if dummy arguments are declared in C. Specify VAL() intrinsic function or %val() specifier to pass the actual argument by value.

See Section "11.2.1 Passing Arguments by Value" for details.

Example: Passing integer and real type arguments:

Fortran main program:

```
PROGRAM FORTRAN_MAIN
WRITE(*,*) '*** START ***'
J=10
K=20
L=30
CALL FUN1(J,K,L)
F=10.0
CALL FUN2(F)
WRITE(*,*) '*** FMAIN END ***'
END
```

C:

```
#include <stdio.h>
void fun1_(j,k,l)
int *j,*k,*l;
{
    printf("J:%d K:%d L:%d\n",*j,*k,*l);
}

void fun2_(f)
float *f;
{
    printf("F:%f\n",*f);
}
```

11.7.2 Passing Arguments from C to Fortran without Procedure Language Binding Specifier

Because Fortran accepts dummy arguments by address in the default, the address is passed if actual arguments appear in C. Specify VALUE attribute for dummy argument to get the actual argument by value. See Section "11.2.2 Getting Arguments by Value" for details.

Example 1: Passing integer type arguments:

C program (MAIN__):

```
#include <stdio.h>
int MAIN__()
{
    int fun_();
```

```

int i,j,k;
i=10;
j=20;
k=fun_(&i,&j);
printf(" k:%d\n",k);
return 0;
}

```

Fortran program:

```

INTEGER FUNCTION FUN(X,Y)
INTEGER(4) X,Y

WRITE(*,*) '**** FUNC1 **** '
FUN =X+Y
END

```

Example 2: Passing character type arguments:

C program (MAIN__):

```

#include <stdio.h>
int MAIN__( )
{
void fun_(char *, long long int, char *, long long int);
char res[3],ch;
ch = 'o';
res[2] = 0;
fun_(res,2,&ch,1);
printf(" res:%s\n",res);
return 0;
}

```

Fortran:

```

CHARACTER(LEN=*) FUNCTION FUN(CH)
CHARACTER(LEN=*) CH
FUN = CH// 'k'
END

```

Note:

Character type function FUN has one character type argument. If this function is called from a C program, four arguments are passed. The first argument is the address of the area for the function value. The second argument is the length of the function value. The third argument is the address of the argument ch. The fourth argument is the length of ch. Then, Fortran receives result length and argument length of character type function FUN as a value of eight byte integer type.

11.8 Passing Using External Variables

- The method of the specification of the BIND attribute.
- The method of no specification of the BIND attribute.

11.8.1 Passing External Variables with BIND Attribute

When COMMON block or module variable with BIND attribute, this COMMON block or variable can associate to the external variable. The external name is the binding label of the variable or COMMON block.

Example 1: Associating the external variable with BIND attribute

```

MODULE MOD
INTEGER, BIND(C)::INT_VARIABLE
! Associating the external variable int_variable in C program
END

```

Example 2: Passing the named COMMON block with BIND attribute

Fortran program:

```
INTERFACE
  SUBROUTINE SUB() BIND(C)
  END SUBROUTINE SUB
END INTERFACE
INTEGER I,J
COMMON /EXT/I,J
BIND(C)::/EXT/
I = 1
J = 1
CALL SUB( )
END
```

C program:

```
#include <stdio.h>
struct tag { int i,j; } ext;
void sub( )
{
  printf("i=%d j=%d\n",ext.i,ext.j);
}
```

Passed COMMON block EXT with BIND attribute.

11.8.2 Passing External Variables without BIND Attribute

Without BIND attribute, the data in Fortran COMMON block can be passed as the external structure type variable in C program.

A COMMON block name of Fortran and an external variable of C program should be the following correspondences.

Specification of COMMON block	External variable name of structure of C
Named COMMON block	COMMON block name_
Unnamed COMMON block	_BLNK__

The COMMON block name is uniformly lowercase if compiler option -AU is not specified. If compiler option -AU is specified, the COMMON block name is same spell in source program.

Example 1: Named common block without BIND attribute

Fortran program:

```
COMMON /EXT/I,J
I=1
J=2
CALL SUB()
END
```

C program:

```
#include <stdio.h>
struct tag {int i,j;} ext_;
void sub_( )
{
  printf("i=%d j=%d\n",ext_.i,ext_.j);
}
```

Note:

To reference common block EXT, change the name to lowercase and add an underscore to the name.

Example 2: Blank common block without BIND attribute

Fortran program:

```
COMMON //X,Y
X=1.0
Y=2.0
CALL SUB()
END
```

C program:

```
#include <stdio.h>
struct tab {float x,y;}_BLNK__ ;
void sub_()
{
printf("x=%f y=%f\n",
_BLNK__.x, _BLNK__.y) ;
}
```

11.9 Passing Using Files

If data is passed by file, the file must be closed. To close files in Fortran and C programs, use the following instructions:

Fortran	C
CLOSE statement	close (2)

11.10 Array Storage Sequence

Fortran and C differ in the order in which they store elements in arrays of two or more dimensions.

Therefore, be careful when passing arrays between Fortran and C.

The following shows the different storage sequences in Fortran and C arrays.

(Array in Fortran)	(Array in C)
INTEGER(4) K(2,3)	int k[2][3];
Storage sequence in Fortran	Storage sequence in C
K(1,1)	k[0][0]
K(2,1)	k[0][1]
K(1,2)	k[0][2]
K(2,2)	k[1][0]
K(1,3)	k[1][1]
K(2,3)	k[1][2]

11.11 Passing a Character String to a C Program

The null character "\0" normally is not appended to the constants of Fortran character type. If a function such as strlen (standard C library function) is called, you must do one of the followings:

- Add the intrinsic module C_NULL_CHAR (null character) of ISO_C_BINDING to the end of character strings.

- Append a null character (CHAR(0)) to the end of constants. Then, compiler option -Ae must be specified.
- Specify the compiler option -Az when the constants of character type is specified in the argument.

Example 1: Passing and receiving the character string using BIND attribute and C_NULL_CHAR

Fortran program:

```
USE, INTRINSIC::ISO_C_BINDING, ONLY:C_NULL_CHAR
INTERFACE
  SUBROUTINE SUBA(STR) BIND(C)
    USE, INTRINSIC::ISO_C_BINDING, ONLY:C_NULL_CHAR,C_CHAR
    CHARACTER(KIND=C_CHAR)::STR(*)
  END SUBROUTINE
END INTERFACE
CALL SUBA('12345'//C_NULL_CHAR) ! Add null character to the end of the character strings
END
```

C program:

```
#include <stdio.h>
#include <string.h>
void suba(char *str)
{
  int i,len;
  printf("len=%d,[%s]\n",len=strlen(str),str);
  for (i=0; i < len; i++)
  {
    printf("0x%02x ",*str++);
  }
  putchar('\n');
}
```

Example 2: Passing and receiving the character strings added NULL character not using BIND attribute.

Fortran program:

```
! CALL SUBA('12345')
CALL SUBA('12345\0') ! Add \0 to the end of the character strings.
END
```

C program:

```
#include <stdio.h>
#include <string.h>
void suba_(char *str)
{
  int i,len;
  printf("len=%d,[%s]\n",len=strlen(str),str);
  for (i=0; i < len; i++)
  {
    printf("0x%02x ",*str++);
  }
  putchar('\n');
}
```

Note:

Compiler option -Ae must be specified for Fortran program.

11.12 Return Value of Executable Program

When the Fortran program is linked with C/C++ language program, the return value of executable program is decided as follows.

- In the following cases, the return value of Fortran is set the return value of executable program. See Section "[3.6 Execution Command Return Values](#)".
 - The program to which control is first passed is Fortran program.
- In the following cases, the return value of C/C++ is set the return value of executable program.
 - The program to which control is first passed is C/C++ program.

11.13 Notes of Compiler Option -Az

If the compiler option -Az is to add \0(Null character) to the end of character strings. However, a part of execution is not guaranteed by the program described by the specification before Fortran 77.

Example: The compiler option -Az

```
CALL SUB('12')
END
SUBROUTINE SUB(I)
INTEGER(4) I
IF (I.NE.'12') PRINT *, 'ok'
END
```

Please change the CALL statement of the above-mentioned program as follows.

```
CALL SUB('12  ')
```

11.14 Restrictions on Linking with C Language Programs

The following restrictions and notes apply to linkage with C programs. And, see Section "[9.19 Attentions for Mixed Language Programming](#)" for notes combined with the optimization functions.

- With BIND attribute, the language entities of Fortran and C should be able to be interoperable.
- The binding label with the BIND attribute and the other processing Fortran procedure names must not be same name.
- When C program is called as a function reference, the function must correspond as shown in "[11.6 Passing and Receiving Function Values](#)".
- When a Fortran program is called from a C program, the Fortran program must not be a main program.
- If the main program is a C program, the name must be "MAIN__". However, if the compiler option -mlcmain=main is specified, main can be used for the function name.
- When using an interrupt-associated Fortran function, do not use signal(2) or clock(3C).
- When signal(2) or clock(3C) function is used in C language programs, the -i option must be specified at execution time to suppress interrupt processing by Fortran objects and followings are restrictions.
 - Specifying time limit to continue execution by -t option at execution time
 - Service function "ALARM"
 - Service function "SIGNAL"
- Always close nonstandard files before calling the other language.
- A CALL statement with an alternate return is valid only when calling Fortran subroutines.
- A subroutine called from a C program must not contain a RETURN statement with an integer expression or an aster of dummy argument. Change such a subroutine to an integer function.
- Fortran procedure must not invoke setjmp or longjmp in C program.
- If C program invokes setjmp or longjmp, that procedure must not cause any Fortran procedure to be invoked.

- When a Fortran procedure is called as a signal processing procedure, the signal processing procedure can refer only the variable that have VOLATILE attribute.
- Fortran procedures are not allowed to call C language function with a variable number of arguments.
- If Fortran variable associates with C external variable (see Section "11.8 Passing Using External Variables") and the C external variable has an initial value, the boundary of Fortran variable is changed to the boundary of the C external variable then warning message is output by linker.
- The variable of the language entity C_PTR type of the intrinsic module ISO_C_BINDING cannot debug using the debugger.
- If C++ program invokes exception processing (try, catch or throw), that procedure must not cause any Fortran procedure to be invoked.
- See "Notes on Linking with Different Languages and Trad/Clang Modes" in the "C++ User's Guide" for how to link C++ program.

11.15 Linking with C++ Language Programs

If link with the C++ programs and the Fortran programs, specify the `--linkstl=stl_kind` option at linking. Specify the same STL that used when compiling the C++ program. It must be specified with the `--linkstl=stl_kind` option.

Fortran program: b.f95

```
SUBROUTINE SUB( )
PRINT *, 'SUB'
END SUBROUTINE
```

C++ program: a.cc

```
#include <iostream>
extern "C" {
    void sub_( );
}
int MAIN__( )
{
    sub_( );
    std::cout << " C++ Program" << std::endl;
    return 0;
}
```

Example 1: STL is libfjc++

The STL used by C++ compiler in Trad Mode.

Compile, link, and execute:

```
$ FCCpx -c a.cc
$ frtpx a.o b.f95 --linkstl=libfjc++
$ ./a.out
SUB
C++ Program
$
```

Example 2: STL is libc++

The STL used by C++ compiler in Clang Mode when specifying the `-stdlib=libc++` option.

Compile, link, and execute:

```
$ FCCpx -Nclang -stdlib=libc++ -c a.cc
$ frtpx a.o b.f95 --linkstl=libc++
$ ./a.out
SUB
C++ Program
$
```

Example 3: STL is libstdc++

The STL used by C++ compiler in Clang Mode by default or used when `-stdlib=libstdc++` option is specified.

Compile, link, and execute:

```
$ FCCpx -Nclang -c a.cc
$ frtpx a.o b.f95 --linkstl=libstdc++
$ ./a.out
SUB
C++ Program
$
```


Chapter 12 Multiprocessing

This chapter describes how to perform parallel processing of a Fortran program using LLVM OpenMP Library.

When using Fujitsu OpenMP Library, see "[Appendix J Fujitsu OpenMP Library](#)".

Note

If the following condition is met, this system uses LLVM OpenMP Library.

- `-Nlibompb` option is effective at linking

Information

The table below shows a connectable combination of library for multiprocessing (LLVM OpenMP Library and Fujitsu OpenMP Library) and connectable object file.

See "C User's Guide" or "C++ User's Guide" about Trad Mode and Clang Mode.

	Fortran object file	C/C++ object file	
		Trad Mode (<code>-Nnoclang</code> option)	Clang Mode (<code>-Nclang</code> option)
LLVM OpenMP Library (<code>-Nlibomp</code> option)	Available	Available	Available
Fujitsu OpenMP Library (<code>-Nfjompplib</code> option)	Available	Available	Not available

12.1 Overview of Multiprocessing

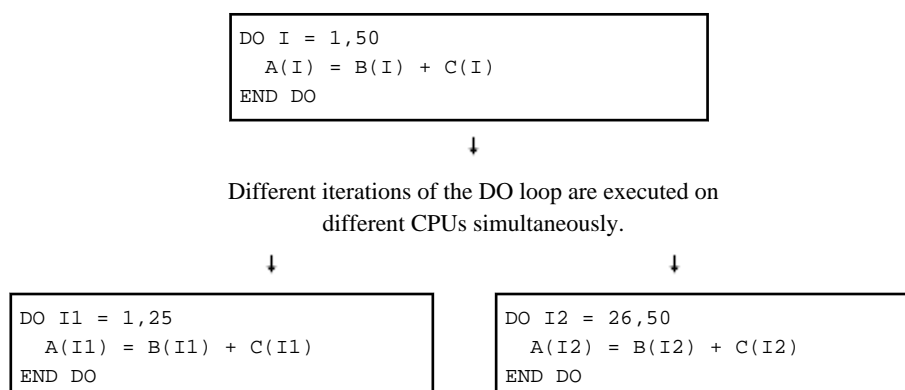
This section outlines the features of parallel processing and parallelization for Fortran in this system.

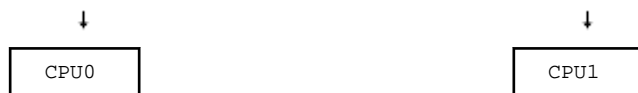
12.1.1 Parallel Processing

Although parallel processing has a broad meaning, the parallel processing mentioned in this section means to perform a single program simultaneously using multiple CPUs working independently. In this section, parallel processing does not mean "multi job", which means running multiple programs at the same time.

The following figure illustrates parallel processing using multiple CPUs simultaneously.

Figure 12.1 Multiprocessing

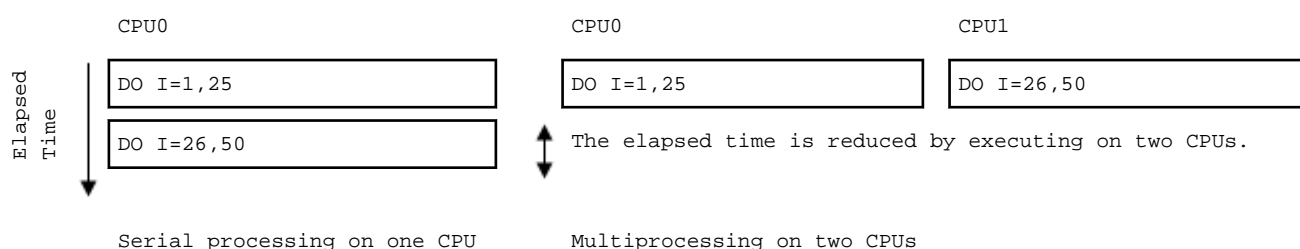




12.1.2 Effect of Multiprocessing

The aim of parallel processing is to reduce the elapsed execution time of a program using multiple CPUs at the same time. For example, as shown in "Figure 12.1 Multiprocessing", ideally the elapsed time of the program is halved by splitting one DO loop into two and performing them simultaneously ("Figure 12.2 Reducing elapsed time by using multiprocessing").

Figure 12.2 Reducing elapsed time by using multiprocessing



Although the elapsed time of a program can be reduced using parallel processing, the CPU time of the program may not be reduced. This is because, even though a number of CPUs are used in parallel processing, the total CPU time used may be the same as the CPU time used in serial processing, or may be increased because of the overhead to perform parallel processing.

However, in general, the performance of a program is evaluated by the elapsed time rather than the CPU time. If a program first executed serially is then executed in parallel, the total CPU time may be longer because of the overhead required for parallel processing.

12.1.3 Requirements for Effective Multiprocessing

A computer environment that can use multiple CPUs at one time is required in order for parallel processing to reduce elapsed time. Although programs generated by this compiler can run on hardware with a single CPU, if this is the case, a reduced elapsed time cannot be expected. Even if the hardware has multiple CPUs, a reduced elapsed time may not be expected when the CPU time is consumed by other jobs. This is because the opportunity to assign multiple CPUs at the same time to a program performing parallel processing may decrease.

In other words, for maximum benefit, parallel processing must be executed in an environment with multiple CPUs and sufficient capacity to assign multiple CPUs to the program.

Further, to minimize the relative rate of overhead due to parallel processing, the program should have either a large number of loop iterations or a large number of instructions within a loop.

12.1.4 Features of Parallel Processing in This Compiler

This compiler provides automatic parallelization and parallelization based on the OpenMP specifications.

The automatic parallelization is a feature of the compiler to parallelize the program automatically. Automatic parallelization is applied only for statements in DO loops and array operations for which the compiler has recognized that parallelization can be applied, but it does not require additional development.

This compiler also supports optimization control lines, which promote automatic parallelization. Information received from optimization control lines can be used to generate efficient object code for automatic parallelization.

To apply automatic parallelization, it is not necessary to update source code that has been designed for serial processing. For this reason, the source code is easily reusable in another system, as long as it conforms to the standard FORTRAN requirements. See Section "12.2 Automatic Parallelization" for information on automatic parallelization.

See Section "12.3 Parallelization by OpenMP Specifications" for information on parallelization in the OpenMP specification.

12.2 Automatic Parallelization

This section explains the automatic parallelization provided by this compiler.

12.2.1 Compilation

To activate automatic parallelization, specify the following compiler options:

```
-Kparallel [-Nlibomp]
```

12.2.1.1 Compiler Option for Automatic Parallelization

The automatic parallelization options are described below. See Section "2.2 Compiler Options" for more information on each option.

```
-Kparallel[ ,reduction,instance=N,array_private,parallel_iteration=N,dynamic_iteration,ocl,  
optmsg=2,independent=pgm_nm,loop_part_parallel,region_extension ] [-Nlibomp]
```

12.2.2 Execution Process

When running a program compiled with automatic parallelization, the number of threads can be specified using the environment variable OMP_NUM_THREADS. The stack area size for each thread can be specified using the environment variable OMP_STACKSIZE. It is also possible to specify a synchronization scenario using the environment variable OMP_WAIT_POLICY.

The other procedures are the same as those used in serial processing.

12.2.2.1 Number of Threads

The priority of the factors that determine the number of threads executed concurrently is as follows.

1. The value specified in the environment variable OMP_NUM_THREADS
2. The number of CPUs that can be used in the system.

The following environment variable for OpenMP specifications can be used in automatic parallelization. See OpenMP specifications for the detail.

Environment variable OMP_NUM_THREADS

Sets the number of threads to use during execution.

12.2.2.2 Stack Size on Execution

The stack area size for each thread can be specified using the following environment variable.

Environment variable OMP_STACKSIZE

The stack area size for each thread can be specified using the environment variable OMP_STACKSIZE in byte, Kbytes, Mbytes, Gbytes, or Tbytes. The default size is 8M bytes.

12.2.2.3 Synchronization Process

The synchronization process can be controlled with the environment variable OMP_WAIT_POLICY.

Environment variable OMP_WAIT_POLICY

ACTIVE	Uses spin wait until all threads are synchronized.
PASSIVE	Uses no spin wait but suspending wait.

The default is "PASSIVE".

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

12.2.2.4 Notes when Using Service Function and Intrinsic Subroutine

- ALARM Service Function

It may not work correctly. It is only for a program using serial processing.

- KILL and SIGNAL Service Functions

It may cause a deadlock. It is only for a program using serial processing.

- FORK Service Function

It may not work correctly. It is only for a program using serial processing.

- SYSTEM, SH and CHMOD Service Functions and EXECUTE_COMMAND_LINE Intrinsic Subroutine

Memory usage doubles and performance may deteriorate. Use only with programs using serial processing.

12.2.2.5 CPU Binding for Thread

Environmental variable `GOMP_CPU_AFFINITY` or `OMP_PROC_BIND` can control the CPU binding of threads. `GOMP_CPU_AFFINITY` takes precedence over `OMP_PROC_BIND`.

Environment variable `GOMP_CPU_AFFINITY`

Threads are bound to CPUs in order of the specified cpuid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid list shall be separated by comma (',') or space(' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. ($0 \leq \text{cpuid1} < \text{CPU_SETSIZE}$)

cpuid2: cpuid of the end of the range. ($0 \leq \text{cpuid2} < \text{CPU_SETSIZE}$)

inc: increment ($1 \leq \text{inc} < \text{CPU_SETSIZE}$)

In addition, it is necessary to be the following.

```
cpuid1 <= cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value. However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See `CPU_SET(3)` about details of `CPU_SETSIZE`.



Note

If cpuid is outside the CPU affinity of the process at the start of execution, an error will be output and the program will terminate. Correct the setting value.



Example

Examples of using environment variable `GOMP_CPU_AFFINITY`

- Example 1:

```
$ export GOMP_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 2:

```
$ export GOMP_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.
When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

- Example 3:

```
$ export GOMP_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.
When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 4:

```
$ export GOMP_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.
When the number of threads is six or more, it is repeatedly used f from the beginning of the list.



Environment variable OMP_PROC_BIND

The environment variable OMP_PROC_BIND can control the thread affinity.
Either TRUE, FALSE or comma separated list of MASTER, CLOSE, or SPREAD can be specified to this environment variable. The values of the list sets the thread affinity policy used by a PARALLEL region corresponding to the nest level.
This environment variable is set to CLOSE by default.
If this environment variable is set to FALSE, thread affinity is disabled, and the PROC_BIND clause on PARALLEL construct is ignored.
Otherwise, thread affinity is enabled and the initial thread is bound to first place in place list.
The MASTER thread affinity policy indicates that all threads are bound to same place as master thread.
The CLOSE thread affinity policy indicates that threads are bound to next to the place where master thread is bound.
The SPREAD thread affinity policy indicates that the threads are bound to the one of sub-partition places divided from master thread's place partition.
The effect of TRUE is same as SPREAD.
See environment variable OMP_PLACES for place.

Environment variable OMP_PLACES

The environment variable OMP_PLACES defines place list.
The explicit place list is defined by ordered set of comma separated non-negative numbers enclosed by braces. The numbers represents the smallest unit of CPU resource in the system.
The value of OMP_PLACES is either abstract name or an explicit place list. The value can be specified using following format.

```
{ abstract-name[(num-places)] | place-list }
```

abstract-name

The following abstract names can be set to this environment variable. The default value is CORES.

Abstract Name	Meaning
THREADS	Each place corresponds to a single hardware thread, which represents smallest unit of CPU resource in the system. The effect of THREADS is equivalent to CORES.

Abstract Name	Meaning
CORES	Each place corresponds to a single hardware core, which represents smallest unit of CPU resource in the system. The effect of CORES is equivalent to THREADS.
SOCKETS	Each place corresponds to a single socket, which represents NUMA node.

num-places

The length of the place list. Positive integer.

place-list

The explicit place list can be specified using following format.

```
{ place:length[:stride] | [!]place[,place-list] } (*1)
```

place

"{"*placeI*" }(*2)

place1

{ *cpuid:length[:stride] | [!]cpuid[,placeI] }*

length

The length of elements. Positive integer.

stride

The value of increment or decrement. Positive integer. Default value is 1.

cpuid

Smallest unit of CPU resource in the system.

*1) An exclusion operator "!" exclude the number or place immediately following the operator.

*2) "{" and "}" are braces.



Example

Example of environmental variable OMP_PLACES

- Example 1:

```
$ export OMP_PLACES="CORES"
```

This example defines place list of CORES abstract name.

- Example 2:

```
$ export OMP_PLACES="CORES(4)"
```

This example defines place list of CORES abstract name of 4 places.

- Example 3:

```
$ export OMP_PLACES="{12,13,14},{15,16,17},{18,19,20},{21,22,23}"
$ export OMP_PLACES="{12:3},{15:3},{18:3},{21:3}"
$ export OMP_PLACES="{12:3}:4:3"
```

All above examples define same place of 12 to 14, 15 to 17, 18 to 20, and 21 to 23.

12.2.3 Example of Compilation and Execution

Example 1:

```
$ frtpx -Kparallel,reduction,ocl -Nlibomp test1.f
$ ./a.out
```

Example 1 compiles the source program using the reduction optimization while enabling optimization control lines. The number of threads using at the runtime is defined by the execution environment. For details about the number of threads, see Section "[12.2.2.1 Number of Threads](#)".

Example 2:

