FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

Development Studio
Fortran使用手引書

J2UL-2473-02Z0(09)
2022年9月

まえがき

本書の目的

本書は、富士通製CPU A64FXを搭載したシステム(以降、FXシステムと呼びます)向けのFortran処理系(以降、本処理系と呼びます)の使用方法を記述しています。
また、本処理系における以下の事項についても記述しています。

- Fortranプログラミングに関する解説
- 制限事項

本書の読者

本書は、本処理系を使用して、Fortran原始プログラムを処理する方およびFortran原始プログラムを作成する方を対象に記述しています。

本書を読むにあたっては、Linux(R)のコマンド、ファイル操作、およびシェルプログラミングの基本的な操作知識が必要です。

本書では、読者はFortran言語仕様を理解していることを前提に記述しています。Fortran言語仕様については、オンラインマニュアルで提供されている“Fortran文法書”、Fortran規格書、または市販本を参照してください。

本書の構成

本書は次の構成になっています。

第1章 概要

本処理系を使用してFortran原始プログラムを処理する場合の概要について記述しています。

第2章 Fortranプログラムの翻訳・結合

本処理系を使用してFortranプログラムを翻訳・結合する方法について記述しています。

第3章 Fortranプログラムの実行

本処理系を使用してFortranプログラムを実行する方法について記述しています。

第4章 Fortranシステムの出力情報

Fortran原始プログラムの翻訳時および実行時に、本処理系が出力する各種の情報について記述しています。

第5章 データの型、種別、および内部表現

本処理系で使用できるデータの型、種別、および内部表現について記述しています。

第6章 プログラミング上の注意

本処理系を用いてプログラミングを行う場合の注意事項について記述しています。入出力文については、“第7章 入出力処理”に記述しています。

第7章 入出力処理

Fortranの入出力文が扱うファイルおよび入出力文を使用する際に必要となる情報について記述しています。

第8章 プログラムのデバッグ

Fortranプログラムを実行した際に、エラーが発生した場合、異常終了した場合または利用者の目的とする結果が得られなかった場合、その原因を追求しFortran原始プログラムを修正しなければなりません。ここでは、その原因を追求するために本処理系が提供しているデバッグ機能およびその操作方法について記述しています。

第9章 最適化機能

最適化機能の使用上の注意および最適化機能の活用方法について記述しています。

第10章 モジュール、サブモジュール、およびモジュール引用の注意事項

モジュールおよびサブモジュールを含むプログラムの翻訳時の注意事項について記述しています。

第11章 言語間結合

FortranプログラムとC/C++言語の結合について記述しています。

第12章 並列化機能

本処理系を使用して、Fortranプログラムを並列処理する方法について記述しています。

付録A 制限事項および注意事項

本処理系における制限および注意事項について記述しています。

付録B エンディアン変換コマンド

本処理系が提供するエンディアン変換コマンドについて記述しています。

付録C プリプロセッサ

C言語プリプロセッサおよびFortranプリプロセッサについて記述しています。

付録D GNU互換オプション

本処理系がサポートするGNU互換オプションについて記述しています。

付録E データおよびメモリ領域

本処理系におけるデータおよびメモリ領域の扱いについて記述しています。

付録F 半精度型を使用できる組込み手続および入出力文

半精度型を使用できる組込み手続および入出力文について記述しています。

付録G コードカバレッジ機能

コードカバレッジ機能について記述しています。

付録H 実行時情報出力機能

実行時情報出力機能について記述しています。

付録I ジョブ運用ソフトウェア連携高速化機能の利用

FXシステム向けの高速化機能を利用するためのプログラム翻訳および実行について記述しています。

付録J 富士通OpenMPライブラリ

富士通独自のOpenMPライブラリを使用してFortran言語プログラムを並列処理する方法について記述しています。

本書の注意事項

本書における最適化のプログラム例は、機能の理解を助けるための概念ソースです。したがって、例のプログラムをそのまま抜き出して実行した場合、翻訳時オプションやプログラム中の変数の宣言文などの条件によっては、期待した機能が動作しない場合があります。

本書の表記について

構文表記記号

構文表記記号とは、構文を記述する上で、特別な意味で定められた記号であり、以下のものがあります。

記号名	記号	説明
選択記号	{ }	この記号で囲まれた項目の中から、どれか1つを選択することを表します。
	|	この記号を区切りとして、複数の項目を列挙することを表します。
省略可能記号	[ ]	この記号で囲まれた項目を省略してもよいことを表します。また、この記号は選択記号“{ }”の意味を含みます。
反復記号	...	この記号の直前の項目を繰り返して指定できることを表します。

並列

本書における並列に関する記述は、並列方式の明示的な記述がない場合、スレッド並列を意味します。

略称

本書では、以下のように略して表記します。

略称	説明
OpenMP 3.1	「OpenMP Application Program Interface Version 3.1 July 2011」で定義された仕様であることを表します。
OpenMP 4.0	「OpenMP Application Program Interface Version 4.0 July 2013」で定義された仕様であることを表します。
OpenMP 4.5	「OpenMP Application Programming Interface Version 4.5 November 2015」で定義された仕様であることを表します。
OpenMP 5.0	「OpenMP Application Programming Interface Version 5.0 November 2018」で定義された仕様であることを表します。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

商標

- Arm is trademark or registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
- OpenMPは、OpenMP Architecture Review Boardの商標です。
- Linux(R)は米国及びその他の国におけるLinus Torvaldsの登録商標です。
- そのほか、本マニュアルに記載されている会社名および製品名は、それぞれ各社の商標または登録商標です。
- 本マニュアルに記載されているシステム名、製品名などには、必ずしも商標表示(TM、(R))を付記しておりません。

謝辞

最適化のガイダンスメッセージは、国立研究開発法人 宇宙航空研究開発機構から委託を受けて実施した事業の成果を基にしています。

出版年月および版数

版数	マニュアルコード
2022年 9月 第2.9版	J2UL-2473-02Z0(09)
2022年 3月 第2.8版	J2UL-2473-02Z0(08)
2021年11月 第2.7版	J2UL-2473-02Z0(07)
2021年 7月 第2.6版	J2UL-2473-02Z0(06)
2021年 3月 第2.5版	J2UL-2473-02Z0(05)
2021年 1月 第2.4版	J2UL-2473-02Z0(04)
2020年11月 第2.3版	J2UL-2473-02Z0(03)
2020年 9月 第2.2版	J2UL-2473-02Z0(02)
2020年 6月 第2.1版	J2UL-2473-02Z0(01)
2020年 3月 第2版	J2UL-2473-02Z0(00)
2020年 1月 初版	J2UL-2473-01Z0(00)

著作権表示

変更履歴

変更内容	変更箇所	版数
翻訳時オプション-N{ Rtrap | Rnotrap }の記事を見直しました。	2.2.3	第2.9版
説明文を修正しました。	9.11	
説明文を修正しました。	I.1	
説明文を見直しました。	―	
注意を追加しました。	4.1.1	第2.8版
以下の最適化指示子を追加しました。 ・ [NO]ILFUNC	9.10.4	
以下のオプションの説明を見直しました。 ・ -K{ array_declaration_opt | noarray_declaration_opt } ・ -K{ region_extension | noregion_extension }	2.2.3	第2.7版
「SIMD化可能な組込み関数」を追加しました。	9.1.1.7.5	
以下の最適化指示子の説明を見直しました。 ・ OCL STRIPING[(*n*)] ・ OCL FISSION_POINT[(*n*)]	9.10.4	
説明を追加しました。	9.11	
「表12.1 自動並列化用の最適化指示子一覧」を改善しました。	12.2.6.4	
「次元引継ぎとMPIルーチンの組合せ」を削除しました。	A.1	
説明文を見直しました。	―	
注意を追加しました。	1.2.1	第2.6版
注意を削除しました。	2.1.2	
-K{ array_declaration_opt | noarray_declaration_opt }オプションのデフォルトを-Knoarray_declaration_optに変更しました。	2.2.3	
以下の環境変数を追加しました。 ・ FCOMP_LINK_FJOBJ	2.3	
利用者定義の入出力手続に関する記事を追加しました。	7.18.1	
注意を追加しました。	8.1.2.3	
説明を改善しました。	9.1.1.7.4 9.1.2.1 9.1.2.10	
「言語間結合の概要」の説明を見直し、「結合時の翻訳時オプションと必須オプション」に変更しました。	11.1	
「次元引継ぎとMPIルーチンの組合せ」を追加しました。	A.1.8	
「リンクエラー(undefined reference to)について」を追加しました。	A.2.4	
説明文を見直しました。	―	
以下の翻訳時オプションの説明を見直しました。 ・ -K{ fp_relaxed | nofp_relaxed } ・ -K{ ilfunc[={loop | procedure}] | noilfunc }	2.2.3	第2.5版
-mldefault=cdeclオプション指定時の注意事項を追加しました。	2.2.3 6.6.6 12.3.5.4	

変更内容	変更箇所	版数
zfillの最適化の説明文を見直しました。	2.2.3 9.1.2.5 9.10.4	
以下の環境変数を追加しました。 ・ FLIB_L1_SCCR_CNTL	9.17.2.2 I.2 J.5.4	
説明を追加しました。	9.20	
以下の翻訳時オプションを追加しました。 ・ --linkstl={ libfjc++ | libc++ | libstdc++ }	2.2.1 2.2.2 2.2.3	第2.4版
言語間結合で使用するコマンドと必須オプションの表を追加しました。	11.1	
「C++プログラムとの結合方法」を追加しました。	11.15	
注意を追加しました。	12.2.2.5 12.3.2.1 J.2.2 J.3.2	
説明文を見直しました。	―	
注意を追加しました。	2.1.2	第2.3版
-N{ reordered_variable_stack | noreordered_variable_stack }オプションに関する記事の誤りを修正しました。	2.2.1 2.2.3 E.4	
割付け代入の記事を修正しました。	2.2.3 6.4.2	
-N{ Rtrap | Rnotrap }オプションの記事を修正しました。	2.2.3	
補足シグナルに関する記事の誤りを修正しました。	8.1.7 8.2.2.1	
ループアンローリングの記事を見直しました。	9.1.1.4	
「除数が0の場合の整数除算例外について」を追加しました。	A.2.3	
説明文を見直しました。	―	
-Kmfuncオプションによる組込み関数および演算の展開方法を変更しました。	2.2.3	第2.2版
4倍精度実数型および4倍精度複素数型の境界の記事を修正しました。	5.2	
SIMD ALIGNEDおよびSIMD UNALIGNED指示子の記事を見直しました。	9.10.4 9.18	
プログラム例を改善しました。	9.10.4 9.17.2.1 9.17.2.2	
-Kilfuncまたは-Kmfuncオプション指定時のatan2の演算結果に関する記事の誤りを修正しました。	9.20	
説明文を見直しました。	―	
以下の翻訳時オプションを追加しました。 ・ -K{ array_declaration_opt | noarray_declaration_opt } ・ -K{ extract_stride_store | noextract_stride_store } ・ -K{ fp_precision | nofp_precision } ・ -K{ subscript_opt | nosubscript_opt }	2.2.1 2.2.3	第2.1版

変更内容	変更箇所	版数
・ -Kswp_policy={ auto | small | large }		
-O3オプションで誘導する最適化に、CLONE最適化を追加しました。	2.2.3	
以下の環境変数を追加しました。 ・ FLIB_TRACEBACK_MEM_SIZE	3.8	
jwe1655を追加しました。	4.2.2 8.1.1 8.1.8	
以下の最適化指示子を追加しました。 ・ [NO]ARRAY_DECLARATION_OPT ・ [NO]EXTRACT_STRIDE_STORE ・ [NO]FULLUNROLL_PRE_SIMD[(*n*)] ・ SWP_POLICY({AUTO|SMALL|LARGE})	9.10.4	
最適化指示子PREFETCH_READ、PREFETCH_WRITEの対象に部分配列を追加しました。	9.10.4	
文字型の種別型パラメタの記事を修正しました。	E.1	
説明文を見直しました。	―	
サポートするOpenMP仕様を変更しました。	1.1	第2版
以下のオプションを追加しました。 ・ { -help | --help }	2.1.1 2.2.1 2.2.2 2.2.3	
以下の環境変数を追加しました。 ・ FCOMP_UNRECOGNIZED_OPTION	2.2 2.3	
以下のオプションを追加/改善しました。 ・ -K{ loop_fission_stripmining[={ *N*| *c-level* }] | loop_nofission_stripmining } ・ -Kloop_fission_threshold=*N* ・ -K{ loop_perfect_nest | loop_noperfect_nest } ・ -Koptions ・ -K{ prefetch_stride[=*kind*] | 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	
翻訳時オプション-Knoaliasの機能を拡張しました。	2.2.1 2.2.3 9.7.1	
既定のマクロを追加/変更しました。	2.2.3	
翻訳時オプション-Kilfunc[=*kind*]の*kind*の省略値を変更しました。	2.2.3	
翻訳時オプション-Kloop_fissionの最適化対象ループを変更しました。	2.2.3	
診断メッセージの出力形式に、ループ識別番号を追加しました。	4.1.1	

変更内容	変更箇所	版数
jwe1051、jwe1052を追加しました。	8.1.1 8.1.8	
「ストリップマイニング」を追加しました。	9.1.2.11.1	
以下の最適化指示子を追加しました。 ・ LOOP_[NO]FISSION_STRIPMINING[({ *n* | *c-level* })] ・ LOOP_FISSION_TARGET[({ CL | LS })] ・ LOOP_FISSION_THRESHOLD(*n*) ・ LOOP_[NO]PERFECT_NEST ・ PREFETCH_[NO]CONDITIONAL ・ PREFETCH_[NO]INDIRECT ・ PREFETCH_LINE(*n*) ・ PREFETCH_LINE_L2(*n*) ・ PREFETCH_NOSEQUENTIAL ・ PREFETCH_[NO]STRIDE[({ SOFT | HARD_AUTO | HARD_ALWAYS })] ・ [NO]PRELOAD ・ SCACHE_ISOLATE_ASSIGN(*array1*[,*array2*]...) END_SCACHE_ISOLATE_ASSIGN ・ SCACHE_ISOLATE_WAY(L2=*n1*[,L1=*n2*]) END_SCACHE_ISOLATE_WAY ・ SIMD_[NO]USE_MULTIPLE_STRUCTURES	9.10.4	
以下の最適化指示子を削除しました。 ・ LOOP_[NO]FISSION	9.10.4	
「型引継ぎ」を追加しました。	11.2.9 12.3.4.11	
「次元引継ぎ」を追加しました。	11.2.10 12.3.4.12	
「ISO_Fortran_binding.h」を追加しました。	11.2.11	
LLVM OpenMPライブラリの環境変数OMP_PROC_BINDの省略値を変更しました。	12.2.2.5 12.3.2.1 12.3.3	
以下の環境変数を追加しました。 ・ OMP_MAX_TASK_PRIORITY	12.3.2.2	
以下を追加しました。 ・ TASKLOOP構文 ・ CRITICAL構文	12.3.3	
「実行時ライブラリの定義」に注意事項を追加しました。	12.3.3	
エラー発生時に関する記事を追加しました。	12.3.5.2	
「注意事項」を追加しました。	A.2	
「GNU互換オプション」を追加しました。	付録 D	

変更内容	変更箇所	版数
半精度型を使用できる組込み手続(CSHIFT、EOSHIFT、MAXVAL、MINVAL、NORM2、PACK、PRODUCT、RESHAPE、SPACING、SPREAD、SUM、TRANSFER、UNPACK)を追加しました。	F.1	
「ジョブ運用ソフトウェア連携高速化機能の利用」を追加しました。	付録 I	
説明文を見直しました。	―	
製品のレベルアップに伴い、体裁を変更しました。	―	

目　次

第1章 概要

この章では、本処理系を利用してFortran原始プログラムを処理する場合の概要について説明します。

1.1 本処理系の構成

本処理系は、言語処理プログラムの1つです。本処理系は、以下で構成されます。

翻訳コマンド(Fortranドライバ)

翻訳コマンドは、Fortranコンパイラ、プリプロセッサ、アセンブラ、およびリンカを呼び出すプログラムです。本処理系には、2種類の翻訳コマンドがあります。

種別	翻訳コマンド	説明
クロスコンパイラ	frtpxコマンド	ログインノード上で使用します。
ネイティブコンパイラ	frtコマンド	計算ノード上で使用します。

以降の説明では、翻訳コマンドとしてfrtpxコマンドを用います。ネイティブコンパイラを利用する場合は、適宜、frtpxコマンドをfrtコマンドに読み替えてください。

Fortranコンパイラ

Fortranコンパイラは、翻訳コマンドから呼び出されます。Fortranコンパイラは、Fortran言語で記述された原始プログラムを翻訳して、オブジェクトプログラムを生成するプログラムです。

Fortranライブラリ

Fortranライブラリは、Fortranプログラムを実行するために、Fortranオブジェクトプログラムから呼び出されます。ライブラリの種類には、スタートアップルーチン、Fortranの入出力文、組込み手続、サービスルーチン、および並列処理があります。
本処理系では、並列処理用に2つのライブラリを提供します。

ライブラリ名	説明
LLVM OpenMPライブラリ	オープンソースソフトウェアであるLLVM OpenMP Runtime Libraryをベースにした並列化機能用のライブラリです。サポートしている仕様は、OpenMP 4.5/OpenMP 5.0(一部)です。 LLVM OpenMPライブラリの仕様については、“第12章 並列化機能”をお読みください。
富士通OpenMPライブラリ	京およびPRIMEHPC FX100以前のシステム向け富士通OpenMPライブラリをベースとした並列化機能用のライブラリです。従来の富士通OpenMPライブラリとの互換を重視する場合に適しています。サポートしている仕様は、OpenMP 3.1/OpenMP 4.5(一部)です。 富士通OpenMPライブラリの仕様については、“付録J 富士通OpenMPライブラリ”をお読みください。

補助コマンド

補助コマンドとして、以下を提供しています。

補助コマンド	クロスコンパイラ	ネイティブコンパイラ
入出力データのエンディアン変換用コマンド	fcvendianpx	fcvendian

以降の説明では、補助コマンドとしてfcvendianpxコマンドを用います。ネイティブコンパイラを利用する場合は、適宜、fcvendianpxコマンドをfcvendianコマンドに読み替えてください。

オンラインマニュアル

manコマンドを使用して本処理系の使用方法を検索することができます。

本処理系向けにfrtpx(1)、frt(1)、fcvendianpx(1)、fcvendian(1)、intrinsic(3)、およびservice(3)を提供しています。

1.2 使用方法

ここでは、本処理系使用方法を簡単に説明します。

1.2.1 準備

本処理系を使用するには、以下の環境変数を正しく設定する必要があります。“*製品インストールパス*”は、システム管理者にお問い合わせください。

環境変数名	設定値
PATH	/*製品インストールパス*/bin
LD_LIBRARY_PATH	/*製品インストールパス*/lib64
MANPATH	/*製品インストールパス*/man

設定例:

```
$ PATH=/製品インストールパス/bin:$PATH ; export PATH
$ LD_LIBRARY_PATH=/製品インストールパス/lib64:$LD_LIBRARY_PATH ; export LD_LIBRARY_PATH
$ MANPATH=/製品インストールパス/man:$MANPATH ; export MANPATH
```

注意

環境変数LANGにロケールを指定するときは、翻訳を行うノードにそのロケールがインストールされていることを確認してください。

インストールされているロケールの一覧は“locale -a”コマンドで確認できます。

例:

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

インストールされていない場合、警告メッセージが出力される、または翻訳エラーが発生することがあります。その場合、以下のどちらかの対応をしてください。

- 環境変数LANGにCを指定
- 指定したいロケールのインストールをシステム管理者に依頼

現在設定されているロケールは、“locale”コマンドで確認できます。

例:

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

1.2.2 翻訳とリンク

利用者は、frtpxコマンドにより翻訳およびリンクを行い、実行可能ファイルを作成することができます。

以下は、prog.f95ファイルに格納されたFortranプログラムから実行可能ファイルを作成する例です。

```
$ frtpx prog.f95
```

これにより、実行可能ファイルa.outが作成されます。

翻訳時オプション-cを指定することにより、リンクを行わず、オブジェクトファイルの作成までを行うこともできます。

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

1.2.3 ファイル名

frtpxコマンドには、Fortranソースファイル、アセンブラファイルおよびオブジェクトファイルを指定することができます。ファイル名についての詳細は、“2.1.2 翻訳コマンドの入力ファイル”を参照してください。

Fortranソースファイルは、サフィックスが.f95、.f90、.f、.for、.F95、.F90、.F、.FOR、.f03、.F03、.f08、または.F08であるファイルです。サフィックスが.f95、.f90、.f03、または.f08であるファイルは、自由形式のFortran原始プログラムと解釈されます。サフィックスが.F95、.F90、.F03、または.F08であるファイルは、自由形式の前処理指令Fortran原始プログラムと解釈されます。サフィックスが.fまたは.forであるファイルは、固定形式のFortran原始プログラムと解釈されます。サフィックスが.Fまたは.FORであるファイルは、固定形式の前処理指令Fortran原始プログラムと解釈されます。ソース形式の解釈は、翻訳時オプション-Freeまたは-Fixedで変更することができます。

アセンブラファイルは、サフィックスが.sのファイルです。

オブジェクトファイルは、サフィックスが.oのファイルです。

1.2.4 代表的な翻訳時オプション

本処理系には、多くの翻訳時オプションが存在します。翻訳時オプションの詳細については、“2.2 翻訳時オプション”を参照してください。ここでは、重要な翻訳時オプションの概要を説明します。

-c

オブジェクトファイルの作成までを行います。リンクを行わず、実行可能ファイルは作成されません。

-fw

wレベル(軽度のエラー)および sレベル(重度のエラー)の診断メッセージだけを出力します。

-fs

sレベル(重度のエラー)の診断メッセージだけを出力します。

-f*msg_num*

*msg_num*にメッセージ番号を指定することにより、特定の診断メッセージの出力を抑止します。

-Nmaxserious=*maxnum*

翻訳時に検出された重度のエラーメッセージの数が *maxnum*に達した場合に翻訳を中止することを指定します。

-o*exe_file*

生成される実行可能ファイル名またはオブジェクトファイル名を変更します。

-Fixed

ソースプログラムが固定形式で記述されていることを指示します。

-Free

ソースプログラムが自由形式で記述されていることを指示します。

-H[a | e | f | o | s | u | x]

翻訳時および実行時に引数の整合性、添字式、部分列式の値、未定義な変数の参照、配列式の形状適合などの検査を行うことを指示します。

-M*directory*

モジュール情報を格納するディレクトリを指定します。

-I*directory*

INCLUDEファイルおよびモジュール情報ファイルを検索するディレクトリを指定します。

-O[{ 0 | 1 | 2 | 3 }]

最適化のレベルを指定します。デフォルトは、-O2です。

-Kfast

ターゲットマシン上で高速に実行させることを指示します。

-X6

言語仕様で解釈の異なる部分をFortran 66仕様と解釈します。

-X7

言語仕様で解釈の異なる部分をFortran 77仕様と解釈します。

-X9

言語仕様で解釈の異なる部分をFortran 95仕様と解釈します。

-X03

言語仕様で解釈の異なる部分をFortran 2003仕様と解釈します。解釈は-X08と同じです。

-X08

言語仕様で解釈の異なる部分をFortran 2008仕様と解釈します。

1.2.5 実行

利用者は、frtpxコマンドを使用して作成した実行可能プログラムを実行コマンドとして起動します。

以下は、実行可能ファイルa.outを実行する例です。

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

利用者は、この実行コマンドの起動時にFortranライブラリへ実行処理に関する指示を行うことができます。この使用方法については、“第3章 Fortranプログラムの実行”を参照してください。

また、frtpxまたはfrtコマンドのオンラインマニュアルの中でもこの使用方法が説明されていますので、manコマンドを利用してください。

1.2.6 デバッグ

Fortran原始プログラムのデバッグ方法として、ソースレベルのデバッグと、本処理系が提供するバッチデバッグ機能があります。

ソースレベルのデバッグは、gdbを利用してください。

バッチデバッグ機能については、“8.2 デバッグのための機能”を参照してください。

1.2.7 チューニング

プロファイラを使って、Fortranプログラムをチューニングすることができます。

プロファイラについては、“プロファイラ使用手引書”を参照してください。

第2章 Fortranプログラムの翻訳・結合

この章では、Fortran言語で記述されたプログラムを翻訳および結合するための手順について説明します。

2.1 翻訳コマンド

利用者は、翻訳コマンドfrtpxを使用することにより、原始プログラムの翻訳から実行可能プログラムの作成までを行うことができます。

frtpxコマンドは、frtpxのコマンド行に指定された各種のオペランドを解析し、必要に応じてFortranコンパイラ、cppコマンド(Cプリプロセッサ)、fppコマンド(Fortranプリプロセッサ)、asコマンド(アセンブラ)、およびldコマンド(リンカ)を呼び出します。

2.1.1 翻訳コマンドの形式

利用者は、frtpxコマンドのオペランドとして、Fortranコンパイラ、asコマンド、およびldコマンドに対するオプションの並びとファイル名の並びを指定することができます。

ここで、Fortranコンパイラに対するオプションを翻訳時オプションといいます。翻訳時オプションの指定については、“2.2 翻訳時オプション”を参照してください。

asコマンドおよびldコマンドに対するオプションについては、manコマンドを使用して参照することができます。

以下に、frtpxコマンドの形式を示します。

コマンド名	オペランド
frtpx	[□オプションの並び]□ファイル名の並び

□: 1個以上の空白が必要なことを意味します。

備考1. “オプションの並び”と“ファイル名の並び”の指定順序に制約はありません。例えば、“ファイル名の並び”のあとに“オプションの並び”を指定してもかまいませんし、混在していてもかまいません。

備考2. -Vまたは{ -help | --help }オプションを指定しない場合、“ファイル名の並び”には、1つ以上のファイル名を指定しなければなりません。

備考3. “ファイル名の並び”については、“2.1.2 翻訳コマンドの入力ファイル”を参照してください。

2.1.2 翻訳コマンドの入力ファイル

以下に、frtpxコマンドに入力ファイルとして指定できるファイルを示します。

ファイル種別	ファイルのサフィックス	渡し先
Fortran原始プログラム	.f	Fortranコンパイラ
	.for	
	.f90	
	.f95	
	.f03	
	.f08	
前処理指令Fortran原始プログラム (注)	.F	プリプロセッサ
	.FOR	
	.F90	
	.F95	
	.F03	
	.F08	
アセンブラ原始プログラム	.s	アセンブラ

ファイル種別	ファイルのサフィックス	渡し先
オブジェクトプログラム	.o	リンカ
その他	上記以外	リンカ

注) 前処理指令Fortran原始プログラムとは、C言語の前処理指令(#行)を含んだFortran原始プログラムのことです。

2.2 翻訳時オプション

翻訳時オプションは、frtpxコマンドのオペランド、翻訳時プロフィルファイル、および環境変数FORT90CPXまたはFORT90Cに指定することができます。翻訳時プロフィルファイルについては、"2.4 翻訳時プロフィルファイル"を、環境変数FORT90CPXまたはFORT90Cについては、"2.3 翻訳コマンドの環境変数"をそれぞれ参照してください。

通常、翻訳コマンドが識別不可能なオプションは、警告メッセージが出力されて無視されます。環境変数FCOMP_UNRECOGNIZED_OPTIONを設定することで、識別不可能なオプションに対する動作を変更することができます。環境変数FCOMP_UNRECOGNIZED_OPTIONについては、"2.3 翻訳コマンドの環境変数"を参照してください。

Fortran原始プログラムおよび前処理指令Fortran原始プログラムのファイルサフィックスは、翻訳時オプションに影響します。
以下に、ファイルサフィックスと翻訳時オプションの関係を示します。

ファイルサフィックス	オプション
.f	-X08 -Fixed (注)
.for	
.F	
.FOR	
.f90	-X9 -Free
.f95	
.F90	
.F95	
.f03	-X03 -Free
.F03	
.f08	-X08 -Free
.F08	

注) -Xd7オプションが有効な場合は、-X7 -Fixedになります。

2.2.1 オプション一覧

表2.1 プリプロセッサに関するオプション

機能概要	オプション名
C言語向けプリプロセッサの選択を指示します	-Ccpp
Fortran向けプリプロセッサの選択を指示します	-Cfpp
プリプロセッサの呼び出しを指示します	-Cpp
*name*と#define前処理指令のように指定された*tokens*の関連付けを指示します	-D*name*[=*tokens*]
プリプロセッサの結果をファイルに出力することを指示します	-P

機能概要	オプション名
プリプロセッサの結果を標準出力に出力することを指示します	-E
#undef前処理命令と同様に*name*の定義を無効にすることを指示します	-U*name*

表2.2 原始プログラムに関するオプション

機能概要	オプション名
英大文字と英小文字を別の文字として扱うことを指示します	-AU
固定形式原始プログラムとして翻訳を行うことを指示します	-Fixed
自由形式原始プログラムとして翻訳を行うことを指示します	-Free
ソースの長さが255の固定形式原始プログラムとして翻訳を行うことを指示します	-Fwide

表2.3 規格および言語仕様に関するオプション

機能概要	オプション名
文字列中の特殊文字を特殊文字として扱わないことを指示します	-AE
文字列中の特殊文字を特殊文字として扱うことを指示します	-Ae
暗黙の型規則を適用せず、英字と型および型パラメタを空とすることを指示します	-AT
モジュールの暗黙の参照許可属性をPRIVATE属性にすることを指示します	-Ap
関数名IACHAR、ACHAR、IBITS 、およびISHFTC を組込み手続とすることを指示します	-Aw
代入文の右辺の符号なし実定数を左辺の変数の型に拡張して取り出すことを指示します	-Ay
外部手続呼出しの実引数に現れた文字定数の末尾に¥0(ヌル文字)を追加することを指示します	-Az
EXTERNAL 文だけに現れる外部名の扱いを指示します	-N{ allextput | noallextput }
割付け代入の動作を指示します	-N{ alloc_assign | noalloc_assign }
規格チェックを行うことを指示します	-Ncheck_std={ 03d | 03e | 03o | 03s | 08d | 08e | 08o | 08s }
COARRAY仕様を有効にするかどうかを指示します	-N{ coarray | nocoarray }
手続引用の実引数に指定されたスカラ定数の扱いを指示します	-N{ copyarg | nocopyarg }
重なりのある文字代入の動作を指示します	-N{ f90move | nof90move }
MALLOC およびFREE の扱いを指示します	-N{ mallocfree | nomallocfree }

機能概要	オプション名
関数名の解釈を指示します	-N{ obsfun | noobsfun }
OpenMPのPRIVATEなどの指示節に指定された割付け変数が割り付けられているときの、対応するプライベート変数の初期状態を指示します。	-N{ privatealloc | noprivatealloc }
RECURSIVE キーワードの付加を指示します	-N{ recursive | norecursive }
SAVE 文の付加を指示します	-N{ save | nosave }
手続の入口およびALLOCATE文で変数にゼロ値を自動的に設定するかを指示します	-N{ setvalue[=*set_arg*] | nosetvalue } *set_arg*: { { heap | noheap } | { stack | nostack } | { scalar | noscalar } | { array | noarray } | { struct | nostruct } }
言語仕様レベルを指示します	-X{ 6 | 7 | 9 | 03 | 08 }
ファイルサフィックスが.f、.for、.F、または.FORの場合、言語仕様レベルのデフォルトを-X7にすることを指示します	-Xd7
C++プログラムとの結合で必要な標準テンプレートライブラリ(STL)の検索を行うよう指示します。	--linkstl=*stl_kind* *stl_kind*: { libfjc++ | libc++ | libstdc++ }

表2.4 精度に関するオプション

機能概要	オプション名
型変換を指示します	-A{ d | i | q }
	-C{ cdII8 | cI4I8 | cdLL8 | cL4L8 | cdRR8 | cR4R8 | cd4d8 | ca4a8 | cdDR16 | cR8R16 }
浮動小数点数の計算で生じる計算精度(丸め誤差)の検証を指示します	-C*l* 0 ≦ *l* ≦ 15

表2.5 データ割付けに関するオプション

機能概要	オプション名
型に応じた境界調整を行わないことを指示します	-AA
局所的な割付け配列を解放するかどうかを指示します	-N{ freealloc | nofreealloc }
実引数の定数を書込み禁止領域に割り付けるかどうかを指示します	-N{ use_rodata | nouse_rodata }
自動割付け変数をスタック領域に割り付ける順序をコンパイラに指示します	-N{ reordered_variable_stack | noreordered_variable_stack }

表2.6 数学関数に関するオプション

機能概要	オプション名
組込み関数および演算のインライン展開を行うかどうかを指示します	-K{ ilfunc[=*kind*] | noilfunc } *kind*: {loop | procedure}
マルチ関数化を行うかどうかを指示します	-K{ mfunc[=*level*] | nomfunc } *level*: { 1 | 2 | 3 }

機能概要	オプション名
Fortran組込み関数ライブラリを静的に結合した実行可能プログラムを作成するかどうかを指示します	-K{ static_fjlib | nostatic_fjlib }
SSL II、SSL IIスレッド並列機能、およびBLAS/LAPACKの逐次機能を結合することを指示します	-SSL2
SSL II、SSL IIスレッド並列機能、およびBLAS/LAPACKのスレッド並列機能を結合することを指示します	-SSL2BLAMP

表2.7 実行性能および最適化に関するオプション

機能概要	オプション名
最適化レベルを指示します	-O{ 0 | 1 | 2 | 3 }
利用者定義の手続のインライン展開を指示します	-x{ - | *proc_name*[,*proc_name*]... | *stmt_no* | *dat_szK* | dir=*dir_name*[,dir=*dir_name*]... | accept=*accept_arg* | 0 }
各プロセッサ向けのオブジェクトファイルを生成することを指示します	-K{ A64FX | GENERIC_CPU }
共通ブロックに属する変数に対する記憶域への割付け処理において、8バイトの整数型、倍精度実数型、4倍精度実数型、倍精度複素数型、および4倍精度複素数型のデータに対して、8バイトの境界調整を行うかどうかを指示します。	-K{ align_commons | noalign_commons }
ループの先頭アライメントを2の累乗バイトの境界に合わせるかどうかを指示します	-K{ align_loops[=*N*] | noalign_loops } 0 ≦ *N* ≦ 32768
Armv8-A、Armv8.1-A、Armv8.2-A、またはArmv8.3-Aで定義されている命令を利用したオブジェクトファイルを生成することを指示します	-K{ ARMV8_A | ARMV8_1_A | ARMV8_2_A | ARMV8_3_A }
配列に対してパディングを行う(配列の内部に隙間を作る)ことを指示します	-Karraypad_const[=*N*] 1 ≦ *N* ≦ 2147483647
	-Karraypad_expr=*N* 1 ≦ *N* ≦ 2147483647
最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にするかどうかを指示します。	-K{ array_declaration_opt | noarray_declaration_opt }
複数の配列をマージすることを指示します	-Karray_merge
	-Karray_merge_common[=*name*]
	-Karray_merge_local
	-Karray_merge_local_size=*N* 2 ≦ *N* ≦ 2147483647
配列の次元移動を行うことを指示します	-Karray_subscript
	-Karray_subscript_element=*N* 2 ≦ *N* ≦ 2147483647
	-Karray_subscript_elementlast=*N*

機能概要	オプション名
	2 ≦ *N* ≦ 2147483647
	-Karray_subscript_rank=*N* 2 ≦ *N* ≦ 30
	-Karray_subscript_variable='*ary_nm*(*rank*,*rank*[,*rank*]...)' 1 ≦ *rank* ≦ 30
-Karraypad_const、-Karray_merge、および -Karray_subscriptオプションが指定された場合と等価です	-Karray_transform
指示されたプログラムの特徴に合わせて、最適化を調整するかどうかを指示します	-Kassume={ { shortloop | noshortloop } | { memory_bandwidth | nomemory_bandwidth } | { time_saving_compilation | notime_saving_compilation } }
局所変数をAUTOMATIC変数にするかどうかを指示します	-K{ auto | noauto }
自動割付けデータ実体をスタックに割り付けるかどうかを指示します	-K{ autoobjstack | noautoobjstack }
利用者定義の配列関数の引用において、関数結果の領域を呼出し元または呼出し先のどちらで確保するかを指示します	-K{ calleralloc[=*level*] | nocalleralloc } *level*: { 1 | 2 }
共通ブロックの各配列変数領域の間に隙間を作ることを指示します	-Kcommonpad[=*N*] 4 ≦ *N* ≦ 2147483644
演算評価方法の変更を行うかどうかを指示します	-K{ eval | noeval }
tree-height-reduction最適化において、命令の並列性を優先するかどうかを指示します	-K{ eval_concurrent | eval_noconcurrent }
SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開するかどうかを指示します。	-K{ extract_stride_store | noextract_stride_store }
ターゲットマシン上で高速に実行させることを指示します	-Kfast
浮動小数点の丸めモードをアクセスする可能性について指示します	-K{ fenv_access | nofenv_access }
ソースプログラムに対してFloating-Point Multiply-Add/Subtract演算命令を使用した最適化を行うかどうかを指示します	-K{ fp_contract | nofp_contract }
浮動小数点演算の計算誤差が生じないようなオプションの組合せを誘導するかどうかを指示します。	-K{ fp_precision | nofp_precision }
浮動小数点除算またはSQRT関数について、逆数近似演算を行うかどうかを指示します	-K{ fp_relaxed | nofp_relaxed }
浮動小数点演算の単純化を行うかどうかを指示します	-K{ fsimple | nofsimple }
Flush-to-zeroモードを使用するかどうかを指示します	-K{ fz | nofz }
A64FXプロセッサのHPCタグアドレスオーバライド機能を利用するかどうかを指示します	-K{ hpctag | nohpctag }

機能概要	オプション名
INTENT属性を指定した仮引数と結合する実引数をもつ手続呼出しを含むプログラム単位において、その特性を利用する最適化を行うかどうかを指示します	-K{ intentopt | nointentopt }
ループブロッキングの最適化を行うかどうかを指示します	-K{ loop_blocking[=*N*] | loop_noblocking } 2 ≦ *N* ≦ 10000
ループを複数のループに分割するかどうかを指示します	-K{ loop_fission | loop_nofission }
ループ分割時にストリップマイニングの最適化を行うかどうかを指示します	-K{ loop_fission_stripmining[={ *N*| *c-level* }] | loop_nofission_stripmining } 2 ≦ *N* ≦ 100000000 *c-level*: { L1 | L2 }
ループ分割における分割後のループの粒度を決める閾値を指示します	-Kloop_fission_threshold=*N* 1 ≦ *N* ≦ 100
ループ融合の最適化を行うかどうかを指示します	-K{ loop_fusion | loop_nofusion }
ループ交換を実施するかどうかを指示します	-K{ loop_interchange | loop_nointerchange }
ループを分割して部分的にSIMD化するかどうかを指示します	-K{ loop_part_simd | loop_nopart_simd }
不完全多重ループを分割して完全多重ループにするかどうかを指示します	-K{ loop_perfect_nest | loop_noperfect_nest }
ループバージョニングの最適化を行うかどうかを指示します	-K{ loop_versioning | loop_noversioning }
リンク時最適化を行うかどうかを指示します	-K{ lto | nolto }
ポインタ変数が他の変数と結合しないことを指示します	-Knoalias[=*spec*] *spec*: s
最適化制御行を有効にするかどうかを指示します	-K{ ocl | noocl }
手続呼出しにおけるフレームポインタレジスタを保証するかどうかを指示します	-K{ omitfp | noomitfp }
!OPTIONS行を翻訳指示行として扱うことを指示します	-Koptions
手続内のコード領域が1MB以内であるとして扱います	-K{ pc_relative_literal_loads | nopc_relative_literal_loads }
位置独立コード(PIC)の生成を指示します	-K{ pic | PIC }
位置独立コード(PIC)での外部シンボルへのアクセス方法にProcedure Linkage Table(PLT)を使用するかどうかを指示します	-K{ plt | noplt }
不変式の先行評価を行うかどうかを指示します	-K{ preex | nopreex }
prefetch命令の生成方法を指示します	-Kprefetch_cache_level= { 1 | 2 | all }
	-K{ prefetch_conditional | prefetch_noconditional }
	-K{ prefetch_indirect | prefetch_noindirect }
	-K{ prefetch_infer | prefetch_noinfer }

機能概要	オプション名
	-Kprefetch_iteration=*N* 1 ≦ *N* ≦ 10000
	-Kprefetch_iteration_L2=*N* 1 ≦ *N* ≦ 10000
	-Kprefetch_line=*N* 1 ≦ *N* ≦ 100
	-Kprefetch_line_L2=*N* 1 ≦ *N* ≦ 100
	-K{ prefetch_sequential[=*kind*] | prefetch_nosequential } *kind*: { auto | soft }
	-K{ prefetch_stride[=*kind*] | prefetch_nostride } *kind*: { soft | hard_auto | hard_always }
	-Knoprefetch
	-K{ prefetch_strong | prefetch_nostrong }
	-K{ prefetch_strong_L2 | prefetch_nostrong_L2 }
ロード命令を投機実行する最適化を行うかどうかを指示します。	-K{ preload | nopreload }
レジスタ割付け後に命令スケジューリングの最適化を行うかどうかを指示します	-K{ sch_post_ra | nosch_post_ra }
レジスタ割付け前に命令スケジューリングの最適化を行うかどうかを指示します	-K{ sch_pre_ra | nosch_pre_ra }
末尾呼出しの最適化を行うかどうかを指示します	-K{ sibling_calls | nosibling_calls }
SIMD化を行うかどうかを指示します	-K{ simd[=*level*] | nosimd } *level*: { 1 | 2 | auto }
packed-SIMD化を促進するかどうかを指示します	-K{ simd_packed_promotion | simd_nopacked_promotion }
乗算のリダクション演算に対して、SIMD化を行うかどうかを指示します	-K{ simd_reduction_product | simd_noreduction_product }
SVEのベクトルレジスタサイズを指定します	-Ksimd_reg_size={ 128 | 256 | 512 | agnostic }
回転数が不明なループに対して、SIMD化を行うかどうかを指示します	-K{ simd_uncounted_loop | simd_nouncounted_loop }
SVEのLoad Multiple Structures命令およびStore Multiple Structures命令 を利用するかどうかを指示します。	-K{ simd_use_multiple_structures | simd_nouse_multiple_structures }
配列の添え字の近傍データの利用を優先した最適化を行うかどうかを指示します。	-K{ subscript_opt | nosubscript_opt }
ループストライピングの最適化を行なうかどうかを指示します	-K{ striping[=*N*] | nostriping } 2 ≦ *N* ≦ 100
SVEを利用するかどうかを指示します	-K{ SVE | NOSVE }
ソフトウェアパイプライニングの最適化を行うかどうかを指示します	-K{ swp | noswp }

機能概要	オプション名
ソフトウェアパイプライニングのレジスタ数に関する条件を変更することを指示します	-K{ swp_freg_rate=*N*| swp_ireg_rate=*N*| swp_preg_rate=*N* }
ソフトウェアパイプライニングで使用する命令スケジューリングアルゴリズムの選択基準を指示します。	-Kswp_policy={ auto | small | large }
ソフトウェアパイプライニング適用の条件を緩和し、ソフトウェアパイプライニングを促進することを指示します	-Kswp_strong
ソフトウェアパイプライニング適用の条件を調整し、実行文の重なりを小さくしたソフトウェアパイプライニングを適用することを指示します	-Kswp_weak
スレッド・ローカル・ストレージ(Thread-Local Storage)へのアクセスに必要なオフセットのサイズを指定します。	-Ktls_size={ 12 | 24 | 32 | 48 }
配列演算の途中結果をスタックに割り付けるかどうかを指示します	-K{ temparraystack | notemparraystack }
ループアンローリングの最適化を行うかどうかを指示します	-K{ unroll[=*N*] | nounroll } 2 ≦ *N* ≦ 100
アンロールアンドジャムの最適化を行うかどうかを指示します	-K{ unroll_and_jam[=*N*] | nounroll_and_jam } 2 ≦ *N* ≦ 100
zfillの最適化を行うかどうかを指示します	-K{ zfill[=*N*] | nozfill } 1 ≦ *N* ≦100

表2.8 並列機能に関するオプション

機能概要	オプション名
ループ内の配列のプライベート化を行うかどうかを指示します	-K{ array_private | noarray_private }
多重ループにおいて、並列効果を意識して、外側ループと内側ループのどちらで並列実行するかを動的に選択するかどうかを指示します	-K{ dynamic_iteration | nodynamic_iteration }
引数 *pgm_nm*で指定された手続が並列化されたDOループ内で引用されても、逐次実行のときと動作が変わらないことを指示します	-Kindependent=*pgm_nm*
自動並列化において、実行時のスレッド数を*N*に想定したオブジェクトプログラムを出力することを指示します	-Kinstance=*N* 2 ≦ *N* ≦ 512
ループを分割して部分的に自動並列化するかどうかを指示します	-K{ loop_part_parallel | loop_nopart_parallel }
OpenMP仕様の指示文を有効にするかどうかを指示します	-K{ openmp | noopenmp }
OpenMPのDO指示文が指定されたループに対して、チャンク内の配列要素が回転を跨いで定義引用されないと解釈し、最適化を促進するかどうかを指示します	-K{ openmp_assume_norecurrence | openmp_noassume_norecurrence }
OpenMPのループ構文において、最内ループまで含めたCOLLAPSE指示節が指定され、	-K{ openmp_collapse_except_innermost | openmp_nocollapse_except_innermost }