```
$ frtpx -Kparallel -Koptmsg=2 -Nlibomp test2.f
Fortran diagnostic messages: program name (main)
  jwd5001p-i "test2.f", line 2: DO loop with DO variable 'i' is parallelized.
$ export OMP_NUM_THREADS=2
$ ./a.out
$ export OMP_NUM_THREADS=4
$ ./a.out
```

Example 2 compiles specifying the `-Koptmsg=2` option to confirm that automatic parallelization is performed. By specifying 2 for the environment variable `OMP_NUM_THREADS`, the program executes with two threads. The next step executes the program with four threads, by specifying 4 for `OMP_NUM_THREADS`.

12.2.4 Tuning of Parallelized Programs

The Instant Profiler can be used to improve the performance of a parallelized program. The Instant Profiler helps to identify the statements in a program that are expensive. Designing a program to perform expensive statements simultaneously can improve its performance. See the "Profiler User's Guide" for information on the Instant Profiler.

12.2.5 Feature Details on Automatic Parallelization

This section describes the automatic parallelization.

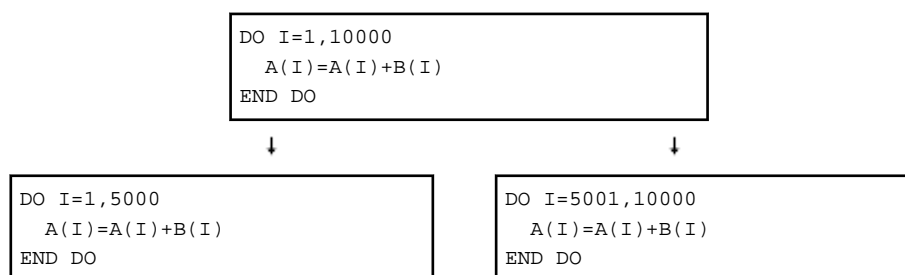
12.2.5.1 Targets for Automatic Parallelization

Automatic parallelization is performed for DO loops (including nested DO loops) and array statements (array operation, array assignments) in the Fortran source code.

12.2.5.2 What is Loop Slicing?

Automatic parallelization may slice a loop into several pieces. The elapsed execution time is reduced by executing the loop slices in parallel. The following figure illustrates the image of the loop slicing.

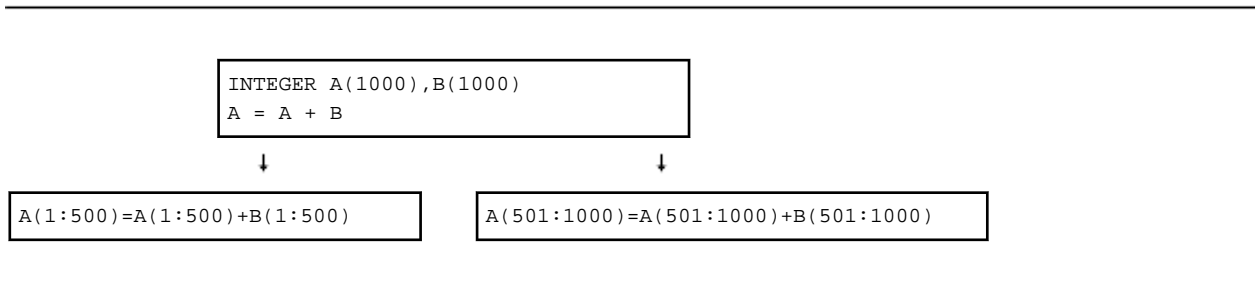
Figure 12.3 Loop slicing



12.2.5.3 Array Operations and Automatic Parallelization

In addition to DO loops, automatic parallelization is also performed on array statements (array operation, array assignments.) The following figure illustrates automatic parallelization on array operation.

Figure 12.4 Automatic parallelization of a statement with an array operation

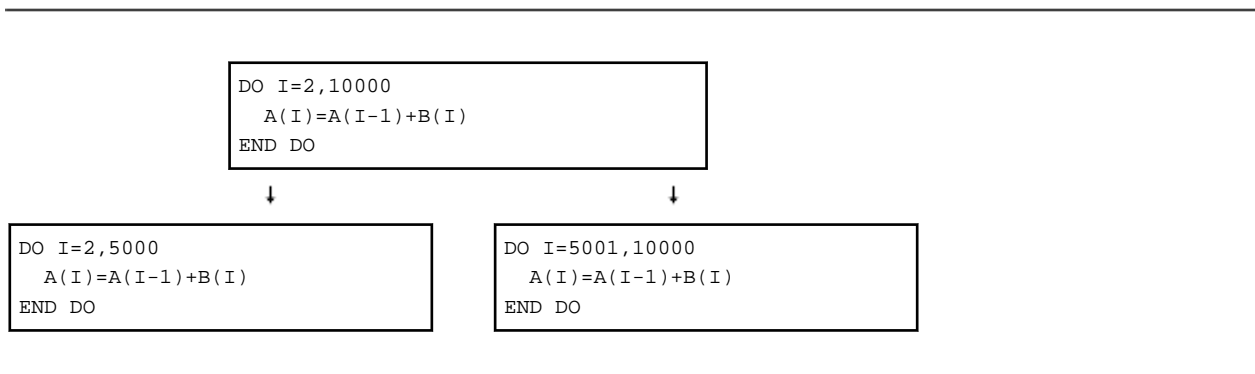


12.2.5.4 Automatic Loop Slicing by Compiler

The system parallelizes a loop if the order of data reference will be the same as with serial execution, and if the system is certain that the result of the parallel loop will be the same as if the loop were processed serially.

"[Figure 12.5 A loop that is not a candidate for loop slicing](#)" shows an example where automatic loop slicing is not applied. In this DO loop, the value of the array element A(5000), which is defined when the loop counter I is 5000, is referenced when I is 5001. If loop slicing is performed for this DO loop, however, the element A(5000) may be referenced by the other loop before its value is defined, causing an error.

Figure 12.5 A loop that is not a candidate for loop slicing

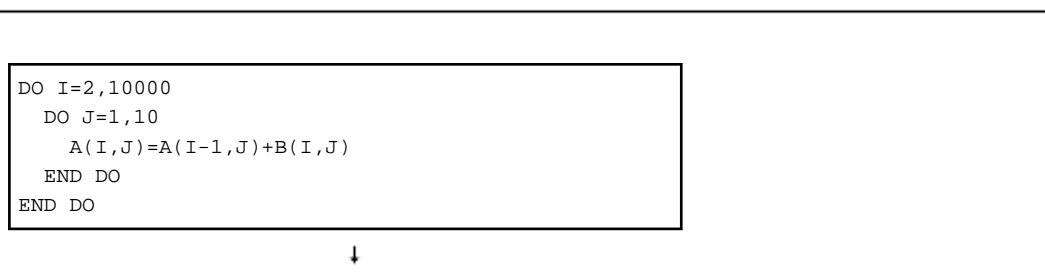


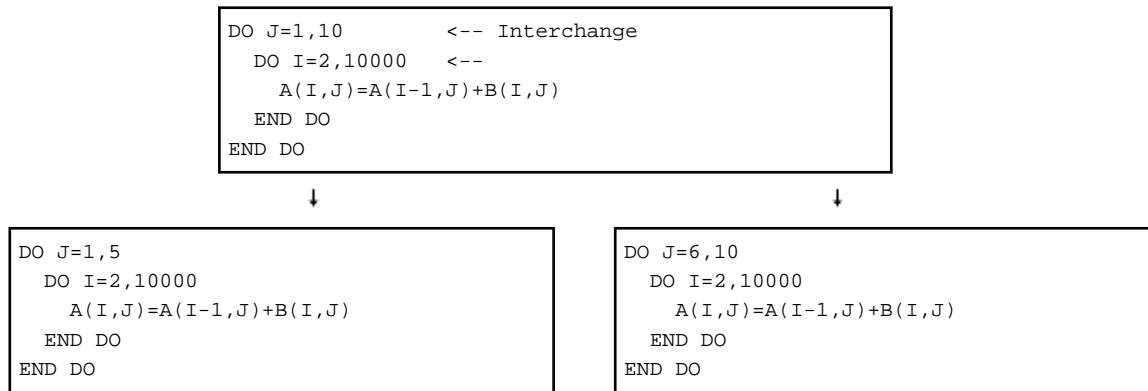
12.2.5.5 Loop Interchange and Automatic Loop Slicing

When nested loops are sliced, the system attempts to parallelize the outermost loop if it can. Therefore, the system selects the loop that can be sliced most cheaply and interchanges it with the outermost loop. The purpose of this is to reduce the overhead of multiprocessing and improve the execution performance.

"[Figure 12.6 Loop interchange in nested DO loops](#)" illustrates the loop slicing after performing loop interchange on a nested DO loop. Before performing parallelization for the DO loop variable "J", loop interchange is performed so that the parallelization overhead is reduced.

Figure 12.6 Loop interchange in nested DO loops





12.2.5.6 Loop Distribution and Automatic Loop Slicing

In the DO loop in "Figure 12.7 Separation of DO loop and loop slicing", loop slicing cannot be performed for array A because the definition or order of data references would be different from the order of data references in serial execution. However, loop slicing can be performed for array B, because the order of parallel data references is the same as the order of serial data references. In this case, the statement where array A is defined and the statement where array B is defined are separated into two loops, and loop slicing is performed on the loop where array B is defined.

The partial automatic parallelization by the distribution of loop is performed when the `-Kloop_part_parallel` option is effective and the compiler assumes that the distribution of loop is effective.

The following figure illustrates separating the statements into DO loops and performing loop slicing.

Figure 12.7 Separation of DO loop and loop slicing

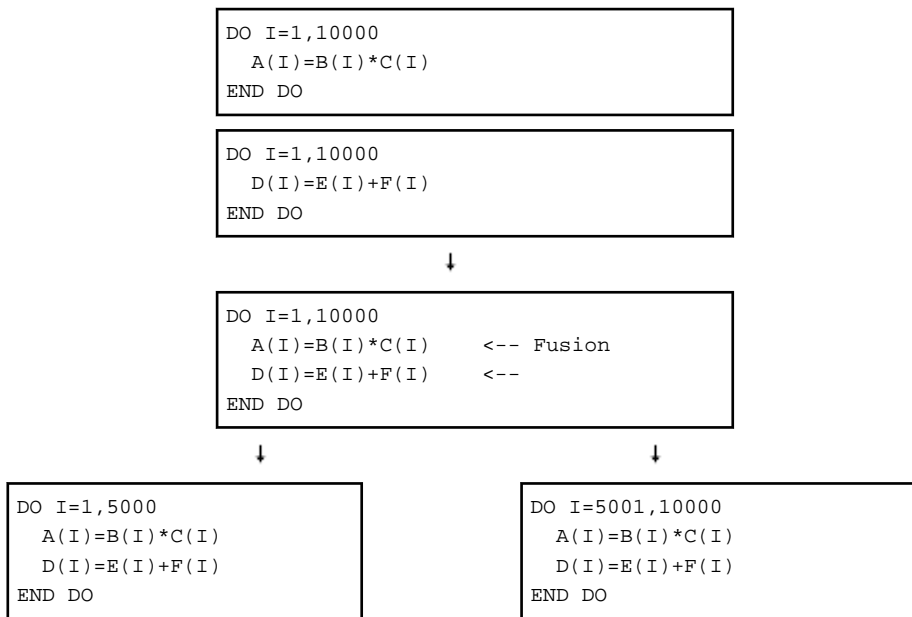


12.2.5.7 Loop Fusion and Automatic Loop Slicing

"Figure 12.8 Fusion of DO loops and loop slicing" contains two DO loops with the same iteration count. In this case, combining the loops into one can reduce the overhead of both parallelization and loop control.

The following figure illustrates the loop slicing after loop fusion.

Figure 12.8 Fusion of DO loops and loop slicing



12.2.5.8 Loop Reduction

If both the compiler options `-Kparallel` and `-Kreduction` are specified at the same time, the automatic parallelization loop reduction will be performed. This performs loop slicing by changing the order of operations such as addition and multiplication if they are interchangeable. However, this may affect precision.

Loop reduction is performed only when the DO loop includes the following operations:

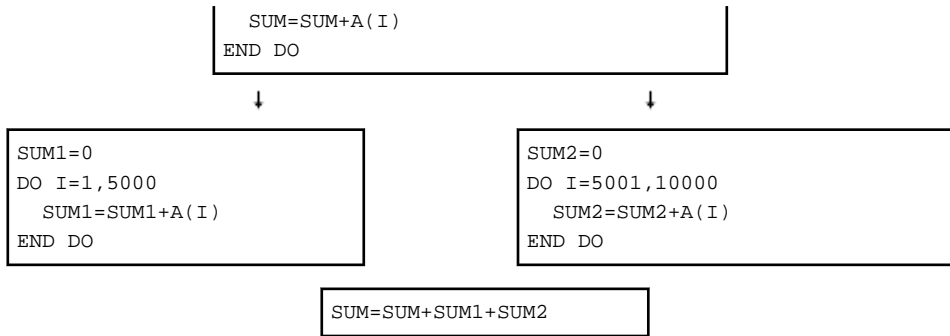
- SUM
 `S=S+A(I)`
- PRODUCT
 `P=P*A(I)`
- DOT PRODUCT
 `P=P+A(I)*B(I)`
- MIN
 `X=MIN(X,A(I))`
- MAX
 `Y=MAX(Y,A(I))`
- OR
 `N=N.OR. A(I)`
- AND
 `M=M.AND.A(I)`

The following figure shows an example of loop reduction and loop slicing.

Figure 12.9 Loop reduction

```

SUM=0
DO I=1,10000
    
```



12.2.5.9 Restrictions on Loop Slicing

Loop slicing does not support the following DO loops:

- If it is unlikely that parallelization will reduce elapsed time
- If operation types not supported by loop slicing are included
- If references to external procedures, internal procedures, or module procedure are included When the form of the DO-loop is complex
- Input-output statements or intrinsic subroutines are included
- Data definition or order of data references order may differ from serial execution
- If it is unlikely that parallelization will reduce elapsed time

1. It can be forecast that the elapsed time would not be reduced in the case of loop slicing.

When the iteration counts of a DO loop is small or there are few statements within the DO loop, the parallelization overhead introduced by loop slicing may deteriorate performance. The compiler will not perform loop slicing when performance improvement cannot be expected for the DO loop.

The following figure shows an example of a DO loop with a small iteration count and few statements.

Figure 12.10 Small loop iteration count and small number of operations

```

DO I=1,10
A(I)=A(I)+B(I)
END DO
! Not parallelized
! because of small loop iteration count and
! small number of operations

```

2. The loop contains operations on types, which are not supported in loop slicing.

Loop slicing does not support DO loops or inner DO loop with the following types of operations:

- Character type
- Derived type

3. References to external procedures, internal procedures, or module procedure are included.

Loop slicing does not support DO loops containing a reference to an external procedure, internal procedure, or module procedure. It is also unsupported when an inner DO loop includes the following operation types.

However, optimization control lines may be used to promote parallelization. See Section "12.2.6 Optimization Control Line" for details.

Inline expanded external procedure and internal procedures are subject to parallelization.

The following figure shows an example of a DO loop that contains a subroutine call.

Figure 12.11 DO loop contains a subroutine call

```
DO J=1,10          ! Not parallelized
  DO I=1,10000    ! because of the subroutine call
    A(I,J)=A(I,J)+B(I,J)
    CALL SUB(A)
  END DO
END DO
```

4. DO loop with complicated structure

The following DO loops cannot be sliced because they are too complicated:

- There is a branch from inside the loop to outside the DO loop.
- The structure of the DO loop is complicated.

The following figure shows an example of a DO loop that has a jump from inside to outside the loop.

Figure 12.12 Jumping out from inside of the DO loop

```
DO J=1,10          ! Not parallelized
  DO I=1,10000    ! because of jumping out of the loop
    A(I,J)=A(I,J)+B(I,J)
    IF(A(I,J).LT.0) GO TO 20
  END DO
END DO
.
.
.
20 CONTINUE
```

The following figure shows an example of DO loop with a complicated structure.

Figure 12.13 Complicated DO loop structure

```
DO J=1,10000      ! Not parallelized, because
  IF (N >0) THEN   ! of complicated DO loop structure
    ASSIGN 10 TO I
  ELSE IF (N <0) THEN
    ASSIGN 20 TO I
  ELSE
    ASSIGN 30 TO I
  END IF
  GO TO I, (10,20,30)
10  A(J)=A(J)+0
20  A(J)=A(J)+1
30  A(J)=A(J)+2
END DO
```

5. Input-output statements or intrinsic subroutines are included.

Loop slicing does not support DO loops containing an input-output statement or intrinsic subroutine.

Intrinsic functions, which are not supported by loop slicing, can be confirmed using diagnostic messages.

6. Data definition or reference order may differ from the serial process

As shown in "Figure 12.5 A loop that is not a candidate for loop slicing", loop slicing does not support DO loops that change data definition or the order of data references when converted from serial execution.

12.2.5.10 Displaying the State of Automatic Parallelization

To confirm whether automatic parallelization is performed, specify compiler options `-Kparallel` and `-Koptmsg=2` at the same time. The compilation diagnostic message will indicate the status.

When automatic parallelization is not performed, the message also outputs the reason.

The following shows an example of diagnosis message:

Example 1: Message when the program in "Figure 12.3 Loop slicing" is compiled

```
jwd5001p-i "test.f", line 2: DO loop with DO variable 'I' is parallelized.
```

Example 2: Message when the program in "Figure 12.5 A loop that is not a candidate for loop slicing" is compiled

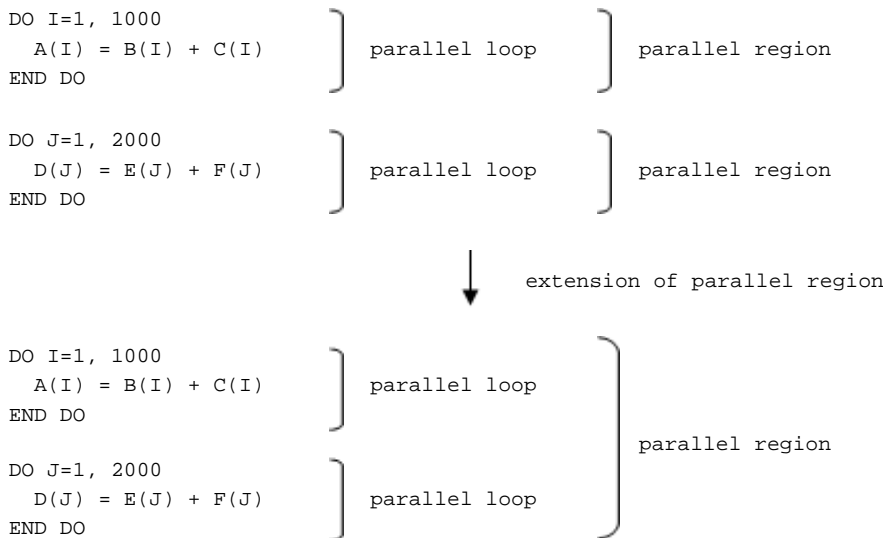
```
jwd5201p-i "test.f", line 2: DO loop is not parallelized: data dependency array 'A' may cause different results from serial execution for loop.
```

12.2.5.11 Extension of Parallel Region

The Parallel region is the range between the creation of multiple threads for automatic parallelization, and the deallocation of each thread when each thread ends. In general, parallelization generates one parallelized loop for one parallel region as shown in "Figure 12.14 Example of extension of parallel region". The cost of generating a loop for parallelization during runtime is considered the parallelization overhead.

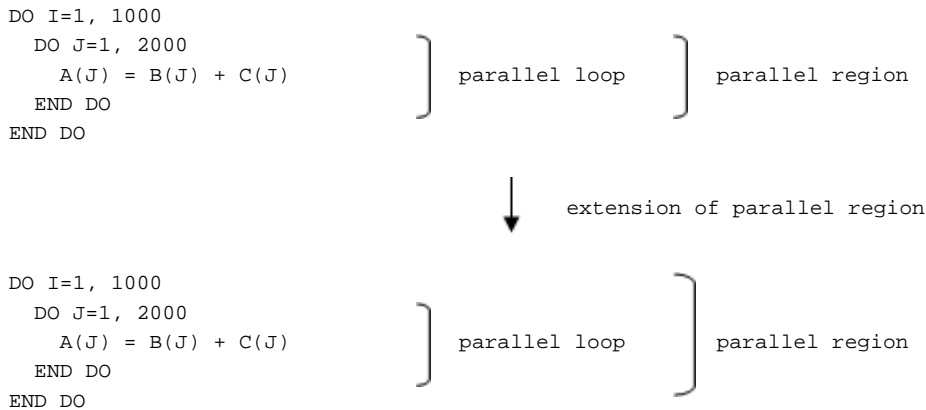
By expanding the parallel region for these two parallelization loops, two parallelized loops will be generated for one parallel region. As a result, parallelization overhead is reduced, because the parallel region is generated only once.

Figure 12.14 Example of extension of parallel region



The parallelization overhead can also be reduced for nested loops as shown in "Figure 12.15 Example of extension of parallel region (nested loop)" by expanding the parallel region. In this example, optimization reduces the number of times the parallel region is generated from 1000 down to 1.

Figure 12.15 Example of extension of parallel region (nested loop)



12.2.5.12 Block Distribution and Cyclic Distribution

The methods of loop slicing in the automatic parallelization are block distribution and cyclic distribution.

The block distribution is to allocate the block that the loop iteration count is distributed by the number of threads.

The cyclic distribution is to sequentially allocate the block that the loop iteration count is distributed by the arbitrary size. The default method of the automatic parallelization is block distribution. When PARALLEL_CYCLIC optimization control specifier is specified, the DO loop is cyclically distributed.

Figure 12.16 Block distribution and cyclic distribution

```

SUBROUTINE SUB(A,B)
INTEGER A(20,20),B(20,20)
DO J=1,20           ! parallelized
  DO I=J,20
    A(I,J) = B(I,J)
  ENDDO
ENDDO
END

```

When block distribution or cyclic distribution is applied to "Figure 12.16 Block distribution and cyclic distribution", the loop iteration count is allocated to each thread as follows:

- Block distribution

The block that the loop iteration count is divided by number of threads (2) is allocated in each thread as follows:

```

Thread 0:  {1,2,3,4,5,6,7,8,9,10}
Thread 1:  {11,12,13,14,15,16,17,18,19,20}

```

- Cyclic distribution (block size: 2)

The block that the loop iteration count is divided by two iteration (block size: 2) is sequentially allocated in each thread as follows:

```

Thread 0:  {1,2}, {5,6}, { 9,10}, {13,14}, {17,18}
Thread 1:  {3,4}, {7,8}, {11,12}, {15,16}, {19,20}

```

12.2.6 Optimization Control Line

This system provides OCLs (Optimization Control Lines) to control automatic parallelization.

To activate the OCL, specify the compiler options `-Kparallel` and `-Kocl` at the same time.

12.2.6.1 Notation Rules for Optimization Control Lines

The column 1 to 5 of the OCL must be `!OCL`. One or more optimization specifiers for automatic parallelization follows the `!OCL`. The notation rules are detailed in the following sections.

```
!OCL i [,i] . . . .
```

i: Optimization control specifier for automatic parallelization

12.2.6.2 Description Position of Optimization Control Line

The position where the optimization control line can be inserted depends on the type of optimization indicator.

The OCL for automatic parallelization must be specified at the position of total, loop, or array assignment statements. These are defined as follows:

- Total position
At the beginning of each program unit.
- Loop position
Immediately in front of DO statements. Other OCLs that can be specified for loop position or comment lines can be specified between the OCL and DO statement.
- Array assignment statement position
Immediately in front of array assignment statements.

Example:

```
!OCL SERIAL <----- total position
SUBROUTINE SUB(B,C,N)
INTEGER A(N),B(N),C(N)
DO I=1,N
  A(I)=B(I)+C(I)
END DO
PRINT *, FUN(A)
!OCL PARALLEL <----- loop position
DO I=1,N
  A(I)=B(I)*C(I)
END DO
PRINT *, FUN(A)
!OCL SERIAL <----- array assignment statement position
A = 0
PRINT *, FUN(A)
END SUBROUTINE
```

12.2.6.3 Automatic Parallelization and Optimization Control Specifiers

The Optimization Specifiers for automatic parallelization are ignored if they are specified for a DO loop for which loop slicing is not supported. See Section "[12.2.5.9 Restrictions on Loop Slicing](#)" for information about DO loops which are not supported by loop slicing.

12.2.6.4 Optimization Control Specifiers for Automatic Parallelization

The functionality of OCLs differs depending on the optimization specifier.

The optimization control specifiers for automatic parallelization specify useful information for the compiler to generate an efficient object module. They are listed in the table below.

Table 12.1 List of optimization control specifiers for automatic parallelization

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		Total	Loop	Statement	Array assignment statement
ARRAY_PRIVATE	Designates to be privatized arrays.	o	o	x	x
NOARRAY_PRIVATE	Designates canceling to be privatized arrays.	o	o	x	x
INDEPENDENT [(<i>e</i> , <i>e</i> ...)]	Asserts that the procedure (<i>e</i>) has no side effects, and that the DO loop that contains a reference to the procedure (<i>e</i>) can be sliced.	o	o	x	x
LOOP_PART_PARALLEL	Designates automatic parallelization that requires dividing loops.	o	o	x	o
LOOP_NOPART_PARALLEL	Suppresses automatic parallelization that requires dividing loops.	o	o	x	o
SERIAL	Suppresses automatic parallelization.	o	o	x	o
PARALLEL	Performs automatic parallelization.	o	o	x	o
PARALLEL_CYCLIC[<i>(n)</i>]	Designates cyclic distribution in block size (<i>n</i>).	x	o	x	o
PARALLEL_STRONG	Designates to parallelize the loop that the loop has small number of loop iterations or the loop has small the number of operations.	o	o	x	o
REDUCTION	Designates to parallel reduction operations.	o	o	x	o
NOREDUCTION	Designates canceling to parallel reduction operations.	o	o	x	o
TEMP [(<i>var</i> , <i>var</i> ...)]	Indicate variables (<i>var</i>) that are only used in the following DO loop and not outside of the loop	o	o	x	x
TEMP_PRIVATE(<i>var</i> , <i>var</i> ...)	Designates to assign variable (<i>var</i>) to local area in thread. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.	x	o	x	x
FIRST_PRIVATE(<i>var</i> , <i>var</i> ...)	Designates to assign variable (<i>var</i>) to local area in thread. And, designates to have a possibility of reference of the initial value of variable. As a result, the data dependency between threads is	x	o	x	x

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		Total	Loop	Statement	Array assignment statement
	solved, and the automatic parallelization is promoted.				
LAST_PRIVATE(<i>var</i> [, <i>var</i>]...)	Designates to assign variable (<i>var</i>) to local area in thread. And, the variable is directed to be referred after this DO loop. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.	x	o	x	x
<i>var1 op var2</i> or <i>var1 op cnst</i>	Designates relationship between variable and variable or between variable and constant.	o	o	x	o

The details of the optimization control specifiers are described below.

For clarity, the source code used in this section has; (p) for statements that are parallelized, (m) for statements that are partially parallelized, and (s) for statements that are not parallelized.

ARRAY_PRIVATE

The ARRAY_PRIVATE specifier directs the system to identify arrays in a loop that can be privatized, and promotes parallelization by privatizing them.

The ARRAY_PRIVATE indicator has the following format:

```
!OCL ARRAY_PRIVATE
```

The ARRAY_PRIVATE indicator can be entered at the loop position or total position.

Its effect differs depending on the position.

- At the loop position

It is effective for the corresponding DO loop.

- At the total position

It is effective for all DO loops in the corresponding program unit.

In "Figure 12.17 Usage of the ARRAY_PRIVATE specifier", using the ARRAY_PRIVATE specifier, the compiler detects that array A can be privatized and it performs parallelization by privatizing the arrays, even if the compiler option -Karray_private is not specified.

Figure 12.17 Usage of the ARRAY_PRIVATE specifier

```

SUBROUTINE SUB(N,M)
  INTEGER(KIND=4),DIMENSION(10000):: A
  INTEGER(KIND=4),DIMENSION(10000,100) :: B,C,D
  DATA A/10000*1/, C/1000000*2/, D/1000000*3/
  !OCL ARRAY_PRIVATE
  p DO I=1,N
  p   DO J=1,M
  p     A(J) = I + D(J,I)
  p     B(J,I) = A(J) + C(J,I)
  p   ENDDO
  p ENDDO
  PRINT*, A,B
  END

```

NOARRAY_PRIVATE

The NOARRAY_PRIVATE specifier directs the compiler not to privatize arrays even if they can be privatized.

The NOARRAY_PRIVATE indicator has the following format:

```
!OCL NOARRAY_PRIVATE
```

The NOARRAY_PRIVATE indicator can be entered at the loop position or total position.

Its effect differs depending on the position.

- At the loop position

It is effective for the corresponding DO loop.

- At the total position

It is effective for all the DO loops in the corresponding program units.

In "Figure 12.18 Usage of the NOARRAY_PRIVATE specifier", array A is not privatized due to NOARRAY_PRIVATE, although it can be privatized and the compiler option -Karray_private is set.

Figure 12.18 Usage of the NOARRAY_PRIVATE specifier

```
SUBROUTINE SUB(N,M)
  INTEGER(KIND=4),DIMENSION(10000):: A
  INTEGER(KIND=4),DIMENSION(10000,100) :: B,C,D
  DATA A/10000*1/, C/1000000*2/, D/1000000*3/
  !OCL NOARRAY_PRIVATE
p   DO I=1,N
p     DO J=1,M
p       A(J) = I + D(J,I)
p       B(J,I) = A(J) + C(J,I)
p     ENDDO
p   ENDDO
  PRINT*, A,B
  END
```

INDEPENDENT[(e[,e]...)]

The INDEPENDENT specifier directs the compiler that performing the procedure call within the DO loop in parallel execution will not change the operation result compared to serial execution. Doing this means that DO-loops with procedure references will be subject to loop slicing.

The INDEPENDENT indicator has the following format:

```
!OCL INDEPENDENT[ (e[,e]... ) ]
```

'e' are the procedure names that will not be affected by loop slicing. Wildcards can be specified in 'e'. When no procedure name is specified, this specifier is applied to all of the procedures within the target range. See Section "12.2.6.5 Wild Card Specification" for information on specifying wildcards.

The procedures specified in 'e' must be compiled with the compiler options -Kthreadsafe and -Kparallel.

The INDEPENDENT indicator can be entered at the loop position or total position.

Its effect differs depending on the position.

- At the loop position

It is effective for the corresponding DO loop.

- At the total position
It is effective for all the DO loops in the corresponding program units.

Figure 12.19 Example program without INDEPENDENT specifier

```

REAL FUN,A(10000)
DO I=1,10000
  J=I
  A(I)=FUN(J)
END DO
.
.
END
FUNCTION FUN(J)
INTEGER J
REAL FUN
FUN=SQRT(REAL(J**2+3*J+6))
RETURN
END

```

In "Figure 12.19 Example program without INDEPENDENT specifier", the procedure FUN is called within the DO loop and the compiler cannot identify whether loop slicing can be performed for this loop.

If performing loop slicing for the DO loop that calls the procedure FUN does not affect the result, using the INDEPENDENT specifier as in "Figure 12.20 Usage of INDEPENDENT specifier" will perform loop slicing for the DO loop.

Figure 12.20 Usage of INDEPENDENT specifier

```

REAL FUN,A(1000)
!OCL INDEPENDENT(FUN)
p DO I=1,1000
p   J=I
p   A(I)=FUN(J)
p END DO
.
.
END
RECURSIVE FUNCTION FUN(J)
INTEGER J
REAL FUN
FUN=SQRT(REAL(J**2+3*J+6))
RETURN
END

```

Note:

Ensure that INDEPENDENT is only specified for procedures for which loop slicing is supported. The following shows an example of a procedure type that is supported by loop slicing:

- Procedures that have dependency relationship with other procedures

LOOP_PART_PARALLEL

The LOOP_PART_PARALLEL specifier directs the system to perform automatic parallelization that requires dividing loops.

The LOOP_PART_PARALLEL indicator has the following format:

```
!OCL LOOP_PART_PARALLEL
```

The LOOP_PART_PARALLEL indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It is effective for the corresponding array assignment statement.
- At the loop position
It is effective for the corresponding DO loop.
- At the total position
It is effective for all DO loops in the corresponding program unit.

The LOOP_PART_PARALLEL specifier is in effect for the code shown in "[Figure 12.21 Usage of the LOOP_PART_PARALLEL specifier](#)", then the loop is divided and automatic parallelization is applied to the part of the divided loop without the -Kloop_part_parallel option.

Figure 12.21 Usage of the LOOP_PART_PARALLEL specifier

```
REAL(KIND=8) A(N),B(N),C(N),D(N)
!OCL LOOP_PART_PARALLEL
m DO I=2,N
p   A(I) = A(I)-B(I)+LOG(C(I)) ! automatic parallelization can be applied to
                               ! this statement
s   D(I) = D(I-1)+A(I)        ! automatic parallelization cannot be applied to
                               ! this statement
p ENDDO
```

LOOP_NOPART_PARALLEL

The LOOP_NOPART_PARALLEL specifier directs to suppress automatic parallelization that requires dividing loops.

The LOOP_NOPART_PARALLEL indicator has the following format:

```
!OCL LOOP_NOPART_PARALLEL
```

The LOOP_NOPART_PARALLEL indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It is effective for the corresponding array assignment statement.
- At the loop position
It is effective for the corresponding DO loop.
- At the total position
It is effective for all DO loops in the corresponding program unit.

The LOOP_NOPART_PARALLEL specifier is in effect for the code shown in "[Figure 12.22 Usage of the LOOP_NOPART_PARALLEL specifier](#)", then the loop is not divided and automatic parallelization is not performed to the loop.

Figure 12.22 Usage of the LOOP_NOPART_PARALLEL specifier

```
REAL(KIND=8) A(N),B(N),C(N),D(N)
!OCL LOOP_NOPART_PARALLEL
s DO I=2,N
p   A(I) = A(I)-B(I)+LOG(C(I))
s   D(I) = D(I-1)+A(I)
p ENDDO
```

SERIAL

The SERIAL indicator is used to prevent loop slicing of DO loops.

This specifier can be used for a DO loop that is faster when executed serially.

The SERIAL indicator has the following format:

```
!OCL SERIAL
```

The SERIAL indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It prevents loop slicing for the corresponding array assignment statement.
- At the loop position
It prevents loop slicing for the corresponding DO loop and its inner DO loops.
- At the total position
It prevents loop slicing for all the DO loops in the corresponding program units.

To suppress loop slicing for the sample program in "[Figure 12.23 Example program without SERIAL specifier](#)", position the SERIAL specifier as in "[Figure 12.24 Usage of the SERIAL specifier](#)".

Figure 12.23 Example program without SERIAL specifier

```
REAL, DIMENSION(100,100) :: A2, B2, C2, D2, E2, F2, G2, H2
p DO J=1,100
p   DO I=1,M
p     A2(I,J)=A2(I,J)+B2(I,J)
p     C2(I,J)=C2(I,J)+D2(I,J)
p     E2(I,J)=E2(I,J)+F2(I,J)
p     G2(I,J)=G2(I,J)+H2(I,J)
p   END DO
p END DO
END
```

Figure 12.24 Usage of the SERIAL specifier

```
REAL, DIMENSION(100,100) :: A2, B2, C2, D2, E2, F2, G2, H2
!OCL SERIAL
DO J=1,100
DO I=1,M
A2(I,J)=A2(I,J)+B2(I,J)
C2(I,J)=C2(I,J)+D2(I,J)
E2(I,J)=E2(I,J)+F2(I,J)
G2(I,J)=G2(I,J)+H2(I,J)
END DO
END DO
END
```

PARALLEL

The PARALLEL specifier is used to suppress the effect of the SERIAL specifier and perform loop slicing for specific DO loops.

PARALLEL indicator format:

```
!OCL PARALLEL
```

The PARALLEL indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It performs loop slicing for the corresponding array assignment statement.
- At the loop position
It performs loop slicing for the corresponding DO loop and its inner DO loop.
- At the total position
It pertains to all the DO loops in the corresponding program unit.

To perform loop slicing for loop 2 in "Figure 12.25 Example program without PARALLEL specifier", specify the SERIAL and PARALLEL specifiers as in "Figure 12.26 Example program with combination of PARALLEL and SERIAL specifiers" so that loop slicing is performed only for the DO loop presented in 2.

Figure 12.25 Example program without PARALLEL specifier

```

SUBROUTINE SUB(A1,A2,A3,B1,B2,B3,M1,M2,M3,N1,N2,N3)
INTEGER :: M1,M2,M3,N1,N2,N3
REAL,DIMENSION(M1,N1) :: A1,B1
REAL,DIMENSION(M2,N2) :: A2,B2
REAL,DIMENSION(M3,N3) :: A3,B3

P   DO J=1,N1                ! loop 1
P     DO I=1,100
P       A1(I,J) = A1(I,J) + B1(I,J)
P     ENDDO
P   ENDDO

P   DO J=1,N2                ! loop 2
P     DO I=1,100
P       A2(I,J) = A2(I,J) + B2(I,J)
P     ENDDO
P   ENDDO

P   DO J=1,N3                ! loop 3
P     DO I=1,100
P       A3(I,J) = A3(I,J) + B3(I,J)
P     ENDDO
P   ENDDO

RETURN
END
```

Figure 12.26 Example program with combination of PARALLEL and SERIAL specifiers

```

!OCL SERIAL
SUBROUTINE SUB(A1,A2,A3,B1,B2,B3,M1,M2,M3,N1,N2,N3)
INTEGER :: M1,M2,M3,N1,N2,N3
REAL,DIMENSION(M1,N1) :: A1,B1
REAL,DIMENSION(M2,N2) :: A2,B2
REAL,DIMENSION(M3,N3) :: A3,B3

DO J=1,N1                ! loop 1
  DO I=1,100
    A1(I,J) = A1(I,J) + B1(I,J)
  ENDDO
ENDDO
```

```

!OCL PARALLEL
p   DO J=1,N2                ! loop 2
p   DO I=1,100
p   A2(I,J) = A2(I,J) + B2(I,J)
p   ENDDO
p   ENDDO

DO J=1,N3                ! loop 3
DO I=1,100
A3(I,J) = A3(I,J) + B3(I,J)
ENDDO
ENDDO

RETURN
END

```

PARALLEL_CYCLIC(*n*)

The PARALLEL_CYCLIC specifier directs the system to perform cyclic distribution for a DO loop that automatic parallelization can be analyzed by compiler. When this specifier is applied, loop is parallelized without estimating the effects of the parallelization.

See Section "12.2.5.12 Block Distribution and Cyclic Distribution" for information on cyclic distribution.

The PARALLEL_CYCLIC indicator has the following format:

```
!OCL PARALLEL_CYCLIC( n )
```

n is the block size of cyclic distribution. *n* is number from 1 to 10000. The default is 1.

The PARALLEL_CYCLIC indicator can be entered at the array assignment statement position or loop position.

Its effect differs depending on the position.

- At the array assignment statement position
It is effective for the corresponding array assignment statement
- At the loop position
It is effective for the corresponding DO loop.

In "Figure 12.27 Usage of the PARALLEL_CYCLIC specifier", DO loop is cyclically distributed with block size of 2.

Figure 12.27 Usage of the PARALLEL_CYCLIC specifier

```

SUBROUTINE SUB(A,B,N)
INTEGER A(N,N),B(N,N)
!OCL PARALLEL_CYCLIC(2)
p   DO J=1,N
p   DO I=J,N
p   A(I,J) = B(I,J)
p   ENDDO
p   ENDDO
END

```

PARALLEL_STRONG

The PARALLEL_STRONG specifier directs to perform loop slicing for loops which have not been sliced because the iteration count or instruction numbers are too small.

The PARALLEL_STRONG indicator has the following format:

```
!OCL PARALLEL_STRONG
```

The PARALLEL_STRONG indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It pertains to the corresponding array assignment statement.
- At the loop position
It pertains to the corresponding DO loop and its inner DO loops.
- At the total position
It pertains to all the DO loops in the corresponding program unit.

The PARALLEL_STRONG specifier forces parallelization for the loop in "[Figure 12.28 Usage of PARALLEL_STRONG specifier](#)", although the loop is usually excluded from parallelization because the iteration count is considered too small to gain any benefit.

Figure 12.28 Usage of PARALLEL_STRONG specifier

```
REAL(KIND=4) :: A(10),B(10),C(10)
!OCL PARALLEL_STRONG
p DO J=1,10
p   A(J) = A(J) + B(J) + C(J)
p END DO
PRINT*, A
END
```

REDUCTION

The REDUCTION specifier directs the compiler to perform parallelization for a DO loop that includes reduction operations.

The REDUCTION indicator has the following format:

```
!OCL REDUCTION
```

The REDUCTION indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It pertains to the corresponding array assignment statement.
- At the loop position
It pertains to the corresponding DO loop and its inner DO loops.
- At the total position
It pertains to all the DO loops in the corresponding program unit.

As in "[Figure 12.29 Usage of the REDUCTION specifier](#)", the REDUCTION specifier forces parallelization for the DO loop that includes reduction operations, even if the compile option -Kreduction is not specified.

Figure 12.29 Usage of the REDUCTION specifier

```
REAL S,A(5000)
!OCL REDUCTION
p DO I=1,5000
p   S = S + A(I)
p END DO
END
```


NOREDUCTION

The NOREDUCTION specifier directs the compiler not to perform parallelization for a DO loop that includes reduction operations.

The NOREDUCTION indicator has the following format:

```
!OCL NOREDUCTION
```

The NOREDUCTION specifier may be specified at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It pertains to the corresponding array assignment statement.
- At the loop position
It pertains to the corresponding DO loop and its inner DO loops.
- At the total position
It pertains to all the DO loops in the corresponding program unit.

As illustrated in "[Figure 12.30 Usage of the NOREDUCTION specifier](#)", the NOREDUCTION specifier can suppress parallelization of the specified DO loop although it includes the reduction operation.

Figure 12.30 Usage of the NOREDUCTION specifier

```
REAL S,A(5000)
!OCL NOREDUCTION
S DO I=1,5000
S   S = S + A(I)
S END DO
END
```

TEMP[(var[,var]...)]

The TEMP specifier directs the compiler that the variable used in the DO loop is a temporary variable, which is only valid within the DO loop. This can improve the performance of the parallelized DO loop.

The TEMP indicator has the following format:

```
!OCL TEMP[(var[,var]...)]
```

'var' is a temporary variable name used in the DO-loop. Wildcards can be specified in 'var'. When no variable name is specified, this specifier is applied to all of the variables within the target range. See Section "[12.2.6.5 Wild Card Specification](#)" for information on specifying wildcards.

The TEMP indicator can be entered at the loop position or total position.

Its effect differs depending on the position.

- At the loop position
It directs that the specified variables, which are used in the corresponding DO loop, are temporary variables.
- At the total position
It directs that the specified variables, which are used in all DO loops in the corresponding program unit, are temporary variables.

In "[Figure 12.31 Example program without a TEMP specifier](#)", the compiler considers that "T" may be used in the SUB because "T" is declared in the common block, and the compiler performs loop slicing while maintaining the value of "T" although it is only used within the DO loop.

Figure 12.31 Example program without a TEMP specifier

```

COMMON T
REAL,DIMENSION(1000,50) :: A,B,C,D
.
.
.
p   DO J=1,50
p       DO I=1,1000
p           T=A(I,J)+B(I,J)
p           C(I,J)=T+D(I,J)
p       END DO
p   END DO
.
.
.
CALL SUB
END

```

If it is known that the value of "T", at the end of the DO loop will not be referenced by the subroutine SUB, use TEMP specifier with T as shown in "[Figure 12.32 Usage of TEMP specifier](#)". This will improve the performance because the instructions that maintain the value of T at the end of the loop are no longer needed.

Figure 12.32 Usage of TEMP specifier

```

COMMON T
REAL,DIMENSION(1000,50) :: A,B,C,D
.
.
.
!OCL TEMP(T)
p   DO J=1,50
p       DO I=1,1000
p           T=A(I,J)+B(I,J)
p           C(I,J)=T+D(I,J)
p       END DO
p   END DO
.
.
.
CALL SUB
END

```

Note:

If variables which are not used for temporary purposes are specified in the TEMP specifier, the compiler will perform loop slicing incorrectly.

TEMP_PRIVATE(*var*[,*var*]...)

In the target loop of automatic parallelization, the TEMP_PRIVATE specifier directs the system to treat a specified variable *var* as a local variable in each thread.

When the optimization message like the following is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```

jwd5208p-i "file", line n : DO loop is not parallelized: the assign-refer order of 'var' in a
parallel execution may differ from the assign-refer order in the serial execution.

```

The TEMP_PRIVATE indicator has the following format:

```
!OCL TEMP_PRIVATE(var[,var]...)
```

Wildcards can be specified in *var*. See Section "12.2.6.5 Wild Card Specification" for information on specifying wildcards.

The TEMP_PRIVATE indicator can be entered at the loop position. It is effective for the corresponding DO loop at the loop position.

The variable that can be specified for this specifier is as follows.

- The scalar variable of integer type, real type or logical type.
- The array variable that meets the following conditions:
 - Integer type, real type or logical type. And,
 - The size of array variable has been fixed when compilation.

When this specifier is applied, the following message is output.

```
jwd5013p-i "file", line n : !OCL TEMP_PRIVATE is applied to variable 'var'.
```

See Section "12.2.7.4 Notes on Using Optimization Control Line" for the notes when specifying this specifier.

In "Figure 12.33 Example program without TEMP_PRIVATE specifier", this loop is not parallelized, because the order of array A definition and the reference is uncertain in an outside loop.

Figure 12.33 Example program without TEMP_PRIVATE specifier

```
INTEGER A(100), B(100), C(100,100)
INTEGER M,N
DO J=1,100           ! Target loop for automatic parallelization.
  DO I=1,M           ! Because the value of variable M is uncertain,
    A(I) = B(I)      ! the range where array A is defined is uncertain.
  ENDDO
  DO I=1,N           ! Because the value of variable N is uncertain,
    C(I,J) = A(I)    ! the range where array A is referred is uncertain.
  ENDDO
ENDDO
PRINT*, C
END
```

In "Figure 12.34 Usage of TEMP_PRIVATE specifier", data dependency is solved by the TEMP_PRIVATE specifier. As a result, loop slicing is performed to the outside loop.

Figure 12.34 Usage of TEMP_PRIVATE specifier

```
!OCL TEMP_PRIVATE(A)
P   DO J=1,100
P   DO I=1,M
P   A(I) = B(I)
P   ENDDO
P   DO I=1,N
P   C(I,J) = A(I)
P   ENDDO
P   ENDDO
```

FIRST_PRIVATE(*var[,var]...*)

In the target loop of automatic parallelization, the FIRST_PRIVATE specifier directs the system to treat a specified variable *var* as a local variable in each thread. And, the initial value of the variable is directed to be referred at the loop entry of each thread.

When the optimization message like the following is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n : DO loop is not parallelized: the assign-refer order of 'var' in a parallel execution may differ from the assign-refer order in the serial execution.
```

The FIRST_PRIVATE indicator has the following format:

```
!OCL FIRST_PRIVATE(var[,var]...)
```

Wildcards can be specified in *var*. See Section "12.2.6.5 Wild Card Specification" for information on specifying wildcards.

The FIRST_PRIVATE indicator can be entered at the loop position. It is effective for the corresponding DO loop at the loop position.

The variable that can be specified for this specifier is as follows.

- The scalar variable of integer type, real type or logical type.
- The array variable that meets the following conditions:
 - Integer type, real type or logical type. And,
 - The size of array variable has been fixed when compilation.

When this specifier is applied, the following message is output.

```
jwd5014p-i "file", line n : !OCL FIRST_PRIVATE is applied to variable 'var'.
```

See Section "12.2.7.4 Notes on Using Optimization Control Line" for the notes when specifying this specifier.

In "Figure 12.35 Example program without FIRST_PRIVATE specifier", this loop is not parallelized, because the order of array A definition and the reference is uncertain in an outside loop.

Figure 12.35 Example program without FIRST_PRIVATE specifier

```

INTEGER A(100), B(100,100), C(100,100), D(100)
INTEGER M,N
DO J=1,100                ! Target loop for automatic parallelization.
  DO I=1,M                ! Because the value of variable M is uncertain,
    B(I,J) = A(I)        ! the range where array A is referred is uncertain.
  ENDDO
  DO I=N,100              ! Because the value of variable N is uncertain,
    A(I) = B(I,J) + D(I) ! the range where array A is defined is uncertain.
    C(I,J) = A(I)
  ENDDO
ENDDO
PRINT*, C
END

```

In "Figure 12.36 Usage of FIRST_PRIVATE specifier", data dependency is solved by the FIRST_PRIVATE specifier. As a result, loop slicing is performed to the outside loop.

Figure 12.36 Usage of FIRST_PRIVATE specifier

```

!OCL FIRST_PRIVATE(A)
P   DO J=1,100
P   DO I=1,M
P     B(I,J) = A(I)
P   ENDDO
P   DO I=N,100
P     A(I) = B(I,J) + D(I)
P     C(I,J) = A(I)

```

```
P      ENDDO
P      ENDDO
```

LAST_PRIVATE(*var*[,*var*]...)

In the target loop of automatic parallelization, the LAST_PRIVATE specifier directs the system to treat a specified variable *var* as a local variable in each thread. And, the variable is directed to be referred after this DO loop.

When the optimization message like the following is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n : DO loop is not parallelized: the assign-refer order of 'var' in a
parallel execution may differ from the assign-refer order in the serial execution.
```

The LAST_PRIVATE indicator has the following format:

```
!OCL LAST_PRIVATE(var[,var]...)
```

Wildcards can be specified in *var*. See Section "12.2.6.5 Wild Card Specification" for information on specifying wildcards.

The LAST_PRIVATE indicator can be entered at the loop position. It is effective for the corresponding DO loop at the loop position.

The variable that can be specified for this specifier is as follows.

- The scalar variable of integer type, real type or logical type.
- The array variable that meets the following conditions:
 - Integer type, real type or logical type. And,
 - The size of array variable has been fixed when compilation.

When this specifier is applied, the following message is output.

```
jwd5015p-i "file", line n : !OCL LAST_PRIVATE is applied to variable 'var'.
```

See Section "12.2.7.4 Notes on Using Optimization Control Line" for the notes when specifying this specifier.

In "Figure 12.37 Example program without LAST_PRIVATE specifier", this loop is not parallelized, because the order of array A definition and the reference is uncertain in an outside loop.

Figure 12.37 Example program without LAST_PRIVATE specifier

```
INTEGER A(100), B(100,100), C(100,100), D(100)
INTEGER M,N
DO J=1,100                                ! Target loop for automatic parallelization.
  DO I=1,M
    B(I,J) = I
  ENDDO
  DO I=N,100                                ! Because the value of variable N is uncertain,
    A(I) = B(I,J) + D(I) ! the range where array A is defined is uncertain.
    C(I,J) = A(I)
  ENDDO
ENDDO
PRINT *, A, C
END
```

In "Figure 12.38 Usage of LAST_PRIVATE specifier", data dependency is solved by the LAST_PRIVATE specifier. As a result, loop slicing is performed to the outside loop.

Figure 12.38 Usage of LAST_PRIVATE specifier

```

!OCL LAST_PRIVATE(A)
P   DO J=1,100
P   DO I=1,M
P   B(I,J) = I
P   ENDDO
P   DO I=N,100
P   A(I) = B(I,J) + D(I)
P   C(I,J) = A(I)
P   ENDDO
P   ENDDO
P   PRINT*, A, C

```

var1 op var2

var1 op cnst

These specifiers are used to indicate to the compiler the relationship between two variables or between a variable and a constant.

'*var1*' and '*var2*' are variable names. '*cnst*' is an integer constant or a named constant. Specify as follows:

```

!OCL var1 op var2
!OCL var1 op cnst

```

The relational operators that can be specified for *op* are listed below.

Operator	Explanation
.lt. and <	Less than
.le. and <=	Less than or equal to
.eq. and ==	Equal to
.ne. and !=	Not equal to
.gt. and >	Greater than
.ge. and >=	Greater than or equal to

The '*var1 op var2*' or '*var1 op cnst*' indicator can be entered at the array assignment statement position, loop position, or total position.

Its effect differs depending on the position.

- At the array assignment statement position
It pertains to the corresponding array assignment statement.
- At the loop position
It pertains to the corresponding DO loop and its inner DO loops.
- At the total position
It pertains to all the DO loops in the corresponding program unit.

In "Figure 12.39 Usage of the *var1 op var2* or *var1 op cnst* specifier", parallelization is not performed when the OCL is not specified because it is not certain that the definition or order of data references of array A remains unchanged by parallelization, unless the magnitude relation between the array indexes is identified. *op* specifier enables parallelization by directing that the definition or order of data references of array A remain unchanged before and after the parallelization.

Figure 12.39 Usage of the *var1 op var2* or *var1 op cnst* specifier

```

REAL, DIMENSION(2000) :: A, B
!OCL M.gt.2000
P   DO I = 1, 2000
P   A(I) = A(I+M) + B(I)

```

```
p   END DO
    END
```

12.2.6.5 Wild Card Specification

In the operand of the following optimization control specifiers, a wild card may be specified for a variable name or a procedure name:

- INDEPENDENT
- TEMP

Wildcard is a combination of alphanumeric characters and a special character called a wildcard. Specifying a wildcard is equivalent to specifying multiple variable names or procedure names that correspond to the criteria specified by the string specified as the wildcard.

The two wildcard characters "*" and "?" have the following meanings:

- "*" matches a string of one or more characters consisting of any alphanumeric characters.
- "?" matches any single alphanumeric character.

Note that multiple wildcards cannot be specified when specifying one wildcard.

```
!OCL TEMP(W*)
```

In this example, "W*" specifies a variable names where the first character is "W" and the length is two or more.

For example, the variable names WORK1, W2, and WORK3 match the wildcard.

```
!OCL INDEPENDENT(SUB?)
```

In this example, "SUB?" specifies a procedure name whose first three characters are "SUB" and the length is four.

For example, the array name SUB1, SUB2, and SUB9 match the wildcard.

12.2.7 Notes on Automatic Parallelization

This section provides notes about using Automatic Parallelization.

12.2.7.1 Caution when Specifying Compile-Time Option -Kparallel,instance=N

When specifying the number of threads for parallel processing with the compiler option -Kparallel, instance=*N*, the *N* must be the same as the number of threads used on runtime. See Section "12.2.2.1 Number of Threads" for information about specifying the number of threads.

When an incorrect value is specified on -Kinstance, the program will abort with the following runtime message.

```
jwe1040i-s This program cannot be executed, because the number of
           threads specified by -Kinstance=N option and the actual
           number of threads are not equal.
```

12.2.7.2 Nested Parallel Processing

Nested parallel processing is when a procedure is called from a sliced DO loop, and the procedure includes another sliced DO loop. When this occurs, the inner DO loop will be executed serially. -Kparallel, instance=*N* must not be specified on compiling a source program that may cause nested parallel processing.

"Figure 12.40 Multiprocessing of nested loops" illustrates a sliced DO loop executed as a serial process. If a program with such a DO loop is compiled with the -Kparallel, instance=*N* option, the program may not work correctly.

Figure 12.40 Multiprocessing of nested loops

File: a.f


```
SUM=SUM+SUM1+SUM2
```

12.2.7.4 Notes on Using Optimization Control Line

Invalid usage of the optimization control lines is shown below.

The system will perform incorrect loop slicing because of the incorrect optimization control lines.

- TEMP specifier

The following program specifies TEMP incorrectly for variable T. There will not be a correct value assigned to variable LAST, because the system cannot guarantee a correct value of variable T at the end of the DO loop.

```
!OCL TEMP(T)
DO I=1, 1000
  T=A(I)+B(I)
  C(I)=T+D(I)
END DO
LAST = T
```

- INDEPENDENT specifier

The following program specifies INDEPENDENT incorrectly for procedure SUB. The execution result is unpredictable when array A is sliced, because the order of the data references for array A in parallel execution is different from the order of data references in serial execution.

```
!OCL INDEPENDENT(SUB)
DO I=2, 1000
  A(I)=B(I)+1.0
  CALL SUB(I-1)
END DO
...
END
RECURSIVE SUBROUTINE SUB(J)
COMMON A(1000)
A(J)=A(J)+1.0
END
```

- TEMP_PRIVATE specifier

The following program specifies TEMP_PRIVATE incorrectly for variable A. Because an uncertain value is referred in each thread, the execution result is incorrect.

```
!OCL TEMP_PRIVATE(A)
DO I=1,1000
  B(I) = A
ENDDO
```

- LAST_PRIVATE specifier

The following program specifies LAST_PRIVATE incorrectly for array A. Because the array A may not be defined by the last thread, the execution result is incorrect.

```
!OCL LAST_PRIVATE(A)
DO J=1,1000
  DO I=J,500
    A(I) = B(J)
  ENDDO
ENDDO
PRINT *, A
```

12.2.7.5 I/O Statement and Intrinsic Procedure Call on Parallel Processing

When a sliced DO loop calls a procedure that includes an input-output statement, an intrinsic subroutine, or an intrinsic function that is not supported by loop slicing, the program may not work correctly. The performance of the program may deteriorate because of the overhead of parallel processing, or the result of the input/output statement may not be the same as with serial processing.

"Figure 12.42 Multiprocessing I/O statement" shows an example of input-output statements in a procedure called from a sliced DO loop.

Figure 12.42 Multiprocessing I/O statement

File: a.f

```
!OCL INDEPENDENT( SUB )
DO I=1,100
  J=I
  CALL SUB(J)
END DO
:
END
RECURSIVE SUBROUTINE SUB(N)
:
PRINT *,N
:
END
```

12.3 Parallelization by OpenMP Specifications

This section explains the parallelization based on the OpenMP specifications provided by this compiler. See the "OpenMP Architecture Review Board" website for details about the OpenMP specifications.

This system supports the following specifications. However, the constructs, the environment variables, and the run time library routines related to the device are not supported.

Supported Specification
OpenMP 4.5
part of OpenMP 5.0 (*1)

*1: The following functions can be used:

- IN_REDUCTION clause of TASK construct
- TASK_REDUCTION clause of TASKGROUP construct
- REDUCTION clause and IN_REDUCTION clause of TASKLOOP construct

12.3.1 Compilation

For compiling source programs and linking, specify the following options to the compile command.

The specified option is different by whether to perform parallelization by the OpenMP specification.

Parallelization by the OpenMP specification	Specified option	
	Compile time	Link time
When using the parallelization and the SIMD Extensions	-Kopenmp	-Kopenmp -Nlibomp
When using the SIMD Extensions only, not using the parallelization	-Kopenmp_simd	There is no specified option

12.3.1.1 Compiler Options for OpenMP Programs

This section explains the compiler options for OpenMP Programs.

-Kopenmp

This option enables OpenMP directives and compiles the source programs.

The -Kopenmp option should also be specified at the linkage phase of an object file, when the object file has previously been compiled with this option.

Example

```
$ frtpx a.f90 -Kopenmp -c
$ frtpx a.o -Kopenmp -Nlibomp
```

-Kopenmp_simd

-Kopenmp_simd option enables only the OpenMP SIMD construct, the DECLARE SIMD construct, the DECLARE REDUCTION directive, and the ORDERED construct with the SIMD clause, and compiles the source programs.

-Nlibomp

This option specifies to use LLVM OpenMP Library.

-Nlibomp option is required at linking.

12.3.1.2 Compiler Options for Obtaining Optimization Information

This section describes the compiler options for obtaining optimization Information.

-Nlst=p

When the compiler option -Nlst=p is specified, the optimization information for the instructions generated by the OpenMP directives are marked as follows:

- "p" is marked for statements that can be executed in parallel processing. It is not marked for the statements executed redundantly. Statements directly enclosed in a DO or SECTIONS statement will meet this condition. Within the WORKSHARE directive, only the parallelized statements are marked as "p".
- "s" is marked for statements that will be executed in only one thread. Statements directly enclosed in MASTER, SINGLE, CRITICAL, or ORDERED directive will meet this condition. For the instructions just after the ATOMIC directive, "s" will be marked only when whole instructions are executed within a single thread.
- "m" is marked for statements which include both instructions that will be executed in parallel and in a single thread. A statement immediately after an ATOMIC directive within a WORKSHARE directive may meet this condition, for example.