機能概要	オプション名
かつ、実行性能が低下する可能性がある場合に、最内ループをCOLLAPSEの対象から外すかどうかを指示します	
OpenMPのREDUCTION指示節が指定されたリージョンの終わりにおける、リダクション演算の演算順序をスレッド番号順に固定するかどうかを指示します	-K{ openmp_ordered_reduction | openmp_noordered_reduction }
OpenMP仕様のSIMD構文、DECLARE SIMD構文、DECLARE REDUCTION指示文、およびSIMD指示節をもつORDERED構文だけを有効にするかどうかを指示します	-K{ openmp_simd | noopenmp_simd }
自動並列化を行うかどうかを指示します	-K{ parallel | noparallel }
スレッド並列数の変化による実数型または複素数型の演算結果に計算誤差が発生しない実行可能プログラムを作成するか否かを指示します	-K{ parallel_fp_precision | parallel_nofp_precision }
翻訳時に繰返し数が *N*以上と判明しているループのみを並列化の対象とすることを指示します	-Kparallel_iteration=*N* 1 ≦ *N* ≦ 2147483647
並列化効果の見積もりを行わず、自動並列化可能と解析されたループをすべて並列化させることを指示します	-Kparallel_strong
reductionの最適化を行うかどうかを指示します	-K{ reduction | noreduction }
パラレルリージョンの拡大を行うかどうかを指示します	-K{ region_extension | noregion_extension }
マルチスレッドセーフなオブジェクトを生成するかどうかを指示します	-K{ threadsafe | nothreadsafe }
VISIMPACT(Virtual Single Processor by Integrated Multicore Parallel Architecture)用の最適なオブジェクトを生成することを指示します	-Kvisimpact
並列処理に使用するライブラリを指定します。	-N{ libomp | fjomplib }

表2.9 デバッグおよび実行時情報に関するオプション

機能概要	オプション名
デバッグ用の情報をオブジェクトプログラム中に出力するかどうかを指示します	{ -g | -g0 }
逐次プログラムをデバッグすることを指示します	-H{ a | e | f | o | s | u | x }
翻訳時に外部手続の定義と引用との間の手続特性の検査、外部手続の定義と引用仕様本体との間の手続特性の検査および共通ブロックの大きさの検査をプログラム単位にまたがって行うことを指示します	-N{ check_global | nocheck_global }
組込み関数のエラー処理を行うことを指示します	-Ncheck_intrfunc
コードカバレッジ機能を利用するための情報を生成するかどうかを指示します	-N{ coverage | nocoverage }

機能概要	オプション名
プロファイル機能を有効にするかどうかを指示します。	-N{ fjprof | nofjprof }
特定箇所からの呼び出しによるフック機能を利用するかどうかを指示します	-N{ hook_func | nohook_func }
一定時間間隔での呼び出しによるフック機能を利用するかどうかを指示します	-N{ hook_time | nohook_time }
エラーが発生した外部手続の呼出し箇所(行位置)とプログラム割込みが発生した行位置の表示を行うかどうかを指示します	-N{ line | noline }
コードカバレッジ機能使用時に必要な.gcdaファイルの格納先ディレクトリを指示します	-Nprofile_dir=*dir_name*
Fortran原始プログラムをデバッグすることを指示します	-Eg
	-Nquickdbg[=*dbg_arg*] *dbg_arg*: {{ argchk | noargchk } | { subchk | nosubchk } | { undef | undefnan | noundef } | { inf_detail | inf_simple }}
実行時情報を出力するかどうかを指示します	-N{ rt_tune | rt_notune }
	-Nrt_tune_func
	-Nrt_tune_loop[=*kind*] *kind*: { all | innermost }
実行時に組込み演算診断メッセージを出力し、浮動小数点演算の割込み事象を検出するかどうかを指示します	-N{ Rtrap | Rnotrap }

表2.10 翻訳情報に関するオプション

機能概要	オプション名
翻訳時メッセージを制御することを指示します	-f{ i | w | s | *msg_num* }
許容値を認めない演算に関するメッセージの出力を指示します	-Ec
最適化の状況やガイダンスをiレベル(インフォメーション)のメッセージで出力するかどうかを指示します	-K{ optmsg[={ *level* | guide }] | nooptmsg } *level*: { 1 | 2 }
プログラムの翻訳が長時間になると予測される場合に、翻訳を中止するかどうかを指示します	-N{ cancel_overtime_compilation | nocancel_overtime_compilation }
翻訳時に配列の大きさを検査して、実行性能に影響を与える可能性がある場合に、iレベルの診断メッセージを出力することを指示します	-Ncheck_cache_arraysize
翻訳しているファイル名およびプログラム単位名の表示をするかどうかを指示します	-N{ compdisp | nocompdisp }
sレベルのエラー数が *maxnum*に達した場合に翻訳を中止することを指示します	-Nmaxserious=*maxnum*
翻訳時の出力情報を制御します	-Nlst[={ a | d | i | m | p | t | x }]
	-Nlst_out=*file*

表2.11 リンカに関するオプション

機能概要	オプション名
共用オブジェクトを作成することを指示します	-shared
ライブラリ lib*name*.soまたは lib*name*.aを検索します	-l*name*
ラージページ機能の使用を指示します	-K{ largepage | nolargepage }
文字列操作関数において、最適化版のライブラリをリンクするかどうかを指示します	-K{ optlib_string | nooptlib_string }
ライブラリを検索するディレクトリのリストに *directory*を加えることを指示します	-L*directory*

表2.12 その他のオプション

機能概要	オプション名
オブジェクトプログラムだけの作成を指示します	-c
ヘルプ情報の出力を指示します	{ -help | --help }
外部名の加工方法を変更することを指示します	-ml={ cdecl | frt }
	-mldefault={ cdecl | frt }
C主プログラム名を指定します	-mlcmain={ main | MAIN__ }
指定したファイル名 *exe_file*で実行可能プログラムの作成を指示します	-o*exe_file*
INCLUDEファイルまたはモジュール情報を検索するためのディレクトリ名 *directory*を指示します	-I*directory*
実行可能プログラムおよび共有オブジェクトの、コード領域と静的データ領域の最大値を指示します	-Kcmodel={ small | large }
モジュール情報の登録／引用 *directory*を指示します	-M*directory*
プリプロセッサ(前処理)の結果をファイルに出力することを指示します	-P
リンカを抑止しアセンブラソースの生成を指示します	-S
各コマンドのバージョン情報の出力を指示します	-V
指定されたオプションを各 *tool*の引数へ渡すことを指示します	-W*tool*,*arg1*[,*arg2*]... *tool*: { p | 0 | a | l } p: プリプロセッサ 0: コンパイラ a: アセンブラ l: リンカ
frtpxコマンドによって実行されるパスおよびオプションの表示を指示します	-#
	-###

2.2.2 オプションの並び

[-c] [-f*msg_lvl*] [{ -g | -g0 }] [{ -help | --help }] [-l*name*] [-ml=*target*] [-mlcmain=*main_program*] [-mldefault=*target*] [-o*exe_file*] [-shared] [-x*inl_arg*] [-A*obj_arg*] [-C*typ_arg*] [-D*name*[=*tokens*]] [-E] [-E*msg_arg*] [-Fixed] [-Free] [-Fwide] [-H*dbg_arg*] [-I*directory*] [-K*opt_arg*] [-L*directory*] [-M*directory*] [-N*src_arg*] [-O[*opt_lvl*]] [-P] [-S] [-SSL2] [-SSL2BLAMP] [-U*name*] [-V] [-W*tool,arg1*[,*arg2*]...] [-X*lan_lvl*] [-#] [-###] [--linkstl=*stl_kind*]

2.2.3 オプションの説明

以下に翻訳時オプションの形式と意味を説明します。

-c

リンカの呼出しを抑止する機能です。-cオプションが有効な場合、オブジェクトプログラム(サフィックス".o"のファイル)を作成します。実行可能プログラムは作成しません。本オプションは、-Pおよび-Sオプションと同時には指定できません。入力ファイルにオブジェクトファイルだけが指定された場合は処理を行いません。

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

-fオプションは、*msg_lvl*で指定されたエラーレベルの診断メッセージのみを出力する機能です。本オプションの省略値は、-fiです。

診断メッセージにプレフィックスとしてjwd*xxxx*z-*y*を付加します。ここで、*xxxx*はメッセージ番号であり、*z*はメッセージ種別、*y*はエラーレベルです。エラーレベルとして、iレベル(インフォメーション)、wレベル(軽度のエラー)とsレベル(重度のエラー)があります。

翻訳時の診断メッセージについては、“4.1.1 翻訳時の診断メッセージ”を参照してください。

-f*msg_num*オプションを有効にすることにより、特定のエラーメッセージの出力を抑止することができます。*msg_num*は、コンマ(,)を区切り子として、複数個指定することもできますし、他のサブオプションと組み合わせて指定することもできます。

例: 翻訳時オプション -f

```
$ frtpx -fw,1040,2005 a.f90
```

翻訳時に検出されるwレベル(軽度のエラー)以上の診断メッセージの出力を指示し、加えてjwd1040i-wおよびjwd2005i-wのメッセージを抑止します。

i

-fiオプションが有効な場合、翻訳時に検出されるすべての診断メッセージを出力します。

w

-fwオプションが有効な場合、翻訳時に検出されるwレベル(軽度のエラー)以上の診断メッセージを出力します。

s

-fsオプションが有効な場合、翻訳時に検出されるsレベル(重度のエラー)以上の診断メッセージを出力します。

msg_num

-f*msg_num*オプションが有効な場合、*msg_num*に対応する翻訳時の診断メッセージの出力を抑止します。*msg_num*には、エラーレベルがiまたはwの診断メッセージのメッセージ番号が指定できます。

{ -g | -g0 }

デバッガで使用されるデバッグ情報を出力するかどうかを指示します。デフォルトは、-g0です。

-g

本オプションが有効な場合、オブジェクトプログラム中にデバッグ用の情報を出力します。

本オプションを使用することで、利用者はソースレベルによるデバッグを行うことができます。

本オプションが有効、かつ、-O1オプション以上を有効にするオプションを指定しない場合、-O0オプションが有効になります。

本オプションが有効な場合、-Karray_merge_common[=*name*]オプション、-Karray_merge_localオプション、-Karray_subscriptオプション、および-Kltoオプションは無効となります。

デバッガによるデバッグの注意事項については、“A.2.1 デバッガを用いたデバッグについて”を参照してください。

-g0

本オプションが有効な場合、オブジェクトプログラム中にデバッグ用の情報を出力しません。

{ -help | --help }

ヘルプ情報を出力します。

-l*name*

-lオプションは、ライブラリlib*name*.soまたはlib*name*.aを検索します。コマンド行内でのこのオプションの位置は重要です。それは、ライブラリやオブジェクトファイルがコマンド行に現れる順に検索されるためです。本オプションと引数はリンカに渡されます。

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

-mlオプションは、外部手続名の加工方法の変更を行います。*target*には、cdeclまたはfrtが指定できます。本機能については、“11.3 外部名の加工”を参照してください。

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

-mlcmainオプションは、C主プログラム名を指定します。C主プログラム名は、*main_program*に指定された関数名になります。関数名には、mainまたはMAIN__を指定できます。詳細は、“11.5.1 Cのプログラムに最初に制御を渡す方法”を参照してください。

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

-mldefaultオプションは、デフォルトの外部手続名の加工方法の変更を行います。*target*には、cdeclまたはfrtが指定できます。本機能については、“11.3 外部名の加工”を参照してください。

サービスルーチンを使用する時の注意事項は、“6.6.6 サービスルーチン”を参照してください。
OpenMPのランタイムライブラリルーチンを使用する時の注意事項は、“12.3.5.4 ランタイムライブラリルーチン”を参照してください。
-mldefault=cdeclを指定してプロファイラを使用する時の注意事項は、“プロファイラ使用手引書”を参照してください。
数学ライブラリのルーチンの外部手続名は-mldefault=frtのみに対応しています。-mldefault=cdeclを指定してコンパイルしたプログラムから数学ライブラリを呼び出す場合は、ルーチン名の末尾にアンダースコアを付けてください。

-o *exe_file*

-oオプションの引数として指定されたファイル名*exe_file*で実行可能プログラムを作成する機能です。

-oオプションと-cオプションの両方が有効な場合、-oオプションの引数として指定されたファイル名でオブジェクトプログラムを作成します。

-oオプションと-Sオプションの両方が有効な場合、-oオプションの引数として指定されたファイル名でアセンブラ原始プログラムを作成します。

-shared

リンカが共有オブジェクトを作成します。本オプションは、リンカに渡されます。

-x*inl_arg*　　*inl_arg*: { - | *proc_name*[,*proc_name*]... | *stmt_no* | *dat_sz*K | dir=*dir_name*[,dir=*dir_name*]... | accept=*accept_arg* | 0 }

-xオプションは、-Kオプションと同じくオブジェクトプログラムの実行性能向上を追求するため、利用者定義の手続のインライン展開の最適化を指示する機能です。最適化機能は、-xオプションの引数として指定します。-O1オプション以上が有効な場合に意味があります。-xオプションの引数はコンマ(,)を区切り子とし、組み合わせて指定することができます。デフォルトは、-x0です。

この最適化機能は、多くの翻訳時間と翻訳作業領域を必要とします。また、最適化が実施されればされるほどオブジェクトプログラムの大きさが増大します。しかし、実行時間の大幅な短縮が期待できますので、オブジェクトプログラムの実行性能向上を追求したいときに利用してください。

また、利用者は-Koptmsg=2オプションを有効にすることにより、この最適化が行われたかどうかを翻訳時の診断メッセージで知ることができます。

インライン展開については、“9.1.2.1 インライン展開”を参照してください。

-

-x-オプションが有効な場合、コンパイラが処理対象の利用者定義の手続を自動的に決定し、その引用箇所にインライン展開する最適化を行います。

proc_name[,*proc_name*]...

-x*proc_name*[,*proc_name*]...オプションが有効な場合、引数*proc_name*として指定された利用者定義の手続だけに閉じてインライン展開の最適化を行います。引数*proc_name*は利用者定義の手続名であり、引数はコンマ(,)を区切り子とし、複数指定することができます。

一般的に-x-オプションが有効な場合、すべての利用者定義の手続がインライン展開の対象となるため、多くの翻訳処理時間や翻訳作業領域を必要とします。これを最小限に抑えるには、呼出し頻度の高い手続名を選び、それを-xオプションの引数として指定することが必要です。

手続名の指定に加え、実行文の数指定(*stmt_no*)またはローカル配列の大きさ指定(*dat_sz*K)を指定した場合、指定したいずれかの条件を満たす手続がインライン展開の対象となります。

モジュール手続名を指定する場合は、モジュール手続名の前にモジュール名と区切り文字"."を付けなければなりません。

内部手続名を指定する場合は、内部手続名の前に親の手続名と区切り文字"."を付けなければなりません。

モジュール手続内の内部手続名については、モジュール名の指定も必要です。

サブモジュールのモジュール手続および内部手続名は指定できません。

例: 翻訳時オプション -x*proc_name*(内部手続名の指定)

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

stmt_no　　1 ≦ *stmt_no* ≦ 2147483647

-x*stmt_no*オプションが有効な場合、引数*stmt_no*として指定された実行文数以下の利用者定義の手続だけに閉じてインライン展開の最適化を行います。引数*stmt_no*は実行文の数です。

引数として指定できる値は1から2147483647の範囲で指定できます。

*dat_sz*K　　1 ≦ *dat_sz* ≦ 2147483

-xオプションの引数として指定する*dat_sz*は、ローカルな配列の大きさです。引数として指定できる値は 1から2147483までの範囲であり、この引数の値に続けて英字K(単位: Kバイト)を必ず指定してください。

-x*dat_sz*Kオプションが有効な場合、ローカルな配列の合計の大きさが、引数*dat_sz*K以下の大きさの利用者定義の手続だけに閉じてインライン展開の最適化を行います。

利用者定義の手続のインライン展開の最適化を行うと、利用者定義の手続がその引用箇所にインライン展開されるとともに関係するデータも一緒に組み込みます。組み込むデータの中に巨大なローカル配列があるときにオブジェクトプログラムの大きさが膨大になる場合があります。これを回避するときにこの機能を利用してください。

dir=*dir_name*[,dir=*dir_name*]...

-xdir=*dir_name*が指定された場合、利用者定義の手続を呼び出しているプログラム単位と同じ入力原始プログラムのファイルに加え、引数*dir_name*として指定されたディレクトリの直下のファイル内の利用者定義の手続を対象にインライン展開の最適化を行います。本オプションが有効な場合、-x-オプションも有効になります。ただし、ディレクトリ*dir_name*下のファイルでインライン展開の対象となる場合、以下の条件を満たす必要があります。

- サフィックスが.f、.for、.f90、.f95、.f03、または.f08である。かつ、
- -Cppオプションの指定を必要としない。

-xdir=*dir_name*オプションが指定された場合、ディレクトリ*dir_name*配下のファイルは、入力ファイルの翻訳時に有効となった翻訳時オプションで一時的に翻訳され、インライン展開の対象となります。

引数*dir_name*はディレクトリ名であり、サブオプション(dir=*dir_name*)はコンマ(,)を区切り子とし、複数指定することができます。複数指定した場合、すべてのディレクトリ下のファイルをインライン展開の対象とします。

-xdir=*dir_name*オプションは、他のサブオプション(-、*proc_name*、*stmt_no*、*dat_sz*K、accept=*accept_arg*)と組み合わせて指定することも可能です。その場合、指定されたディレクトリ下のすべてのファイルに対して、他のサブオプションを有効にします。

-xdir=*dir_name*オプションを指定した場合、より多くの翻訳処理時間や翻訳作業領域を必要とします。これを最小限に抑えるために、指定するディレクトリ配下には、呼出し頻度の高い手続が存在するファイル群だけを置くような使用方法を推奨します。

*dir_name*には、ワイルドカードを使うことはできません。

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

他のサブオプション(-、*proc_name*、*stmt_no*、*dat_sz*K、dir=*dir_name*)により、インライン展開の対象手続であり、かつ、以下のいずれかの条件の手続に対してインライン展開の最適化を行うかどうかを指示します。

- 親有効域のモジュールが割付け変数をもつモジュール手続
- 割付け変数を参照結合している手続

なお、以下の条件を満たす割付け変数が手続に含まれる場合、一部の最適化機能が行われないことで、実行性能が低下する場合があるので注意が必要です。

1. POINTER属性をもたない派生型である。かつ、
2. 1.の成分にALLOCATABLE属性をもつ派生型がある。かつ、
3. 2.の成分にPOINTER属性をもつ変数がある。

本オプションは、他のサブオプション(-、*proc_name*、*stmt_no*、*dat_sz*K、dir=*dir_name*)のいずれか1つ以上を有効にし、同時に指定する必要があります。単独で指定した場合は無効となります。デフォルトは、-xaccept=nomodule_allocatableです。

module_allocatable

インライン展開を行うことを指示します。

nomodule_allocatable

インライン展開を行わないことを指示します。

例: **翻訳時オプション** -xaccept=module_allocatable

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

0

インライン展開の最適化を行いません。

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

-Aオプションの引数としてA、E、e、T、U、d、i、p、q、w、y、またはzが指定できます。また、-AeTUのように引数を組み合わせて指定することができます。

A

共通ブロックに属する変数および派生型の成分に対する記憶域への割付け処理において、型に応じた境界調整を行わないことを指示します。

データの境界調整については、“5.2 データの正しい境界”を参照してください。

E

文字列の中の特殊文字列を特殊文字列として扱わないことを指示します。デフォルトは、-AEです。

e

文字列の中の特殊文字列を特殊文字列として扱うことを指示します。

T

暗黙の型規則を適用せず、英字と型、および型パラメタとの対応付けを空とします。

IMPLICIT文がある場合には、IMPLICIT文の対応付けを有効とします。

U

原始プログラム内の英小文字と英大文字を別の文字として扱うことを指示します。

以下の注意が必要です。

- 規格合致プログラムとはみなされません。Fortran規格では、英小文字は英大文字と等価な文字です。
- 組込みモジュールの言語要素、サービスサブルーチン、サービス関数、およびマルチ関数の外部手続名を引用する場合、英小文字でなければなりません。

- 組込み手続の名前は、すべての宣言および参照において文字のつづり(英小文字と英大文字)を同じにしてください。同じでない場合は、動作を保証しません。
- IMPLICIT文において、英字範囲並びに指定された英小文字は、英大文字と等価です。
- デバッガを用いてデバッグを行う場合、利用者が定義した手続名は、英小文字と英大文字は非等価になり、変数名は、英小文字と英大文字は等価になります。

d

単精度実数型、単精度複素数型の定数、変数、および関数(組込み関数、文関数、外部関数、モジュール関数、内部関数)を一段高い精度に拡張することを指示します。

-Adオプションが有効な場合、単精度実数型の定数、変数、および関数(組込み関数、文関数、外部関数、モジュール関数、内部関数)を倍精度実数型に、単精度複素数型の定数、変数、および関数(組込み関数、文関数、外部関数、モジュール関数、内部関数)を倍精度複素数型にそれぞれ精度拡張します。

本機能については、“5.3.1 精度の拡張”を参照してください。

本オプションは、-CcR4R8、-CcdRR8、-Ccd4d8、-Cca4a8、-CcdDR16、および-CcR8R16オプションと同時には指定できません。

i

4バイトの整数型のデータを2バイトの整数型に変更することを指示します。-Aiオプションが有効な場合、以下に示す4バイトの整数型のデータを2バイトの整数型に変更します。

- 絶対値が0から32767までの整定数
- I、J、K、L、M、Nで始まる名前の暗黙の型
- INTEGERの型指定における種別指定の省略時の型
- 組込み関数(実引数が1バイトの整数型、4バイトの整数型、および8バイトの整数型を除くINT、NINT、IDNINT、ICHAR、MAX1、MIN1、LEN、およびINDEX)の結果の型

本オプションは、-CcI4I8、-CcdII8、-Ccd4d8、および-Cca4a8オプションと同時には指定できません。

p

モジュールの暗黙の参照許可属性をPRIVATE属性にします。これは、モジュールに参照対象並びのないPRIVATE文を記述するのと同じ効果をもちます。

q

倍精度実数型、倍精度複素数型の定数、変数、および関数(組込み関数、文関数、外部関数、モジュール関数、内部関数)を一段高い精度に拡張することを指示します。

-Aqオプションが有効な場合、倍精度実数型の定数、変数、および関数(組込み関数、文関数、外部関数、モジュール関数、内部関数)を4倍精度実数型に、倍精度複素数型の定数、変数、および関数(組込み関数、文関数、外部関数、モジュール関数、内部関数)を4倍精度複素数型にそれぞれ精度拡張します。

本機能については、“5.3.1 精度の拡張”を参照してください。

本オプションは、-CcR4R8、-CcdRR8、-Ccd4d8、-Cca4a8、-CcdDR16、および-CcR8R16オプションと同時には指定できません。

w

関数名IACHAR、ACHAR、IBITS、およびISHFTCを組込み手続とすることを指示します。

-Awオプションが有効な場合、これらの関数名は、-X6または-X7オプションが有効でも組込み手続となります。

y

代入文の右辺の符号なし実定数を左辺の変数の型に拡張して取り出すことを指示します。

-Ayオプションが有効な場合

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

は、以下のプログラムとして解釈されます。

```
REAL(8) A
A=1.23456789012345D0
```

z

外部手続呼出しの実引数に現れた文字定数の末尾に¥0(ヌル文字)を追加することを指示します。

-Azオプションを有効にすることにより、対応する引数をC言語のstrlen関数の引数にすることができます。

この際、引数として渡される文字型の長さには¥0(ヌル文字)を含みません。

以下のプログラムを-Azオプションを有効にして翻訳しても、実行結果は1234 4となります。

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

-C *typ_arg*　　*typ_arg* : { *l* | cpp | fpp | pp | cdII8 | cI4I8 | cdLL8 | cL4L8 | cdRR8 | cR4R8 | cd4d8 | ca4a8 | cdDR16 | cR8R16 }

-Cオプションの引数として*l*、cpp、fpp、pp、cdII8、cI4I8、cdLL8、cL4L8、cdRR8、cR4R8、cd4d8、ca4a8、cdDR16、またはcR8R16が指定できます。

l　　0 ≦ *l* ≦ 15

浮動小数点数の計算で生じる計算誤差(丸め誤差)が、利用者プログラムの計算結果にどの程度の影響を及ぼしているかを検証することを指示します。

-C*l*オプション(*l*は0から15までの範囲です)が有効な場合、実数型、倍精度実数型、4倍精度実数型、複素数型、倍精度複素数型、および4倍精度複素数型のデータが代入文で設定されるたびに、そのデータの仮数部の下位*l*ビットをゼロにします。複素数型、倍精度複素数型、および4倍精度複素数型のデータの場合は、それぞれ実部と虚部の下位*l*ビットをゼロにします。

本機能については、“5.3.2 精度縮小と誤差の影響分析”を参照してください。

cpp

C言語のプリプロセッサを選択することを指示します。本オプションは単独で指定しても効果はなく、Fortran原始プログラムのサフィックスが.F、.FOR、.F90、.F95、.F03、.F08のいずれかのとき、または、-Cppオプションが有効な場合に効果があります。

本オプションを指定すると、前処理指令Fortran原始プログラム(C言語の前処理指令を含んだ原始プログラム)をC言語のプリプロセッサで前処理します。

fpp

Fortranのプリプロセッサを選択することを指示します。本オプションは単独で指定しても効果はなく、Fortran原始プログラムのサフィックスが.F、.FOR、.F90、.F95、.F03、.F08のいずれかのとき、または、-Cppオプションが有効な場合に効果があります。

本オプションを指定すると前処理指令Fortran原始プログラム(C言語の前処理指令を含んだ原始プログラム)を、Fortran言語仕様を優先したプリプロセッサで前処理します。

仕様の詳細については“付録C プリプロセッサ”を参照してください。

pp

C言語またはFortranのプリプロセッサを呼び出すことを指示します。前処理指令 Fortran原始プログラム(C言語の前処理指令を含んだ原始プログラム)を翻訳する場合に指定します。

デフォルトは、Fortranのプリプロセッサが呼び出されます。C言語のプリプロセッサを呼び出したい場合は、-Cppオプションに加え-Ccppオプションを指定してください。

cdII8

基本整数型を8バイトの整数型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名INT、IFIX、IDINT、IQINT、NINT、IDNINT、およびIQNINTは、実引数に現すことはできません。

本オプションが有効な場合、組込み手続の個別名NINTおよびIDNINTは、ポインタ代入文の右辺式および構造体構成子の成分に現すことはできません。

本オプションは、-Aiオプションと同時には、指定できません。

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

上記は、以下のプログラムとして解釈します。

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

cI4I8

すべての4バイトの整数型を8バイトの整数型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名INT、IFIX、IDINT、IQINT、NINT、IDNINT、およびIQNINTは、実引数に現すことはできません。

本オプションが有効な場合、組込み手続の個別名NINTおよびIDNINTは、ポインタ代入文の右辺式および構造体構成子の成分に現すことはできません。

本オプションは、-Aiオプションと同時には、指定できません。

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

上記は、以下のプログラムとして解釈します。

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

cdLL8

基本論理型を8バイトの論理型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名BTESTは、実引数に現すことはできません。

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

上記は、以下のプログラムとして解釈します。

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

cL4L8

すべての4バイトの論理型を8バイトの論理型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名BTESTは、実引数に現すことはできません。

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

上記は、以下のプログラムとして解釈します。

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

cdRR8

基本実数型および基本複素数型を倍精度実数型および倍精度複素数型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名REAL、FLOAT、SNGL、およびSNGLQは、実引数に現すことはできません。

本オプションは、-Adおよび-Aqオプションと同時には、指定できません。

```
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)
```

上記は、以下のプログラムとして解釈します。

```
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

すべての単精度実数型および単精度複素数型を倍精度実数型および倍精度複素数型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名REAL、FLOAT、SNGL、およびSNGLQは、実引数に現すことはできません。

本オプションは、-Adオプションと同等の機能です。

本オプションは、-Adおよび-Aqオプションと同時には、指定できません。

```
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)
```

上記は、以下のプログラムとして解釈します。

```
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

本オプションは、-CcdII8、-CcdLL8、および-CcdRR8オプションすべてが有効な場合と等価です。

```
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)
```

上記は、以下のプログラムとして解釈します。

```
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

本オプションは、-CcI4I8、-CcL4L8、および-CcR4R8オプションすべてが有効な場合と等価です。

```
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)
```

上記は、以下のプログラムとして解釈します。

```
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

基本倍精度実数型を4倍精度実数型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名DFLOAT、DBLE、DBLEQ、DREAL、およびDPRODは、実引数に現すことはできません。