For nested parallelization, only the parallelization information of the innermost PARALLEL region, which includes the target instructions, is displayed. Whether the PARALLEL region is executed in parallel or serially does not affect the display.

-Koptmsg=2

The diagnostic message related to the execution performance such as SIMD extension and COLLAPSE of the OpenMP specification is output.

12.3.2 Execution Process

The procedure to run the OpenMP program is the same as executing it as a serial process.

12.3.2.1 Environment Variable at Execution

The execution process of an OpenMP program is the same as the procedure for a serial-processing program.

Environment variable OMP_WAIT_POLICY

The synchronization process can be controlled with the environment variable OMP_WAIT_POLICY.

ACTIVE : Uses spin wait until all threads are synchronized.
 PASSIVE : Uses no spin wait but suspending wait.

The default is "PASSIVE".

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

Environment variable OMP_PROC_BIND

The environment variable OMP_PROC_BIND can control the thread affinity.

Either TRUE, FALSE or comma separated list of MASTER, CLOSE, or SPREAD can be specified to this environment variable. The values of the list sets the thread affinity policy used by a PARALLEL region corresponding to the nest level.

This environment variable is set to CLOSE by default.

If this environment variable is set to FALSE, thread affinity is disabled, and the PROC_BIND clause on PARALLEL construct is ignored.

Otherwise, thread affinity is enabled and the initial thread is bound to first place in place list.

The MASTER thread affinity policy indicates that all threads are bound to same place as master thread.

The CLOSE thread affinity policy indicates that threads are bound to next to the place where master thread is bound.

The SPREAD thread affinity policy indicates that the threads are bound to the one of sub-partition places divided from master thread's place partition.

The effect of TRUE is same as SPREAD.

See environment variable OMP_PLACES for place.

Environment variable OMP_PLACES

The environment variable OMP_PLACES defines place list.

The explicit place list is defined by ordered set of commas separated non-negative numbers enclosed by braces. The numbers represents the smallest unit of CPU resource in the system.

The value of OMP_PLACES is either abstract name or an explicit place list. The value can be specified using following format.

```
{ abstract-name[(num-places)] | place-list }
```

abstract-name

The following abstract names can be set to this environment variable. The default value is CORES.

Abstract Name	Meaning
THREADS	Each place corresponds to a single hardware thread, which represents smallest unit of CPU resource in the system. The effect of THREADS is equivalent to CORES.
CORES	Each place corresponds to a single hardware core, which represents smallest unit of CPU resource in the system. The effect of CORES is equivalent to THREADS.
SOCKETS	Each place corresponds to a single socket, which represents NUMA node.

num-places

The length of the place list. Positive integer.

place-list

The explicit place list can be specified using following format.

```
{ place:length[:stride] | [!]place[,place-list] } (*1)
```

place

"{"*place1*" }"(*2)

place1

{ *cpuid.length[:stride]* | [!]*cpuid*[,*place1*] }

length

The length of elements. Positive integer.

stride

The value of increment or decrement. Positive integer. Default value is 1.

cpuid

Smallest unit of CPU resource in the system.

*1) An exclusion operator "!" exclude the number or place immediately following the operator.

*2) "{" and "}" are braces.



Example

Example of environmental variable OMP_PLACES

- Example 1:

```
$ export OMP_PLACES="CORES"
```

This example defines place list of CORES abstract name.

- Example 2:

```
$ export OMP_PLACES="CORES(4)"
```

This example defines place list of CORES abstract name of 4 places.

- Example 3:

```
$ export OMP_PLACES="{12,13,14},{15,16,17},{18,19,20},{21,22,23}"
$ export OMP_PLACES="{12:3},{15:3},{18:3},{21:3}"
$ export OMP_PLACES="{12:3}:4:3"
```

All above examples define same place of 12 to 14, 15 to 17, 18 to 20, and 21 to 23.

Environment variable OMP_STACKSIZE

The stack area size for each thread can be specified using the environment variable OMP_STACKSIZE in byte, Kbytes, Mbytes, Gbytes, or Tbytes. The default size is 8M bytes.

Environment variable GOMP_CPU_AFFINITY

Threads are bound to CPUs in order of the specified cpuid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid list shall be separated by comma (',') or space(' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. (0<=*cpuid1*<CPU_SETSIZE)

cpuid2: cpuid of the end of the range. (0<=*cpuid2*<CPU_SETSIZE)

inc: increment (1<=*inc*<CPU_SETSIZE)

In addition, it is necessary to be the following.

```
cpuid1<=cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The *cpuid* can be used the above-mentioned value. However, *cpuid* which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See CPU_SET(3) about details of CPU_SETSIZE.

Note

If *cpuid* is outside the CPU affinity of the process at the start of execution, an error will be output and the program will terminate. Correct the setting value.

Example

Examples of using environment variable GOMP_CPU_AFFINITY

- Example 1:

```
$ export GOMP_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 2:

```
$ export GOMP_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.

When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

- Example 3:

```
$ export GOMP_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 4:

```
$ export GOMP_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.

When the number of threads is six or more, it is repeatedly used from the beginning of the list.

12.3.2.2 Environment Variable for OpenMP Specifications

The following environment variables are supported by the OpenMP specifications.

For environment variables available on OpenMP 4.0 and later, the specifications they are supported on are enclosed in a pair of parentheses after each environment variable.

See the "OpenMP specifications" for details.

Environment variable OMP_SCHEDULE

This specifies the schedule type and the chunk size that should be performed for the DO and PARALLEL DO directives with schedule type RUNTIME.

Environment variable OMP_NUM_THREADS

This specifies the number of threads used during execution.

Environment variable OMP_DYNAMIC

This specifies to enable and disable dynamic thread adjustment.

Environment variable OMP_PROC_BIND

This specifies whether threads are bound to CPUs.

Environment variable OMP_NESTED

This enables or disables nested parallelism.

Environment variable OMP_STACKSIZE

This specifies the stack size for each thread.

Environment variable OMP_WAIT_POLICY

This specifies the behavior of the synchronization process.

Environment variable OMP_MAX_ACTIVE_LEVELS

This specifies the maximum number of nested active PARALLEL regions.

Environment variable OMP_THREAD_LIMIT

This specifies the maximum number of threads executed on an Open MP program.

Environment variable OMP_PLACES (OpenMP 4.0 and later)

This specifies the list of places for thread affinity.

Environment variable OMP_CANCELLATION (OpenMP 4.0 and later)

This enables or disables the cancellation.

Environment variable OMP_MAX_TASK_PRIORITY (OpenMP 4.5 and later)

This specifies maximum task priority.

Environment variable OMP_DISPLAY_ENV (OpenMP 4.0 and later)

This enables or disables to display the OpenMP version number and the initial value of the internal control variables.

The OpenMP version number is 201611.

12.3.2.3 Notes during Execution

The following points should be noted when using parallelization features based on the OpenMP specifications on this compiler.

Allocating Variables at Runtime

If the size of the stack area for every thread is defined as a specific size, the environment variable OMP_STACKSIZE can be specified.

See Section "[12.3.2.1 Environment Variable at Execution](#)" for information about environment variables OMP_STACKSIZE.

Maximum number of CPUs

- The number of CPUs that can be used in the system.

Number of Threads

- The number of threads for OpenMP will be determined using the following priority. The number of threads for automatic parallelization follows the description in Section "[12.2 Automatic Parallelization](#)".
 1. The value specified in the NUM_THREADS clause in the PARALLEL statement
 2. The value specified in the OMP_SET_NUM_THREADS routine.
 3. The value specified in the environment variable OMP_NUM_THREADS
 4. Maximum number of CPUs

When the dynamic thread adjustment feature is enabled, the number of threads is adjusted to be less or equal to the number of CPUs.

When the dynamic thread adjustment feature is disabled, the thread number determined by the above priority will be used. When more than one thread is assigned for a CPU, the thread that is to be executed in parallel will be executed in a time sharing manner. In this case, the overhead caused by synchronizing the threads may sacrifice the performance of the program. It is recommended to conserve resources by setting one thread per CPU, unless there is a specific reason to do otherwise.

Notes when using service function and intrinsic subroutine

- ALARM Service Function
It may not work correctly. It is only for a program using serial processing.
- KILL and SIGNAL Service Functions
It may cause a deadlock. It is only for a program using serial processing.
- FORK Service Function
It may not work correctly. It is only for a program using serial processing.
- SYSTEM, SH, and CHMOD Service Functions and EXECUTE_COMMAND_LINE Intrinsic Subroutine
Memory usage doubles and performance may deteriorate. Use only with programs using serial processing.

CPU binding for thread

Environmental variable OMP_PROC_BIND or GOMP_CPU_AFFINITY can control thread affinity. GOMP_CPU_AFFINITY takes precedence over OMP_PROC_BIND.

12.3.3 Implementation-Dependent Specifications

The implementation-dependent specification in the OpenMP is implemented in the following way in this compiler. Refer to the OpenMP specifications for details about these items.

Memory Model

When multiple threads attempt to update a variable that exceeds 4 bytes or is stored across a 4-byte boundary, the value of the variable may be left undefined without storing any of the values, unless lock control is explicitly performed.

If read and write to the variable occurs from multiple threads at the same time, and this exceeds 4 bytes or crosses a 4-byte boundary, the read value may be undefined rather than the written value, unless lock control is explicitly performed.

Internal Control Variables

The initial values for each of the internal control variables are as follows:

- nthreads-var: Maximum number of CPUs.
- dyn-var: FALSE.
- run-sched-var: STATIC without chunk size.
- def-sched-var: STATIC without chunk size.
- bind-var:FALSE.
- stacksize-var: the stack area size for threads. Default size is 8M bytes .
- wait-policy-var: PASSIVE.
- thread-limit-var: 2147483647.
- max-active-levels-var: 2147483647.
- place-partition-var: Default value is CORES, which is equivalent to THREADS in the system.

Dynamic Thread Adjustment Features

This compiler provides dynamic thread adjustment for parallelization performed based on OpenMP specifications. See Section "[12.3.2.3 Notes during Execution](#)" for information about the effect of this feature.

The dynamic thread adjustment feature is not set by default.

Loop Statement

An eight byte integer type is used to calculate the iteration count of a collapsed loop.

When AUTO is set for the internal control variable run-sched-var, the effect of the SCHEDULE(RUNTIME) construct is set to SCHEDULE(STATIC).

SECTIONS Construct

Assignment to a thread of a structured block in a SECTIONS construct is performed in the same way as a static schedule.

SINGLE Construct

SINGLE region is executed by the thread that reaches the region first.

SIMD Construct

An eight byte integer type is used to calculate the iteration count of a collapsed loop.

SIMD length is the value to which the compiler decides automatically.

When the *alignment* parameter is not specified in the ALIGNED clause, it is considered that the following value was specified for the *alignment* parameter.

When -KSVE option is effective at the compile time.	Alignment of the type of the list item.
When -KNOSVE option is effective at the compile time.	16

DECLARE SIMD Construct

SIMD length when the SIMDLEN clause is not specified is as follows.

- When -KSVE option is effective at the compile time

SIMD length is decided at the run time.

- When -KNOSVE option is effective at the compile time

SIMD length is decided by the size of the minimum type among all input parameters and return value. SIMD length is as follows by the size of the type.

Size of the type (byte)	SIMD length
1	16, 8
2	8, 4
4	4, 2
8	2
16	

When the *alignment* parameter is not specified in the ALIGNED clause, it is considered that the following value was specified for the *alignment* parameter.

When -KSVE option is effective at the compile time	Alignment of the type of the list item.
When -KNOSVE option is effective at the compile time	16

TASKLOOP Construct

When GRAINSIZE and NUM_TASKS clause are not specified, these values will be set appropriately considering the loop iteration.

An eight byte integer type is used to calculate the iteration count of a collapsed loop.

CRITICAL Construct

This implementation do not support lock hint. Therefore there is no the effect of using a hint clause.

ATOMIC Construct

Two ATOMIC regions will be executed independently (not exclusively), when a variable type to be updated or a type parameter is different. When the type and type parameter match, it may be executed exclusively even if the address is different.

- When updating a logical type, complex number type, 1 byte integer type, 2 byte integer type, or quadruple precision (16 byte) real type variable
- When updating array element and index expression is not the same
- The target expression has explicit or implicit type conversion

OMP_SET_NUM_THREADS Routine

If the argument is not a positive integer, the effect of OMP_SET_NUM_THREADS routine is same as the argument is 1. A value that exceeds the number of threads supported by the system must not be specified.

OMP_SET_SCHEDULE Routine

There are no schedule types that are implementation-dependent.

OMP_SET_MAX_ACTIVE_LEVELS Routine

A call to the OMP_SET_MAX_ACTIVE_LEVELS routine will be ignored when it is performed from an explicit PARALLEL region. It will also be ignored when the argument is an integer that less than 0.

OMP_GET_MAX_ACTIVE_LEVELS Routine

The OMP_GET_MAX_ACTIVE_LEVELS routine can be called from anywhere in the program and it returns the value of the internal control variable max-active-levels-var.

OMP_GET_PLACE_PROC_IDS Routine

The OMP_GET_PLACE_PROC_IDS Routine returns identifiers of the available processors in the specified place. The definition of processor is the CPU controlled by operating system.

OMP_INIT_LOCK_WITH_HINT and OMP_INIT_NEST_LOCK_WITH_HINT Routine

This implementation do not support lock hint. Therefore there is no the effect of using these routines.

Environment Variable OMP_SCHEDULE

When the schedule type specified for OMP_SCHEDULE is invalid, the schedule type is ignored, and the default schedule (STATIC without chunk size) is used.

When the schedule type specified for OMP_SCHEDULE is STATIC, DYNAMIC, or GUIDED, and the chunk size is not a positive number, the chunk size will be as follows:

Schedule Type	Chunk Size
STATIC	no chunk size
DYNAMIC, or GUIDED	1

Environment Variable OMP_NUM_THREADS

When 0 is specified for the list of OMP_NUM_THREADS, it works in the same ways as when 1 is specified. When a value less than 0 is specified for the list of OMP_NUM_THREADS, the number of threads is determined by "[12.3.2.3 Notes during Execution](#)" procedure. A value that exceeds the number of threads supported by the system must not be specified.

Environment Variable OMP_PROC_BIND

If the value does not conform to the specified format for OMP_PROC_BIND, the value is ignored, and the default value (CLOSE) is used.

If all of the following conditions are met, the excess threads are assigned to places evenly as possible.

- SPREAD or CLOSE is specified to OMP_PROC_BIND.
- T (number of threads) is greater than P (number of places).
- P does not divide T evenly.

Environment Variable OMP_PLACES

If the value does not conform to the specified format, the value is ignored, and the default value (CORES) is used.

Environment Variable OMP_DYNAMIC

When a value other than TRUE or FALSE is specified for OMP_DYNAMIC, the value is ignored and the default value (FALSE) is used.

Environment Variable OMP_NESTED

When a value other than TRUE or FALSE is specified for OMP_NESTED, it is ignored and the default value (FALSE) is used.

Environment Variable OMP_STACKSIZE

When the value specified for OMP_STACKSIZE does not meet the defined format, it is ignored and the default value (8M bytes) is used.

Environment Variable OMP_WAIT_POLICY

ACTIVE performs spin wait. PASSIVE performs suspend wait.

Environment Variable OMP_MAX_ACTIVE_LEVELS

When the value specified for OMP_MAX_ACTIVE_LEVELS is an integer that is less than 0, it is ignored and the default value (2147483647) is used.

Environment Variable OMP_THREAD_LIMIT

When the value specified for OMP_THREAD_LIMIT is not a positive integer, it is ignored and the default value (2147483647) is used.

THREADPRIVATE directing statement

The value, allocation state, and association state of the thread private variables will be maintained across the consecutive PARALLEL regions, only when the following conditions are satisfied. These conditions are not affected by the configuration of the dynamic thread adjustment feature and nested parallelization.

- The consecutive PARALLEL region is not nested within another explicit PARALLEL region.
- The same number of threads is used to execute both PARALLEL regions.

If these conditions are not satisfied, the value, allocation status, and association status of the threadprivate variable at the entry point of the subsequent PARALLEL region will have an undefined status. However, the allocation status will be not allocated at the entry point of the subsequent PARALLEL region, when the following is true:

- The allocation status is not allocated just before entering the subsequent PARALLEL region.
- The allocation status is not allocated by any thread at the exit of the all preceding PARALLEL regions that are already executed.

SHARED Clause

When passing a shared variable to a non-intrinsic procedure, the values of the shared variables are stored in a temporary storage area, and the values are stored in the actual argument area after the procedure call. This can occur when the following conditions are satisfied:

- a. The actual argument is one of the following:
 - A shared variable
 - A subobject of a shared variable
 - An object associated to a shared variable
 - An object associated to a subobject of shared variable
- b. In addition the actual argument is one of the following:
 - An array section
 - An array section with a vector subscript
 - An assumed-shape array
 - A pointer array
- c. The associated dummy argument for this actual argument is an explicit-shape array or an assumed-size array.

Runtime Library Definitions

This compiler provides an include file omp_lib.h and a module omp_lib for the parallelization feature based on OpenMP specifications. This does not include an interface block with generic name.

The following notes must be considered when using the module `omp_lib` and `omp_lib.h`:

- When the compiler option `-AU` is specified, module name `omp_lib` and each library routine name must be specified using lower case alphabetic characters.
- When the compiler option `-CcdI18`, `-Ccl4I8`, `-CcdLL8`, `-Ccl4L8`, `-Ccd4d8`, or `-Cca4a8` is specified, the corresponding type of the library routine argument and the result will be converted. On execution, the corresponding runtime option `-Lb` or `-Li` must be specified.

12.3.4 Clarifying OpenMP Specifications and Restrictions

This section explains the interpretation of the OpenMP Specifications and some restrictions in this system.

12.3.4.1 Specifying Label with ASSIGN Statement and Assigned GO TO Statement

An `ASSIGN` statement cannot reference a label outside of the structured block that the `ASSIGN` statement itself belongs to. An `ASSIGN` statement cannot reference a label in a `PARALLEL` region from outside of the `PARALLEL` region.

An assigned `GO TO` statement must not jump into, or jump out of the structured block range.

12.3.4.2 Additional Functions in ATOMIC Construct and Reduction Identifier

The following intrinsic functions can be specified in an `ATOMIC` construct and reduction identifier.

Intrinsic function : AND, OR, XOR

12.3.4.3 Notes on Using THREADPRIVATE

When using the `THREADPRIVATE` directive, the same common block name must be used for all program units and procedures in the program specified on `THREADPRIVATE`.

The common block, specified with `THREADPRIVATE`, cannot have the size extended.

When a module or submodule containing the `THREADPRIVATE` directive is compiled with compiler option `-Kopenmp`, the program that references the `THREADPRIVATE` variable also has to be compiled with compiler option `-Kopenmp`.

12.3.4.4 Inline Expansion

The following procedures are not expanded inline.

- User defined procedures which include OpenMP directives except `DECLARE SIMD` directive.

12.3.4.5 Internal Procedure Calling from PARALLEL Region

All the parent procedure variables, which are referenced in the internal procedure called from a `PARALLEL` region, are regarded as `SHARED`, even if they are privatized in the `PARALLEL` region.

Example:

```
...
I=1          THIS "I" IS SHARED.
!$OMP PARALLEL PRIVATE(I)
I=2          *
PRINT *,I    *"I" IS PRIVARE IN THIS RANGE.
CALL C_PROC() *
!$OMP END PARALLEL
CONTAINS
SUBROUTINE C_PROC()
...
PRINT *,I    *"I" is SHARED IN THIS RANGE
...
END SUBROUTINE
...
```

12.3.4.6 Polymorphic Variable

Polymorphic variable cannot be specified in `THREADPRIVATE` directive, `REDUCTION` clause, `IN_REDUCTION` clause, and `TASK_REDUCTION` clause. Unlimited polymorphic variable cannot be specified as following clauses:

- `PRIVATE`
- `FIRSTPRIVATE`
- `LASTPRIVATE`
- `COPYPRIVATE`
- `COPYIN`
- `DEPEND`
- `ALIGNED`
- `UNIFORM`

12.3.4.7 OPTIONAL Attribute

When a variable declared with `OPTIONAL` attribute is specified in `PRIVATE`, `FIRSTPRIVATE`, `LASTPRIVATE`, `REDUCTION`, `IN_REDUCTION`, `TASK_REDUCTION`, or `LINEAR` clause, the variable cannot be specified as actual argument of `PRESENT` intrinsic function in the scope.

12.3.4.8 ASSOCIATE NAME

An associate name of `SELECT TYPE` construct cannot be specified in OpenMP directive.

12.3.4.9 Selector

The variable of selector with the associate name in a `SELECT TYPE` construct cannot be specified as following clauses:

- `PRIVATE`
- `FIRSTPRIVATE`
- `LASTPRIVATE`
- `COPYPRIVATE`
- `COPYIN`
- `REDUCTION`
- `IN_REDUCTION`
- `TASK_REDUCTION`

12.3.4.10 Derived Type Variable with Length Type Pparameter

The derived type variable with length type parameter cannot be specified in `THREADPRIVATE` directive. The derived type variable with length type parameter cannot be specified as following clauses:

- `PRIVATE`
- `FIRSTPRIVATE`
- `LASTPRIVATE`
- `REDUCTION`
- `IN_REDUCTION`
- `TASK_REDUCTION`
- `COPYPRIVATE`

- COPYIN
- DEPEND
- ALIGNED
- UNIFORM

12.3.4.11 Assumed Type Variable

An assumed type variable cannot be specified as following clauses:

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- REDUCTION
- IN_REDUCTION
- TASK_REDUCTION
- COPYPRIVATE
- COPYIN
- DEPEND
- ALIGNED
- UNIFORM

12.3.4.12 Assumed Rank Variable

An assumed rank variable cannot be specified as following clauses:

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- REDUCTION
- IN_REDUCTION
- TASK_REDUCTION
- COPYPRIVATE
- COPYIN
- DEPEND
- ALIGNED
- UNIFORM

12.3.5 Notes on OpenMP Programming

12.3.5.1 Implementation of PARALLEL Region and Explicit TASK Region

The structured block within a PARALLEL construct or a TASK construct is compiled as an internal procedure.

The internal procedures generated from a PARALLEL construct will be named "_OMP_id-numberA_". The "id-numberA" is a unique number that identifies PARALLEL construct in a program unit.

The internal procedures generated from a TASK construct will be named "_TSK_id-numberB_". The "id-numberB" is a unique number that identifies TASK construct in a program unit.

12.3.5.2 Implementation of THREADPRIVATE Variable

The THREADPRIVATE variable is implemented by using the Thread-Local Storage. When the size of the Thread-Local Storage exceeds the range of the offset, an error occurs at link time.

Example of error message:

```
relocation truncated to fit: R_AARCH64_TLSLE_ADD_TPREL_HI12 against symbol 'varname_'
```

When the error occurs, specify the size of an appropriate offset by `-Ktls_size={ 12|24|32|48 }` option. For details about the `-Ktls_size` option, see Section "2.2.3 Description of Compiler Options".

12.3.5.3 Automatic Parallelization for OpenMP Programs

This compiler allows the specification of the compiler options `-Kopenmp` and `-Kparallel` at the same time. When they are specified together, the DO loop automatic parallelization performed is restricted as follows:

- Automatic parallelization is not performed for DO loops within the OpenMP constructs.
- Automatic parallelization is not performed for DO loops that statically include OpenMP directives.
- When there is a DO loop which is parallelized by the OpenMP DO directive, automatic parallelization is not performed for the following DO loops:
 - DO loops parallelized by the OpenMP DO directive
 - DO loops within a DO loop parallelized by the OpenMP DO directive

12.3.5.4 Runtime Library Routines

The compiler option `-mldefault=cdecl` is specified, refer the module `omp_lib`. In a program that does not refer the module `omp_lib`, the function with the same name in C/C++ is executed. Therefore, when a routine with parameters is executed, the execution result is different, the performance deteriorates, a runtime error is output, and the program terminates. The program may not work properly.



Example

Example of executing an `omp_set_num_threads` routine

- Example program

```
program example
  integer :: omp_get_max_threads
  call omp_set_num_threads(12)
!$omp single
  print *, "Max Threads=", omp_get_max_threads()
!$omp end single
end
```

- Execution result

- If the compiler option `-mldefault=cdecl` is not specified

```
Max Threads= 12
```

- If the compiler option `-mldefault=cdecl` is specified

```
Max Threads= 4199088  (*)
```

*) The correct value is not output.

To get the correct results, refer the module `omp_lib` as follows.

```
program example
  use omp_lib
  call omp_set_num_threads(12)
!$omp single
  ! refer the module omp_lib
```

```

    print *, "Max Threads=", omp_get_max_threads()
!$omp end single
end

```

12.3.6 Debugging OpenMP Programs

See Section "[Chapter 8 Debugging](#)" for the debug feature in this system.

The following restrictions are applied to the debugging of the OpenMP programs:

- A function generated by the DECLARE SIMD construct cannot be debugged using the debugger.
- The compiler option -H, which provides the checking feature for debugging, is not effective within the PARALLEL region.

12.4 Runtime Messages

LLVM OpenMP Library outputs runtime messages to standard error as following format.

```
OMP: type message
```

The meaning of runtime messages of LLVM OpenMP Library is explained as follows.

<i>type</i>	Hint	Hint about OpenMP.
	Info	Information about OpenMP.
	Warning	Warning about OpenMP. Not fatal error. Even if this message is displayed, execution continues.
	Error	Fatal error about OpenMP. If this message is displayed, execution is aborted.
	System error	Fatal error about the system. If this message is displayed, execution is aborted.
<i>message</i>	content of message	



Example

```

$ export OMP_STACKSIZE=1Z
$ ./a.out
OMP: Warning #80: OMP_STACKSIZE="1Z": value too large.
OMP: Info #107: OMP_STACKSIZE value "9223372036854775807" will be used.
OMP: Error #34: System unable to allocate necessary resources for OMP thread:
OMP: System error #11: Resource temporarily unavailable
OMP: Hint Try decreasing the value of OMP_NUM_THREADS.

```

The above example is series of messages when unacceptable value in FX system is set to environmental value OMP_STACKSIZE as stack size.



Note

The following runtime options are invalidated for the control of the runtime message of LLVM OpenMP Library. For details about each option, see Section "[3.3 Runtime Options](#)".

- *-enum*
- *-lvl*
- *-Re*
- *-Rp*

Appendix A Restrictions and Notes

A.1 Restrictions

The Fortran compiler restricts the size and complexity of source programs. Some restrictions apply to one Fortran statement and others apply to a combination of several statements.

A.1.1 Nesting Level of Functions, Array Sections, Array Elements, and Substring References

In a combination of function, array section, array element, and substring references, nesting must not exceed 255 levels. Depending on the compiler, nesting may be subject to further restrictions.

Example: Nesting level of functions and array element references

```
INTEGER IA(10)
EXTERNAL FUN,IFUN
:
X=FUN(IFUN(IA(1))) ! The nesting level is 3
```

A.1.2 DO, CASE, IF, SELECT TYPE, BLOCK, CRITICAL, and ASSOCIATE Constructs

In a combination of DO, CASE, IF, SELECT TYPE, BLOCK, CRITICAL, and ASSOCIATE constructs, the total nesting level must not exceed 50.

Example: Nesting levels of DO and IF constructs

```
DO I=1,10
  A(I)=A(I)+1 ! The nesting level is 1
  IF (I.LE.5)THEN
    CALL S! The nesting level is 2
    DO J=1,5
      B(J)=A(J)! The nesting level is 3
    END DO
  CALL T! The nesting level is 2
END IF
END DO
```

A.1.3 Implied DO-Loops

The nesting level of implied DO-loops in input/output or DATA statements must not exceed 25.

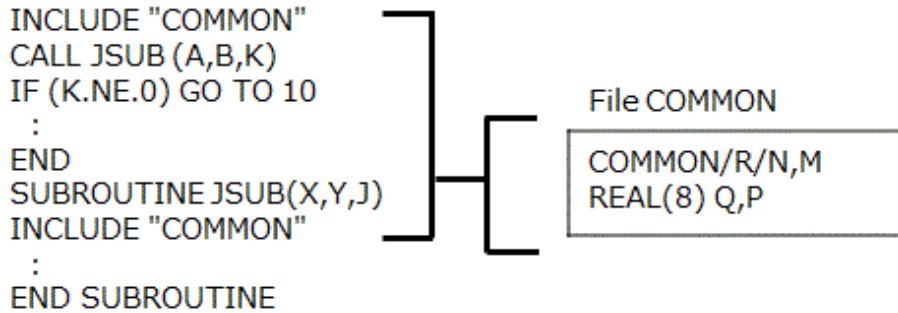
Example: Nesting level of implied DO-loops

```
PRINT *, ((A(I,J),I=1,2),J=1,3) ! The nesting level is 2
```

A.1.4 INCLUDE Line

The nesting level of INCLUDE lines must not exceed 16.

Example: Nesting level of INCLUDE lines



In this case, the nesting level of INCLUDE lines is 1.

A.1.5 Array Declarations

For array declarations, there are two restrictions.

Restriction for All Array Declarations

The compiler calculates T for each array declaration to reduce the number of calculations needed for array sections or array element addresses. The absolute value of T obtained by following formula must not exceed the limitation value, and the absolute value must not exceed the limitation value during calculation:

$$T = l_1 * s + \sum_{i=2}^n \{ l_i * (\prod_{m=2}^i d_{m-1} * s) \}$$

n: Array dimension number

s: Array element length

l: Lower bound of each dimension

d: Size of each dimension

T: Value calculated for the array declaration

The limitation value is 9223372036854775807.

Example: Restrictions on array declarations

```

PARAMETER (IL=10000000,IU=10000999)
INTEGER(8) A(IL:IU,IL:IU,IL:IU,IL:IU,IL:IU)

```

Because T (the value calculated for array declarations) for array A is 8008008008008000000, this array declaration must not be specified.

Restriction for Array Declaration of Zero-Sized Array

In an array declaration of zero-sized array that has bounds that are nonconstant specification expressions, the absolute value of T obtained by following formula must not exceed the limitation value, and the absolute value must not exceed the limitation value during calculation:

$$T = s * (\prod_{i=1}^n u_i - l_i + 1)$$

n: Array dimension number
s: Array element length
l: Lower bound of each dimension
u: Upper bound of each dimension
T: Value calculated for the array declaration

The limitation value is 9223372036854775807.

A.1.6 Array Section

In the following subscript triplet,

```
subscript1 : subscript2 : stride
```

overflow must not occur during calculation.

```
(subscript2 - subscript1 + stride) / stride
```

A.1.7 Distance between a Branch Instruction and Branch Destination Instruction

The immediate field size of the branch destination address in a branch instruction is restricted. For this reason, when the branch instruction is separated from the branch destination instruction over a distance that exceeds a certain fixed value, the assembler may not be able to assemble the program. In this case, the assembler displays the following messages:

```
/var/tmp/asmAAAa006jR.s: Assembler messages:  
/var/tmp/asmAAAa006jR.s:10: Error: relocation overflow
```

The likelihood for this symptom becomes high when the program is assembled with the `-H` and `-Kthreadsafe` options specified simultaneously for a loop with several thousand or more executable statements or when there are numerous array expressions in a huge loop.

If this symptom occurs, correct the program (e.g., perform loop splitting) or `-O0` option is in effect. It is necessary to note it for the execution performance because `-O0` option suppresses the optimization.

A.2 Notes

A.2.1 Debug Using Debugger

This section explains the notes when the executable file is debugged by using the debugger.

Debugging of optimized program

Note the following restrictions when using debugger with program compiled with `-O1` or higher option:

- Debugging of the variables is guaranteed at the entry of each procedure, and debugging of only the dummy arguments, variables declared in modules or submodules, and variables declared in common blocks is guaranteed. And note the following restrictions:
 - Debugging of the dummy arguments of which type is quadruple precision REAL, single precision COMPLEX, double precision COMPLEX, quadruple precision COMPLEX, derived type, or polymorphic is restricted.
 - Debugging of the dummy arguments of which type is CHARACTER whose length type parameter is not constant, array whose lower or upper bound is not constant, or assumed-shape array is restricted.
 - Debugging of the dummy arguments which are not referred in its procedure is restricted.
- Debugging of the local variables is not guaranteed.
- Debugging of the inlined procedures is restricted.
- The displaying of line number is not guaranteed.

Compile the program with the compiler option `-O0` when using debugger without the restrictions above.

A.2.2 Changing SVE Vector Register Size

This system assumes that the SVE vector register size is not changed during executing a program.

If the effective size of the SVE vector register is changed during executing a program using the system call `prctl(2)` or in other ways, the behavior is undefined.

If you want to change the SVE vector register size, you should set it before executing a program that uses SVE.

A.2.3 Integer Division Exception when Divisor Is Zero

Note that "Integer divide by zero" is not detected in the Fujitsu CPU A64FX with Arm architecture.

If the compiler option `-NRtrap` is specified and the divisor is 0, the program that expects "Integer divide by zero" to be detected may have different results.

It is recommended to check the value of the divisor just before the integer division.



Example

Example of checking the value of a divisor just before integer division

```
program main
  integer a,b,c
  a=1
  b=0
  if (b == 0) then
    error stop "Integer divide by zero"
  endif
  c = a/b
end program
```

A.2.4 Link Error (undefined reference to)

When compiling a program, if the following operation is performed by configure etc., a link error (undefined reference to) may occur because the combining of this system original objects that normally performed at linking is suppressed.

- The `-L` option which is passed to the linker by this system is directly specified on the compile command to pass the linker by user.



Example

Example of a link error

```
/opt/FJSVxos/devkit/aarch64/rfs/usr/lib64/crt1.o: In function `_start':
(.text+0x18): undefined reference to `main'
```

This can be avoided by setting the environment variable `FCOMP_LINK_FJOB`. Refer to "[2.3 Compile Command Environment Variable](#)" for the environment variable `FCOMP_LINK_FJOB`.

Appendix B Endian Convert Command

The endian convert commands `fcvendian(1)` and `fcvendianpx(1)` swap all bytes in type values within input file, and output the swapped values to the output file.

B.1 Command Format

The format of the `fcvendian(1)` command and `fcvendianpx(1)` command is as follows:

Command	Command argument
<code>fcvendian</code>	<code>_file1_file2_type</code>
<code>fcvendianpx</code>	

`_`: At least one blank is required.

`file1`: An input file with values to swap.

`file2`: An output file to output the swapped values.

`type`: A type of values to swap (The value which can be specified is 2, 4, 8, or 16).

If no command arguments are specified, usage of `fcvendian(1)` and `fcvendianpx(1)` is output.

Examples are as follows:

Example1: To byteswap a file "in.data" with 8 byte values.

```
$ fcvendian in.data out.data 8
```

Example2: To output the usage of the `fcvendian` command.

```
$ fcvendian
```

B.2 Return Value of `fcvendian(1)` and `fcvendianpx(1)`

The following table lists the return values set by the `fcvendian(1)` command and the `fcvendianpx(1)` command.

Return value	Status
0	Normal termination
1	<code>fcvendian(1)</code> or <code>fcvendianpx(1)</code> command error

B.3 Diagnostic Messages of `fcvendian(1)` and `fcvendianpx(1)`

If `fcvendian(1)` is not successful completion, the following diagnostics message is output to standard error file.

```
Usage: fcvendian infile outfile type
```

```
fcvendian: Invalid operand
```

```
fcvendian: Input file open error
```

```
fcvendian: Output file open error
```

```
fcvendian: Memory allocate error
```

If `fcvendianpx(1)` is not successful completion, the following diagnostics message is output to standard error file.

```
Usage: fcvendianpx infile outfile type
```

fcvendianpx: Invalid operand

fcvendianpx: Input file open error

fcvendianpx: Output file open error

fcvendianpx: Memory allocate error

B.4 Note

If the file which the type of values is mixed in is swapped, the byteswap may be not successful completion.

Appendix C Preprocessor

This appendix explains the C language preprocessor and the Fortran preprocessor.

When a suffix of the source file is .F, .FOR, .F90, .F95, .F03, or F08, or when the -Cpp option is specified, the preprocessor is executed. In this case, the -Ccpp or -Cfpp option can be specified to select the C language preprocessor or the Fortran preprocessor. The Fortran preprocessor is selected by default.

For the details of the preprocessor options, see Section "[2.2 Compiler Options](#)".

C.1 C Language Preprocessor

The C language preprocessor preprocesses in conformance with C language specification.

C.2 Fortran Preprocessor

The Fortran preprocessor gives more priority to the Fortran language specification than to the C language specification, and it preprocesses considering Fortran constructs, such as comment and continuation line indicator. Note the following:

- The Fortran constructs, such as continuation line indicator, comment, fixed and free source form and etc, are considered.
- The Fortran comment construct is also a target of macro substitution. However, the message is not output even if the error is in Fortran comment construct.
- The Hollerith constant and H edit descriptor are not considered as character contexts, and are the targets of macro substitution.
- The B, O, X, Z, b, o, x, and z which represent binary, octal and hexadecimal constants are not affected by preprocessing directives, and are not the targets of macro substitution.
- If a character context is to be continued in free source form, an '&' character must be the last nonblank character on the line and an '&' character must be the first nonblank character on the next line.
- Because the line number may be changed by the preprocessing of a Fortran source programs, be careful when this option is used. In this case, specify -P option and refer to the produced preprocessed output file.
- When the line exceeds limited column by the preprocessing directives and when -P option is specified, the control lines, such as "#line ! fpp start" and "#line !fpp end", may be produced in the produced preprocessed output file. When the preprocessed output file is specified as input file to the compiler, do not change those control lines because they are used by the compiler.

The following appendixes explain the features of the Fortran preprocessor with examples.

C.2.1 Continuation Line of Fortran

The continuation line of Fortran is considered.

- In fixed source form, column 6 indicates a continuation line. It is analyzed as the last character of the previous line continues to the column 7 of the continuation line.

```
Example: aaa is substituted to bbb.
#define aaa bbb
!23456789
    aa
    *a=1
```

- In free source form, the last character & and the first character & are analyzed as a character to indicate a continuation line. It is analyzed as the character before & continues to the character after &.

```
Example: aaa is substituted to bbb.
#define aaa bbb
aa&
    &a=1
```

C.2.2 Comment Line of Fortran

The comment line of Fortran is considered.

- The comments of C language and Fortran can be specified. The comment of C language form is substituted to blanks.

```
Example: The comments of C language and Fortran are effective.
/* comment of C language
   comment of C language */
! comment of Fortran
```

C.2.3 Comment of Fortran and Comment of C Language

- When the beginning of C language comment "/*" is specified in the Fortran comment, the beginning of the C language comment is not effective.

```
Example: The beginning of C language form comment is not
         effective.
! /* comment of Fortran
Not comment of C language */
```

- When the comment of Fortran is specified in the C language comment, the comment of Fortran is not effective.

```
Example: Comment of Fortran is not effective and assignment
         statement is effective.
/* ! */ kkk=1
```

C.2.4 Macro Substitution in Fortran Comment Construct

The macro substitution in Fortran comment construct is considered.

- The macros in Fortran comment construct are the targets of substitution. It is the specification that considers that the OpenMP prefix and the optimization control line and etc begin with comment character.

```
Example: aaa in Fortran comment construct is substituted to bbb.
#define aaa bbb
!ocl noarraypad(aaa)
      integer::aaa(100)
      aaa=10

! aaa=20
!$omp parallel shared(aaa)
      :
```

- A message is not output even if the macro substitution error is detected in the Fortran comment construct.

```
Example: A message is not output even if the macro substitution
         error is detected in comment.
#define a(i) b(2,i)
a(5,10)=1 ! message is output.
! a(5,10)=1 message is not output.
```

C.2.5 Treatment of Characters Specified after Column 72 in Fixed Source Form

In fixed source form, the characters specified after column 72 are ignored except comment and preprocessing directives. However, when -Fwide option is specified, columns 73 through 255 are effective.

```
Example: In fixed source form, the character C is specified at column 73,
         ABC is considered as AB, and ABC is not substituted to D.
         Specify -Fwide option in order to make columns 73 through 255
```



```
effective.
#define ABC D
!      1      2      3      4      5      6      7
!2345678901234567890123456789012345678901234567890123456789012
ABC
* = 100
print *,D
end
```

Appendix D GNU Compatible Options

This system partially supports the GNU Fortran compiler options (called "GNU compatible options" in this manual).

This appendix describes GNU compatible options supported by this system.

For more detail about these GNU compatible options, refer to the GCC website.

GNU compatible options supported by this system

`--print-file-name=include`

This option specifies to print the include directory.

`--print-prog-name={ as | ld | objdump | ranlib | ar }`

This option specifies to print the name of programs which are called by compile command.

`--shared`

This option makes the linker create shared objects and not dynamic link executable files.

`--version`

This option specifies to emit the version and copyright information of the compiler to the standard output.

`-cpp`

This option specifies that the C language preprocessor preprocesses Fortran source programs.

`-Xlinker option`

This option directs that the *option* is to be passed as arguments to the linker.

`-f{ align-loops[=M] | no-align-loops }`

The `-falign-loops` option specifies to align the loop to the boundary of power-of-two byte.

The `-fno-align-loops` option invalidates the `-falign-loops` option. `-fno-align-loops` is set by default.

`-ffp-contract=fast`

This option specifies to perform optimization using Floating-Point Multiply-Add/Subtract instructions.

When this option is set, side effects (calculation errors in rounding error extent) may occur in the execution results and produce unexpected results.

`-f{ inline-functions | no-inline-functions }`

The `-finline-functions` option specifies to perform the inline expansion for user-defined procedures. The user-defined function which is the subject of inline expansion is determined by the compiler automatically. The `-fno-inline-functions` option is set by default.

`-floop-parallelize-all`

This option specifies to perform automatic parallelization. However, if the effect of parallel execution is not expected, automatic parallelization is not performed.

If the object program compiled with this option is included in the filename list, this option must also be specified when linking.

`-flto`

This option specifies to perform the link time optimization. This option should be specified when programs are compiled and linked.

`-f{ omit-frame-pointer | no-omit-frame-pointer }`

These options specify whether or not to keep the frame pointer in a register from procedure calling. The `-fno-omit-frame-pointer` option is set by default.

`-fopenmp`

The `-fopenmp` option specifies to enable Specification of OpenMP Application Program Interface.

The `-fopenmp` option is needed if an object program compiled with the `-fopenmp` option exists in the command line as input files.

-f{ openmp-simd | no-openmp-simd }

The `-fopenmp-simd` specifies that only the OpenMP SIMD construct, the DECLARE SIMD construct, the DECLARE REDUCTION directive, and the ORDERED construct with the SIMD clause are effective.

`-fno-openmp-simd` is set by default.

-f{ optimize-sibling-calls | no-optimize-sibling-calls }

The `-foptimize-sibling-calls` option specifies to perform optimization of the sibling call. The `-fno-optimize-sibling-calls` option invalidates the `-foptimize-sibling-calls` option. `-fno-optimize-sibling-calls` is set by default.

-f{ pic | PIC }

This option specifies to generate position-independent code (PIC).

This option is effective at compilation time only.

-f{ pie | PIE }

This option specifies to generate position-independent code.

This option is effective at compilation time only.

-f{ plt | no-plt }

The `-fplt` option specifies to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC).

The `-fno-plt` option invalidates the `-fplt` option. `-fplt` is set by default.

-fprofile-dir=*path*

This option specifies storage location directory of the `.gcca` file which is necessary for the use of the code coverage. For *path*, the storage location directory name is specified by the relative path or the absolute path.

-f{ schedule-insns | no-schedule-insns }

The `-fschedule-insns` option specifies to perform the optimization of an instruction scheduling before the register is allocated. The `-fnoschedule-insns` option invalidates the `-fschedule-insns` option. `-fno-schedule-insns` is set by default.

-f{ schedule-insns2 | no-schedule-insns2 }

The `-fschedule-insns2` option specifies to perform the optimization of an instruction scheduling after the register is allocated. The `-fnoschedule-insns2` option invalidates the `-fschedule-insns2` option. `-fno-schedule-insns2` is set by default.

-funroll-loops

This option specifies to apply loop unrolling.

-f{ unsafe-math-optimizations | no-unsafe-math-optimizations }

The `-funsafe-math-optimizations` option specifies to use flush-to-zero mode. The default is `-fno-unsafe-math-optimizations`.

When `-funsafe-math-optimizations` option is specified, side effects may occur in the execution results.

These options must be set at linking.

-g{ dwarf | dwarf-4 }

This option specifies to add DWARF4 debugging information to the object file.

This option is equivalent to the `-g` option.

-isystem *dir*

This option adds the *dir* directory to the search paths to search INCLUDE files or module information files.

If multiple directories are specified in multiple `-isystem` options, the directories are searched in the specified order.

If a file is specified with an absolute path name, only the specified absolute path name is searched.

If the specified directory does not exist, this option is invalidated.

-march=*arch*[+*features*]

This option specifies the architecture.

In *archi*, specifies one of `armv8-a`, `armv8.1-a`, `armv8.2-a`, or `armv8.3-a`.

For *features*, specifies sve or nosve. The default for *features* is sve.

If `-march` option is omitted, `-march=armv8.3-a+sve` is set.

`-mcmmodel={ small | large }`

This option specifies the possible largest size of the text area and the static data area in an executable program or a shared object. `-mcmmodel=small` is set by default.

`-mcmmodel=small`

The total size of the text area and the static data area is limited to 4GB at linking. This option creates an efficient object program.

`-mcmmodel=large`

Only the size of the text area is limited to 4GB at linking. This option is used when the static data area is large and an error occurs at linking.

`-mcpu=cpu`

This option specifies the processor which optimizations targets. In *cpu*, specifies one of a64fx, thunderx2t99, or generic. `-mcpu=a64fx` is set by default.

`-m{ pc-relative-literal-loads | no-pc-relative-literal-loads }`

The `-mpc-relative-literal-loads` option specifies to access the literal pool by one instruction assuming that the code area in the function is within 1MB.

The `-mno-pc-relative-literal-loads` option invalidates the `-mpc-relative-literal-loads`. `-mno-pc-relative-literal-loads` is set by default.

`-mtls-size={ 12 | 24 | 32 | 48 }`

This option specifies the size of an offset necessary for the access to Thread-Local Storage. Units are bits.

`-mtune=cpu`

This option specifies the processor which optimizations targets. In *cpu*, specifies one of a64fx, thunderx2t99, or generic. `-mtune=a64fx` is set by default.

`-pthread`

This option specifies to create thread safe objects by using the POSIX thread library.

`-rdynamic`

This option specifies to the linker to add all symbols to the dynamic symbol table.

This option passes the `-export-dynamic` option to the linker.

`-v`

This option specifies to display the executing command which compile command called as compiler, assembler and linker.

`-w`

This option suppresses the output of warning messages.

Appendix E Data and Memory Regions

This appendix describes attribute of the data and memory regions in this system.

E.1 Data Size and Alignment

Data type, kind type parameter, size, and alignment of basic data are shown below.

Data Type	Kind Type Parameter	Size (byte)	Alignment (byte)
integer	1	1	1
	2	2	2
	4	4	4
	8	8	8
real	4	4	4
	8	8	8
	16	16	16
double precision	-	8	8
complex	4	8	4
	8	16	8
	16	32	16
logical	1	1	1
	2	2	2
	4	4	4
	8	8	8
character	1	-	1
byte	-	-	1
derived type	-	-	The largest value among the maximum alignment of the member variables and 8 byte.

E.2 Memory Regions

The main usages of the memory regions are shown below.

Memory regions	Usage
Text	Memory regions where instructions are arranged.
Static data (.data)	Memory regions where static data with initialization is stored.
Static data (.bss)	Memory regions where static data without initialization is stored.
Process stack	Stack regions for the process and the main thread.
Thread stack	Stack regions for the child process.
Dynamically allocated memory region	Memory regions allocated when requiring memory to be allocated dynamically, or memory regions allocated as thread heap region or thread stack
Shared memory	Memory regions shared among processes

E.3 Data Allocation

Variables in a program are arranged to different memory regions depending on each type of the variables and existence of initial values.

Example

```
module mod
  integer(kind=4),dimension(10) :: f = 3
end module mod

program main
use mod
integer(kind=4),parameter :: n = 10
integer(kind=4),dimension(10) :: a
integer(kind=4),dimension(10) :: b = 1
real(kind=4),dimension(10) :: c
real(kind=4),dimension(10) :: d = 2.0
save c,d
common /com1/e
integer(kind=4),dimension(10) :: e
real(kind=4),allocatable,dimension(:) :: g
allocate(g(n))
```

In the above example, each array is allocated as follows.

Array name	Region used
a	Allocated to the process stack region, because it is a local arrays and not initialized.
b	Allocated to the static data (.data) region, because it is a local arrays and initialized.
c	Allocated to the static data (.bss) region, because it is a static array and not initialized.
d	Allocated to the static data (.data) region, because it is a static array and initialized.
e	Allocated to the static data (.bss) region, because it is a global array and not initialized.
f	Allocated to the static data (.data) region, because it is a global array and initialized
g	Allocated to the dynamically allocated memory region, because it is an allocatable array.

E.4 Data Allocation to Stack Region

In this processing system, the order of data allocation to the stack region can be controlled by the compiler option.

The compiler option that controls the order of data allocation is shown as follows.

`-N{ reordered_variable_stack | noreordered_variable_stack }`

The `-Nreordered_variable_stack` option directs the compiler the order in which to allocate the automatic variables to the stack area.

If the `-Nreordered_variable_stack` option is set, the allocation order of automatic variables is determined in the following order.

1. Descending order of their alignments
2. Descending order of data sizes if the alignments are equal
3. The order of appearance of the declaration statements in the source program if both the alignments and data sizes are equal

The stack area of the entire program can be reduced by allocating the automatic variables in descending order of their alignments.

When the `-Nnoreordered_variable_stack` option is specified, automatic variables are allocated in order of the appearance of the declaration statements in the source program.

The default is `-Nnoreordered_variable_stack`.

The order of allocation is not guaranteed when the `-Nnoline`, `-g0`, or `-Kocl` option is set.

Example

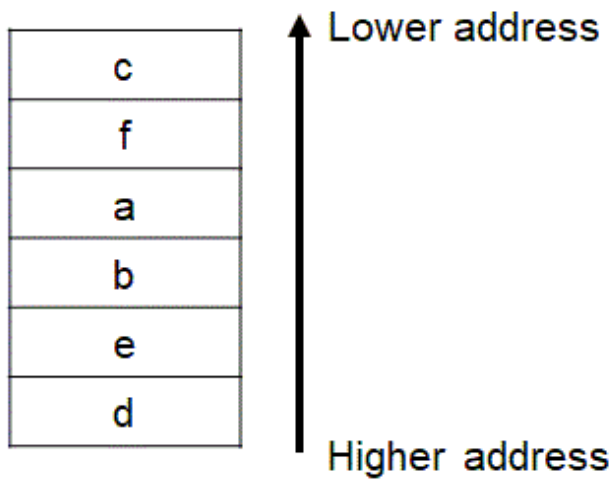
- Source Program

```
integer(kind=4) ::a
real(kind=8) ::b
character ::c
real(kind=8),dimension(2) ::d,e
integer(kind=2) ::f
a = 1
b = 2.0
c = "3"
d = 4.0
e = 5.0
f = 6
stop
end
```

- When specifying the order of data allocation for the example source program

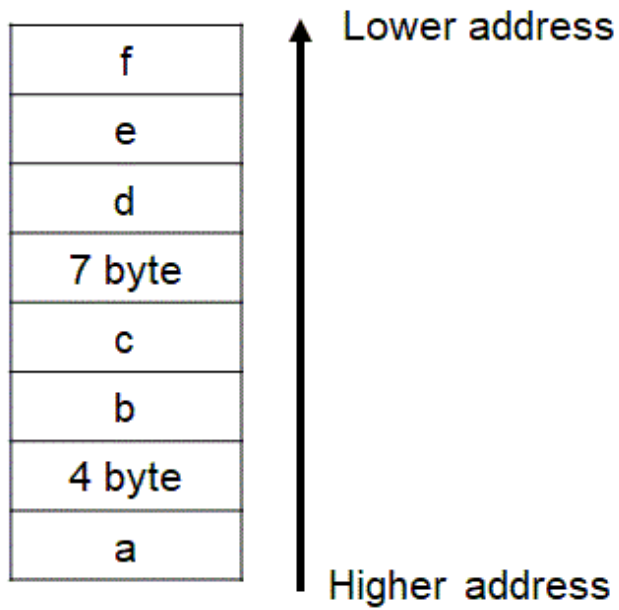
Automatic variables are stored in the stack as shown below. The stack region can be decreased.

Figure E.1 When the `-Nreordered_variable_stack` option is valid



- When not specifying the order of data allocation for the example source program Automatic variables are stored in the stack as shown below.

Figure E.2 When the -Nnoreordered_variable_stack option is valid



Appendix F Intrinsic Procedures and I/O Statements Compatible with Half-Precision Type

F.1 Intrinsic Procedures

The table below shows the intrinsic procedures that are compatible with the half-precision REAL type and half-precision COMPLEX.

Generic name	Half precision REAL	Half precision COMPLEX
ABS	Available	Available
AIMAG	Not available	Available
ALLOCATED	Available	Available
ASSOCIATED	Available	Available
CMPLX	Available	Available
CSHIFT	Available	Available
DCMPLX	Available	Available
DBLE	Available	Available
DIGITS	Available	Not available
DOT_PRODUCT	Available	Available
EOSHIFT	Available	Available
EPSILON	Available	Not available
HUGE	Available	Not available
INT	Available	Available
IS_CONTIGUOUS	Available	Available
KIND	Available	Available
LBOUND	Available	Available
LOC	Available	Available
MATMUL	Available	Available
MAX	Available	Not available
MAXEXPONENT	Available	Not available
MAXVAL	Available	Not available
MIN	Available	Not available
MINEXPONENT	Available	Not available
MINVAL	Available	Not available
MOVE_ALLOC	Available	Available
NORM2	Available	Not available
PACK	Available	Available
PRECISION	Available	Available
PRESENT	Available	Available
PRODUCT	Available	Available
QCMLX	Available	Available
QEXT	Available	Available

Generic name	Half precision REAL	Half precision COMPLEX
RADIX	Available	Not available
RANGE	Available	Available
REAL	Available	Available
RESHAPE	Available	Available
SHAPE	Available	Available
SIGN	Available	Not available
SIZE	Available	Available
SIZE_OF	Available	Available
SPACING	Available	Not available
SPREAD	Available	Available
STORAGE_SIZE	Available	Available
SUM	Available	Available
TINY	Available	Not available
TRANSFER	Available	Available
UBOUND	Available	Available
UNPACK	Available	Available
VAL	Available	Available

F.2 Input/Output Statements

The table below shows the input/output statements that are compatible with the half-precision REAL type and half-precision COMPLEX.

Input/output statement	Half precision REAL	Half precision COMPLEX
Unformatted sequential access READ statement	Available	Available
Unformatted sequential access WRITE statement	Available	Available
Unformatted direct access READ statement	Available	Available
Unformatted direct access WRITE statement	Available	Available
Unformatted stream access READ statement	Available	Available
Unformatted stream access WRITE statement	Available	Available
Output list INQUIRE statement	Available	Available

Appendix G Code Coverage

This appendix explains the code coverage.

The code coverage is the function to measure the execution frequency of the executable statement at execution, and to examine the code coverage rate of the program.

The code coverage uses the code coverage tool based on LLVM compiler infrastructure (hereafter called llvm-cov), which is an open source software.

Refer to an online manual of the llvm-cov command by the man command for the use of the llvm-cov command.

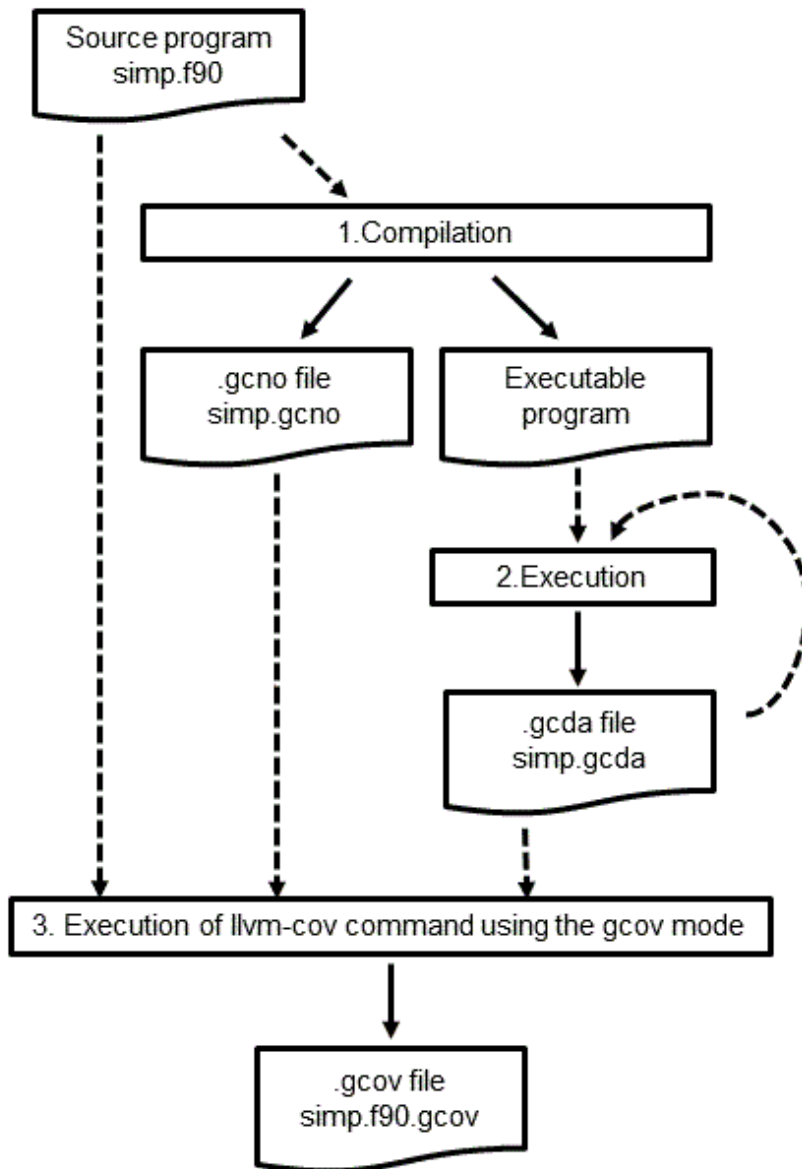
G.1 How to Use Code Coverage

The code coverage is used the following procedures.

1. Compilation
2. Execution
3. Execution of llvm-cov command using the gcov mode

A following figure shows the procedure of the code coverage.

Figure G.1 Procedure of the code coverage



G.2 Necessary Files to Use Code Coverage

This section explains the necessary files to use the code coverage.

G.2.1 The ".gcno" File

The .gcno file is a binary file which contains the information obtained at compilation.

File generation step	The file is generated at compilation
File name	The file name is the same as the source program with file extension .gcno.
File generated location	The .gcno file is generated in the directory where the compilation is performed.

An example of generating .gcno file is shown as follows.

Example