本オプションが有効な場合、組込み手続の個別名DPRODは、ポインタ代入文の右辺式および構造体構成子の成分に現すことはできません。

本オプションは、-Adおよび-Aqオプションと同時には、指定できません。

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

上記は、以下のプログラムとして解釈します。

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

cR8R16

すべての倍精度実数型および倍精度複素数型を4倍精度実数型および4倍精度複素数型とすることを指示します。変数、定数、および関数に適用します。

本オプションが有効な場合、組込み手続の個別名DFLOAT、DBLE、DBLEQ、DREAL、およびDPRODは、実引数に現すことはできません。

本オプションが有効な場合、組込み手続の個別名DPRODは、ポインタ代入文の右辺式および構造体構成子の成分に現すことはできません。

本オプションは、-Aqオプションと同等の機能です。

本オプションは、-Adおよび-Aqオプションと同時には、指定できません。

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

上記は、以下のプログラムとして解釈します。

```
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*]

#define前処理指令のように、*name*と*tokens*を関連付けます。*tokens*が指定されない場合は、1を与えます。

-Dオプションと-Uオプションに同一の*name*を指定した場合は、その順序によらず*name*は定義されません。

既定のマクロには、以下のものがあります。

マクロ	値	備考
__FRT_major__	コンパイラのメジャーバージョン番号	
__FRT_minor__	コンパイラのマイナーバージョン番号	
__FRT_patchlevel__	コンパイラのパッチレベル番号	
__FRT_version__	コンパイラのバージョンを表す文字列	
__FUJITSU	1	
__arch64__	1	
__unix	1	
__linux	1	
unix	1	
linux	1	
_OPENMP	201511	-Kopenmpオプション有効時

-E

-Eオプションは、プリプロセッサ(前処理)の結果を標準出力に出力することを指示します。

原始プログラムファイルのサフィックスが.F、.FOR、.F90、.F95、.F03、および.F08でない場合、-Cppオプションを指定しなければなりません。

-Eオプションを指定した場合、プリプロセッサ(前処理)だけが実行されます。

本オプションは、-Pオプションと同時には指定できません。

-E *msg_arg*　　*msg_arg* : { c | g }

許容値を認めない演算の検出メッセージ、プログラム単位間での手続特性のエラー検査をFortranコンパイラに指示することを指示します。-E オプションの引数として、cまたはgが指定できます。また、これらの引数は組み合わせて指定することができます。

c

-Ecオプションが有効な場合、許容値を認めていない関係式(半精度実数型、単精度実数型、倍精度実数型、4倍精度実数型、半精度複素数型、単精度複素数型、倍精度複素数型、または4倍精度複素数型の関係演算で、演算子が==、/=、.EQ.、または.NE.)または算術IF文における算術式が半精度実数型、単精度実数型、倍精度実数型、または4倍精度実数型のときに、翻訳時にiレベルの診断メッセージを出力します。

翻訳時の診断メッセージについては、“4.1.1 翻訳時の診断メッセージ”を参照してください。

許容量を認めていない関係式については、“6.3.2 実数型データの演算”を参照してください。

g

-Egオプションが有効な場合、-Ncheck_global、-Haefosux、および-O0オプションも有効にします。

-Fixed

-Fixedオプションが有効な場合、固定形式のFortran原始プログラムとして翻訳します。

-Free

-Freeオプションが有効な場合、自由形式のFortran原始プログラムとして翻訳します。

-Fwide

-Fwideオプションが有効な場合、ソースの長さが255の固定形式のFortran原始プログラムとして翻訳します。

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

Fortran原始プログラムをデバッグするための機能です。-Hオプションが有効な場合、Fortran原始プログラムの誤りを翻訳時に検査し、かつ、オブジェクトプログラム内にデバッグするために必要な情報を組み込み、実行時に自動検査します。-Hオプションの引数として、a、e、f、o、s、u、またはxが指定できます。また、これらの引数は-Haefosuのように組み合わせて指定することができます。

同様な機能として、-Egオプションを提供します。

-Hオプションが有効な場合、-O0オプションも有効にします。-Oオプションを有効にする場合には、-Hオプションよりあとに-Oオプションを指定する必要があります。

本機能については、“8.2 デバッグのための機能”を参照してください。

a

手続引用において、誤りを検査する機能です。Fortran原始プログラムの翻訳時に、引用元と引用先の妥当性を検査します。また、Fortran原始プログラムの翻訳時に、引用元と引用先の妥当性を検査するオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査します。

-Haオプションが有効な場合、引数や関数結果に関する手続引用の妥当性の検査を行います。誤りを検出した場合は、診断メッセージを出力します。

本機能については、“8.2.1.1 引数の妥当性の検査(ARGCHK機能)”を参照してください。

e

配列式または配列式をもつ代入文において、Fortran原始プログラムの翻訳時に、形状適合を検査するオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査する機能です。

-Heオプションが有効な場合、形状適合の検査を利用者プログラムの実行時に行います。実行時に誤りを検出した場合は、診断メッセージを出力します。

本機能については、“8.2.1.4 形状適合の検査(SHAPECHK機能)”を参照してください。

f

Fortran原始プログラムの翻訳時に、次の検査をするオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査する機能です。誤りを検出した場合は、診断メッセージを出力します。

- 入出力文において1つのファイルを同時に2つ以上の装置に接続しているか検査します。
- 入出力文の実行中に、関数副プログラムの引用を介して、入出力文を呼び出しているか検査します。

本機能については、“8.2.1.7 同時OPENおよびI/Oの再帰呼び出し検査(I/O OPEN機能)”を参照してください。

o

Fortran原始プログラムの翻訳時に、次の検査をするオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査する機能です。誤りを検出した場合は、診断メッセージを出力します。

- 同じ手続の2つの異なる仮引数の実体が重なっている場合、手続の実行中に重なっている部分の値を確定、または、不定にしてはいけません。重なっている部分の値を確定、または、不定にしていないか検査します。
- INTENT(OUT)属性をもつ大きさ引継ぎ配列を引用した場合に、その変数に値が定義されているか検査します。
- モジュールまたはサブモジュールの宣言部および共通ブロックに属さないSAVE属性をもつ変数を引用した場合に、その変数に値が定義されているか検査します。

本機能については、“8.2.1.6 重複仮引数および拡張未定義検査(OVERLAP機能)”を参照してください。

-Hoオプションを指定すると、-Haおよび-Huオプションも同時に指定されたと解釈されます。

-Haについては“8.2.1.1 引数の妥当性の検査(ARGCHK機能)”を、-Huについては“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照してください。

s

部分配列、配列要素、または部分列の引用において、誤りを検査する機能です。

Fortran原始プログラムの翻訳時に、部分配列、配列要素、または部分列の引用範囲の正当性を検査します。また、部分配列、配列要素、または部分列の宣言と引用において、Fortran原始プログラムの翻訳時に、宣言した大きさと引用した範囲の正当性を検査するオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査します。

-Hsオプションが有効な場合、部分配列、配列要素、または部分列の引用で、その添字式や部分列式の値が宣言した範囲内かどうかの検査を行います。誤りを検出した場合は、診断メッセージを出力します。

本機能については、“8.2.1.2 添字式および部分列範囲の検査(SUBCHK機能) ”を参照してください。

u

モジュールまたはサブモジュールの宣言部および共通ブロックに属さない変数の引用において、Fortran原始プログラムの翻訳時に、その変数に値が定義されているかを検査するオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査する機能です。

-Huオプションが有効な場合、モジュールまたはサブモジュールの宣言部および共通ブロックに属さない変数の引用において、その変数に値が定義されているかの検査を利用者プログラムの実行時に行います。実行時に誤りを検出した場合は、診断メッセージを出力します。

本機能については、“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照してください。

x

Fortran原始プログラムの翻訳時に、次の検査をするオブジェクトプログラムを出力し、そのプログラムの実行時に誤りを検査する機能です。誤りを検査した場合は、診断メッセージを出力します。

- モジュールまたはサブモジュールの宣言部および共通ブロックに属する変数を引用した場合、その変数に値が定義されているか検査します。
- ポインタを引用した場合、ポインタが結合されているか検査します。
- DO構文の増分値、FORALL構文の刻み幅、配列構成DO制御の増分値および添字三つ組の刻み幅の値が0かを検査します。

-Hxオプションが有効な場合、-Huオプションも有効にします。

-Hxオプションを有効にする場合、以下の注意が必要です。

- 共通ブロックに属する変数に値を設定しているプログラム単位は、初期値設定プログラム単位を含め、すべて-Hxオプションを指定して翻訳しなければなりません。
- モジュールまたはサブモジュールを-Hxオプションを指定して翻訳した場合、そのモジュールまたはサブモジュールを引用しているすべてのプログラム単位を、-Hxオプションを指定して翻訳しなければなりません。

本機能については、“8.2.1.5 拡張検査(EXTCHK機能)”を参照してください。

-I *directory*

INCLUDEファイルまたはモジュール情報を検索するためのディレクトリ名を指示する機能です。

-I*directory*オプションが有効な場合、引数*directory*のディレクトリ名で検索します。-Iオプションの引数として指定できるディレクトリ名は1つです。ただし、-Iオプションは複数指定でき、検索は-Iオプションが指定された順に行います。

INCLUDEファイルの検索は以下の順序で行います。

1. INCLUDE行が属するファイルが置かれているディレクトリ
2. frtpxコマンドを起動しているカレントディレクトリ
3. -Iオプションの引数に指定されたディレクトリ

モジュール情報の検索は以下の順序で行います。

1. -Mオプションの引数に指定されたディレクトリ
2. frtpxコマンドを起動しているカレントディレクトリ
3. -Iオプションの引数に指定されたディレクトリ

-K*opt_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 | eval_noconcurrent } | { 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=*pgm_nm* | 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 | prefetch_noinfer } | prefetch_iteration=*N* | prefetch_iteration_L2=*N* | prefetch_line=*N* | prefetch_line_L2=*N* | { prefetch_sequential[=*kind*] | 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[=*N*] | nostriping } | { SVE | NOSVE } | { swp | noswp } | { swp_freg_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[=*N*] | nounroll } | { unroll_and_jam[=*N*] | nounroll_and_jam } | visimpact | { zfill[=*N*] | nozfill } }

オブジェクトプログラムの実行性能向上を追求するための最適化を指示する機能です。最適化機能は -Kオプションの引数として指定します。

-Kオプションの引数はコンマ(,)を区切り子とし、組み合わせて指定することができます。

{ align_commons | noalign_commons }

共通ブロックに属する変数に対する記憶域への割付け処理において、8バイトの整数型、倍精度実数型、4倍精度実数型、倍精度複素数型、および4倍精度複素数型のデータに対して、8バイトの境界調整を行うかどうかを指示します。デフォルトは、-Kalign_commonsです。

align_commons

共通ブロックに属する変数に対する記憶域への割付け処理において、8バイトの整数型、倍精度実数型、4倍精度実数型、倍精度複素数型、および4倍精度複素数型のデータに対して、8バイトの境界調整を行います。

noalign_commons

共通ブロックに属する変数に対する記憶域への割付け処理において、8バイトの整数型、倍精度実数型、4倍精度実数型、倍精度複素数型、および4倍精度複素数型のデータに対して、コンパイラが自動的に境界調整を行います。

本オプションを使用する場合、すべてのプログラム単位は-Knoalign_commonsを指定して翻訳されていなければなりません。

{ align_loops[=*N*] | noalign_loops }　　0 ≦ *N* ≦ 32768

ループの先頭アライメントを2の累乗バイトの境界に合わせることを指示します。*N*はループの先頭アライメントのバイト境界の値です。*N*には1から32768までの2の累乗または0が指定できます。*N*の指定を省略した場合または*N*に0を指定した場合、コンパイラが自動的に値を決定します。-O0または-O1オプションが有効な場合のデフォルトは-Knoalign_loops、-O2オプション以上が有効な場合のデフォルトは-Kalign_loopsです。

align_loops[=*N*]

ループの先頭アライメントを2の累乗バイトの境界に合わせることを指示します。*N*の指定を省略した場合または*N*に0を指定した場合、コンパイラが自動的に値を決定します。本オプションは、-O1オプション以上が有効な場合に意味があります。

noalign_loops

ループの先頭アライメントを合わせることを指示しません。-Kalign_loops=1は同じ意味です。

archi

アーキテクチャを指定して、規定される命令を含んだオブジェクトファイルを生成することを指示します。*archi*には、ARMV8_A、ARMV8_1_A、ARMV8_2_A、またはARMV8_3_Aを指定します。デフォルトは、-KARMV8_3_Aです。

ARMV8_A

Armv8-Aで定義されている命令を利用したオブジェクトファイルを生成することを指示します。

ARMV8_1_A

Armv8-AおよびArmv8.1-Aで定義されている命令を利用したオブジェクトファイルを生成することを指示します。

ARMV8_2_A

Armv8-A、Armv8.1-A、およびArmv8.2-Aで定義されている命令を利用したオブジェクトファイルを生成することを指示します。

ARMV8_3_A

Armv8-A、Armv8.1-A、Armv8.2-A、およびArmv8.3-Aで定義されている命令を利用したオブジェクトファイルを生成することを指示します。

arraypad_const[=*N*]　　1 ≦ *N* ≦ 2147483647

1次元目が明示上下限であり、かつその上下限式が定数式の配列に対して*N*要素のパディングを行います。*N*が省略された場合、対象の配列ごとにコンパイラがパディングの量を決めます。パディングとは、配列の内部に隙間を作ることです。

-Karraypad_expr=*N*オプションと同時には指定できません。

本オプションを指定する場合の注意事項については、“9.12 配列の形状を変更する機能の影響”を参照してください。

arraypad_expr=*N*　　1 ≦ *N* ≦ 2147483647

上下限式が定数式かどうかにかかわらず、1次元目が明示上下限の配列に対して *N*要素のパディングを行います。

-Karraypad_const[=*N*]オプションと同時には指定できません。

本オプションを指定する場合の注意事項については、“9.12 配列の形状を変更する機能の影響”を参照してください。

{ array_declaration_opt | noarray_declaration_opt }

最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にするかどうかを指示します。

デフォルトは、-Knoarray_declaration_optです。

array_declaration_opt

最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にします。SIMD化など最適化が促進されます。

前提条件を満足していない場合に本オプションを指定すると、実行結果は保証されません。

本オプションは、-O1オプション以上が有効な場合に意味があります。

noarray_declaration_opt

最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にしません。

array_merge

本オプションは、-Karray_merge_localおよび-Karray_merge_commonオプションが指定された場合と等価です。

本オプションを指定する場合の注意事項については、“9.15 配列変数マージ”を参照してください。

array_merge_common[=*name*]

共通ブロック内の複数配列をマージすることを指示します。-Ncheck_global、-Nquickdbg=argchk、-Nquickdbg=undef、-Nquickdbg=undefnan、-Nquickdbg=subchk、-gまたは-Hオプションが有効な場合は、-Karray_merge_common[=*name*]オプションを無効とします。

*name*には、共通ブロック名を指定できます。

*name*を省略した場合、すべての名前付き共通ブロック内の配列が対象となります。

array_merge_local

ローカル配列の複数配列をマージすることを指示します。-Ncheck_global、-Nquickdbg=argchk、-Nquickdbg=undef、-Nquickdbg=undefnan、-Nquickdbg=subchk、-gまたは-Hオプションが有効な場合は、-Karray_merge_localオプションを無効とします。

-Karray_merge_local_size=1000000も同時に有効になります。

array_merge_local_size=*N*　　2 ≦ *N* ≦ 2147483647

マージ対象とするローカル配列の大きさが、*N*バイト以上であることを指示します。本オプションは、-Karray_merge_localオプションが有効な場合に意味があります。

{ array_private | noarray_private }

ループ内のプライベート化可能な配列に対して、プライベート化するかどうかを指示します。本オプションは、-Kparallelオプションが有効な場合に意味があります。デフォルトは、-Knoarray_privateです。

array_private

ループ内のプライベート化可能な配列に対して、プライベート化を行います。本オプションは、-Kparallelオプションが有効な場合に意味があります。

noarray_private

ループ内のプライベート化可能な配列に対して、プライベート化を行いません。

array_subscript

4次元以上の割付け配列および最終次元の要素が10以下で、最終次元以外の要素が100以上の4次元以上の配列に対して配列の次元移動を行うことを指示します。本オプションを指定する場合の注意事項については、“9.14 配列宣言の次元移動”を参照してください。-Ncheck_global、-Nquickdbg=argchk、-Nquickdbg=undef、-Nquickdbg=undefnan、-Nquickdbg=subchk、-g、または-Hオプションが有効な場合は、-Karray_subscriptオプションを無効とします。

-Karray_subscript_element=100,-Karray_subscript_elementlast=10、および-Karray_subscript_rank=4も同時に有効になります。

array_subscript_element=*N*　　2 ≦ *N* ≦ 2147483647

本オプションが指定された場合、配列の次元移動の対象となる配列の最終次元以外の要素数が*N*以上であることを指示します。本オプションは、-Karray_subscriptオプションが有効な場合に意味があります。ただし、割付け配列では、本オプションの意味はありません。

array_subscript_elementlast=*N*　　2 ≦ *N* ≦ 2147483647

本オプションが指定された場合、配列の次元移動の対象となる配列の最終次元の要素数が*N*以下であることを指示します。本オプションは、-Karray_subscriptオプションが有効な場合に意味があります。ただし、割付け配列では、本オプションの意味はありません。

array_subscript_rank=*N*　　2 ≦ *N* ≦ 30

本オプションが指定された場合、配列の次元移動の対象となる配列の次元数が*N*次元以上であることを指示します。本オプションは、-Karray_subscriptオプションが有効な場合に意味があります。

array_subscript_variable='*ary_nm*(*rank*,*rank*[,*rank*]...)'　　1 ≦ *rank* ≦ 30

'*ary_nm*(*rank*,*rank*[,*rank*]...)'で指定した配列に対して配列の次元移動を行うことを指示します。配列変数名(次元位置,次元位置,...)の形式で指定してください。指定する次元位置の個数は、配列変数名の次元数と同じでなければなりません。次元位置には、移動させたい次元の位置を指定します。このとき、同じ次元位置を2回以上指定することはできません。本オプションは、-Karray_subscriptオプションが有効な場合に意味があります。本オプションが有効な場合、-Karray_subscript_element=*N*、-Karray_subscript_elementlast=*N*、-Karray_subscript_rank=*N*で指定した値は無効になります。

array_transform

本オプションは、-Karraypad_const、-Karray_merge、および-Karray_subscriptオプションが指定された場合と等価です。

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

-Kassumeオプションは、プログラムの特徴に合わせて、最適化を調整するかどうかを指示するためのオプションです。複数の-Kassumeオプションを同時に指定することもできます。

デフォルトは、-Kassume=noshortloop,assume=nomemory_bandwidth,assume=notime_saving_compilationです。

コンパイラは以下の方針で最適化を実施します。

- 最内ループの繰返し数が翻訳時に不明な場合は、繰返し数が大きいとみなす
- 最内ループをメモリバンド幅がボトルネックではないとみなし、CPU演算のボトルネックを優先的に解消する
- 翻訳時間を消費するプログラムでも実行可能プログラムの高速化を優先する

本オプションは、-O1オプション以上が有効な場合に意味があります。

{ shortloop | noshortloop }

プログラム中の最内ループにおいて、ループの繰返し数が翻訳時に不明な場合に、繰返し数が小さいとみなすかどうかを指示します。デフォルトは、-Kassume=noshortloopです。

shortloop

ループの繰返し数が翻訳時に不明なときは、繰返し数が小さいとみなします。-Kassume=shortloopオプションを指定した場合は、自動並列化、ループアンローリング、ソフトウェアパイプライニングなどの最適化が調整または抑止される可能性があります。

noshortloop

ループの繰返し数が大きいとみなして最適化を実施します。

{ memory_bandwidth | nomemory_bandwidth }

プログラム中の最内ループにおいて、メモリバンド幅がボトルネックになるとみなすかどうかを指示します。デフォルトは、-Kassume=nomemory_bandwidthです。

memory_bandwidth

メモリバンド幅がボトルネックになるとみなして最適化を実施します。-Kassume=memory_bandwidthオプションを指定した場合は、zfillの最適化の促進およびソフトウェアパイプライニングなどの最適化が調整または抑止される可能性があります。

nomemory_bandwidth

メモリバンド幅がボトルネックにならないとみなして最適化を実施します。

{ time_saving_compilation | notime_saving_compilation }

プログラムの翻訳時間が短くなるように最適化を調整するかどうかを指示します。デフォルトは、-Kassume=notime_saving_compilationです。

time_saving_compilation

プログラムの翻訳時間が短くなるように最適化を調整します。

notime_saving_compilation

プログラムの翻訳時間の高速化より、実行可能プログラムの高速化を優先します。

{ auto | noauto }

SAVE属性をもつ変数および初期値をもつ変数を除く局所変数を、AUTOMATIC変数にするかどうかを指示します。AUTOMATIC変数は、手続の終了時に不定になります。デフォルトは、-Kautoです。

auto

SAVE属性をもつ変数および初期値をもつ変数を除く局所変数を、AUTOMATIC変数として扱い、スタックに割り付けます。ただし、-Kopenmpオプションが有効な場合、主プログラムの局所変数は、スタックに割り付けられません。本オプションを指定する場合の注意事項については、“9.11 スタック割付けの影響”を参照してください。

noauto

SAVE属性をもつ変数および初期値をもつ変数を除く局所変数を、AUTOMATIC変数にしません。

本オプションを-Kopenmpオプションと同時に指定する場合は、-Kopenmpオプションのあとに指定する必要があります。本オプションは、-Knothreadsafeオプションと同時に指定すると、指定した手続についてスタック領域の使用を小さくする効果があります。本オプションが指定できる手続は、主プログラムまたはPARALLELリージョンの外で呼び出される副プログラムでなければならず、また、以下のものを除くOpenMP指示文を含んでいてはなりません。

- プライベート変数の宣言を含まないPARALLEL構文

- THREADPRIVATE指示文

指定できない手続に対して指定した場合は、プログラムが正しく実行されないことがあるので注意してください。

{ autoobjstack | noautoobjstack }

自動割付けデータ実体をスタックまたはヒープのどちらに割り付けるかを指示します。デフォルトは、-Kautoobjstackです。

autoobjstack

自動割付けデータ実体をスタックに割り付けることを指示します。本オプションを指定する場合の注意事項については、“9.11 スタック割付けの影響”を参照してください。

noautoobjstack

自動割付けデータ実体をヒープに割り付けることを指示します。

{ callerallocl[=*level*] | nocallerallocl }　　*level*: { 1 | 2 }

利用者定義の配列関数の引用において、関数結果の領域を呼出し元または呼出し先のどちらで確保するかを指示します。デフォルトは、-Knocallerallocです。

*level*には、1または2が指定できます。*level*の省略値は1です。*level*により条件に一致した関数結果を呼出し元に確保します。条件を満たさない関数結果は、関数の実行時に動的な領域の割付けを行い、復帰後に呼出し元で解放を行います。

本オプションの指定により、関数結果の領域の動的な割付けおよび解放によって生じる実行時オーバヘッドを削減できる可能性があります。

本オプションを指定する場合には、該当する配列関数を定義または引用するすべてのファイルで指定しなければなりません。該当する配列関数に対して呼出し元と呼出し先で本オプションの有無または *level*の値が異なる場合、正しい動作は保証されません。

callerallocl=1

利用者定義の配列関数の引用において、以下の条件を満たす関数結果は、関数の呼出し元で領域を確保します。

- 関数の結果変数の型は、組込みデータ型である。かつ、

- 関数の結果変数は、形状明示配列であり、すべての明示された上下限並びの宣言式は定数式である。かつ、

- 関数の結果変数が文字型であるとき、その変数の文字長パラメタは定数式である。

callerallocl=2

callerallocl=1の条件に加え、以下の条件を満たす関数結果についても関数の呼出し元で領域を確保します。

- 翻訳時オプション -Karraypad_expr=*N*が無効である。かつ、

- 関数の結果変数の型は、組込みデータ型である。かつ、

- 関数の結果変数が文字型であるとき、その変数の文字長パラメタは、定数式である。かつ、

- 関数の結果変数の宣言において、すべての明示された上下限並びの宣言式は、定数式または組込み関数SIZEの引用のいずれかである。かつ、

- 組込み関数SIZEの実引数は、形状引継ぎ配列の変数名であり、OPTIONAL属性をもたない。かつ、

- 組込み関数SIZEは、第2引数をもたない。

nocallerallocl

利用者定義の配列関数の引用において関数結果の領域は、常に関数の実行時に動的な割付けを行います。また、復帰後に呼出し元で解放を行います。

cmodel={ small | large }

実行可能プログラムおよび共有オブジェクトの、コード領域と静的データ領域の最大値を指示します。デフォルトは、-Kcmodel=smallです。

cmodel=small

リンク後のコード領域と静的データ領域の合計が4GB以内になると仮定し、効率の良いオブジェクトプログラムを生成します。

cmodel=large

リンク後のコード領域が4GB以内になると仮定します。静的データ領域の大きさに制約はありません。静的データ領域が大きくリンク時にエラーが発生する場合は、本オプションを指定します。

本オプションが有効な場合、-Kpicおよび-KPICオプションを同時に指定することはできません。

commonpad[=*N*]　　4 ≦ *N* ≦ 2147483644

-Kcommonpadオプションが有効な場合、データキャッシュの使用効率をあげるために共通ブロック宣言した各配列変数の領域の間に、隙間をつくることを指定します。*N*は、4～2147483644の範囲で指定でき、単位はバイトです。*N*を省略した場合は、コンパイラが自動的に最良な値を決定します。

-Kcommonpad オプションを指定する場合、その共通ブロックを引用する他のファイルに対しても、同じ翻訳時オプション-Kcommonpadを指定しなければなりません。

この共通ブロック内の成分がすべて配列の場合に有効です。

cpu

ターゲットのプロセッサを指定します。*cpu*には、A64FXまたはGENERIC_CPUが指定できます。デフォルトは、-KA64FXです。

A64FX

-KA64FXオプションが有効な場合、A64FXプロセッサ向けのオブジェクトファイルを出力するように指示します。本オプションが有効な場合には、-Khpctagオプションが有効になります。

GENERIC_CPU

-KGENERIC_CPUオプションが有効な場合、Armプロセッサ向けのオブジェクトファイルを出力するように指示します。

{ dynamic_iteration | nodynamic_iteration }

多重ループにおいて、外側ループと内側ループの両方で並列化可能な場合、どちらのループで並列実行するかをスレッド並列数を意識して動的に選択するかどうかを指示します。本オプションは、-Kparallelオプションが有効な場合に意味があります。デフォルトは、-Knodynamic_iterationです。

dynamic_iteration

多重ループにおいて、外側ループと内側ループの両方で並列化可能な場合、どちらのループで並列実行するかをスレッド並列数を意識して動的に選択します。なお、対象となるループのネストレベルは3までです。

nodynamic_iteration

多重ループにおいて、外側ループと内側ループの両方で並列化可能な場合、どちらのループで並列実行するかを動的に選択しません。

{ eval | noeval }

ソースプログラムに対して演算評価方法を変更する最適化を行うかどうかを指示します。デフォルトは、-Knoevalです。

-Kevalオプションを指定した場合、実行結果に副作用(計算誤差および実行時の例外発生など)を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

eval

ソースプログラムに対して演算評価方法を変更する最適化を行います。本オプションが有効な場合、-Kfsimpleオプションが有効になります。

また、本オプションおよび-Kparallelオプションが有効な場合には、-Kfsimpleオプションおよび-Kreductionオプションが有効になります。

本オプションおよび-Ksimd[={ 1 | 2 | auto }]オプションが有効な場合には、-Kfsimpleオプションおよび-Ksimd_reduction_productオプションが有効になります。

本オプションは、-O1オプション以上が有効な場合に意味があります。

noeval

ソースプログラムに対して演算評価方法を変更する最適化を行いません。本オプションが有効な場合、-Knofsimpleオプション、-Knoreductionオプション、および-Ksimd_noreduction_productオプションも有効になります。

{ eval_concurrent | eval_noconcurrent }

tree-height-reduction最適化において、命令の並列性を優先するかどうかを指示します。デフォルトは、-Keval_noconcurrentです。tree-height-reduction最適化については、“9.1.2.10 tree-height-reduction最適化”を参照してください。

eval_concurrent

tree-height-reduction最適化において、命令の並列性を優先することを指示します。本オプションは、-O1オプション以上が有効、かつ、-Kevalオプションが有効な場合に意味があります。

ループの繰返し数が小さく、ソフトウェアパイプライニングが動作しない場合に効果が期待できます。

eval_noconcurrent

tree-height-reduction最適化において、命令の並列性を抑え、FMA命令の利用を優先することを指示します。

{ extract_stride_store | noextract_stride_store }

SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開するかどうかを指示します。

デフォルトは、-Knoextract_stride_storeです。

extract_stride_store

スカラ命令に展開します。

本機能が適用されたループでは、“9.1.1.7.4 SIMD長の倍数冗長実行によるSIMD化機能”が抑止されます。

本オプションを指定すると、オブジェクトプログラムサイズおよび翻訳時間が増加することがあります。

本オプションは、-O2オプション以上が有効な場合に意味があります。

noextract_stride_store

スカラ命令に展開しません。

fast

本オプションは、ターゲットマシン上で高速に実行するオブジェクトプログラムを作成することを指示します。

本オプションは、

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

を指定した場合と等価です。

本オプションは、プログラムの翻訳時およびリンク時に指定する必要があります。

本オプションは、-Kevalオプション、-Kfp_contractオプション、-Kfp_relaxedオプション、-Kilfuncオプション、および-Kmfuncオプションの影響により、実行結果に副作用を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

{ fenv_access | nofenv_access }

プログラムが浮動小数点の丸めモードにアクセスする可能性があるかどうか、および、丸めモードがデフォルトであるかどうかを示します。デフォルトは、-Knofenv_accessです。

fenv_access

プログラムが浮動小数点の丸めモードにアクセスする可能性があること、または、丸めモードがデフォルトでないことを示します。

本オプションが有効な場合、いくつかの浮動小数点最適化が抑止されます。

nofenv_access

プログラムが浮動小数点の丸めモードにアクセスしないこと、および、丸めモードがデフォルトであることを示します。

{ fp_contract | nofp_contract }

ソースプログラムに対してFloating-Point Multiply-Add/Subtract演算命令を使用した最適化を行うかどうかを指示します。デフォルトは、-Knofp_contractです。

-Kfp_contractオプションを指定した場合、実行結果に副作用(丸め誤差程度の違い)を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

fp_contract

Floating-Point Multiply-Add/Subtract演算命令を使用した最適化を行います。

本オプションは、-O1オプション以上が有効な場合に意味があります。

nofp_contract

Floating-Point Multiply-Add/Subtract演算命令を使用した最適化を行いません。

{ fp_precision | nofp_precision }

浮動小数点演算の計算誤差が生じないようなオプションの組合せを誘導するかどうかを指示します。デフォルトは、-Knofp_precisionです。

fp_precision

浮動小数点演算の計算誤差が生じないようなオプションの組合せを誘導します。

本オプションは、本オプションを

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

に置き換えた場合と等価です。そのため、本オプションが有効な場合、一部の最適化機能が制限されるので、実行性能が低下する可能性があります。

nofp_precision

浮動小数点演算の計算誤差が生じないようなオプションの組合せを誘導しません。

本オプションは、-Kfp_precisionオプションの指定を無効にしますが、-Kfp_precisionオプションが誘導する個々のオプションを同時に指定していても、その指定には影響を与えません。

{ fp_relaxed | nofp_relaxed }

単精度浮動小数点除算、倍精度浮動小数点除算、およびSQRT関数について、逆数近似演算を行うかどうかを指示します。デフォルトは、-Knofp_relaxedです。

-Kfp_relaxedオプションを指定した場合、実行結果に副作用を生じることがあります。また、-NRtrapオプションが無効な場合でも、プログラムの論理上は発生しない浮動小数点例外を発生させることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

fp_relaxed

単精度または倍精度の浮動小数点除算またはSQRT関数について、逆数近似演算命令とFloating-Point Multiply-Add/Subtract演算命令を利用することを指示します。本オプションは、-O1オプション以上が有効な場合に意味があります。

ただし、-NRtrapオプションが有効、かつ-Knosimdまたは-KNOSVEオプションのいずれかが有効である場合、SQRT関数を逆数近似演算に変換する最適化が抑止されます。したがって、-NRnotrapオプションが有効な場合と比べ、実行性能が低下する場合があります。

nofp_relaxed

単精度または倍精度の浮動小数点除算またはSQRT関数について、通常の除算命令またはSQRT命令を利用することを指示します。

{ fsimple | nofsimple }

ソースプログラムに対して浮動小数点演算の単純化を行うかどうかを指示します。例えば、x*0のような演算は、0に単純化されます。デフォルトは、-Knofsimpleです。

-Kfsimpleオプションを指定した場合、実行結果に副作用を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

fsimple

ソースプログラムに対して浮動小数点演算の単純化を行います。本オプションは、-O1オプション以上が有効な場合に意味があります。

nofsimple

ソースプログラムに対して浮動小数点演算の単純化を行いません。

{ fz | nofz }

flush-to-zeroモードを使用するかどうかを指示します。本オプションは、プログラムのリンク時に指定する必要があります。デフォルトは、-Knofzです。

-Kfzオプションを指定した場合、実行結果に副作用を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

fz

flush-to-zeroモードを使用します。flush-to-zeroモードは、演算結果またはソースオペランドが非正規化数の場合、それらを同符号の0で置き換えます。プログラムによっては、高速な性能を得られる場合があります。本オプションは、-O1オプション以上が有効な場合に意味があります。

nofz

flush-to-zeroモードを使用しません。

{ hpctag | nohpctag }

A64FXプロセッサのHPCタグアドレスオーバライド機能を利用するかどうかを指示します。HPCタグアドレスオーバライド機能を利用することで、セクタキャッシュ機能やハードウェアプリフェッチアシスト機能を有効にできます。本オプションは、-KA64FXオプションが有効な場合に意味があります。また、本オプションは、プログラムの翻訳時およびリンク時に指定する必要があります。デフォルトは、-Khpctagです。

hpctag

HPCタグアドレスオーバライド機能を利用します。

nohpctag

HPCタグアドレスオーバライド機能を利用しません。

{ ilfunc[=*kind*] | noilfunc }　　*kind*: { loop | procedure }

組込み関数および演算をインライン展開するかどうかを指示します。以下に、対象となる組込み関数およびべき乗演算を示します。

- 単精度実数型および倍精度実数型の組込み関数ATAN、ATAN2、COS、EXP、EXP10、LOG、LOG10、SIN、およびTAN
- 単精度複素数型および倍精度複素数型の組込み関数ABSおよびEXP
- 基数と指数が単精度実数型および基数と指数が倍精度実数型の場合のべき乗演算

*kind*には、loopおよびprocedureが指定できます。*kind*の省略値は、procedureです。デフォルトは、-Knoilfuncです。

コンパイラは、性能低下が生じる可能性があると判断した場合、インライン展開を行いません。

本最適化によってインライン展開が行われると、実行結果に副作用を生じることがあります。また、-NRtrapオプションが無効な場合でも、プログラムの論理上は発生しない浮動小数点例外を発生させることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

ilfunc=loop

ループに含まれる組込み関数およびべき乗演算をインライン展開します。ループに含まれない組込み関数およびべき乗演算はインライン展開されません。

本オプションは、-O1オプション以上が有効な場合に意味があります。

ilfunc=procedure

プログラム単位に含まれる組込み関数およびべき乗演算をインライン展開します。

本オプションは、-O1オプション以上が有効な場合に意味があります。

noilfunc

組込み関数およびべき乗演算をインライン展開しません。

independent=*pgm_nm*

本オプションは、引数*pgm_nm*で指定された手続が並列化されたDOループ内で引用されても、逐次実行のときと動作が変わらないことを指示します。これにより、手続引用のあるDOループを自動並列化の対象にします。本オプションは、-Kparallelオプションが有効な場合に意味があります。手続*pgm_nm*は、-Kthreadsafeオプションを指定して翻訳しなければなりません。また、-Kthreadsafeオプションが指定されていない場合、実行結果が保証されないことがあります。モジュール手続名を指定する場合は、モジュール手続名の前にモジュール名と区切り文字“.”を付けなければなりません。内部手続名を指定する場合は、内部手続名の前に親の手続名と区切り文字“.”を付けなければなりません。モジュール手続内の内部手続名については、モジュール名の指定も必要です。サブモジュールのモジュール手続および内部手続名は指定できません。

例: 親手続名 sub内の内部手続名 insubを指定

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

本オプション使用上の注意事項については、最適化指示子INDEPENDENTと同じです。詳細については、“12.2.7.2 並列処理の入れ子での注意”および“12.2.7.4 最適化制御行の使い方の注意”を参照してください。

instance=*N*　　2 ≦ *N* ≦ 512

-Kinstance=*N*オプションは、実行時のスレッド数を指定します。*N*は、2～512の範囲で指定できます。本オプションは、-Kparallelオプションが有効な場合に意味があります。*N*の値が実行時のスレッド数と異なる場合、実行時に実行を打ち切ります。詳細については、“12.2.7.1 翻訳時オプション-Kparallel,instance=N指定時の注意”を参照してください。

{ intentopt | nointentopt }

INTENT属性を指定した仮引数と結合する実引数をもつ手続呼出しを含むプログラム単位において、その特性を利用する最適化を行うかどうかを指示します。-O0オプションが有効な場合、デフォルトは-Knointentoptです。-O1オプション以上が有効な場合、デフォルトは-Kintentoptです。

intentopt

INTENT属性を指定した仮引数と結合する実引数をもつ手続呼出しを含むプログラム単位において、その特性を利用する最適化を行います。本オプションは、-O1オプション以上が有効な場合に意味があります。

nointentopt

INTENT属性を指定した仮引数と結合する実引数をもつ手続呼出しを含むプログラム単位において、その特性を利用する最適化を行いません。

{ largepage | nolargepage }

ラージページ機能を使用する実行可能プログラムを作成するかどうかを指示します。本オプションはプログラムのリンク時に指定する必要があります。デフォルトは、-Klargepageです。

largepage

ラージページ機能を使用する実行可能プログラムを作成します。

nolargepage

ラージページ機能を使用しない実行可能プログラムを作成します。

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

ループブロッキングの最適化を行うかどうかを指示します。*N*には、ブロックの大きさを2～ 10000の範囲で指定します。*N*の指定を省略した場合、コンパイラが自動的に最良な値を決定します。-O0または-O1オプションが有効な場合、デフォルトは-Kloop_noblockingです。-O2オプション以上が有効な場合、デフォルトは-Kloop_blockingです。

ループブロッキングについては、“9.1.1.5 ループブロッキング”を参照してください。

loop_blocking[=*N*]

ループブロッキングの最適化を行います。本オプションは、-O2オプション以上が有効な場合に意味があります。

loop_noblocking

ループブロッキングの最適化を行いません。

{ loop_fission | loop_nofission }

ソフトウェアパイプライニングの促進、キャッシュメモリ利用効率の改善、およびレジスタ不足の解消のために、ループを複数のループに分割するかどうかを指示します。-O0または-O1オプションが有効な場合、デフォルトは-Kloop_nofissionです。-O2オプション以上が有効な場合、デフォルトは-Kloop_fissionです。

ループ分割の対象ループおよび分割位置は、以下の最適化指示子を用いて指定します。

- 最適化指示子LOOP_FISSION_TARGET

  指定されたループを自動で分割します。

- 最適化指示子FISSION_POINT

  指定された実行文の位置でループを分割します。

最適化指示子LOOP_FISSION_TARGETおよびFISSION_POINTについては、“9.10.4 最適化指示子”を参照してください。なお、最適化指示子を有効にするためには、-Koclオプションを指定する必要があります。

本機能については、“9.1.2.11 ループ分割”を参照してください。

loop_fission

ループを複数のループに分割します。本オプションは、-O2オプション以上が有効な場合に意味があります。本オプションは、上記の最適化指示子と組み合わせて使用した場合にのみ、動作します。

loop_nofission

ループを複数のループに分割しません。

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

自動ループ分割時にストリップマイニングの最適化を行うかどうかを指示します。ストリップの長さを指定する方法は、直接その長さ(*N*)を指示する方法と、各キャッシュ階層(*c-level*)に適したサイズとなるように指示する方法があります。*c-level*には、L1またはL2を指定できます。*N*および*c-level*を省略した場合、コンパイラが自動的に値を決定します。

本最適化により、ループ分割したループ間でアクセスされるデータに対して、キャッシュメモリの利用効率の向上が期待できます。最適化制御行に最適化指示子LOOP_FISSION_TARGETが指定されており、-Kloop_fissionオプション、-Koclオプション、および-O2オプション以上が有効な場合に、本オプションは有効になります。デフォルトは、-Kloop_nofission_stripminingです。

本機能については、“9.1.2.11.1 ストリップマイニング”を参照してください。

loop_fission_stripmining=*N*

ストリップの長さを*N*にします。*N*は、2～100000000の範囲で指定できます。

loop_fission_stripmining=L1

キャッシュメモリの利用効率を考慮し、ストリップの長さを1次キャッシュのサイズに合わせます。

loop_fission_stripmining=L2

キャッシュメモリの利用効率を考慮し、ストリップの長さを2次キャッシュのサイズに合わせます。

loop_nofission_stripmining

ストリップマイニングの最適化を行いません。

loop_fission_threshold=*N*　　1 ≦ *N* ≦ 100

自動ループ分割における分割後のループの粒度(ループ内の命令数やレジスタ数など)を決める閾値*N*を指示します。*N*は1～100の範囲で指定します。*N*の値を小さくした場合、分割後のループが小さくなり、ループの分割数が増える傾向があります。最適化制御行に最適化指示子LOOP_FISSION_TARGETが指定されており、-Kloop_fissionオプション、-Koclオプション、および-O2オプション以上が有効な場合に、本オプションは有効になります。デフォルトは、-Kloop_fission_threshold=50です。

{ loop_fusion | loop_nofusion }

隣接するループを融合するかどうかを指示します。-O2オプション以上が有効な場合、デフォルトは-Kloop_fusionです。

loop_fusion

隣接するループを融合します。本オプションは、-O2オプション以上が有効な場合に意味があります。

loop_nofusion

隣接するループを融合しません。

{ loop_interchange | loop_nointerchange }

ループ交換を実施するかどうかを指示します。-O2オプション以上が有効な場合、デフォルトは-Kloop_interchangeです。

loop_interchange

ループ交換を実施します。本オプションは、-O2オプション以上が有効な場合に意味があります。

loop_nointerchange

ループ交換を実施しません。

{ loop_part_parallel | loop_nopart_parallel }

ループ中に自動並列化できる部分と自動並列化できない部分が存在した場合、そのループを分割して部分的に自動並列化するかどうかを指示します。対象ループは最内ループです。ループを分割することにより、翻訳時間または実行時間が増加する場合があるため注意が必要です。本オプションは、-Kparallelが有効な場合に意味があります。デフォルトは、-Kloop_nopart_parallelです。

loop_part_parallel

ループを分割することにより部分的に自動並列化します。

loop_nopart_parallel

部分的に自動並列化しません。

{ loop_part_simd | loop_nopart_simd }

ループ中にSIMD化できる部分とSIMD化できない部分が存在する場合、そのループを分割して部分的にSIMD化するかどうかを指示します。対象ループは最内ループです。ループを分割することにより、翻訳時間または実行時間が増加する場合があるため注意が必要です。本オプションは、-Ksimd[={ 1 | 2 | auto }]が有効な場合に意味があります。デフォルトは、-Kloop_nopart_simdです。

loop_part_simd

ループを分割することにより部分的にSIMD化します。

loop_nopart_simd

部分的にSIMD化しません。

{ loop_perfect_nest | loop_noperfect_nest }

不完全多重ループを分割して完全多重ループにするかどうかを指示します。本機能により、ループ交換、ループ一重化などの最適化が促進されます。-O2オプション以下が有効な場合のデフォルトは-Kloop_noperfect_nest、-O3オプションが有効な場合のデフォルトは-Kloop_perfect_nestです。

loop_perfect_nest

不完全多重ループを分割して完全多重ループにします。本オプションは、-O2オプション以上が有効な場合に意味があります。

loop_noperfect_nest

不完全多重ループを完全多重ループにしません。

{ loop_versioning | loop_noversioning }

ループバージョニングの最適化を行うかどうかを指示します。

本機能により、SIMD化、ソフトウェアパイプライニング、自動並列化などの最適化が促進されます。

ループバージョニングは、ループを複数生成するため、オブジェクトプログラムサイズおよび翻訳時間が増加する場合があります。また、生成したループを選択するための判定処理がオーバヘッドとなり、実行性能が低下する場合があります。

本オプションは、-O2オプション以上が有効な場合に意味があります。デフォルトは、-Kloop_noversioningです。

ループバージョニングについては、“9.1.2.6 ループバージョニング”を参照してください。

loop_versioning

ループバージョニングの最適化を行います。

loop_noversioning

ループバージョニングの最適化を行いません。

{ lto | nolto }

リンク時最適化を行うかどうかを指示します。-Kltoオプションは、プログラムの翻訳時およびリンク時に指定する必要があります。デフォルトは、-Knoltoです。

-gまたは-Ncoverageオプションが有効な場合、-Kltoオプションは無効となります。また、-Kltoオプションと-xdir=*dir_name*オプションを同時に指定した場合、-xdir=*dir_name*オプションは無効となります。

本機能については、“9.1.2.8 リンク時最適化”を参照してください。

lto

リンク時最適化を行います。本オプションは、-O1オプション以上が有効な場合に意味があります。

nolto

リンク時最適化を行いません。

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

組込み関数および演算をマルチ演算関数に変換する最適化を行うかどうかを指示します。マルチ演算関数とは、1回の呼出しで複数の引数に対する同種の組込み関数計算および演算を行うことにより、実行性能を向上させた関数です。ただし、アルゴリズムが異なることにより実行結果に副作用(精度の違い)を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

本オプションは、-O2オプション以上が有効な場合に意味があります。*level*には、1、2、および3が指定できます。*level*の省略値は1です。デフォルトは、-Knomfuncです。

以下に対象となる組込み関数および演算を示します。

関数名／演算	単精度実数型	倍精度実数型	単精度複素数型	倍精度複素数型	整数型
ACOS	○ (注1)	○ (注1)	○ (注2)	○ (注2)	―
ACOSH	×	×	○ (注2)	○ (注2)	―
ASIN	○ (注1)	○ (注1)	○ (注2)	○ (注2)	―
ASINH	×	×	○ (注2)	○ (注2)	―
ATAN	○	○	○ (注2)	○ (注2)	―
ATAN2	○	○	―	―	―
ATANH	×	×	○ (注2)	○ (注2)	―
COS	○	○	○ (注2)	○ (注2)	―
COSH	× (注3)	× (注3)	○ (注2)	○ (注2)	―
COSQ	×	×	○ (注2)	○ (注2)	―
ERF	○ (注1)	○ (注1)	―	―	―
ERFC	○ (注1)	○ (注1)	―	―	―
EXP	○	○	○ (注2)	○	―
EXP10	○	○	―	―	―
LOG	○	○	○ (注2)	○ (注2)	―
LOG10	○	○	―	―	―
SIN	○	○	○ (注2)	○ (注2)	―
SINH	× (注3)	× (注3)	○ (注2)	○ (注2)	―
SINQ	×	×	○ (注2)	○ (注2)	―
TAN	×	×	○ (注2)	○ (注2)	―
TANH	× (注3)	× (注3)	○ (注2)	○ (注2)	―
べき乗演算	○	○	×	×	× (注3)
ISHFT	―	―	―	―	○ (注2)

○: 対象 ×: 対象外 ―: 組込み関数なし

注1) マルチ演算関数内で演算を逐次実行しているため、通常のマルチ演算関数と比べて実行性能向上率は低いですが、分岐の最適化などで実行性能が向上します。

注2) マルチ演算関数内でスカラ組込み関数を複数回呼び出しています。そのため、マルチ演算関数の単体性能は、スカラ組込み関数を複数回呼び出した場合と変わりませんが、組込み関数を含むループのSIMD化が促進されます。

注3) スカラ組込み関数の複数回呼び出しに展開されます。

mfunc=1

引数の多重度をSIMD長と同じ値としたマルチ演算関数を使用します。

mfunc=2

-Kmfunc=1の機能に加え、多重度をコンパイラが自動的に決定する値としたマルチ演算関数も使用します。本機能は、スタック領域を多く必要とします。

mfunc=3

-Kmfunc=2の機能に加え、IF構文などを含むループに対しても多重度をコンパイラが自動的に決定する値としたマルチ演算関数を使用します。IF構文の真率が低い場合、-Kmfunc=1または-Kmfunc=2より実行性能が低下することがあるので注意が必要です。また、この最適化を行うことで実行時異常終了となる副作用を生じることがあるため注意が必要です。副作用については“9.16.2 翻訳時オプション-Kmfunc=3の影響”を参照してください。本機能は、スタック領域を多く必要とします。

nomfunc

組込み関数および演算をマルチ演算関数に変換する最適化を行いません。

noalias[=*spec*]　　*spec*: s

-Knoaliasオプションは、ポインタ変数またはポインタ成分が他の変数と記憶域を結合しないものとして、最適化を行うことを指示します。

*spec*には、sを指定することができます。*spec*にsを指定した場合、コンパイラはFortran規格のポインタ属性をもつ実体が、他の変数と結合していないものとして最適化を行います。

以下の文脈では、ポインタ属性をもつ実体が他の変数と結合する可能性があります。そのため、実行可能プログラムは誤った結果を出力する可能性があるので、注意が必要です。

- ポインタ代入文
- ポインタ成分をもつ派生型の代入文
- ポインタ成分をもつ派生型がSOURCE=指定子に現れるALLOCATE文
- ポインタ属性またはポインタ成分をもつ仮引数
- ポインタ属性またはポインタ成分をもつ変数への初期設定

noprefetch

prefetch命令を使用したオブジェクトを生成しないことを指示します。

{ ocl | noocl }

最適化制御行を有効にするかどうかを指示します。最適化制御行については、“9.10 最適化制御行(OCL)の利用”を参照してください。デフォルトは、-Knooclです。

ocl

最適化制御行を有効にします。本オプションは、-O1オプション以上が有効な場合に意味があります。

noocl

最適化制御行を無効にします。

{ omitfp | noomitfp }

手続呼出しにおけるフレームポインタレジスタを保証するかどうかを指示します。デフォルトは、-Knoomitfp です。

omitfp

手続呼出しにおけるフレームポインタレジスタを保証しない最適化を行うことを指示します。本オプションが有効な場合、トレースバック情報が保証されません。本オプションは、-O1オプション以上が有効な場合に意味があります。

noomitfp

手続呼出しにおけるフレームポインタレジスタを保証することを指示します。

{ openmp | noopenmp }

OpenMP仕様の指示文を有効にするかどうかを指示します。

本機能については“12.3 OpenMP仕様による並列化”を参照してください。デフォルトは、-Knoopenmpです。

openmp

OpenMP仕様の指示文を有効にします。-Kopenmpオプションが有効な場合、-Kthreadsafeおよび-Kautoオプションも有効にします。ただし、主プログラムの局所変数は、-Kautoオプションが有効であってもスタックには割り付けません。

-Knoautoおよび-Knothreadsafeオプションは、-Kopenmpオプションよりあとに指定すれば有効になります。-Knoautoまたは-Knothreadsafeオプションを-Kopenmpオプションと同時に指定する場合には、注意が必要です。

リンクだけ行う場合でも、-Kopenmpオプションを指定して翻訳されたオブジェクトプログラムが含まれる場合は、本オプションを指定する必要があります。

-O1オプション以下が有効な場合、SIMD構文またはDECLARE SIMD構文が有効であったとしても、SIMD化は行われません。

noopenmp

OpenMP仕様の指示文を注釈として扱います。

{ openmp_assume_norecurrence | openmp_noassume_norecurrence }

OpenMPのDO指示文が指定されたループに対して、チャンク内の配列要素が回転を跨いで定義引用されないと解釈し、最適化を促進するかどうかを指示します。

本機能により、SIMD化、ソフトウェアパイプライニングなどの最適化が促進されます。

本オプションは、最内ループにDO指示文が指定されている場合のみ対象とします。

本オプションは、-Kopenmpオプションおよび-O2オプション以上が有効な場合に意味があります。デフォルトは、-Kopenmp_noassume_norecurrenceオプションです。

openmp_assume_norecurrence

OpenMPのDO指示文が指定されたループに対して、チャンク内の配列要素が回転を跨いで定義引用されないものとして解釈することを指示します。これにより最適化が促進されます。

openmp_noassume_norecurrence

OpenMPのDO指示文が指定されたループに対して、チャンク内の回転順序が回転を跨いで定義引用されないものとして解釈することを指示しません。

{ openmp_collapse_except_innermost | openmp_nocollapse_except_innermost }

OpenMPのループ構文において、以下の条件をすべて満たしている場合に、最内ループをCOLLAPSEの対象から外すか否かを指示します。

- 最内ループまで含めたCOLLAPSE指示節が指定されている。
- 最内ループまで含めたCOLLAPSEにより実行性能が低下する可能性があると、コンパイラが翻訳時に判断できる。

本オプションは、-Kopenmpオプションが有効な場合に意味があります。デフォルトは、-Kopenmp_nocollapse_except_innermostです。

openmp_collapse_except_innermost

最内ループをCOLLAPSEの対象から外します。これにより、COLLAPSEによる実行性能の低下を防止できる場合があります。また、-Koptmsg=2オプションを有効にすることにより、COLLAPSEの対象から外した最内ループを翻訳時の診断メッセージで知ることができます。

openmp_nocollapse_except_innermost

ユーザの指示どおりにCOLLAPSEの展開を行います。

{ openmp_ordered_reduction | openmp_noordered_reduction }

OpenMPのREDUCTION指示節が指定されたリージョンの終わりにおける、リダクション演算の演算順序をスレッド番号順に固定するかどうかを指示します。リダクション演算の演算順序を固定することにより、使用されるスレッド数が同じであれば、常に同じ結果を得ることができます。ただし、スケジュール種別がDYNAMICまたはGUIDEDであるループ構文、またはSECTIONS構文のスケジューリングの影響によって演算順序が変わる場合、丸め誤差程度の違いが生じることがあります。デフォルトは、-Kopenmp_noordered_reductionです。

openmp_ordered_reduction

REDUCTION指示節が指定されたリージョンの終わりにおける、リダクション演算の演算順序をスレッド番号順に固定します。演算順序を固定しない場合に比べて、実行性能が低下する場合があります。本オプションは、-Kopenmp オプションが有効な場合に意味があります。

openmp_noordered_reduction

REDUCTION指示節が指定されたリージョンの終わりにおける、リダクション演算の演算順序を固定しません。

{ openmp_simd | noopenmp_simd }

OpenMP仕様のSIMD構文、DECLARE SIMD構文、DECLARE REDUCTION指示文、およびSIMD指示節をもつORDERED構文だけを有効にするかどうかを指示します。デフォルトは、-Knoopenmp_simdです。

openmp_simd

OpenMP仕様のSIMD構文、DECLARE SIMD構文、DECLARE REDUCTION指示文、およびSIMD指示節をもつORDERED構文だけを有効にします。

OpenMP仕様によるSIMD化だけを行い、並列化は行いません。

-O1オプション以下が有効な場合、SIMD構文またはDECLARE SIMD構文が有効であっても、SIMD化は行われません。

noopenmp_simd

-Kopenmp_simdオプションを無効にします。

options

!optionsで始まる行を翻訳指示行として扱うことを指示します。

翻訳指示行については、“2.6 翻訳指示行”を参照してください。

{ optlib_string | nooptlib_string }

以下の文字列操作関数において、最適化版のライブラリをリンクするかどうかを指示します。

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

本オプションはプログラムのリンク時に指定する必要があります。デフォルトは、-Knooptlib_stringです。

optlib_string

最適化版のライブラリを静的にリンクします。

-Koptlib_stringオプションは、-KSVEおよび-KA64FXのオプションと同時に指定した場合に有効になります。

nooptlib_string

最適化版のライブラリをリンクしません。

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

最適化状況やガイダンスをメッセージ出力するかどうかを指示します。*level*には、1、2が指定できます。*level*およびguideを省略した場合は、1を指定したものと解釈します。デフォルトは、-Koptmsg=1です。

optmsg=1

実行結果に副作用を生じる可能性がある最適化をしたことをメッセージ出力します。

optmsg=2

optmsg=1のメッセージに加えて、自動並列化、SIMD化、ループアンローリング、インライン展開などの最適化機能が動作したことをメッセージ出力します。

optmsg=guide

optmsg=2のメッセージに加えて、以下の最適化が適用できなかった場合の阻害要因と対処方法をガイダンスメッセージとして出力します。

- SIMD化

- 自動並列化

- ソフトウェアパイプライニング

- インライン展開

なお、本オプションに関する注意事項を以下に示します。

- optmsg=2のメッセージ内容で対処方法が判断できる場合は、ガイダンスメッセージは出力されません。

- プログラムによっては、ガイダンスメッセージの対処方法でも期待した効果が得られない場合があります。

nooptmsg

最適化状況およびガイダンスメッセージの出力を抑止します。

{ parallel | noparallel }

自動並列化を行うかどうかを指示します。ただし、並列化の効果が期待できないものに対しては自動並列化を行いません。本オプションは、-O2オプション、-Kregion_extension、-Kloop_part_parallel、および-Kloop_perfect_nestを誘導します。ただし、-O3オプションが有効な場合は、最適化レベルは3となります。

-Hまたは-Egオプションが有効な場合は、-Kparallelオプションを無効とします。デフォルトは、-Knoparallelです。

本機能については“12.2 自動並列化”を参照してください。

リンクだけ行う場合でも、-Kparallelオプションを指定して翻訳されたオブジェクトプログラムが含まれる場合には、本オプションを指定する必要があります。

parallel

自動並列化を行うことを指示します。

noparallel

自動並列化しないことを指示します。

{ parallel_fp_precision | parallel_nofp_precision }

スレッド並列数の変化による浮動小数点型または複素数型の演算結果に計算誤差を考慮し、最適化の適用をコンパイラが制御するかどうかを指示します。

デフォルトは、-Kparallel_nofp_precisionです。

parallel_fp_precision

計算誤差が発生しない範囲の最適化を適用します。

本オプションは、-Kparallelまたは-Kopenmpオプションが有効な場合に意味があります。

本オプションおよび-Kopenmpオプションが有効な場合、-Kopenmp_ordered_reductionオプションが有効となります。

本オプションを指定すると一部の最適化機能が制限されるため、実行性能が低下する可能性があります。

OpenMPのREDUCTION指示節が指定された場合には、本オプションが有効であったとしても、スレッド並列数の変化による実数型または複素数型の演算結果に計算誤差が発生する可能性があるため注意が必要です。

parallel_nofp_precision

スレッド並列数の変化による実数型または複素数型の演算結果に計算誤差が発生する可能性がある実行可能プログラムを作成します。

parallel_iteration=*N*　　1 ≦ *N* ≦ 2147483647

翻訳時に繰返し数が*N*以上と判明しているループのみを並列化の対象とすることを指示します。本オプションは、-Kparallelオプションが有効な場合に意味があります。

例:

```
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)
  ENDDO
ENDDO
```

上記の例では、-Kparallel_iteration=6オプションが有効な場合、外側ループは繰返し数が5のため並列化の対象になりません。

parallel_strong

並列化効果の見積もりを行わず、自動並列化可能と解析されたループをすべて並列化させることを指示します。-Kevalおよび-Kpreexオプションも同時に有効になります。

上記を除き、機能や注意事項は、-Kparallelオプションと同じです。

{ pc_relative_literal_loads | nopc_relative_literal_loads }

手続内のコード領域が1MB以内であるとして扱い、リテラルプールに1命令でアクセスします。本オプションは、翻訳時の指定だけが有効となります。デフォルトは、-Knopc_relative_literal_loadsです。

pc_relative_literal_loads

PC-relative literal loads命令を使用します。

nopc_relative_literal_loads

PC-relative literal loads命令を使用しません。

{ pic | PIC }

位置独立コード(PIC)を生成することを指示します。本オプションは、-Kcmodel=smallオプションと同時に指定したときに有効となります。-Kcmodel=largeオプションが有効な場合、-Kpicおよび-KPICオプションを同時に指定することはできません。本オプションは、翻訳時の指定だけが有効となります。

{ plt | noplt }

位置独立コード(PIC)での外部シンボルへのアクセスにProcedure Linkage Table(PLT)を使用するかどうかを指示します。本オプションは、-Kpicまたは-KPICオプションが有効な場合に意味があります。デフォルトは、-Kpltです。

plt

位置独立コード(PIC)での外部シンボルへのアクセスにProcedure Linkage Table(PLT)を使用します。

noplt

位置独立コード(PIC)での外部シンボルへのアクセスにProcedure Linkage Table(PLT)を使用しません。

{ preex | nopreex }

ソースプログラムに対して不変式の先行評価の最適化を行うかどうかを指示します。なお、この最適化は、プログラムの論理上実行されないはずの命令が実行され、エラーになる場合があります。利用者は、翻訳時の診断メッセージの出力により、この最適化が行われたかどうかを知ることができます。本機能による副作用については、“9.1.2.2 不変式の先行評価”を参照してください。

デフォルトは、-Knopreexです。

preex

ソースプログラムに対して不変式の先行評価の最適化を行います。本オプションは、-O1オプション以上が有効な場合に意味があります。

nopreex

ソースプログラムに対して不変式の先行評価の最適化を行いません。

prefetch_cache_level=*N*　　*N*: { 1 | 2 | all }

どのキャッシュレベルにデータをプリフェッチするかを指示します。本オプションは、-Kprefetch_sequential、-Kprefetch_strideまたは-Kprefetch_indirectオプションが有効な場合に意味があります。*N*には、1、2、およびallが指定できます。デフォルトは、-Kprefetch_cache_level=allです。

prefetch_cache_level=1

データを1次キャッシュにプリフェッチすることを指示します。

prefetch_cache_level=2

データを2次キャッシュにプリフェッチすることを指示します。

prefetch_cache_level=all

データをすべての階層のキャッシュにプリフェッチすることを指示します。各階層へのprefetch命令を組み合わせることにより、より高度なプリフェッチを実現することができます。

{ prefetch_conditional | prefetch_noconditional }

IF構文やCASE構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成するかどうかを指示します。本オプションは、-Kprefetch_sequential、-Kprefetch_stride、または-Kprefetch_indirectオプションが有効な場合に意味があります。デフォルトは、-Kprefetch_noconditionalです。

prefetch_conditional

IF構文やCASE構文に含まれるブロックの中で使用される配列データに対して、prefetch命令を使用したオブジェクトを生成します。

prefetch_noconditional

IF構文やCASE構文に含まれるブロックの中で使用される配列データに対して、prefetch命令を使用したオブジェクトを生成しません。

{ prefetch_indirect | prefetch_noindirect }

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を使用したオブジェクトを生成するかどうかを指示します。本オプションは、-O1オプション以上が有効な場合に意味があります。デフォルトは、-Kprefetch_noindirectです。

prefetch_indirect

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を使用したオブジェクトを生成します。

prefetch_noindirect

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を使用したオブジェクトを生成しません。

{ prefetch_infer | prefetch_noinfer }

プリフェッチする距離が不明な場合でも連続アクセスのprefetch命令を生成するかどうかを指示します。

本オプションは、-Kprefetch_sequential、-Kprefetch_stride、または-Kprefetch_indirectが有効な場合に意味があります。デフォルトは、-Kprefetch_noinferです。

prefetch_infer

プリフェッチの距離が不明な場合でも連続アクセスのプリフェッチを出力することを指示します。

prefetch_noinfer

プリフェッチの距離が不明な場合は、連続アクセスのプリフェッチを出力しないことを指示します。

prefetch_iteration=*N*　　1 ≦ *N* ≦ 10000

prefetch命令を生成する際、ループの*N*回転後に参照または定義されるデータを対象とすることを指示します。SIMD化が適用されるループの場合、*N*にはSIMD化された後のループの繰返し数を指定してください。

ただし、本機能では1次キャッシュにプリフェッチするprefetch命令を対象としますので、-Kprefetch_sequential、-Kprefetch_stride、-Kprefetch_indirectのいずれかのオプションが有効、かつ、-Kprefetch_cache_level=1または-Kprefetch_cache_level=allが有効な場合に意味があります。

-Kprefetch_line=*N*オプションと同時には指定できません。

prefetch_iteration_L2=*N*　　1 ≦ *N* ≦ 10000

prefetch命令を生成する際、ループの*N*回転後に参照または定義されるデータを対象とすることを指示します。SIMD化が適用されるループの場合、*N*にはSIMD化された後のループの繰返し数を指定してください。

ただし、本機能では2次キャッシュにのみプリフェッチするprefetch命令を対象としますので、-Kprefetch_sequential、-Kprefetch_stride、-Kprefetch_indirectのいずれかのオプションが有効、かつ、-Kprefetch_cache_level=2または-Kprefetch_cache_level=allが有効な場合に意味があります。

-Kprefetch_line_L2=*N*オプションと同時には指定できません。

prefetch_line=*N*　　1 ≦ *N* ≦ 100

prefetch命令を生成する際、*N*キャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。

ただし、本機能では1次キャッシュにプリフェッチするprefetch命令を対象としますので、 -Kprefetch_sequentialまたは-Kprefetch_indirectが有効、かつ、-Kprefetch_cache_level=1または-Kprefetch_cache_level=allが有効な場合に意味があります。

-Kprefetch_iteration=*N*オプションと同時には指定できません。

prefetch_line_L2=*N*　　1 ≦ *N* ≦ 100

prefetch命令を生成する際、*N*キャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。

ただし、本機能では 2次キャッシュにプリフェッチするprefetch命令を対象としますので、 -Kprefetch_sequentialまたは-Kprefetch_indirectが有効、かつ、-Kprefetch_cache_level=2または-Kprefetch_cache_level=allが有効な場合に意味があります。

-Kprefetch_iteration_L2=*N*オプションと同時には指定できません。

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

ループ内で使用される連続的にアクセスされる配列データに対して、prefetch命令を使用したオブジェクトを生成するかどうかを指示します。*kind*には、autoおよびsoftが指定できます。*kind*の省略値はautoです。-O0または-O1オプションが有効な場合、デフォルトは-Kprefetch_nosequentialです。-O2オプション以上が有効な場合、デフォルトは-Kprefetch_sequentialです。

prefetch_sequential=auto

ループ内で使用される連続的にアクセスされる配列データに対して、ハードウェアプリフェッチを利用するか、prefetch命令を出力するかをコンパイラが自動的に選択します。本オプションは、-O1オプション以上が有効な場合に意味があります。

prefetch_sequential=soft

ループ内で使用される連続的にアクセスされる配列データに対して、ハードウェアプリフェッチを利用せずに、prefetch命令を出力します。本オプションは、-O1オプション以上が有効な場合に意味があります。

prefetch_nosequential

ループ内で使用される連続的にアクセスされる配列データに対して、prefetch命令を使用せずにオブジェクトを生成します。

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

ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、プリフェッチを実施するかどうかを指示します。プリフェッチするアドレスが翻訳時に確定しないループを含みます。*kind*には、soft、hard_auto、およびhard_alwaysが指定できます。*kind*の省略値はsoftです。デフォルトは、-Kprefetch_nostrideです。

-Kprefetch_stride=softオプションおよび-Kprefetch_nostrideオプションは、-O1オプション以上が有効な場合に意味があります。

-Kprefetch_stride=hard_autoオプションおよび-Kprefetch_stride=hard_alwaysオプションは、-Khpctagオプションおよび-O1オプション以上が有効な場合に意味があります。

prefetch_stride=soft

prefetch命令を生成して、プリフェッチを実施します。

prefetch_stride=hard_auto

ハードウェアストライドプリフェッチャーを利用して、プリフェッチを実施します。

本オプションを指定した場合、キャッシュ上にないデータのみプリフェッチするようストライドプリフェッチャーを設定します。

prefetch_stride=hard_always

ハードウェアストライドプリフェッチャーを利用して、プリフェッチを実施します。

本オプションを指定した場合、-Kprefetch_stride=hard_autoオプションと異なり、常にプリフェッチを行うようストライドプリフェッチャーを設定します。

prefetch_nostride

prefetch命令を生成しません。

{ prefetch_strong | prefetch_nostrong }

1次キャッシュに対するprefetch命令を生成する際、strong prefetch命令を生成するかどうかを指示します。

ただし、本オプションは1次キャッシュにプリフェッチするprefetch命令を対象としますので、-Kprefetch_sequential、-Kprefetch_stride、-Kprefetch_indirectのいずれかのオプションが有効、かつ、-Kprefetch_cache_level=1または-Kprefetch_cache_level=allオプションが有効な場合に意味があります。

デフォルトは、-Kprefetch_strong です。

prefetch_strong

1次キャッシュに対して生成されるprefetch命令をstrong prefetchとします。

prefetch_nostrong

1次キャッシュに対して生成されるprefetch命令をstrong prefetchとしません。

本オプションは、-Khpctagオプションが有効な場合に意味があります。

{ prefetch_strong_L2 | prefetch_nostrong_L2 }

2次キャッシュに対するprefetch命令を生成する際、strong prefetch命令を生成するかどうかを指示します。

ただし、本オプションは2次キャッシュにプリフェッチするprefetch命令を対象としますので、-Kprefetch_sequential、-Kprefetch_stride、-Kprefetch_indirectのいずれかのオプションが有効、かつ、-Kprefetch_cache_level=2 または-Kprefetch_cache_level=allが有効な場合に意味があります。

デフォルトは、-Kprefetch_strong_L2 です。

prefetch_strong_L2

2次キャッシュに対して生成されるprefetch命令をstrong prefetchとします。

prefetch_nostrong_L2

2次キャッシュに対して生成されるprefetch命令をstrong prefetchとしません。

本オプションは、-Khpctagオプションが有効な場合に意味があります。

{ preload | nopreload }

ロード命令を投機実行する最適化を行うかどうかを指示します。本最適化を行うことで、命令スケジューリングの最適化が促進され、実行性能の向上が期待できます。ただし、この最適化は、プログラムの論理上、実行されないはずのロード命令が実行され不当な領域を参照して、プログラムの実行が中断することがあります。利用者は、翻訳時の診断メッセージの出力により、この最適化が行われたかどうかを知ることができます。

デフォルトは、-Knopreloadです。

preload

ロード命令を投機実行する最適化を行います。本オプションは、-O1オプション以上が有効な場合に意味があります。

nopreload

ロード命令を投機実行する最適化を行いません。

{ reduction | noreduction }

リダクションの最適化を行うかどうかを指示します。本オプションは、-Kparallelオプションが有効な場合に意味があります。デフォルトは、-Knoreductionです。

最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

本機能については“12.2.5.8 リダクションによるループスライス”を参照してください。

reduction

リダクションの最適化を行います。

noreduction

リダクションの最適化を行いません。

{ region_extension | noregion_extension }

パラレルリージョンを拡大することにより、並列化オーバヘッドを削減することを指示します。本オプションは、-Kparallelオプションが有効な場合に意味があります。デフォルトは、-Knoregion_extensionです。

本機能については、“12.2.5.11 パラレルリージョンの拡大”を参照してください。

region_extension

パラレルリージョンを拡大することにより、並列化オーバヘッドを削減することを指示します。

並列化効果の小さいループでは、実行性能が低下する場合があります。

noregion_extension

パラレルリージョンを拡大しないことを指示します。

{ sch_post_ra | nosch_post_ra }

レジスタ割付け後に、命令スケジューリングの最適化を行うかどうかを指示します。実行する命令の順序を入れ替えることで、実行性能を向上させます。-Ksch_post_raオプションを指定しても、レジスタのメモリへの退避・復元命令は増加しません。

-O0オプションが有効な場合、デフォルトは-Knosch_post_raです。-O1オプション以上が有効な場合、デフォルトは-Ksch_post_raです。

sch_post_ra

レジスタ割付け後に、命令スケジューリングの最適化を行うことを指示します。本オプションは、-O1オプション以上が有効な場合に意味があります。

nosch_post_ra

レジスタ割付け後に、命令スケジューリングの最適化を行わないことを指示します。

{ sch_pre_ra | nosch_pre_ra }

レジスタ割付け前に、命令スケジューリングの最適化を行うかどうかを指示します。実行する命令の順序を入れ替えることで、実行性能を向上させます。-Ksch_pre_raオプションを指定した場合、レジスタのメモリへの退避・復元命令が増加し、実行性能が低下することがあります。

-O0オプションが有効な場合、デフォルトは-Knosch_pre_raです。-O1オプション以上が有効な場合、デフォルトは-Ksch_pre_raです。

sch_pre_ra

レジスタ割付け前に、命令スケジューリングの最適化を行うことを指示します。本オプションは、-O1オプション以上が有効な場合に意味があります。

nosch_pre_ra

レジスタ割付け前に、命令スケジューリングの最適化を行わないことを指示します。

{ sibling_calls | nosibling_calls }

末尾呼出しの最適化を行うかどうかを指示します。-O0または-O1オプションが有効な場合、デフォルトは-Knosibling_callsです。-O2オプション以上が有効な場合、デフォルトは-Ksibling_callsです。

sibling_calls

末尾呼出しの最適化を行うことを指示します。本オプションが有効な場合、トレースバック情報が保証されません。本オプションは、-O1オプション以上が有効な場合に意味があります。

nosibling_calls

末尾呼出しの最適化を行わないことを指示します。

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

ループ内の命令に対してSIMD命令を利用したオブジェクトを生成するかどうかを指示します。-Ksimd[={ 1 | 2 | auto }]オプションは、-O2オプション以上が有効な場合に意味があります。*level*には、1、2、およびautoが指定できます。*level*の省略値はautoです。-O2オプション以上が有効な場合、デフォルトは-Ksimd=autoです。-Ksimd[={ 1 | 2 | auto }]オプションが有効な場合、-Kloop_part_simdも有効になります。

SIMD化については、“9.1.1.7 SIMD化”を参照してください。

simd=1

SIMD命令を利用したオブジェクトを生成することを指示します。

simd=2

simd=1オプションの機能に加え、IF構文などを含むループに対して、SIMD命令を利用したオブジェクトを生成することを指示します。IF構文内の命令を冗長実行するため、IF構文の真率によっては実行性能が低下する場合があります。また、-Kpreexオプションと同様にIF構文内の式を投機実行するため、プログラムの論理上実行されないはずの命令が実行され、エラーになる場合があります。

simd=auto

ループをSIMD化するかどうかをコンパイラが自動的に判定することを指示します。IF構文を含むループのSIMD化が促進されます。

nosimd

SIMD命令を利用せずにオブジェクトを生成することを指示します。

{ simd_packed_promotion | simd_nopacked_promotion }

単精度実数型ならびに4バイト整数型の配列要素のインデックス計算が4バイトの範囲を超えないと仮定して、packed-SIMD化を促進するかどうかを指示します。本オプションは、-Ksimd[={ 1 | 2 | auto }]が有効な場合に意味があります。

デフォルトは、-Ksimd_nopacked_promotionです。

配列要素のインデックス計算が4バイトの範囲を超える場合、不当な領域をアクセスし、実行時異常終了または実行結果に誤りが生じることがあります。

simd_packed_promotion

packed-SIMD化を促進します。

simd_nopacked_promotion

packed-SIMD化を促進しません。

{ simd_reduction_product | simd_noreduction_product }

乗算のリダクション演算に対して、SIMD化を行うかどうかを指示します。本オプションは、-Ksimd[={ 1 | 2 | auto }]オプションが有効な場合に意味があります。デフォルトは、-Ksimd_noreduction_productです。

本オプションは、-Ksimd_reg_size=agnosticオプションと同時には指定できません。

最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

simd_reduction_product

乗算のリダクション演算に対して、SIMD化を行います。

simd_noreduction_product

乗算のリダクション演算に対して、SIMD化を行いません。

simd_reg_size={ 128 | 256 | 512 | agnostic }

SVEのベクトルレジスタサイズを指定します。本オプションは、-KSVEオプションが有効な場合に意味があります。デフォルトは、-Ksimd_reg_size=512です。

simd_reg_size={ 128 | 256 | 512 }

SVEのベクトルレジスタサイズを指定します。単位はビットです。

翻訳時に、本オプションで指定した値がSVEのベクトルレジスタサイズであるとみなして、最適化を実施します。そのため、最適化が促進されるので、実行性能の向上を期待できます。ただし、生成した実行可能プログラムは、本オプションで指定したサイズのSVEのベクトルレジスタを実装しているCPUアーキテクチャでのみ正常に動作します。

詳細は、“9.21 SVEのベクトルレジスタのサイズを指定する場合の注意事項”を参照してください。

simd_reg_size=agnostic

SVEのベクトルレジスタを特定のサイズとみなさず翻訳を行い、実行時にSVEのベクトルレジスタサイズを決定する実行可能プログラムを作成します。この実行可能プログラムは、CPUアーキテクチャに実装されたSVEのベクトルレジスタサイズによらず実行可能です。ただし、-Ksimd_reg_size={128|256|512}オプションを指定した場合と比べて、実行性能が低下する場合があります。

{ simd_uncounted_loop | simd_nouncounted_loop }

DO WHILEループ、DO UNTILループ、およびループを終了する文を含むDOループ内の命令に対して、SIMD命令を利用したオブジェクトを生成するかどうかを指示します。本オプションは、-Ksimd[={ 1 | 2 | auto }]オプションおよび-KSVEオプションが有効な場合に意味があります。デフォルトは、-Ksimd_nouncounted_loopです。

simd_uncounted_loop

SIMD命令を利用したオブジェクトを生成することを指示します。

simd_nouncounted_loop

SIMD命令を利用せずにオブジェクトを生成することを指示します。

{ simd_use_multiple_structures | simd_nouse_multiple_structures }

SIMD化の際、SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用するかどうかを指示します。本オプションは、-Ksimd[={ 1 | 2 | auto }]オプションおよび-KSVEが有効な場合に意味があります。デフォルトは、-Ksimd_use_multiple_structuresです。

simd_use_multiple_structures

SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用します。SIMD化対象のロードおよびストアに上記の命令を利用することで、実行性能を向上させます。ただし、データのアライメントによっては性能劣化する場合があります。

simd_nouse_multiple_structures

SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用しません。

{ subscript_opt | nosubscript_opt }

配列の添字の近傍データの利用を優先した最適化(例えば、ステンシル計算)を行うかどうかを指示します。

デフォルトは、-Knosubscript_optです。

subscript_opt

配列の添字の近傍データの利用を優先した最適化を行います。

本オプションを指定した場合、使用するレジスタが増加するため実行性能が低下する場合があるので注意が必要です。

nosubscript_opt

配列の添字の近傍データの利用を優先した最適化を行いません。

{ static_fjlib | nostatic_fjlib }

Fortran組込み関数ライブラリを静的に結合した実行可能プログラムを作成するかどうかを指示します。本オプションはプログラムのリンク時に指定する必要があります。デフォルトは、-Knostatic_fjlibです。

static_fjlib

Fortran組込み関数ライブラリを静的に結合した実行可能プログラムを作成することを指示します。Fortran組込み関数ライブラリを静的に結合した場合、実行可能プログラムのサイズが大きくなることがあります。

nostatic_fjlib

Fortran組込み関数ライブラリを動的に結合した実行可能プログラムを作成することを指示します。

{ striping[=*N*] | nostriping }　　$2 \leqq N \leqq 100$

ループストライピングの最適化を行うかどうかを指示します。ストライプ長(展開数)を*N*で指定することができます。*N*は2から100までの値を指定することができます。*N*の省略値は2です。ソースプログラムでループの繰返し数が既知の場合、*N*に繰返し数を超える値を指定しても、コンパイラが自動的に決定した展開数が有効となります。デフォルトは、-Knostripingです。

ループストライピングについては、“9.1.2.4 ループストライピング”を参照してください。

ループストライピングは、ループ内の文が多重に展開されるので、ループアンローリング同様、オブジェクトプログラムの大きさが増加します。また、多くの翻訳時間を必要とする場合があります。実行性能に関しても、使用するレジスタが増加するため実行性能が低下する場合があるので注意が必要です。

striping[=*N*]

ループストライピングの最適化を行うことを指示します。本オプションは、-O2オプション以上が有効な場合に意味があります。

nostriping

ループストライピングの最適化を行わないことを指示します。

{ SVE | NOSVE }

Armv8-Aアーキテクチャの拡張であるSVEを利用するかどうかを指示します。-KSVEオプションが指定された場合、SVEを利用したオブジェクトファイルを出力します。SVEをサポートしないプロセッサ向けのオブジェクトファイルを作成したい場合は、-KNOSVEオプションを指定する必要があります。デフォルトは、-KSVEです。

SVE

SVEを利用したオブジェクトファイルを出力することを指示します。

NOSVE

SVEを利用しないオブジェクトファイルを出力することを指示します。

{ swp | noswp }

ソフトウェアパイプライニングの最適化を行うかどうかを指示します。ただし、最適化の効果が期待できないものに対しては最適化を行いません。-O0または-O1オプションが有効な場合、デフォルトは-Knoswpです。-O2オプション以上が有効な場合、デフォルトは-Kswpです。

本オプションは、-O1オプション以上が有効な場合に意味があります。

-Kswpオプションを、-Kswp_weakまたは-Kswp_strongオプションと同時に指定した場合、後に指定した方が有効となります。

ソフトウェアパイプライニングについては、“9.1.1.6 ソフトウェアパイプライニング”を参照してください。

swp

ソフトウェアパイプライニングの最適化を行うことを指示します。本オプションは、-O2オプション以上が有効な場合に意味があります。

noswp

ソフトウェアパイプライニングの最適化を行わないことを指示します。

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

以下のレジスタについて、ソフトウェアパイプライニングで使用可能な割合(百分率)を指示します。

- 浮動小数点レジスタおよびSVEのベクトルレジスタ
- 整数レジスタ
- SVEのプレディケートレジスタ

*N*は、1から1000までの整数値です。デフォルトは、-Kswp_freg_rate=100,swp_ireg_rate=100,swp_preg_rate=100です。

レジスタ数に関する条件を変更することで、ソフトウェアパイプライニングの適用を調整できます。レジスタが不足するためソフトウェアパイプライニングが適用されない場合、100より大きな整数値を本オプションに指定することで、ソフトウェアパイプライニングが適用できることがあります。

本オプションを指定することで、レジスタのメモリへの退避・復元命令が変化し、実行性能が低下する場合があります。

本オプションは、-O2オプション以上が有効な場合に意味があります。

swp_freg_rate=*N*

ソフトウェアパイプライニングを適用する際、浮動小数点レジスタならびにSVEのベクトルレジスタの*N*%が使用可能であるとみなします。

swp_ireg_rate=*N*

ソフトウェアパイプライニングを適用する際、整数レジスタの*N*%が使用可能であるとみなします。

swp_preg_rate=*N*

ソフトウェアパイプライニングを適用する際、SVEのプレディケートレジスタの*N*%が使用可能であるとみなします。

swp_policy={ auto | small | large }

ソフトウェアパイプライニングで使用する命令スケジューリングアルゴリズムの選択基準を指示します。

ソフトウェアパイプライニングは、翻訳時オプション-Kswp、-Kswp_weak、-Kswp_strong、またはそれぞれのオプションに対応した最適化制御行が有効な場合に行われます。

デフォルトは、-Kswp_policy=autoです。

swp_policy=auto

ループ毎に命令スケジューリングアルゴリズムを自動で選択します。

swp_policy=small

小さなループ(例えば、必要レジスタ数が少ないループ)に適した命令スケジューリングアルゴリズムを使用します。

swp_policy=large

大きなループ(例えば、必要レジスタ数が多いループ)に適した命令スケジューリングアルゴリズムを使用します。

swp_strong

ソフトウェアパイプライニング適用の条件を緩和し、ソフトウェアパイプライニングを促進することを指示します。本オプションを指定することで、翻訳メモリや翻訳時間が大幅に増加する場合があります。

-Kswp_strongオプションを、-Kswpまたは-Kswp_weakオプションと同時に指定した場合、後に指定した方が有効となります。

その他の機能や注意事項は、-Kswpオプションと同じです。

swp_weak

ソフトウェアパイプライニングを調整し、ループ内の実行文の重なりを小さくすることを指示します。実行時にソフトウェアパイプライニングが適用されたループを実行するために必要なループの繰返し数が小さくなるため、ループの繰返し数が翻訳時に不明であり、かつループの繰返し数が小さい場合に最適化の効果が期待できます。本オプションを指定することで、実行文の重なりが小さくなるため、実行性能が低下する場合があります。

-Kswp_weakオプションを、-Kswpまたは-Kswp_strongオプションと同時に指定した場合、後に指定した方が有効となります。

その他の機能や注意事項は、-Kswpオプションと同じです。

{ temparraystack | notemparraystack }

以下の結果領域をスタックに割り付けるかどうかを指示します。デフォルトは、-Ktemparraystackです。

- 配列演算の途中結果
- DO CONCURRENTの繰返し数が定数の場合、マスク式評価結果

temparraystack

配列演算の途中結果およびマスク式評価結果をスタックに割り付けます。本オプションを指定する場合の注意事項については、“9.11 スタック割付けの影響”を参照してください。

notemparraystack

配列演算の途中結果およびマスク式評価結果をスタックに割り付けません。

{ threadsafe | nothreadsafe }

スレッドセーフなオブジェクトプログラムを生成するかどうかを指示します。スタック領域の大きさに関する注意事項は、“9.11 スタック割付けの影響”を参照してください。

デフォルトは、-Knothreadsafeです。

threadsafe

スレッドセーフなオブジェクトプログラムを生成します。

nothreadsafe

スレッドセーフでないオブジェクトプログラムを生成します。

本オプションを-Kopenmpオプションと同時に指定する場合は、-Kopenmpオプションの後に指定する必要があります。本オプションを指定すると、プログラムの実行から、スレッドセーフを保証するための処理が省略されます。本オプションが指定できる手続は、主プログラムまたは動的なパラレルリージョンの外で呼び出される副プログラムでなければならず、また、以下のものを除くOpenMP指示文を含んでいてはなりません。

- プライベート変数の宣言を含まないPARALLEL構文
- THREADPRIVATE指示文

指定できない手続に対して指定した場合は、プログラムが正しく実行されないことがあるので注意してください。

tls_size={ 12 | 24 | 32 | 48 }

スレッド・ローカル・ストレージ(Thread-Local Storage)へのアクセスに必要なオフセットのサイズを指定します。単位はビットです。

スレッド・ローカル・ストレージのサイズに合わせ、-Ktls_size=12(4Kバイト)、-Ktls_size=24(16Mバイト)、-Ktls_size=32(4Gバイト)、-Ktls_size=48(256Tバイト)を指定することができます。

スレッド・ローカル・ストレージのサイズがオフセットの範囲を超えた場合、リンク時にエラーが生じます。

-Kpicまたは-KPICオプションが有効な場合、本オプションは無効となります。

{ unroll[=*N*] | nounroll }　　2 ≦ *N* ≦ 100

ループアンローリングの最適化を行うことを指示します。*N*にはループ展開数の上限を2～100の範囲で指定できます。*N*の指定を省略した場合、コンパイラが自動的に最良な値を決定します。-O0または-O1オプションが有効な場合のデフォルトは-Knounroll、-O2オプション以上が有効な場合のデフォルトは-Kunrollです。

ループアンローリングについては“9.1.1.4 ループアンローリング”を参照してください。

unroll[=*N*]

ループアンローリングの最適化を行います。本オプションは、-O1オプション以上が有効な場合に意味があります。

nounroll

ループアンローリングの最適化を行いません。

{ unroll_and_jam[=*N*] | nounroll_and_jam }　　2 ≦ *N* ≦ 100

アンロールアンドジャムの最適化を行うかどうかを指示します。*N*にはループ展開数の上限を2～100の範囲で指定できます。*N*の指定を省略した場合、コンパイラが自動的に最良な値を決定します。デフォルトは、-Knounroll_and_jamです。

アンロールアンドジャムを適用することで、共通式の除去が促進され、実行性能が向上する場合があります。ただし、データストリーム数の増加やデータのアクセス順序の変化は、キャッシュの利用効率を低下させ、実行性能が低下する場合があります。

また、アンロールアンドジャムの最適化による実行性能への影響はループ単位で異なります。このため、本最適化は、-Kunroll_and_jam[=*N*]オプションでプログラム全体へ適用せず、最適化指示子UNROLL_AND_JAMや最適化指示子UNROLL_AND_JAM_FORCEでループ単位に適用することを推奨します。

アンロールアンドジャムについては、“9.1.2.9 アンロールアンドジャム”を参照してください。

unroll_and_jam [=*N*]

アンロールアンドジャムの最適化を行います。本オプションは、-O2オプション以上が有効な場合に意味があります。ただし、以下の場合には最適化を行いません。

- 最適化の効果が期待できないと判断した場合

- 回転を跨いだデータ依存があると判断した場合

nounroll_and_jam

アンロールアンドジャムの最適化を行いません。

visimpact

VISIMPACT(Virtual Single Processor by Integrated Multicore Parallel Architecture)用の自動並列化された最適なオブジェクトを生成することを指示します。本オプションは、

-Kfast,parallel

を指定した場合と等価です。

本オプションは、プログラムの翻訳時およびリンク時に指定する必要があります。

最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

{ zfill[=*N*] | nozfill }　　1 ≦ *N* ≦ 100

zfillの最適化を行うかどうかを指示します。zfillの最適化は、ループ内で書き込みのみを行う配列データについて、データをメモリからロードすることなく、キャッシュ上に書き込み用の領域を確保する命令(DC ZVA)を使って、書き込み動作を高速化します。zfillの最適化は対象となるストア命令の指すアドレスを起点として、256バイトを1ブロックとする単位で*N*ブロック分先のデータを最適化の対象とします。*N*は1～100の範囲の整数値で指定できます。*N*の指定を省略した場合、コンパイラが自動的に値を決定します。デフォルトは、-Knozfillです。

-Kzfillオプションは、-KA64FXおよび-O2オプション以上と同時に指定した場合に有効となります。

なお、-KzfillオプションをDC ZVA命令の1回のキャッシュ書き込み動作が256バイトであるCPU以外に適用した場合、実行時異常終了または実行結果に誤りが生じることがあります。A64FXでのDC ZVA命令の1回のキャッシュ書き込み動作は256バイトです。

また、以下の場合に本最適化を適用すると実行性能が低下することがあります。

- メモリバンド幅のボトルネック影響を受けていないプログラム
- 繰返し数が小さいループ
- -Kzfill=*N*オプションでブロック数を指定している、かつ、ループによって書き込まれるメモリ領域のサイズが*N*ブロックよりも小さい場合

本最適化による実行性能への影響はループ単位で異なります。このため、本最適化は、-Kzfill[=*N*]オプションでプログラム全体へ適用せず、最適化指示子ZFILLでループ単位に適用することを推奨します。

zfillについては、“9.1.2.5 zfill”を参照してください。

zfill[=*N*]

zfillの最適化を行うことを指示します。*N*を指定することで、*N*ブロック先のデータを最適化の対象とします。

nozfill

zfillの最適化を行わないことを指示します。

-L *directory*

リンカがライブラリを検索するディレクトリのリストに*directory*を加えます。このオプションと引数はリンカに渡されます。

ライブラリの検索は以下の順序で行われます。

1. -Lオプションの引数に指定されたディレクトリ
2. 本処理系が提供するライブラリを格納するディレクトリ
3. 標準ライブラリのディレクトリ
4. 環境変数LIBRARY_PATHに指定されたディレクトリ

-M *directory*

原始プログラム中のモジュール情報を*directory*に登録します。また、原始プログラム中に存在しないモジュールまたはサブモジュールを*directory*から引用したい場合に指定します。モジュールおよびサブモジュールの翻訳については、“第10章 モジュール、サブモジュール、およびモジュール引用の注意事項”を参照してください。

-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 } }

-Nオプションの引数はコンマ(,)を区切り子とし、組み合わせて指定することができます。

{ allextput | noallextput }

EXTERNAL文だけに現れる外部名を有効にするかどうかを指示します。デフォルトは、-Nallextputです。

allextput

EXTERNAL文だけに現れる外部名を有効にします。

noallextput

EXTERNAL文だけに現れる外部名を無効にします。

{ alloc_assign | noalloc_assign }

Fortran規格の割付け代入(“6.4.2 割付け代入”参照)の動作を指示します。-X9、-X03、または-X08オプションが有効な場合だけ、-Nalloc_assignは有効となります。-X6または-X7が有効な場合、-Nalloc_assignは有効になりません。

-X03または-X08オプションが有効な場合のデフォルトは-Nalloc_assign、-X9オプションが有効な場合のデフォルトは-Nnoalloc_assignです。

alloc_assign

-X9、-X03または-X08オプションが有効な場合、本オプションを指定するとFortran規格の割付け代入の動作を行います。Fortran 95規格仕様プログラムでは、-Nalloc_assign オプションを指定すると実行性能が低下します。

noalloc_assign

-Nnoalloc_assignオプションを指定すると、Fortran規格の割付け代入は行われません。

{ cancel_overtime_compilation | nocancel_overtime_compilation }

プログラムの翻訳が長時間(24時間以上を目安とします)になると予測される場合に、翻訳を中止するかどうかを指示します。

デフォルトは、-Ncancel_overtime_compilationです。

cancel_overtime_compilation

翻訳時間が長時間になると予測される場合に、翻訳を中止します。

nocancel_overtime_compilation

翻訳が長時間になる場合でも、翻訳を中止しません。

check_cache_arraysize

-Ncheck_cache_arraysize オプションが有効な場合、翻訳時に配列の大きさを検査して、実行性能に影響を与える可能性がある場合に、iレベルの診断メッセージを出力します。

本機能は、配列の大きさが2次キャッシュの大きさの倍数である場合にキャッシュの競合を起こすとみなします。

メッセージが出力された配列については、2次キャッシュの大きさの倍数とならないように配列の形状を書き換えることで影響を避けられることがあります。

本機能を使用する場合の注意事項として、セクタキャッシュの利用時には正確な診断ができない場合があります。

-Karraypad_const[=*N*] または -Karraypad_expr=*N*オプションが指定された場合、配列の大きさはパディングされた大きさを加算した値が対象になります。

翻訳時の診断メッセージについては、“4.1.1 翻訳時の診断メッセージ”を参照してください。

{ check_global | nocheck_global }

-Ncheck_globalオプションが有効な場合、翻訳時に外部手続の定義と引用との間の手続特性の検査、外部手続の定義と引用仕様本体との間の手続特性の検査および共通ブロックの大きさの検査をプログラム単位にまたがって行います。誤りを検出した場合は、診断メッセージを出力します。

-Ncheck_globalオプションが有効な場合、検査を行うことで翻訳時間が増加することがあります。

-Nnocheck_globalオプションが有効な場合、翻訳時にプログラム単位にまたがって検査を行いません。

check_intrfunc

オブジェクトプログラムに対して、組込み関数のエラー処理を有効にします。

-Ncheck_intrfuncオプションが有効な場合、プログラムの実行時にエラー検査が実施されます。検査の結果、エラーに適合する場合には以下の処理が行われます。

- 診断メッセージ、トレースバックマップ、エラー集計情報の出力
- ERRSET/ERRSTR サービスサブルーチンによるエラー項目の修正
- 利用者修正出口処理ルーチンの実行

-Ncheck_intrfuncオプションの指定で検出されるエラーの番号および組込み関数のエラー処理については、“8.1.5 組込み関数エラーの処理”を参照してください。

check_std=*std_arg std_arg* :{ 03d | 03e | 03o | 03s | 08d | 08e | 08o | 08s }

引数を複数指定する場合は、-Ncheck_std=03d,check_std=03oのようにコンマで区切って指定してください。

引数03dと08d、03eと08e、03oと08o、03sと08sは、同時に指定できません。

03d

03dが有効な場合、原始プログラム中のFortran 2003規格の廃止事項の検査を行います。

03e

03eが有効な場合、原始プログラム中のFortran 95規格からFortran 2003規格に対して追加された事項の検査を行います。

03o

03oが有効な場合、原始プログラム中のFortran 2003規格の廃止予定事項の検査を行います。

03s

03sが有効な場合、原始プログラムがFortran 2003規格に準拠しているか検査を行います。

08d

08dが有効な場合、原始プログラム中のFortran 2008規格の廃止事項の検査を行います。

08e

08eが有効な場合、原始プログラム中のFortran 2003規格からFortran 2008規格に対して追加された事項の検査を行います。

08o

08oが有効な場合、原始プログラム中のFortran 2008規格の廃止予定事項の検査を行います。

08s

08sが有効な場合、原始プログラムがFortran 2008規格に準拠しているか検査を行います。

{ coarray | nocoarray }

COARRAY仕様を有効にするかどうかを指示します。

本機能については“Fortran使用手引書 別冊 COARRAY”を参照してください。デフォルトは、-Nnocoarrayです。

coarray

COARRAY仕様を有効にします。

リンクだけ行う場合でも、-Ncoarrayオプションを指定して翻訳されたオブジェクトプログラムが含まれる場合は、本オプションを指定する必要があります。

nocoarray

COARRAY仕様を無効にします。

本オプションを指定してCOARRAYプログラムを翻訳すると、エラーメッセージを出力して翻訳を中止します。

{ compdisp | nocompdisp }

翻訳しているファイル名およびプログラム単位名を表示するかどうかを指示します。デフォルトは、-Nnocompdispです。

compdisp

翻訳しているファイル名およびプログラム単位名を表示します。

nocompdisp

翻訳しているファイル名およびプログラム単位名を表示しません。

{ copyarg | nocopyarg }

手続引用の実引数に指定されたスカラ定数をコピーして引数として渡すかどうかを指示します。デフォルトは、-Nnocopyargです。

copyarg

手続引用の実引数に指定されたスカラ定数をコピーして引数とすることを指示します。-Ncopyargオプションが有効な場合、副プログラムで仮引数の値を更新しても、実引数の定数値には影響しません。ただし、本オプションの指定により動作可能になるプログラムは、Fortran規格に合致しないプログラムです。

nocopyarg

手続引用の実引数に指定されたスカラ定数を、そのまま引数とします。

例: -Ncopyarg