```
$ frt -Ncoverage ../simp.f90
$ ls
a.out simp.gcno
```

When the assembler source file and the object file are generated with specifying the `-o` option, the `-o` option is also applied to the `.gcno` file. An example of generating files with specifying the `-o` option is shown as follows.

Example

```
$ frt -Ncoverage ../simp.f90 -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
```

G.2.2 The ".gcda" File

The `.gcda` file is a binary file which contains the information obtained at execution.

File generation step	The file is generated at execution. The execution frequency is accumulated when the <code>.gcda</code> file with the same name already exists.
File name	The file name is the same as the source program with file extension <code>.gcda</code> .
File generated location	The <code>.gcda</code> file is generated in the directory where the compilation is performed.

An example of generating `.gcda` file is shown as follows.

Example

```
$ frt -Ncoverage ../simp.f90
$ ls
a.out simp.gcno
$ ./a.out
$ ls
a.out simp.gcda simp.gcno
```

When the assembler source file and the object file are generated with specifying the `-o` option, the `-o` option is also applied to the `.gcda` file. An example of generating files with specifying the `-o` option is shown as follows.

Example

```
$ frt -Ncoverage ../simp.f90 -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
$ frt -Ncoverage www/yyy.o
$ ./a.out
$ ls www/
yyy.gcda yyy.gcno yyy.o
```

The storage location directory of the `.gcda` file can be changed by specifying the `-Nprofile_dir=dir_name` option.

An example of generating files with specifying the `-Nprofile_dir=dir_name` option is shown as follows.



Example

```
$ frt -Ncoverage ../simp.f90 -c -o www/yyy.o -Nprofile_dir=/tmp
$ ls www/
yyy.gcno yyy.o
$ frt -Ncoverage www/yyy.o
$ ./a.out
$ ls www/
yyy.gcno yyy.o
$ ls /tmp/www/
yyy.gcda
```

G.3 Notes on Code Coverage

This section explains notes of the code coverage.

- The execution performance may decrease because the instructions that measure the execution frequency are added in the object program.
- The storage directory of the .gcda file, which is generated at execution, is determined at compilation.
 - The location of the .gcda file storage directory does not change even if the executable program is moved after compilation.
 - Use the `-Nprofile_dir=dir_name` option at compilation to change the location of the .gcda file storage directory.
- The code coverage function is not available for programs that need the `-Kcmodel=large` option.
- The execution frequency may be measured incorrectly in the following cases:
 - Two or more executable statements are included in one line.
 - STOP statement, ERROR STOP statement, EXIT service subroutine, or SETRCD service subroutine is called.
 - An exception is caught.
 - The `#line` directive is included in the source program.
 - Optimizations of the compiler are applied.

Appendix H Runtime Information Output Function

This appendix describes the Runtime Information Output Function.

This function outputs information for speedup and performance confirmation of the program, by specifying the compiler option and setting the environment variable, when the program is executed.

H.1 Usage of Runtime Information Output Function

This section describes the usage of Runtime Information Output Function.

H.1.1 Range of Obtainable Information

This section describes the range of information that can be fetched using this function.

Table H.1 Range of obtainable information

Compiler Options	Runtime environment variables	Range of obtainable information
-Nrt_tune	FLIB_RTINFO	from the start of the program to the end
-Nrt_tune_func		each user-defined procedure
-Nrt_tune_loop		each loop

When the -Nrt_tune compile option and FLIB_RTINFO runtime environment variable is specified, this function is enabled and information can be fetched from the start of the program to the end.

Also, if the -Nrt_tune_func compile option or -Nrt_tune_loop compile option is specified, additional information concerning each user-defined procedure, each loop can be fetched.

Refer to "2.2 Compiler Options" for further details.

If the MPI Programs, information on the rank 0 is output. If the COARRAY Programs, information on the image 1 is output.

Each User-defined Function

When the -Nrt_tune_func compile option is specified, information for each user-defined procedure is output.



Note

- If a procedure is called from within a procedure:
Information for the called procedure is not included in the calling source procedure information.
- If the same procedure is called from multiple locations :
The total values are used as the information for that procedure.
- Only user-defined procedures are output. Information concerning system calls and library procedures is not fetched.

Each Loop

When the -Nrt_tune_loop compile option is specified, the following information for each loop is output.

- Information for each loop operates in serial processing.
- Information for each loop operates in PARALLEL region by OpenMP.

The cost of the serial loop is output as information for serial loop. If the loop exists in PARALLEL region by OpenMP, it is output as each loop in PARALLEL region by OpenMP. As the information of loop in PARALLEL region by OpenMP, information of the master thread is output.



- If a procedure is called from within a loop :
Information for the called procedure is included in the calling source loop information.
- If a loop is called from within a loop :
Information for the called loop is included in the calling source loop information.

H.1.2 Runtime Environment Variables

This section describes the runtime environment variables using this function.

The English small letter and the English capital letter specified for the value of the environment variable at execution time are distinguished. For environment variables that do not require operands, the specified operand value is ignored.

Table H.2 Environment variables that control the information output

Runtime environment variables		Explanation
Variable name	Operand	
FLIB_RTINFO	(None)	Outputs runtime information.
FLIB_RTINFO_NOFUNC	(None)	Even if the compiler option -Nrt_tune_func is effective, the output of information for each procedure is suppressed.
FLIB_RTINFO_NOLOOP	(None)	Even if the compiler option -Nrt_tune_loop is effective, the output of information for each loop is suppressed.
FLIB_RTINFO_CSV	Filename	Outputs the fetched information to a CSV file.
	-	Operand is optional. Any file name can be specified as operand of the environment variable. If operand is omitted, flib_rtinfo.csv is assumed.

H.2 Output of Runtime Information Output Function

This section describes the output of Runtime Information Output Function.

H.2.1 Output Information

This section describes the information output by this function.

Table H.3 Output information

Compiler Options	Runtime environment variables	Output information
-Nrt_tune	FLIB_RTINFO	The following information from the start of the program to the end is output <ul style="list-style-type: none"> - Elapsed time - User CPU time - System CPU time
-Nrt_tune_func		In addition to the -Nrt_tune output, following Information of each procedure is output <ul style="list-style-type: none"> - Procedure cost - Percentage of total cost accounted for by the procedure cost - Procedure cost per called once - Procedure start line number

Compiler Options	Runtime environment variables	Output information
		<ul style="list-style-type: none"> - Procedure end line number - Call frequency of procedure - Procedure name
-Nrt_tune_loop		<p>In addition to the -Nrt_tune output, following Information of each loop in serial and each loop that in PARALLEL region by OpenMP is output</p> <ul style="list-style-type: none"> - Loop cost - Percentage of total cost accounted for by the loop cost - Loop cost per called once - Loop start line number - Loop end line number - Call frequency of Loop - Loop nest level - Generated procedure name

H.2.2 Output Format

Runtime information can be output by the text format or CSV format.

Text format

When FLIB_RTINFO_CSV environment variable is not specified, information is output to the standard output by text format.

Refer to "[H.2.3 Output Example](#)" for details of output format of information.

CSV format

When the FLIB_RTINFO_CSV environment variable is specified, information is output in CSV format. If a filename is specified in the operands, the fetched information is output to the specified file. If the specified file already exists, the information overwrites the existing file. If the file fails to open, runtime message jwe1653i-w is output and the information is output to the standard output.

When runtime information is output by CSV, data line follows the following compositions. When there is no output information, "-" is output.

Table H.4 Composition of CSV format

Column position	Output information
First	Identifier that shows type [COST] : Information for entire program [COST_ROUTINE] : Information for each procedure [COST_LOOP_SERIAL] : Information for each loop that operates in serial processing [COST_LOOP_PRL] : Information for each loop that in PARALLEL region by OpenMP
Second and subsequent	It is depending on output information.

Refer to "[H.2.3 Output Example](#)" for details of output format of information.

H.2.3 Output Example

Output example of Runtime Information Output Function is shown below.

Output example of runtime information

```

+=====+
| | EXECUTION PERFORMANCE INFORMATION | |
+=====+

+-----+-----+-----+
| Elapsed | User | System |
| (sec) | (sec) | (sec) |
+-----+-----+-----+
| 0.8820 | 0.8909 | 0.0030 |
+-----+-----+-----+

Routine (4)
+-----+-----+-----+-----+-----+-----+-----+
| Cost(sec) | % | Once(sec) | Start | End | Count | Routine name |
+-----+-----+-----+-----+-----+-----+-----+
| 0.7790 | 98.94 | 0.0779 | 28 | 37 | 10 | sub1 | (12)
| 0.0084 | 1.06 | 0.0000 | 39 | 41 | 2000 | sub2 |
+-----+-----+-----+-----+-----+-----+-----+
| 0.7874 | 100.00 | - | - | - | - | - | (13)
+-----+-----+-----+-----+-----+-----+-----+

Serial Loop (14)
+-----+-----+-----+-----+-----+-----+-----+-----+
| Cost(sec) | % | Once(sec) | Start | End | Count | Nest | Routine name |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0.8668 | 55.53 | 0.8668 | 15 | 17 | 1 | 1 | sample1 | (16)
| 0.6826 | 43.73 | 0.0683 | 29 | 31 | 10 | 2 | (sample1.sub1_) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1.5609 | 100.00 | - | - | - | - | - | - | (17)
+-----+-----+-----+-----+-----+-----+-----+-----+

Parallel Loop (18)
+-----+-----+-----+-----+-----+-----+-----+-----+
| Cost(sec) | % | Once(sec) | Start | End | Count | Nest | Routine name |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0.1841 | 50.28 | 0.0184 | 33 | 35 | 10 | 2 | (sample1.sub1._OMP_2_) |
| 0.1820 | 49.72 | 0.0000 | 34 | 34 | 10000 | 3 | (sample1.sub1._OMP_2_) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0.3661 | 100.00 | - | - | - | - | - | - |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

- (1) Elapsed time
- (2) User CPU time
- (3) System CPU time
- (4) Information for each procedure that operates in serial processing (-Nrt_tune_func is specified)
- (5) Cost (sec)
- (6) Percentage of the entire cost
- (7) Cost called once
- (8) Start line number
- (9) End line number
- (10) Call frequency
- (11) Procedure name
- (12) Information for each procedure
- (13) Total value for each procedure

(14) Information for each loop that operates in serial processing (-Nrt_tune_loop is specified)

(15) Nest level

(16) Information for each loop

(17) Total value for each loop

(18) Information for each loop that in PARALLEL region by OpenMP (-Nrt_tune_loop is specified)

Note: When information is fetched for each loop, the generated procedure name is output as the procedure name. The procedure name is enclosed with parentheses at the loop that is nested.

Output example of runtime information (environmental variable FLIB_RTINFO_CSV is specified)

```
=====
EXECUTION PERFORMANCE INFORMATION
=====
Type,Elapsed(sec),User(sec),System(sec)
[COST],0.8944,1.7747,0.0040

Routine
Type,Cost(sec),%,Once(sec),Start,End,Count,Routine name
[COST_ROUTINE],0.7885,98.92,0.0788,28,37,10,"sub1"
[COST_ROUTINE],0.0086,1.08,0.0000,39,41,2000,"sub2"
[COST_ROUTINE],0.7971,100.00,-,-,-,-,-

Serial Loop
Type,Cost(sec),%,Once(sec),Start,End,Count,Nest,Routine name
[COST_LOOP_SERIAL],0.8745,55.41,0.8745,15,17,1,1,"sample1"
[COST_LOOP_SERIAL],0.6924,43.87,0.0692,29,31,10,2,"(sample1.sub1_)"
[COST_LOOP_SERIAL],0.0883,100.00,-,-,-,-,-

Parallel Loop
Type,Cost(sec),%,Once(sec),Start,End,Count,Nest,Routine name
[COST_LOOP_PRL],0.1820,50.38,0.0182,33,35,10,2,"(sample1.sub1._OMP_2_)"
[COST_LOOP_PRL],0.1793,49.62,0.0000,34,34,10000,3,"(sample1.sub1._OMP_2_)"
[COST_LOOP_PRL],0.7971,100.00,-,-,-,-,-
```

H.3 Notes on Runtime Information Output Function

This section describes the notes on Runtime Information Output Function.

- If the information is fetched each procedure or loop by specifying -Nrt_tune_func or -Nrt_tune_loop, fetched information is increases, and execute time may increase. If the information cannot be obtained, the diagnostic message jwe1655i-i may be output. If this diagnostic message is output during error processing, the order in which the diagnostic messages are output may be incorrect.
- When there are GOTO statement, STOP statement during a program or an exception is checked, information may be not correctly fetched.
- There is a possibility of receiving the following influences by optimization of the compiler.
 - When fetched the information for each loop, it may be different which loop is worked as a target of the measurement.
 - Line number may not accord with line number of the source program.
 - The nest relations of the loop may not accord with a nest of the source program. By the following optimization, I may not recognize loop information definitely. By the following optimization, it may not be correctly recognized loop information.
 - inline expansion for procedures
 - loop unrolling
 - loop blocking
 - loop interchange
 - loop fusion

- loop striping
- loop splitting
- loop peeling
- loop unswitching
- change the function to a multi-operation function, use SIMD instructions

Appendix I Using High-Speed Facility on Job Operation Software

On the job of Job Operation Software, the high-speed facility (the inter-core hardware barrier and the Sector cache) can be used.

This appendix provides information on the compilation and execution processes for using the high-speed facility.

See manuals of Job Operation Software for details about how to execute and submit a job.

I.1 Inter-Core Hardware Barrier

The inter-core hardware barrier (hereafter called hardware barrier) is a hardware mechanism which facilitates high speed synchronization among threads in a CPU Chip, and raises the execution performance. This section describes compiling and executing programs to take advantage of hardware barriers using the LLVM OpenMP library.

I.1.1 Compilation

When generating a program using the hardware barrier as the thread barrier, specify `-Kparallel` or `-Kopenmp` option.

I.1.2 Execution

FLIB_BARRIER

The barrier-kind of OpenMP and Automatic Parallelization can be controlled using the environment variable `FLIB_BARRIER`.

Allowable values and their meanings are as follows:

HARD

The hardware barrier is used.

When the hardware barrier cannot be used, the diagnostic message (jwe1050i-w) is output, and the processing is continued using the software barrier.

When the hardware barrier is used, the part or everything of the following OpenMP functions, cannot be used. When these OpenMP functions are used, the diagnostic message (jwe1051i-w) is output, and these functions are ignored, the processing is continued using the software barrier.

- The controlling number of threads

The number of threads will be determined using the following priority.

1. The value specified in the environment variable `OMP_NUM_THREADS`
2. The limit on the number of CPUs (The number of CPUs assigned to the process by the job)

When the value specified in the environment variable `OMP_NUM_THREADS` exceeds the limit on the number of CPUs, the number of threads is the limit on the number of CPUs.

The number of threads specified with the `NUM_THREADS` clause or the `OMP_SET_NUM_THREADS` routine must be the same as defined by the execution environment.

Irrespective of the environment variable `OMP_DYNAMIC` and the `OMP_SET_DYNAMIC` routine, the number of threads cannot exceed the limit on the number of CPUs.

- The controlling thread affinity

The thread affinity is set using CPUs assigned to the process by the job.

Irrespective of the environment variable `OMP_PROC_BIND` and the `PROC_BIND` clause of `PARALLEL` construct, OpenMP thread affinity is ignored.

When the thread affinity is set using other ways, the operation is not guaranteed.

- Nested parallelism.

Irrespective of the environment variable `OMP_NESTED` and the `OMP_SET_NESTED` routine, nested parallel regions are always serialized.

- Task

Always, the undeferred task is generated.

- Cancellation

Irrespective of the environment variable `OMP_CANCELLATION`, the `CANCEL` construct and cancellation points are effectively ignored.

SOFT

The software barrier is used. (default)

Note

- When the number of threads is 1, the hardware barrier cannot be used.
- The hardware barrier in FX system is used only inside the NUMA node. When one process is allocated to multiple NUMA nodes, hardware barrier is used inside NUMA node and software barrier is used between NUMA nodes. Therefore, it is recommended to allocate one process to one NUMA node, in case of giving priority to the performance of threads barrier.
- On the following cases, the high speed runtime library cannot determine whether one process can reserve a Barrier resource or not. Therefore behavior of hardware barrier is indeterminate, program may be ended with diagnostic-message (jwe1044i-u or jwe1045i-u). or the program may terminate abnormally.

- When Process is generated by functions other than Fujitsu MPI

Example:

- FORTRAN FORK/SYSTEM/SH service function is used.
- C fork system call/system function is used.

Example

Example 1: A program that uses the hardware barrier is executed on a thread parallel processing job.

```
$ cat job.sh
#!/bin/sh
export FLIB_BARRIER=HARD
./a.out
$ pjsub -L "node=1" job.sh
```

Example 2: A program that uses the hardware barrier is executed on a hybrid (process and thread) parallel processing job (One job is executed in one dimension: The number of nodes is set to the number of processes.).

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=12"
#PJM --mpi "shape=12"
#PJM --mpi "rank-map-bynode"
export FLIB_BARRIER=HARD
mpiexec -n 12 a.out
$ pjsub job.sh
```

I.2 Sector Cache

See Section "9.17 Software Control of Sector Cache" for details on how to control the Sector cache.

This section explains only notes on execution.

Notes on execution

The Sector cache for the primary and secondary level caches are available in the following execution environments.

The execution environments	The Sector cache for the first level cache	The Sector cache for the second level cache
Only one process runs on a NUMA node	Available	Available
Multiple processes run on a NUMA node	Available (*1)	Not available

*1) Not available if the environment variable FLIB_L1_SCCR_CNTL is assigned to the value FALSE. See Section "[9.17.2.2 Software Control with Environment Variables and Optimization Control Lines](#)" for details.

On the following cases, the high speed runtime library cannot determine whether one process can reserve a NUMA node or not. Therefore behavior of Sector cache is indeterminate. Performance of program may be reduced, program may be ended with diagnostic-message (jwe1048i-u), or the program may terminate abnormally.

- Process is generated by functions other than Fujitsu MPI.

Example:

- FORTRAN FORK/SYSTEM/SH service function is used.
- C fork system call/system function is used.
- Threads are generated by functions other than Fujitsu Automatic Parallelization/OpenMP.

Example:

- Threads are controlled by pthread functions.

To invalidate the Sector cache control function unconditionally regardless of the execution environment, "FALSE" is set for the environment variable FLIB_SCCR_CNTL.

See Section "[9.17.2.2 Software Control with Environment Variables and Optimization Control Lines](#)" for details.

Appendix J Fujitsu OpenMP Library

This appendix describes how parallel processing is performed for a Fortran program using Fujitsu OpenMP Library.

When using LLVM OpenMP Library, see "[Chapter 12 Multiprocessing](#)".

Note

If the following condition is met, this system uses Fujitsu OpenMP Library.

- `-Nfjomplib` option is effective at linking

Information

The table below shows a connectable combination of library for multiprocessing (LLVM OpenMP Library and Fujitsu OpenMP Library) and connectable object file.

See "C User's Guide" or "C++ User's Guide" about Trad Mode and Clang Mode.

	Fortran object file	C/C++ object file	
		Trad Mode (<code>-Nnoclang</code> option)	Clang Mode (<code>-Nclang</code> option)
LLVM OpenMP Library (<code>-Nlibomp</code> option)	Available	Available	Available
Fujitsu OpenMP Library (<code>-Nfjomplib</code> option)	Available	Available	Not available

J.1 Overview of Multiprocessing

See Section "[12.1 Overview of Multiprocessing](#)".

J.2 Automatic Parallelization

This section explains the automatic parallelization provided by this compiler.

J.2.1 Compilation (Automatic Parallelization)

For compiling source programs and linking, specify the following options to the compile command.

- Compile time

```
-Kparallel
```

- Link time

```
-Kparallel -Nfjomplib
```

Compiler Option for Automatic Parallelization

The meaning and format of the automatic parallelization options are described below. See Section "[2.2 Compiler Options](#)" for more information on each option.

```
-K{ parallel | parallel_strong | visimpact } [,array_private,dynamic_iteration,independent=pgm_nm,  
instance=N,loop_part_parallel,ocl,optmsg=2,parallel_fp_precision,parallel_iteration=N,  
reduction,region_extension] -Nfjomplib
```


J.2.2 Execution Process (Automatic Parallelization)

When running a program compiled with automatic parallelization, the number of threads can be specified using the environment variable `PARALLEL` or `OMP_NUM_THREADS`. The stack area size for each thread can be specified using the environment variable `THREAD_STACK_SIZE` or `OMP_STACKSIZE`. It is also possible to specify a synchronization scenario using the environment variable `FLIB_SPINWAIT` or `OMP_WAIT_POLICY`.

The other procedures are the same as those used in serial processing.

Environment Variable `PARALLEL`

The environment variable `PARALLEL` specifies the number of threads performed at the same time. See "[Number of Threads](#)" for details on determining the number of threads.

Number of Threads

The priority of the factors that determine the number of threads executed concurrently is as follows.

1. The value specified in the environment variable `PARALLEL`
2. The value specified in the environment variable `OMP_NUM_THREADS`
3. Number of CPUs that can be used with jobs
4. 1 thread



Value of `PARALLEL`

If the `-Kopenmp` option has been specified at the linkage phase of the compilation and the environment variable `FLIB_FASTOMP` is `TRUE`, the value of the environment variable `PARALLEL` must be the same as the value of the environment variable `OMP_NUM_THREADS`. When the value does not match, a message will be output, and smaller value will be used for the number of threads.

The number determined above is compared with the available maximum CPU number, and smaller number is used as the number of threads.

Refer to "[Number of CPUs that can be used on the job](#)" in Section "[J.5.1 Management of CPU Resources](#)" for information on the number of CPUs that can be used with jobs.

The maximum number of CPUs is determined in the following way.

For jobs	Number of CPUs that can be used with jobs
Not for job	Number of CPUs in the system

See "[Environment Variable for OpenMP Specifications](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for information on the environment variable `OMP_NUM_THREADS`.

Environment variables `THREAD_STACK_SIZE` and `OMP_STACKSIZE`

`THREAD_STACK_SIZE`

Specify the stack size (in Kbytes) for each thread using the environment variable `THREAD_STACK_SIZE`.

When the environment variable `OMP_STACKSIZE` is also specified, the greater value of the two is used as the stack area size for each thread.

`OMP_STACKSIZE`

The stack area size for each thread can be specified using the environment variable `OMP_STACKSIZE` in byte, Kbytes, Mbytes, or Gbytes.

When the environment variable `THREAD_STACK_SIZE` is also specified, the greater value of the two is used as the stack area size.

See Section "[Stack Size on Execution](#)" for information on stack area.

Stack Size on Execution

Local scalar variables within a parallelized DO loop (except the automatic data object when the compiler option `-Knoautoobjstack` is in effect) are allocated in the stack area for each thread. Allocate sufficient stack area, when there are many variables of this type.

When the stack area size for each thread needs to be specified, specify the environment variable `THREAD_STACK_SIZE` or `OMP_STACKSIZE`.

When the environment variable is not specified, the stack size for each thread is the same as the stack size for the process.

However, if the maximum stack size for the process is larger than the value calculated by the expression "min(the size of available memory in this system, the size of a virtual memory)/(the number of threads)", the system will determine the stack area size and allocate it as follows:

```
the stack size for each thread (byte) =
(min(the size of available memory in this system, the size of a virtual memory) / (the number of
threads)) / 5
```

However, the assumed value does not guarantee that the system works correctly. It is recommended that an appropriate value for maximum process stack size be specified manually. The maximum process stack area size can be specified using the bash built-in command "ulimit". Note that the maximum virtual memory size can also be confirmed using ulimit (bash built-in command).

See "Number of Threads" in Section "J.2.2 Execution Process (Automatic Parallelization)" for information about the number of threads.



Note

Some types of pthread library do not support setting the size of the stack area because of fixed stack. Static types of pthread library tend to be fixed stack, so ensure that you use a pthread library that is of a shared type.

Process of Waiting Threads

The standby threads in a serial process can be controlled with the environment variable `FLIB_SPINWAIT` or `OMP_WAIT_POLICY`.

FLIB_SPINWAIT

unlimited	Uses spin wait. The default is unlimited.
0	Uses suspending wait.
<i>n</i> s	Uses spin wait until after <i>n</i> seconds, and uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "s" is specified following <i>n</i> .
<i>n</i> ms	Uses spin wait until after <i>n</i> milliseconds, and uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "ms" is specified following <i>n</i> .

Spin waiting is a technique to wait while consuming CPU. Suspend waiting is a technique to wait without consuming CPU. To give priority to elapsed time, specify unlimited, as spin waiting requires less overhead for parallel process. Select 0 to give priority to the CPU time.

OMP_WAIT_POLICY

ACTIVE	Uses spin wait. The default value is ACTIVE.
PASSIVE	Uses suspending wait.

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

The settings of environment variable `FLIB_SPINWAIT` will be valid when environment variable `FLIB_SPINWAIT` is specified.

Notes when setting time limits

Operation may not be correct when time limits are set with the `ulimit(1)` command. If operation is not correct, use the Fortran exception handling process, the runtime option `-Wl, -i` to invalidate it. Alternatively, use the runtime option `-Wl, -t` to set the time limit.

Notes when using service function and intrinsic subroutine

See Section "[12.2.2.4 Notes when Using Service Function and Intrinsic Subroutine](#)".

CPU binding for thread

CPU binding for thread differs between job execution and non-job execution.

When executing program on non-job

The thread is not bound to a CPU. But you can control CPU binding for thread using environment variable `FLIB_CPU_AFFINITY`.

Environment variable `FLIB_CPU_AFFINITY`

Threads are bound to CPUs in order of the specified `cpuid` list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The `cpuid` list shall be separated by comma (',') or space(' ').

The `cpuid` list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1 : cpuid of the beginning of the range. ($0 \leq cpuid1 < CPU_SETSIZE$)

cpuid2 : cpuid of the end of the range. ($0 \leq cpuid2 < CPU_SETSIZE$)

inc : increment ($1 \leq inc < CPU_SETSIZE$)

In addition, it is necessary to be the following.