```
CALL SUB(1)
PRINT *,1
END
SUBROUTINE SUB(I)
I=2
END
```

上記の例では、-Ncopyargオプションが有効な場合、1を出力します。

{ coverage | nocoverage }

コードカバレッジ機能を利用するための情報を生成するかどうかを指示します。-Ncoverageオプションは、プログラムの翻訳時およびリンク時に指定する必要があります。デフォルトは、-Nnocoverageです。

-Ncoverageオプションが有効な場合、-Kltoオプションは無効となります。

コードカバレッジ機能の詳細については、“付録G コードカバレッジ機能”を参照してください。

coverage

コードカバレッジ機能を利用するための情報を生成します。

実行回数を計測する命令を追加するため、実行性能が低下する場合があります。

nocoverage

コードカバレッジ機能を利用するための情報を生成しません。

{ f90move | nof90move }

重なりのある文字代入(“6.4.1 重なりのある文字代入”参照)の動作を指示します。-X9、-X03または-X08オプションが有効な場合、-Nf90moveは常に有効となり、-Nnof90moveは、同時に指定できません。-X7または -X6オプションが有効な場合、-Nnof90moveがデフォルトで指定されます。

f90move

-X7または-X6オプションが有効でも、-Nf90moveオプションを指定するとFortran規格仕様の文字代入を行います。

nof90move

-Nnof90moveオプションを指定すると、Fortran規格仕様の文字代入は行われません。

例: -Nf90move