```
cpuid1 <= cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The `cpuid` can be used the above-mentioned value. However, `cpuid` which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See `CPU_SET(3)` about details of `CPU_SETSIZE`.

Note

If `cpuid` is outside the CPU affinity of the process at the start of execution, the program outputs error messages. Correct the setting value.

Example

Example 1:

```
$ export FLIB_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 2:

```
$ export FLIB_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.

When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

Example 3:

```
$ export FLIB_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 4:

```
$ export FLIB_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.

When the number of threads is six or more, it is repeatedly used from the beginning of the list.



When executing program on job

When thread is not bound to CPU, you may control CPU binding with `FLIB_CPU_AFFINITY` as executing on non-job.

When thread is bound to CPU on job, the environment variable `FLIB_CPU_AFFINITY` becomes invalid.

For details of CPU binding, see Section "[J.5.2 CPU Binding](#)".

J.2.3 Example of Compilation and Execution

Example 1:

```
$ frtpx -Kparallel,reduction,ocl -Nfjomplib test1.f
$ ./a.out
```

Example 1 compiles the source program using the reduction optimization while enabling optimization control lines. The number of threads using at the runtime is defined by the execution environment. For details about the number of threads, see "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)".

Example 2:

```
$ frtpx -Kparallel -Koptmsg=2 -Nfjomplib test2.f
Fortran diagnostic messages: program name(main)
jwd5001p-i "test2.f", line 2: DO loop with DO variable 'i' is parallelized.
$ export PARALLEL=2
$ ./a.out
$ export PARALLEL=4
$ ./a.out
```

Example 2 compiles specifying the `-Koptmsg=2` option to confirm that automatic parallelization is performed. By specifying 2 for the environment variable `PARALLEL`, the program executes with two threads. The next step executes the program with 4 threads, by specifying 4 for `PARALLEL`.

J.2.4 Tuning of Parallelized Programs

See Section "[12.2.4 Tuning of Parallelized Programs](#)".

J.2.5 Automatic Parallelization

See Section "[12.2.5 Feature Details on Automatic Parallelization](#)".

J.2.6 Optimization Control Line

See Section "[12.2.6 Optimization Control Line](#)".

J.2.7 Notes on Automatic Parallelization

See Section "[12.2.7 Notes on Automatic Parallelization](#)".

J.2.8 Linkage with Other Multi-Thread Programs

This section describes the restrictions of linking with object program that perform in multiple threads rather than automatic parallelization.

OpenMP program by this Compiler

OpenMP object modules, which are compiled with the `-Kopenmp` option, can be linked with object programs that are compiled with the `-Kparallel` option.

Other Multi-Thread Programs

Linking with object programs that use multiple threads, but which are not compiled with the system, is not supported.

However, the program may be linked with a pthread program when using the environment variable `FLIB_PTHREAD`.

Environment Variable `FLIB_PTHREAD`

You can control the linking with a pthread program, using the environment variable `FLIB_PTHREAD`.

Valid values are as follows. Default value is 0.

Value	Explanation
0 (Default)	<p>The program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none"> - The pthread thread must use suspending wait. - The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*) - The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*) - The Fortran routine must not be used in the pthread program. (*)
1	<p>The program can be linked with a pthread program which executes parallel processing.</p> <p>And, the program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none"> - A pthread parallel processing, and an OpenMP or automatic parallel processing must be executed in order. (*) - An OpenMP or automatic parallel processing must not be executed in a pthread parallel processing. - And, a pthread parallel processing must not be executed in OpenMP or automatic parallel processing. - The pthread thread must use suspending wait. - The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*) - The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*) - The Fortran routine must not be used in the pthread program. (*) - Fujitsu's math libraries must not be used in the pthread program. (The pthread program must not be compiled with <code>-SSL2BLAMP</code> option.) (*) - You must execute OpenMP parallel processing with two or more threads, before creating pthread threads. And, this number of OpenMP threads cannot be changed later. <p>And, when this function is used, the following functions are effective.</p> <ul style="list-style-type: none"> - <code>FLIB_SPINWAIT=0</code>

Value	Explanation
	- FLIB_CPUBIND=off Operation is not guaranteed when a value which is different in the above-mentioned environment variable is specified.

*: Operation is not guaranteed when the restriction function is used.

J.3 Parallelization by OpenMP Specification

This section details parallelization as described in the OpenMP Specification. See the "OpenMP Architecture Review Board" website for details about the OpenMP specifications.

This system supports the following specifications.

Supported Specifications
OpenMP 3.1
part of OpenMP 4.0 (*1)
part of OpenMP 4.5 (*1)

*1: The following functions can be used:

- SIMD construct
- DECLARE SIMD construct

J.3.1 Compilation (OpenMP Parallelization)

For compiling source programs and linking, specify the following options to the compile command.

The specified option is different by whether to perform parallelization by the OpenMP specification.

Parallelization by the OpenMP specification	specified option	
	Compile time	Link time
When using the parallelization and the SIMD Extensions	-Kopenmp	-Kopenmp -Nfjomplib
When using the SIMD Extensions only, not using the parallelization	-Kopenmp_simd	None

Compiler Options for OpenMP Programs

This section describes the compilation options used to compile OpenMP programs.

-Kopenmp

This option enables OpenMP directives and compiles the source programs.

The -Kopenmp option should also be specified at the linkage phase of an object file, when the object file has previously been compiled with this option.



Example

```
$ frtpx a.f90 -Kopenmp -c
$ frtpx a.o -Kopenmp -Nfjomplib
```

-Kopenmp_simd

-Kopenmp_simd option enables only the OpenMP SIMD construct, the DECLARE SIMD construct, the DECLARE REDUCTION directive, and the ORDERED construct with the SIMD clause, and compiles the source programs.

-Nfjompilib

This option specifies to use Fujitsu OpenMP Library.

-Nfjompilib option is required at linking.

Compiler Options for Obtaining Optimization Information

See Section "[12.3.1.2 Compiler Options for Obtaining Optimization Information](#)".

J.3.2 Execution Process (OpenMP Parallelization)

The execution process of an OpenMP program is the same as the procedure for a serial-processing program.

Environment Variable at Execution

Environment Variable	Explanation
FLIB_FASTOMP	<p>The user can control the high-speed runtime library using the environment variable FLIB_FASTOMP.</p> <p>Allowable values and their meanings are as follows:</p> <p>TRUE (default)</p> <p>The high-speed runtime library is adopted instead of the normal runtime library.</p> <p>FALSE</p> <p>The normal runtime library is adopted.</p> <p>The high-speed runtime library achieves a fast execution time by partly limiting the OpenMP specification. Details of the high-speed runtime library are shown below:</p> <ul style="list-style-type: none">- It is well optimized on the assumption that nested parallelism is inhibited and every thread is connected to a CPU respectively.- The hardware barrier can be used in the Job Operation Software environment. <p>To use the high-speed runtime library, OpenMP programs are restricted as follows:</p> <ul style="list-style-type: none">- The number of threads specified with the NUM_THREADS clause or the OMP_SET_NUM_THREADS subroutine must be the same as defined by the execution environment. <p>See "Notes on Execution" or "J.5 Using High-Speed Facility on Job Operation Software" for the number of threads determined in the runtime environment.</p> <p>When the high-speed runtime library is used, control of parallel execution with OpenMP environment variables and the OpenMP API is restricted as follows:</p> <ul style="list-style-type: none">- Irrespective of the environment variable OMP_NESTED and the OMP_SET_NESTED subroutine, nested PARALLEL regions are always serialized.- Irrespective of the environment variable OMP_DYNAMIC and the OMP_SET_DYNAMIC subroutine, the number of threads cannot exceed the limit on the number of CPUs. <p>See "Notes on Execution" or "J.5 Using High-Speed Facility on Job Operation Software" for information on the number of available CPUs.</p>
FLIB_SPINWAIT	<p>The standby threads during the serial execution can be controlled with the environment variable FLIB_SPINWAIT.</p> <p>The settings of FLIB_SPINWAIT will be valid when FLIB_SPINWAIT and OMP_WAIT_POLICY are specified at the same time.</p> <p>Allowable values and their meanings are as follows:</p>

Environment Variable	Explanation
	<p>unlimited(default) Uses spin wait.</p> <p>0 Uses suspending wait.</p> <p><i>n</i> s Uses spin wait until after <i>n</i> seconds, and uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "s" is specified following <i>n</i>.</p> <p><i>n</i> ms Uses spin wait until after <i>n</i> milliseconds, and uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "ms" is specified following <i>n</i>.</p> <p>Spin waiting is a technique to wait while consuming CPU. Suspend waiting is a technique to wait without consuming CPU. To give priority to elapsed time, specify unlimited, as spin waiting requires less overhead for parallel process. Select 0 to give priority to the CPU time.</p>
OMP_WAIT_POLICY	<p>The standby threads during the serial execution can be controlled with the environment variable OMP_WAIT_POLICY.</p> <p>The settings of FLIB_SPINWAIT will be valid when FLIB_SPINWAIT and OMP_WAIT_POLICY are specified at the same time.</p> <p>Allowable values and their meanings are as follows.</p> <p>ACTIVE(default) Uses spin wait.</p> <p>PASSIVE Uses suspending wait.</p> <p>Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.</p>
OMP_PROC_BIND	<p>A user can control the CPU binding for thread using the environment variable OMP_PROC_BIND.</p> <p>Allowable values and their meanings are as follows:</p> <p>TRUE The threads are bound to CPUs in order of cpuid list. However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.</p> <p>FALSE The thread is not bound to a CPU.</p> <p>The environment variable FLIB_CPUBIND or FLIB_CPU_AFFINITY is given to priority more than OMP_PROC_BIND.</p> <p>However, OMP_PROC_BIND is effective in the following case.</p> <ul style="list-style-type: none"> - FLIB_CPUBIND=off is specified, and FLIB_CPU_AFFINITY is not specified. <p>See Section "J.5.2 CPU Binding" for the environment variable FLIB_CPUBIND.</p> <p>See "Notes on Execution" for the environment variable FLIB_CPU_AFFINITY.</p>
THREAD_STACK_SIZE	<p>A user can specify the stack area size for each thread using the environment variable THREAD_STACK_SIZE in Kbytes. When the environment variable OMP_STACKSIZE is specified, the greater value is used as the stack area size for each thread.</p>

Environment Variable	Explanation
	See " Notes on Execution " for details.
OMP_STACKSIZE	A user can specify the stack area size for each thread using the environment variable OMP_STACKSIZE in Kbytes, Mbytes, or Gbytes. When the environment variable THREAD_STACK_SIZE is specified, the greater value is used as the stack area size. See " Notes on Execution " for details.

Environment Variable for OpenMP Specifications

The user can set the following environment variables, which are in accordance with the OpenMP Specifications. For the details, refer to the OpenMP specifications.

Environment Variable for OpenMP Specifications	Meaning
OMP_SCHEDULE	Sets the schedule type and chunk size for directives that have the schedule type at runtime.
OMP_NUM_THREADS	Sets the number of threads to use during execution.
OMP_DYNAMIC	Enables or disables dynamic thread adjustment.
OMP_PROC_BIND	Specifies whether threads are bound to CPUs.
OMP_NESTED	Enables or disables nested parallelism.
OMP_STACKSIZE	Specifies the stack size for each thread.
OMP_WAIT_POLICY	Specifies the behavior of the standby thread.
OMP_MAX_ACTIVE_LEVELS	Specifies the maximum number of nested active PARALLEL regions.
OMP_THREAD_LIMIT	Specifies the maximum number of threads performed on an Open MP program.

Notes on Execution

The following points should be noted when using parallelization features based on the OpenMP specifications on this compiler.

Allocating Variables at Runtime

The programs on this system that use parallelization features based on OpenMP specifications allocate local variables (other than the automatic data object while compiler option `-Knoautoobjstack` is in effect) and private variables on the stack, except for the main program. When a procedure requires a large area for local and private variables, a sufficiently large stack must be allocated.

When the stack area size for each thread needs to be specified, specify the environment variable `THREAD_STACK_SIZE` or `OMP_STACKSIZE`.

When the environment variable is not specified, the stack size for each thread is the same as the stack size for the process. However, if the maximum stack size for the process is larger than the value calculated by the expression " $\min(\text{the size of available memory in this system, the size of a virtual memory}) / (\text{the number of threads})$ ", the system will determine the stack area size and allocate it as follows:

```
the stack size for each thread (byte) =
(min(the size of available memory in this system, the size of a virtual memory) / (the number of
threads)) / 5
```

However, the assumed value does not guarantee that the system works correctly. It is recommended that an appropriate value for maximum process stack size be specified manually. The maximum process stack area size can be specified using the bash built in command `ulimit`.

Note that the maximum virtual memory size can also be confirmed using `ulimit` (bash built-in command).

The compiler calculates the stack area size without considering the feature that can dynamically change the thread numbers to align the stack area size of each thread. For this reason, the formula presented above uses the thread number assuming `"FLIB_FASTOMP=TRUE"`. When the thread number is to be changed dynamically, specify the maximum thread numbers using `OMP_NUM_THREADS` in advance.

See "[Environment Variable at Execution](#)" and "[Environment Variable for OpenMP Specifications](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for information about the environment variables.

Maximum number of CPUs

One of the following values is defined as the limit on the number of CPUs.

For jobs	The number of CPUs that can be used with jobs.
Not for job	The number of CPUs in the system.

Number of Threads

When the high-speed runtime library is used, a common number of threads is used for parallel execution of both the OpenMP program and automatic parallelization, otherwise the numbers of threads are determined independently of each other. Use of the high-speed runtime library is controlled by setting the environment variable `FLIB_FASTOMP`. See "[Environment Variable at Execution](#)" and "[Environment Variable for OpenMP Specifications](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for information about the environment variables.

When the high-speed runtime library is used (`FLIB_FASTOMP` is TRUE)

The number of threads is determined with the following priority:

1. The value of environment variable `OMP_NUM_THREADS`
2. The value of environment variable `PARALLEL`
3. The number of CPUs that can be used with jobs
4. 1 thread

The number of threads may not exceed the limit on the number of CPUs. If this occurs, the number of threads becomes equal to the limit on the number of CPUs.

The numbers of threads specified in the program with the `NUM_THREADS` clauses and the `OMP_SET_NUM_THREADS` subroutine must adhere to this limitation. If a different number of threads is specified in the program, the execution is stopped with the message `jwe1041i-s`.

When the high-speed runtime library is not used (`FLIB_FASTOMP` is FALSE)

The number of threads used for OpenMP code is determined with the following priority (while the one for automatic parallelization is described in Section "[J.2 Automatic Parallelization](#)"):

1. The value specified with the `NUM_THREADS` clause of the `PARALLEL` directive
2. The value specified with the `OMP_SET_NUM_THREADS` subroutine
3. The value of the environment variable `OMP_NUM_THREADS`
4. The value of the environment variable `PARALLEL`
5. The number of CPUs that can be used with jobs
6. 1 thread

When the dynamic thread adjustment feature is enabled, the actual number of threads is the minimum number of threads determined by the rule above and the limits on the number of CPUs.

When the dynamic thread adjustment feature is disabled, multiple threads can be executed on the same processor via time-sharing. Such parallel execution is not effective because the overhead of synchronization among threads is very high. Therefore, it is recommended to set the number of threads to be less than the limit on the number of CPUs.

For the execution processing on the job, see Section "[J.5.1 Management of CPU Resources](#)".

Notes when setting time limits

The program may not work correctly when time limits are set using the command such as `ulimit(1)`. In such a case, use the Fortran exception handling process, execution time option `-Wl, -i` to invalidate it.

Alternatively, use the execution time option `-Wl,-t` to set the time limit.

Notes when using service routines

ALARM Service Function

It may not work correctly. It is only for a program using serial processing.

KILL and SIGNAL Services Functions

It may cause a deadlock. It is only for a program using serial processing.

FORK Service Function

It may not work correctly. It is only for a program using serial processing.

SYSTEM, SH and CHMOD Service Functions

Memory usage doubles and performance may deteriorate. Use only with programs using serial processing.

CPU binding for thread

CPU binding for thread differs between job execution and non-job execution.

When executing program on non-job

The thread is not bound to a CPU. But you can control CPU binding for thread using environment variable `FLIB_CPU_AFFINITY`.

Environment variable `FLIB_CPU_AFFINITY`

Threads are bound to CPUs in order of the specified `cpuid` list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The `cpuid` list shall be separated by comma (',') or space(' ').

The `cpuid` list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: `cpuid` of the beginning of the range. ($0 \leq cpuid1 < CPU_SETSIZE$)

cpuid2: `cpuid` of the end of the range ($0 \leq cpuid2 < CPU_SETSIZE$)

inc: increment ($1 \leq inc < CPU_SETSIZE$)

In addition, it is necessary to be the following.

```
cpuid1 <= cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The `cpuid` can be used the above-mentioned value.

However, `cpuid` which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See `CPU_SET(3)` about details of `CPU_SETSIZE`.



If `cpuid` is outside the CPU affinity of the process at the start of execution, the program outputs error messages. Correct the setting value.



Example 1:

```
$ export FLIB_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 2:

```
$ export FLIB_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.
When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

Example 3:

```
$ export FLIB_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.
When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 4:

```
$ export FLIB_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.
When the number of threads is six or more, it is repeatedly used f from the beginning of the list.



When executing program on job

- When thread is not bound to CPU, you may control CPU binding with `FLIB_CPU_AFFINITY` as executing on non-job.
- When thread is bound to CPU on job, the environment variable `FLIB_CPU_AFFINITY` becomes invalid.
- For details of CPU binding, see Section "[J.5.2 CPU Binding](#)".

Output from More than One Threads

When errors are detected from more than one thread within a program, traceback maps are output in thread units. Therefore, in the `PARALLEL` region, the information is displayed at the same number as threads.

J.3.3 Implementation-Dependent Specifications

The OpenMP specification includes the following processor-dependent specification. For details on each item, refer to the OpenMP Specifications.

Memory Model

- If a variable is longer than 4 bytes or crosses a 4-byte boundary:
- The value of the variable written from two threads without synchronization may be undefined.
 - The value read by a thread from a variable that is written from another thread without synchronization may be undefined.

Internal Control Variables

The initial values for each of the internal control variables are as follows:

Internal Control Variable	Initial Value
nthreads-var	1
dyn-var	TRUE
run-sched-var	STATIC without chunk size
def-sched-var	STATIC without chunk size
bind-var	FALSE
stacksize-var	Stack area size for each thread in " Notes on Execution ", Section " J.3.2 Execution Process (OpenMP Parallelization) ".
wait-policy-var	ACTIVE
thread-limit-var	2147483647

Internal Control Variable	Initial Value
max-active-levels-var	2147483647

Dynamic Thread Adjustment Features

This system supports the dynamic thread adjustment described in "[Notes on Execution](#)", Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)".

By default, the dynamic adjustment features are on.

Loop Statement

The type used to calculate the iteration count of a loop that is not nested is 8 byte integer.

When auto is set for the internal control variable run-sched-var, the effect of the SCHEDULE(RUNTIME) construct is set to SCHEDULE(STATIC).

SECTIONS Construct

Assignment to a thread of a structured block in a SECTIONS construct is performed in the same way as a DYNAMIC schedule.

Each thread that encounters the SECTIONS region or that finishes the execution of a structured block in the sections region is moved on the next structured block.

SINGLE Construct

A SINGLE region is executed by the thread that encounters the region first.

SIMD Construct

An eight byte integer type is used to calculate the iteration count of a collapsed loop.

SIMD length is the value to which the compiler decides automatically.

When the *alignment* parameter is not specified in the ALIGNED clause, it is considered that the following value was specified for the *alignment* parameter.

When -KSVE option is effective at the compile time	Alignment of the type of the list item.
When -KNOSVE option is effective at the compile time	16

DECLARE SIMD Construct

SIMD length when the SIMDLEN clause is not specified is as follows.

- When -KSVE option is effective at the compile time

SIMD length is decided at the run time.

- When -KNOSVE option is effective at the compile time

SIMD length is decided by the size of the minimum type among all input parameters and return value. SIMD length is as follows by the size of the type.

Size of the type (byte)	SIMD length
1	16, 8
2	8, 4
4	4, 2
8	2
16	

When the *alignment* parameter is not specified in the ALIGNED clause, it is considered that the following value was specified for the *alignment* parameter.

When -KSVE option is effective at the compile time	Alignment of the type of the list item.
When -KNOSVE option is effective at the compile time	16

ATOMIC Construct

Two ATOMIC regions will be executed independently (not exclusively), when a variable type to be updated is different. When the types match, it may be executed exclusively even if the address is different.

- When updating a logical type, complex type, 1 byte integer type, 2 byte integer type, or quadruple precision (16 byte) real type variable
- When updating array element and index expression is not the same
- The target expression has explicit or implicit type conversion

OMP_SET_NUM_THREADS Routine

Calling the OMP_NUM_THREADS routine has no effect when the argument value is equal to or less than 0. A value that exceeds the number of threads supported by the system must not be specified.

OMP_SET_SCHEDULE Routine

There are no schedule types that are implementation-dependent.

OMP_SET_MAX_ACTIVE_LEVELS Routine

A call to the OMP_SET_MAX_ACTIVE_LEVELS routine will be ignored when it is performed from an explicit PARALLEL region. It will also be ignored when the argument is an integer that less than 0.

OMP_GET_MAX_ACTIVE_LEVELS Routine

The OMP_GET_MAX_ACTIVE_LEVELS routine can be called from anywhere in the program and it returns the value of the internal control variable max-active-levels-var.

Environment Variable OMP_SCHEDULE

When the schedule type specified for OMP_SCHEDULE is invalid, the schedule type is ignored, and the default schedule (STATIC without chunk size) is used.

When the schedule type specified for OMP_SCHEDULE is STATIC, DYNAMIC, or GUIDED, and the chunk size is not a positive number, the chunk size will be as follows:

Schedule Type	Chunk Size
STATIC	no chunk size
DYNAMIC, or GUIDED	1

Environment Variable OMP_NUM_THREADS

When a value equal to or less than 0 is specified for the list of OMP_NUM_THREADS, it works in the same ways as when 1 is specified. A value that exceeds the number of threads supported by the system must not be specified.

Environment Variable OMP_PROC_BIND

When a value other than TRUE or FALSE is specified for OMP_PROC_BIND, the value is ignored, and the default value (FALSE) is used.

Environment Variable OMP_DYNAMIC

When a value other than TRUE or FALSE is specified for OMP_DYNAMIC, the value is ignored and the default value (TRUE) is used.

Environment Variable OMP_NESTED

When a value other than TRUE or FALSE is specified for OMP_NESTED, it is ignored and the default value (FALSE) is used.

Environment Variable OMP_STACKSIZE

When the value specified for OMP_STACKSIZE does not meet the defined format, it is ignored and the default value is used.

Environment Variable OMP_WAIT_POLICY

ACTIVE performs spin wait. PASSIVE performs suspend wait.

Environment Variable OMP_MAX_ACTIVE_LEVELS

When the value specified for OMP_MAX_ACTIVE_LEVELS is an integer that is less than 0, it is ignored and the default value (2147483647) is used.

Environment Variable OMP_THREAD_LIMIT

When the value specified for OMP_THREAD_LIMIT is not a positive integer, it is ignored and the default value (2147483647) is used.

THREADPRIVATE directing statement

The value, allocation state, and association state of the thread private variables will be maintained across the consecutive PARALLEL regions, only when the following conditions are satisfied. These conditions are not affected by the configuration of the dynamic thread adjustment feature and nested parallelization.

- The consecutive PARALLEL region is not nested within another explicit PARALLEL region.
- The same number of threads is used to execute both PARALLEL regions.

If these conditions are not satisfied, the value, allocation status, and association status of the threadprivate variable at the entry point of the subsequent PARALLEL region will have an undefined status. However, the allocation status will be not allocated at the entry point of the subsequent PARALLEL region, when the following is true:

- The allocation status is not allocated just before entering the subsequent PARALLEL region.
- The allocation status is not allocated by any thread at the exit of the all preceding PARALLEL regions that are already executed.

SHARED Clause

When passing a shared variable to a non-intrinsic procedure, the values of the shared variables are stored in a temporary storage area, and the values are stored in the actual argument area after the procedure call. This can occur when the following conditions are satisfied:

- a. The actual argument is one of the following:
 - A shared variable
 - A subobject of a shared variable
 - An object associated to a shared variable
 - An object associated to a subobject of shared variable
- b. In addition the actual argument is one of the following:
 - An array section
 - An array section with a vector subscript
 - An assumed-shape array
 - A pointer array
- c. The associated dummy argument for this actual argument is an explicit-shape array or an assumed-size array.

Runtime Library Definitions

This compiler provides an include file omp_lib.h and a module omp_lib for the parallelization feature based on OpenMP specifications. This does not include an interface block with generic name.

The following notes must be considered when using the module omp_lib and omp_lib.h:

- When the compiler option -AU is specified, module name omp_lib and each library routine name must be specified using lower case alphabetic characters.
- When the compiler option -CcdII8, -CciI4I8, -CcdLL8, -CclL4L8, -Ccd4d8, or -Cca4a8 is specified, the corresponding type of the library routine argument and the result will be converted. On execution, the corresponding runtime option -Lb or -Li must be specified.
- Although the diagnostic message jwd2004i - i will be displayed if the named constant declared in omp_lib.h was not used, it does not affect the result.

J.3.4 Clarifying OpenMP Specifications and Restrictions

See Section "[12.3.4 Clarifying OpenMP Specifications and Restrictions](#)".

J.3.5 Notes on OpenMP Programming

See Section "[12.3.5 Notes on OpenMP Programming](#)".

J.3.6 Linking with Other Multi-Thread Programs

This section describes the restrictions to link with multi-threaded programs other than OpenMP.

Programs Automatically Parallelized Using this Compiler

Object modules compiled with this compiler using the `-Kparallel` option can be linked with other modules compiled with this compiler using the `-Kparallel` option.

Other multi-thread programs

The object module created by this system cannot be linked with multi-threaded object modules created by another multi-thread system.

However, the program may be linked with a pthread program when using the environment variable `FLIB_PTHREAD`.

Environment Variable `FLIB_PTHREAD`

You can control the linking with a pthread program, using the environment variable `FLIB_PTHREAD`.

Valid values are as follows. Default value is 0.

Value	Explanation
0 (Default)	<p>The program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none">- The pthread thread must use suspending wait.- The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*)- The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*)- The Fortran routine must not be used in the pthread program. (*)
1	<p>The program can be linked with a pthread program which executes parallel processing.</p> <p>And, the program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none">- A pthread parallel processing, and an OpenMP or automatic parallel processing must be executed in order. (*) An OpenMP or automatic parallel processing must not be executed in a pthread parallel processing. And, a pthread parallel processing must not be executed in OpenMP or automatic parallel processing.- The pthread thread must use suspending wait.- The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*)- The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*)- The Fortran routine must not be used in the pthread program. (*)

Value	Explanation
	<ul style="list-style-type: none"> - Fujitsu's math libraries must not be used in the pthread program. (The pthread program must not be compiled with -SSL2BLAMP option.) (*) - You must execute OpenMP parallel processing with two or more threads, before creating pthread threads. And, this number of OpenMP threads cannot be changed later. <p>And, when this function is used, the following functions are effective.</p> <ul style="list-style-type: none"> - FLIB_SPINWAIT=0 - FLIB_CPUBIND=off <p>Operation is not guaranteed when a value which is different in the above-mentioned environment variable is specified.</p>

*) Operation is not guaranteed when the restriction function is used.

J.3.7 Debugging OpenMP Programs

See Section "[12.3.6 Debugging OpenMP Programs](#)".

J.4 I/O Buffer Parallel Transfer

The I/O buffer parallel transfer is a feature that provides parallel threading to improve the performance of data transfer between the buffer area used in the array area and the input-output statement in a Fortran program.

J.4.1 Compilation

Automatic parallelization

The compile time option `-Kparallel` is an option to perform automatic parallelization. See Section "[J.2.1 Compilation \(Automatic Parallelization\)](#)" for details.

OpenMP parallelization

The compiler option `-Kopenmp` is an option to perform parallelization based on the OpenMP specifications. See Section "[J.3.1 Compilation \(OpenMP Parallelization\)](#)" for details.

J.4.2 Execution Process

Following environment variables are used as necessary.

FLIB_IOBUFCPY

FLIB_IOBUFCPY is used for I/O buffer parallel transfer function. Only "MP" can be specified to FLIB_IOBUFCPY. When the value except "MP" is specified for this environment variable, I/O buffer parallel transfer function is not executed.

Example:

```
$ export FLIB_IOBUFCPY=MP
```

FLIB_IOBUFCPY_SIZE size

The environment variable FLIB_IOBUFCPY_SIZE is used to change the condition to execute I/O buffer parallel transfer function. When the size is specified to the environment variable FLIB_IOBUFCPY_SIZE, an unformatted I/O statement transfers data in parallel if data transfer size and I/O buffer size are more than size Kbytes. The size argument is in Kilobytes. The size argument must be an integer with a value of 1 or more.

Example:

```
$ export FLIB_IOBUFCPY_SIZE=16
```

Automatic parallelization

- Specify MP for the environment variable FLIB_IOBUFCPY.
- Specify the number of threads using the environment variable PARALLEL. See "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" for details.

OpenMP parallelization

- Specify MP for the environment variable FLIB_IOBUFCPY.
- Specify the number of threads using the environment variable OMP_NUM_THREADS. See "[Number of Threads](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for details.

J.4.3 Conditions to Perform I/O Buffer Parallel Transfer

The following conditions must be met to perform I/O buffer parallel transfer:

- Target I/O statements are unformatted I/O statements
- Target I/O statements are outside the parallel region or outside the automatically parallelized area
- When the environment variable FLIB_IOBUFCPY_SIZE is not specified, the amount of the data transferred and the I/O buffer value must be more than "NumberOfThread + 30" Kbyte.
- When size is specified on the environment variable FLIB_IOBUFCPY_SIZE, the amount of the data transferred and the I/O buffer value must be more than "size" Kbyte.

For sequential access I/O statements or stream access I/O statements, specify the I/O buffer size using the runtime option -g or the environment variable fuxxbf. For direct access I/O statements, specify using the recl specifier of the open statement and the runtime option -d.

See "[3.3 Runtime Options](#)" for more information on runtime option -g.

See Section "[3.8 Using Environment Variables for Execution](#)" for information on the environment variables fuxxbf and FLIB_IOBUFCPY_SIZE.

J.4.4 Notes on I/O Buffer Parallel Transfer

When the I/O buffer size or the transferred data size is small, the performance may deteriorate even if I/O buffer parallel transfer is performed.

J.4.5 Example

- test.f

```
CHARACTER CH((30+4)*1024)
CH='X'
OPEN(10,FORM="UNFORMATTED")
WRITE(10) CH
CLOSE(10)
END
```

- Automatic parallelization

```
$ frtpx -kparallel -Nfjompilib test.f
$ export FLIB_IOBUFCPY=MP
$ export PARALLEL=4
$ ./a.out -Wl,-g34
```

This specifies MP for the environment variable FLIB_IOBUFCPY to enable I/O buffer parallel transfer. It also sets 4 for the environment variable PARALLEL to make the number of threads 4. The runtime option -g34 sets the I/O buffer size to 34 Kbytes (Number of thread 4 + 30 Kbytes).

- OpenMP parallelization

```
$ frtpx -Kparallel -Nfjompilib test.f
$ export FLIB_IOBUFPCPY=MP
$ export OMP_NUM_THREADS=4
$ ./a.out -Wl,-g34
```

This specifies MP for the environment variable `FLIB_IOBUFPCPY` to enable I/O buffer parallel transfer. It also sets 4 for the environment variable `OMP_NUM_THREADS` to make the number of threads 4. The runtime option `-g34` sets the I/O buffer size to 34 Kbytes (Number of thread 4 + 30 Kbytes).

J.5 Using High-Speed Facility on Job Operation Software

On the job of Job Operation Software, the high-speed facility (the inter-core hardware barrier and the Sector cache) can be used.

This section provides information on the compilation and execution processes for using the high-speed facility on FX system.

J.5.1 Management of CPU Resources

The CPUs resources that can be used are strictly managed on the job of the Job Operation Software.

- The Number of CPUs that can be used on the job

This is the number of CPUs that can be used for a process on the job. See "[Number of CPUs that can be used on the job](#)" for details.

- Limits on Number of CPUs

The number of CPUs that can be used on the job is defined as the limit on the number of CPUs.

The number of threads when the program is executed is depends on the limit values of the number of CPUs. See "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" for details on the number of the threads for automatic parallelization. See "[Number of Threads](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for details on the number of threads for OpenMP.

Number of CPUs that can be used on the job

The limit on the number of CPUs for one process is determined by considering CPU resources and the number of processes in the virtual node on the job. This value is defined as the number of CPUs that can be used on the job. Details are shown as follows:

```
"Number of CPUs that can be used on the job" = cpunum_on_node / procnum_on_node
```

cpunum_on_node : The number of CPUs that can be used on one virtual node.

The value is from "1" to "the number of all CPUs on one virtual node".

procnum_on_node : The number of processes on one virtual node.

For thread parallel processing jobs: the value is "1".

For hybrid (process and thread) parallel processing jobs: the value is from "1" to "the number of all CPUs on one virtual node".

It is rounded down when it cannot be divided.

When MPI program is executed with VCOORD file that is set the number of CPUs to a process, the value is the number of CPUs that can be used on the job. See the "MPI User's Guide" for details on MPI.

FLIB_USE_ALLCPU

"The number of CPUs that can be used on the job" can be controlled using the environment variable `FLIB_USE_ALLCPU`.

The allowed values their meanings are as follows. The default value is "FALSE".

TRUE

The number of CPUs that can be used on the job is *cpunum_on_node*. This means all CPUs that can be used on the virtual node are used regardless of the number of processes. Therefore, when the number of processes on the virtual node is two or more, these CPUs resource is shared among the processes.

When the frequency calculated at the same time in each process is low etc., an improvement in execution performance can be expected.

In general use, an improvement in execution performance cannot be expected.

You should use the following functions.

- The `-Kopenmp` option is used at compiling and linking time.
- The environment variable `FLIB_FASTOMP=FALSE` is defined at run time.
- The environment variable `FLIB_SPINWAIT=0` is defined at run time.

When the above-mentioned is not used, the performance might be greatly downed like the dead lock. See "[J.3 Parallelization by OpenMP Specification](#)" for details of `-Kopenmp`, `FLIB_FASTOMP` and `FLIB_SPINWAIT`.

When `procnum_on_node` is 1, "TRUE" and "FALSE" have the same effect. When `procnum_on_node` is not 1, "[J.5.3 Inter-Core Hardware Barrier](#)" and "[J.5.4 Sector Cache](#)" cannot be used.

Refer to "[Number of CPUs that can be used on the job](#)" in Section "[J.5.1 Management of CPU Resources](#)" for details about `cpunum_on_node` and `procnum_on_node`.

FALSE

Refer to "[Number of CPUs that can be used on the job](#)" in Section "[J.5.1 Management of CPU Resources](#)".

When MPI program is executed with VCOORD file that is set the number of CPUs to a process, the value is ignored. See the "MPI User's Guide" for details on MPI.

FLIB_USE_CPURESOURCE

The environment variable `FLIB_USE_CPURESOURCE` controls the CPU resource manager on the job.

The allowed values their meanings are as follows. The default value is "TRUE".

TRUE

The CPU resources on the job are managed.

FALSE

The CPU resources on the job are not managed. Therefore the number of CPUs is not managed, and you cannot use the following functions.

- "[J.5.2 CPU Binding](#)"
- "[J.5.3 Inter-Core Hardware Barrier](#)"
- "[J.5.4 Sector Cache](#)"

J.5.2 CPU Binding

The thread is bound to one CPU when the program using the automatic parallelization or OpenMP is executed on the job. CPUs resources can be strictly managed as described in Section "[J.5.1 Management of CPU Resources](#)". Therefore an improvement on the performance can be expected using CPU binding.

See "[J.2 Automatic Parallelization](#)" and "[J.3 Parallelization by OpenMP Specification](#)" for details.

FLIB_CPUBIND

The environment variable `FLIB_CPUBIND` controls the CPU binding for threads.

The following value can be set to `FLIB_CPUBIND`. The default value is "chip_pack".

chip_pack	All threads are bound to the same CPU Chip as much as possible.
chip_unpack	Each thread is bound to a different CPU Chip as much as possible.
off	The thread is not bound to the CPU. If you control CPU binding by yourself, this must be set.

When only one CPU Chip can be used, "chip_pack" and "chip_unpack" have the same effect.

When only one CPU on a CPU Chip can be used, "chip_pack" and "chip_unpack" have the same effect.

J.5.3 Inter-Core Hardware Barrier

FX system has the inter-core hardware barrier. On the job of Job Operation Software, it can be used as the thread barrier. The inter-core hardware barrier is a hardware mechanism which facilitates high speed synchronization among threads in a CPU Chip, and raises the execution performance.

See manuals of Job Operation Software for details on Job Operation Software about how to execute and submit a job.

Compilation

When generating a program using the inter-core hardware barrier as the thread barrier, specify `-Kparallel` or `-Kopenmp` option.

The hardware barrier is automatically used during execution in an environment in which it can be used. By contrast, the software barrier is used during execution in an environment that cannot use the hardware barrier.

Execution

The inter-core hardware barrier can be used only on the job. The inter-core hardware barrier is a hardware mechanism for synchronization among threads in a CPU Chip.

FLIB_CNTL_BARRIER_ERR

When the inter-core hardware barrier cannot be used, the following message is output, and the processing is continued using the software barrier.

```
jwe1050i-w The hardware barrier couldn't be used and continues processing using the software barrier.
```

This diagnostic message (jwe1050i-w) error can be controlled using the environment variable `FLIB_CNTL_BARRIER_ERR`.

The allowed values and their meanings are as follows. Default is "TRUE".

TRUE

The diagnostic message (jwe1050i-w) error is detected.

When the inter-core hardware barrier cannot be used, this message is output, and the processing is continued using the software barrier.

FALSE

The diagnostic message (jwe1050i-w) error is not detected.

When the inter-core hardware barrier cannot be used, the processing is continued using the software barrier.

FLIB_NOHARDBARRIER

When the environment variable `FLIB_NOHARDBARRIER` is set, the inter-core hardware barrier is not used and the software barrier is used. Always outputs the diagnostic message jwe1050i-w.

See "[FLIB_CNTL_BARRIER_ERR](#)" in Section "[J.5.3 Inter-Core Hardware Barrier](#)" for details.

Note

- When the inter-core hardware barrier is used by a program made using the compile option `-Kopenmp`, the environment variable `FLIB_FASTOMP` shall not be FALSE. See Section "[J.3 Parallelization by OpenMP Specification](#)" for the meaning and use of the environment variable `FLIB_FASTOMP`.
- When the inter-core hardware barrier is used, CPU binding is needed. Therefore, when "off" is set for the environment variable `FLIB_CPUBIND`, the inter-core hardware barrier cannot be used. See Section "[J.5.2 CPU Binding](#)" for details of CPU binding.
- When the number of threads is 1, the inter-core hardware barrier cannot be used. See "[Number of Threads](#)" in "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" or "[Number of Threads](#)" in "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for the number of threads.
- When the number of CPUs to a process is 3 or less, an inter-core hardware barrier may not be able to use. The number of CPUs to a process should be set to 4 or more.
- When the job is a Swap Target Job and a Node-sharing Job, and the number of CPUs to a process is three or less, an inter-core hardware barrier cannot be used. The number of CPUs to a process should be set to 4 or more.

- The inter-core hardware barrier in FX system is used only inside the NUMA node. When one process is allocated to multiple NUMA nodes, inter-core hardware barrier is used inside NUMA node and software barrier is used between NUMA nodes. Therefore, it is recommended to allocate one process to one NUMA node, in case of giving priority to the performance of threads barrier.
- On the following cases, the high speed runtime library cannot determine whether one process can reserve a Barrier resource or not. Therefore behavior of hardware barrier is indeterminate, program may be ended with diagnostic-message (jwe1044i-u or jwe1045i-u), or the program may terminate abnormally.
 - When Process is generated by functions other than Fujitsu MPI:

Example:

- FORTRAN FORK/SYSTEM/SH service function is used
- C fork system call/system function is used

Example

Example 1: A program that uses the inter-core hardware barrier is executed on a thread parallel processing job.

```
$ cat job.sh
#!/bin/sh
./a.out
$ pjsub -L "node=1" job.sh
```

Example 2: A program that uses the inter-core hardware barrier is executed on a hybrid (process and thread) parallel processing job (One job is executed in one dimension: The number of virtual nodes is set to the number of processes).

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=12"
#PJM --mpi "shape=12"
#PJM --mpi "rank-map-bynode"
mpixec -n 12 a.out
$ pjsub job.sh
```

J.5.4 Sector Cache

FX system has Sector cache. On the job, the Sector cache can be controlled.

See Section "[9.17 Software Control of Sector Cache](#)" for details on how to control the Sector cache.

This section explains only notes on execution.

Notes on execution

The Sector cache for the primary and secondary level caches are available in the following execution environments.

The execution environments	The Sector cache for the first level cache	The Sector cache for the second level cache
Only one process runs on a NUMA node	Available	Available
Multiple processes run on a NUMA node	Available (*1)	Not available

*1) Not available if the environment variable FLIB_L1_SCCR_CNTL is assigned to the value FALSE. See Section "[9.17.2.2 Software Control with Environment Variables and Optimization Control Lines](#)" for details.

On the following cases, the high speed runtime library cannot determine whether one process can reserve a NUMA node or not. Therefore behavior of Sector cache is indeterminate. Performance of program may be reduced, program may be ended with diagnostic-message (jwe1048i-u), or the program may terminate abnormally.

- When Process is generated by functions other than Fujitsu MPI:

Example:

- FORTRAN FORK/SYSTEM/SH service function is used
- C fork system call/system function is used

- When threads are generated by functions other than Fujitsu Automatic Parallelization/OpenMP:

Example:

- Threads are controlled by pthread functions

To invalidate the Sector cache control function unconditionally regardless of the execution environment, "FALSE" is set for the environment variable FLIB_SCCR_CNTL.

See Section "[9.17.2.2 Software Control with Environment Variables and Optimization Control Lines](#)" for details.

Index

	[Special characters]		
-#.....		71	
-###.....		72	
\$ edit descriptor.....		137,157	
&END.....		141	
&namelist name.....		141	
	[A]		
-A.....		19	
-a.....		76	
-AA.....		19,111	
ACTION= Specifier.....		145,181	
-Ad.....		20,112	
-AE.....		19	
-Ae.....		19	
-Ai.....		20	
ALLOCATE Statement.....		128	
-Ap.....		20	
-Aq.....		20,113	
arithmetic IF statement.....		125	
array declaration.....		414	
array element.....		28,229	
array section.....		28,229,413,415	
assembler.....		5	
assigned GO TO statement.....		125,255	
assignment statements with overlapping character positions.....		124	
ASSOCIATE construct.....		413	
assumed rank.....		344	
assumed type.....		343	
-AT.....		20	
-AU.....		20	
-Aw.....		20	
-Ay.....		20	
-Az.....		21	
	[B]		
BACKSPACE statement.....		175	
big endian data.....		179	
BLOCK construct.....		413	
BLOCKSIZE= Specifier.....		146	
	[C]		
-c.....		16	
-C.....		21	
CASE construct.....		413	
-Cca4a8.....		24	
-Ccd4d8.....		24	
-CcdDR16.....		25	
-CcdII8.....		21	
-CcdLL8.....		22	
-CcdRR8.....		23	
-CcI4I8.....		22	
-CcL4L8.....		23	
-Ccpp.....		21	
-CcR4R8.....		23	
-CcR8R16.....		25	
-Cfpp.....		21	
character type data.....		111	
-Cl.....		21,114	
code coverage.....		431	
command.....		5	
common block.....		28,111	
COMMON statement.....		119	
Compile command.....		1	
compiler option.....		6	
complex type data.....		107	
control information.....		154	
CONVERT= Specifier.....		146	
-Cpp.....		21	
cpp command.....		5	
CRITICAL construct.....		413	
	[D]		
-D.....		26	
-d.....		77	
DEALLOCATE Statement.....		128	
debugging.....		188	
debugging check function.....		224	
debugging functions.....		224	
default implicit typing.....		20	
diagnostic message.....		16	
direct access input/output statement.....		158	
DO construct.....		413	
DO statement.....		126	
	[E]		
-E.....		26	
-e.....		77	
-Ec.....		26,123	
-Eg.....		27	
endfile record.....		141	
ENDFILE statement.....		176	
Endian convert command.....		1	
EQUIVALENCE statement.....		120	
error.....		188	
error condition.....		156	
error level.....		85	
error monitor.....		188	
error processing.....		188	
ERROR STOP statement.....		127	
executable program.....		16	
executable program output.....		103	
execution command.....		76	
execution command environment variable.....		81	
execution command format.....		76	
execution command return value.....		82	
expression.....		122	
	[F]		
-f.....		16	
FCOMP_LINK_FJOB.....		72	
FCOMP_UNRECOGNIZED_OPTION.....		73	

-Kinstance.....	38	-Knoprefetch.....	43
-Kintentopt.....	38	-Knopreload.....	50
-Klargepage.....	38	-Knoreduction.....	50
-Kloop_blocking.....	39	-Knoregion_extension.....	50
-Kloop_fission.....	39	-Knosch_post_ra.....	51
-Kloop_fission_threshold.....	40	-Knosch_pre_ra.....	51
-Kloop_fusion.....	40	-Knosibling_calls.....	51
-Kloop_fusion_stripmining.....	39	-Knosimd.....	52
-Kloop_interchange.....	40	-Knostatic_fjlib.....	53
-Kloop_noblocking.....	39	-Knostriping.....	54
-Kloop_nofission.....	39	-Knosubscript_opt.....	53
-Kloop_nofission_stripmining.....	39	-KNOSVE.....	54
-Kloop_nofusion.....	40	-Knoswp.....	54
-Kloop_nointerchange.....	40	-Knotemparraystack.....	55
-Kloop_nopart_parallel.....	40	-Knothreadsafe.....	56
-Kloop_nopart_simd.....	40	-Knounroll.....	56
-Kloop_noperfect_nest.....	41	-Knounroll_and_jam.....	56
-Kloop_noversioning.....	41	-Knozfill.....	57
-Kloop_part_parallel.....	40	-Kocl.....	43
-Kloop_part_simd.....	40	-Kopenmp.....	43,399,450
-Kloop_versioning.....	41	-Kopenmp_assume_norecurrence.....	44
-Klto.....	41	-Kopenmp_collapse_except_innermost.....	44
-Kassume=memory_bandwidth.....	32	-Kopenmp_noassume_norecurrence.....	44
-Kmfunc.....	41	-Kopenmp_nocollapse_except_innermost.....	44
-Knoalias.....	43	-Kopenmp_noordered_reduction.....	44
-Knoalign_commons.....	29	-Kopenmp_ordered_reduction.....	44
-Knoalign_loops.....	29	-Kopenmp_simd.....	45,399,450
-Knoarray_declaration_opt.....	30	-Koptlib_string.....	45
-Knoauto.....	32	-Koptmsg.....	45,260
-Knoautoobjstack.....	33	-Kparallel.....	46
-Knocalleralloc.....	33	-Kparallel_fp_precision.....	46
-Knodynamic_iteration.....	34	-Kparallel_iteration.....	46
-Knoeval.....	34	-Kparallel_nofp_precision.....	46
-Knoextract_stride_store.....	35	-Kparallel_strong.....	46
-Knofenv_access.....	35	-Kpc_relative_literal_loads.....	47
-Knofp_contract.....	36	-Kloop_perfect_nest.....	41
-Knofp_precision.....	36	-Kpic.....	47
-Knofp_relaxed.....	36	-KPIC.....	47
-Knofsimple.....	37	-Kplt.....	47
-Knofz.....	37	-Kpreex.....	47
-Knohpctag.....	37	-Kprefetch_cache_level.....	47
-Knoifunc.....	38	-Kefetch_conditional.....	48
-Knointentopt.....	38	-Kprefetch_indirect.....	48
-Knolargepage.....	39	-Kprefetch_infer.....	48
-Knolto.....	41	-Kprefetch_iteration.....	48
-Knomfunc.....	43	-Kprefetch_iteration_L2.....	48
-Knocl.....	43	-Kprefetch_line.....	48
-Komitfp.....	43	-Kprefetch_line_L2.....	48
-Knoomitfp.....	43	-Kprefetch_noconditional.....	48
-Knoopenmp.....	44	-Kprefetch_noindirect.....	48
-Knoopenmp_simd.....	45	-Kprefetch_noinfer.....	48
-Knooptlib_string.....	45	-Kprefetch_nosequential.....	49
-Knooptmsg.....	46	-Kprefetch_nostride.....	49
-Knoparallel.....	46	-Kprefetch_nostrong.....	49
-Knopc_relative_literal_loads.....	47	-Kprefetch_sequential.....	49
-Knoplt.....	47	-Kprefetch_stride.....	49
-Knopreex.....	47	-Kprefetch_strong.....	49

-Kprefetch_strong_L2.....	50
-Kpreload.....	50
-Kreduction.....	50
-Kregion_extension.....	50
-Ksch_post_ra.....	51
-Ksch_pre_ra.....	51
-Ksibling_calls.....	51
-Ksimd.....	51
-Ksimd=1.....	51
-Ksimd=2.....	51
-Ksimd=auto.....	52
-Ksimd_nopacked_promotion.....	52
-Ksimd_noreduction_product.....	52
-Ksimd_nouncounted_loop.....	53
-Ksimd_nouse_multiple_structures.....	53
-Ksimd_packed_promotion.....	52
-Ksimd_reduction_product.....	52
-Ksimd_reg_size.....	52
-Ksimd_uncounted_loop.....	53
-Ksimd_use_multiple_structures.....	53
-Kstatic_fjlib.....	53
-Kstriping.....	54
-Ksubscript_opt.....	53
-KSVE.....	54
-Kswp.....	54
-Kswp_freq_rate.....	54
-Kswp_ireg_rate.....	54
-Kswp_policy=auto.....	55
-Kswp_policy=large.....	55
-Kswp_policy=small.....	55
-Kswp_preg_rate.....	55
-Kswp_strong.....	55
-Kswp_weak.....	55
-Ktemparraystack.....	55
-Kthreadsafe.....	55
-Ktls_size.....	56
-Kunroll.....	56
-Kunroll_and_jam.....	56
-Kvisimpact.....	56
-Kzfill.....	57

[L]

-l.....	17,77
-L.....	57
-Lb.....	79
-le.....	78
L edit descriptor.....	183
-li.....	78
-Li.....	79
LIBRARY_PATH.....	73
line-feed character.....	137
linker.....	5
--linkstl.....	72
--linkstl=libc++.....	72
--linkstl=libfj++.....	72
--linkstl=libstdc++.....	72
list-directed input/output statement.....	166

little endian data.....	179
local array.....	18
logical IF statement.....	125
logical type data.....	107
-Lr.....	79
-ls.....	78
-Lu.....	79
-lw.....	78

[M]

-M.....	57,324
-m.....	78
man command.....	5
-ml.....	17
-mlcmain.....	17
-mldefault.....	17
module.....	324

[N]

-N.....	57
-n.....	78
-Nallextput.....	57
-Nalloc_assign.....	58
namelist input/output statement.....	159
NAMELIST statement.....	159
-Nargchk.....	64
-Ncancel_overtime_compilation.....	58
-Ncheck_cache_arraysize.....	58
-Ncheck_global.....	58
-Ncheck_intrfunc.....	58
-Ncheck_std.....	59
-Ncoarray.....	59
-Ncompdisp.....	59
-Ncopyarg.....	60
-Ncoverage.....	60
NEWUNIT= Specifier.....	146
-Nf90move.....	60,124
-Nfjmplib.....	62,451
-Nfjprof.....	61
-Nfreealloc.....	61
-Nhook_func.....	61
-Nhook_time.....	61
-Ninf_detail.....	65
-Ninf_simple.....	65
-Nlibomp.....	61,399
-Nline.....	62
-Nlst=p.....	102
-Nlst=t.....	102
-Nmallocfree.....	63
-Nmaxserious.....	63
-Nnoallextput.....	57
-Nnoalloc_assign.....	58
-Nnoargchk.....	64
-Nnocancel_overtime_compilation.....	58
-Nnocheck_global.....	58
-Nnocarray.....	59
-Nnocompdisp.....	59
-Nnocopyarg.....	60

-Nnocoverage.....	60	OMP_THREAD_LIMIT.....	403,407
-Nnof90move.....	60	OMP_WAIT_POLICY.....	399,403,407
-Nnofjprof.....	61	Online manual.....	1
-Nnofreealloc.....	61	online manual.....	5
-Nnohook_func.....	61	OpenMP.....	398
-Nnohook_time.....	61	OPEN statement.....	142
-Nnoline.....	62	OPEN statement specifier.....	142
-Nnomallocfree.....	63	option.....	5
-Nnoobsfun.....	63	option-list.....	5
-Nnoprivatealloc.....	63	output of runtime information output function.....	436
-Nnorecursive.....	65		
-Nnoreordered_variable_stack.....	65	[P]	
-Nnosave.....	66	-P.....	70
-Nnosetvalue.....	68	-p.....	78
-Nnosubchk.....	64	PAUSE statement.....	126
-Nnoundef.....	64	precision conversion.....	112
-Nnose_rodatta.....	68	precision improving.....	112
-Nobsfun.....	63	precision lowering and analysis of error.....	113
nonadvancing input/output statement.....	170	preprocessor.....	3
notes on runtime information output function.....	439	PUBLIC attribute.....	326
-Nprivatealloc.....	63		
-Nprofile_dir.....	63	[Q]	
-Nquickdbg[=dbg_arg].....	63	-q.....	78
-Nrecursive.....	65	-Q.....	79
-Nreordered_variable_stack.....	65		
-NRnotrap.....	69	[R]	
-NRtrap.....	68	-r.....	78
-Nrt_notune.....	65	-Re.....	80
-Nrt_tune.....	65	REAL and CMPLX used as generic names.....	130
-Nrt_tune_func.....	65	real type data.....	107
-Nrt_tune_loop.....	66	RECL= Specifier.....	144
-Nsave.....	66	relational expressions.....	123
-Nsetvalue.....	66	relational operation.....	123
-Nsubchk.....	64	removing a common expression.....	255
-Nundef.....	64	REWIND statement.....	179
-Nundefnan.....	64	rounding error.....	21
-Nuse_rodatta.....	68	-Rp.....	80
		runtime information output function.....	435
[O]		runtime option.....	76
-o.....	18	-Ry.....	80
-O.....	69		
-O0.....	69	[S]	
-O1.....	69	-S.....	70
-O2.....	69	SAVE attribute for initialized variable.....	122
-O3.....	70	SELECT TYPE construct.....	413
OCL.....	262	sequential access.....	141
OMP_CANCELLATION.....	403	service routine.....	131
OMP_DISPLAY_ENV.....	403	-shared.....	18
OMP_DYNAMIC.....	403,407	size of an array element.....	121
OMP_MAX_ACTIVE_LEVELS.....	403,407	-SSL2.....	70
OMP_MAX_TASK_PRIORITY.....	403	-SSL2BLAMP.....	71
OMP_NESTED.....	403,407	STAT= Specifier.....	128
OMP_NUM_THREADS.....	402,406	statement function reference.....	129
OMP_PLACES.....	369,400,403,406	STATUS= Specifier.....	143
OMP_PROC_BIND.....	369,400,403,406	STOP statement.....	127
OMP_SCHEDULE.....	402	substring.....	28,228,413
OMP_STACKSIZE.....	401,403,407		

[T]

-t.....	79
-T.....	80
TMPDIR.....	74,84
trace back map.....	104

[U]

-U.....	71
unformatted sequential input/output statement.....	157
usage of runtime information output function.....	435
USE statement.....	324
Using High-Speed Facility on Job Operation Software.....	441
using the optimization control line.....	262

[V]

-V.....	71
version and release information.....	71

[W]

-W.....	71
-Wl.....	76

[X]

-x.....	18,79,258
-X.....	71
-x-.....	18
-X03.....	71
-X08.....	71
-X6.....	71
-X7.....	71
-X9.....	71
-xaccept=accept_arg.....	19
-Xd7.....	71
-xdat_szK.....	18
-xdir=dir_name.....	19
-xproc_name.....	18
-xstmt_no.....	18