```
CHARACTER(LEN=5) C
C='12345'
C(2:5)=C(1:4)
PRINT *,C
END
```

上記の例では、-X7または-X6オプションが有効な場合、-Nf90moveオプションを指定することにより、11234の結果が得られます。

{ fjprof | nofjprof }

プロファイラ機能を有効にするかどうかを指示します。本オプションはリンク時に指定する必要があります。デフォルトは、-Nfjprofです。

プロファイラ機能については、“プロファイラ使用手引書”をお読みください。

fjprof

プロファイラ機能を有効にします。

nofjprof

プロファイラ機能を無効にします。

{ freealloc | nofreealloc }

局所的な割付け配列を、手続の終了時に解放するかどうかを指示します。デフォルトは、-Nfreeallocです。

freealloc

副プログラムの復帰時に局所的な割付け配列を解放することを指示します。

nofreealloc

副プログラムの復帰時に局所的な割付け配列を解放しないことを指示します。-Nnofreeallocオプションを指定するとFortran規格に合致しないプログラムになります。原始プログラムを修正し、SAVE属性を付加することを推奨します。

{ hook_func | nohook_func }

特定箇所からの呼び出しによるフック機能を利用するかどうかを指示します。フック機能については、“8.2.3 フック機能”を参照してください。

デフォルトは、-Nnohook_funcです。

リンクだけ行う場合でも、-Nhook_funcオプションを指定して翻訳されたオブジェクトプログラムが含まれる場合は、本オプションを指定する必要があります。

hook_func

-Nhook_funcオプションが有効な場合、次の箇所からユーザ定義ルーチンが呼び出されます。

- プログラムの入口/出口

- 手続の入口/出口

- パラレルリージョン(OpenMP/自動並列化)の入口/出口

nohook_func

特定箇所からの呼び出しによるフック機能を利用しません。

{hook_time | nohook_time}

一定時間間隔での呼び出しによるフック機能を利用するかどうかを指示します。

フック機能については、“8.2.3 フック機能”を参照してください。

本オプションは、リンク時に指定する必要があります。デフォルトは、-Nnohook_timeです。

hook_time

-Nhook_timeオプションが有効な場合、一定時間間隔ごとにユーザ定義関数が呼び出されます。

時間間隔は環境変数FLIB_HOOK_TIMEで指定可能です。環境変数FLIB_HOOK_TIMEを指定しない場合は、1分ごとにユーザ定義関数が呼び出されます。

環境変数FLIB_HOOK_TIMEについては、“3.8 実行時の環境変数”を参照してください。

nohook_time

一定時間間隔での呼び出しによるフック機能を利用しません。

{ libomp | fjomplib }

並列処理に使用するライブラリを指定します。本オプションは、リンク時に指定する必要があります。デフォルトは、-Nlibompです。

libomp

並列処理にLLVM OpenMPライブラリを使用します。LLVM OpenMPライブラリについては、“第12章 並列化機能”を参照してください。

fjomplib

並列処理に富士通OpenMPライブラリを使用します。富士通OpenMPライブラリについては、“付録J 富士通OpenMPライブラリ”を参照してください。

{ line | noline }

トレースバックマップ(逆追跡情報)に対する情報量を選択する機能です。Fortranプログラムの実行時にエラーが発生した場合にトレースバックマップを標準エラー出力ファイルに出力します。このトレースバックマップに対する情報を、その情報の種類の指定に応

じてオブジェクトプログラム中に出力します。デフォルトは、-Nlineです。トレースバックマップについては、“4.2.2トレースバックマップ”を参照してください。

line

トレースバックマップ上にエラーが発生したユーザ手続の呼出し箇所(行位置)およびエラーが発生した行位置を表示するオブジェクトプログラムを出力します。

noline

トレースバックマップ上にエラーが発生した行位置を表示しないオブジェクトプログラムを出力します。

lst[=*lst_arg*]　　*lst_arg*:{ a | d | i | m | p | t | x }

翻訳情報を得る機能です。翻訳情報を.lstファイルに出力します。Fortran原始プログラムを複数ファイル指定した場合には、最初のファイル名.lstファイルに出力します。

引数を複数指定する場合は、-Nlst=a,lst=d,lst=xのようにコンマで区切って指定してください。引数を省略した場合、翻訳情報として、原始プログラムリストおよびエラーメッセージを出力します。

a

-Nlst=aオプションが有効な場合、-Nlstオプションでの出力に加えて、名前の属性情報を出力します。

d

-Nlst=dオプションが有効な場合、-Nlstオプションでの出力に加えて、派生型の構成情報を出力します。

i

-Nlst=iオプションが有効な場合、-Nlstオプションでの出力に加えて、インクルードされたファイルのプログラムリストおよびインクルードファイル名一覧を出力します。

m

-Nlst=mオプションが有効な場合、-Nlst=pオプションでの出力に加えて、自動並列化の状況をOpenMP指示文によって表現した原始プログラムを出力します。出力される原始プログラムは、入力ファイルの拡張子の直前に.ompが挿入されたファイルに格納されます。

本オプションは、-Kparallelオプションが有効な場合に有効にします。

本オプションは、-Nlst=iオプションが有効な場合には、無効にします。

p

-Nlst=pオプションが有効な場合、-Nlstオプションでの出力に加えて、プログラムリスト中に最適化情報を出力します。最適化情報により、自動並列化、インライン展開、アンローリングの状況がわかります。

t

-Nlst=tオプションが有効な場合、-Nlst=pオプションでの出力に加えて、より詳細な最適化情報および統計情報を出力します。

x

-Nlst=xオプションが有効な場合、-Nlstオプションでの出力に加えて、名前および文番号の相互参照情報を出力します。

lst_out=*file*

指定されたファイル名*file*に翻訳情報を出力します。

{ mallocfree | nomallocfree }

MALLOCおよびFREEを組込み手続とすることを指示するオプションです。デフォルトは、-Nnomallocfreeです。

mallocfree

MALLOCおよびFREEを組込み手続とすることを指示します。

nomallocfree

MALLOCおよびFREEを組込み手続とはしません。本オプションが有効な場合、MALLOCおよびFREEは、サービスルーチンとみなします。

maxserious=*maxnum*

翻訳時に検出されるsレベル(重度のエラー)のエラーメッセージの数が*maxnum*に達した場合に翻訳を中止することを指定します。*maxnum*は、1以上の値でなければなりません。本オプションを指定しない場合、翻訳は最後まで行います。

{ obsfun | noobsfun }

次の関数名を組込み関数として解釈するかを指示します。デフォルトは、-Nnoobsfunです。

obsfun

-Nobsfunオプションが有効な場合、次の関数名を組込み関数として解釈します。

noobsfun

-Nnoobsfunオプションが有効な場合、次の関数名を利用者定義の手続として解釈します。

関数名:

AIMAX0、AJMAX0、I2MAX0、IMAX0、JMAX0、IMAX1、JMAX1、AIMIN0、AJMIN0、I2MIN0、IMIN0、JMIN0、IMIN1、JMIN1、FLOATI、FLOATJ、DFLOTI、DFLOTJ、IIABS、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、IIAND、JIAND、IIEOR、JIEOR、IIOR、JIOR、INOT、JNOT、IISHFT、JISHFT、IISHFTC、JISHFTC、IZEXT、JZEXT、IZEXT2、JZEXT2、JZEXT4、VAL

{ privatealloc | noprivatealloc }

OpenMPのPRIVATEなどの指示節に指定された割付け変数が割り付けられているときの、対応するプライベート変数の初期状態を指示します。

本オプションは、-Kopenmpオプションが有効な場合に意味があります。デフォルトは、-Nnoprivateallocです。

privatealloc

プライベート変数は未割付けの初期状態になります。

本オプションが有効な場合、OpenMP 3.0以降の規格に合致しない振舞いになります。本オプションは、下方互換を維持するためにあります。

noprivatealloc

プライベート変数は割り付けられている初期状態になります。

profile_dir=*dir_name*

コードカバレッジ機能使用時に必要な.gcdaファイルの格納先ディレクトリを指示します。*dir_name*には、格納先ディレクトリ名を相対パスまたは絶対パスで指定します。

.gcdaファイルは、コードカバレッジ用の情報を含むオブジェクトファイルを結合した実行可能プログラムを実行すると、生成されます。実行時に、*dir_name*で指定されたディレクトリが存在しない場合は、新規に作成されます。

コードカバレッジ機能、およびデフォルトの格納先ディレクトリについては、“付録G コードカバレッジ機能”を参照してください。

本オプションは、-Ncoverageオプションが有効、かつ-Sまたは-cオプションが有効な場合に意味があります。

quickdbg[=*dbg_arg*]　　*dbg_arg*: { { argchk | noargchk }| { subchk | nosubchk } | { undef | undefnan | noundef } | { inf_detail | inf_simple } }

Fortran原始プログラムをデバッグするための機能です。デバッグ機能が有効な場合、Fortran原始プログラムの誤りを翻訳時に検査し、かつ、オブジェクトプログラム内にデバッグするために必要な情報を組み込み、実行時に自動検査します。

検査は、-Nquickdbg=argchk、-Nquickdbg=subchk、-Nquickdbg=undef、または-Nquickdbg=undefnanオプションが有効な場合に行われます。

*dbg_arg*には、argchk、noargchk、subchk、nosubchk、undef、undefnan、noundef、inf_detail、またはinf_simpleのいずれか1つを指定できます。これらの指定は、-Nquickdbg[=*dbg_arg*]オプションを複数個指定することで、組み合わせて利用することができます。

-Nquickdbg=inf_detailおよび-Nquickdbg=inf_simpleオプションは、-Nquickdbg、-Nquickdbg=argchk、-Nquickdbg=subchk、-Nquickdbg=undef、-Nquickdbg=undefnanオプションのいずれかと組み合わせて指定することで有効となります。

*dbg_arg*が省略された場合、-Nquickdbg=argchk,quickdbg=subchk,quickdbg=undef が指定されたものと見なします。

-Nquickdbg=argchk、-Nquickdbg=subchk、-Nquickdbg=undef、または-Nquickdbg=undefnanオプションが有効な場合、-Nquickdbg=inf_detail、-Ncheck_globalオプションも有効になります。

本機能については、“8.2 デバッグのための機能”を参照してください。

-Hオプションが有効な場合は、-Nquickdbgオプションを無効とします。

{ argchk | noargchk }

手続の引用において、引用元と引用先の妥当性を検査するかどうかを指示します。

本検査はプログラムの翻訳時および実行時に行われます。本検査機能については、“8.2.1.1 引数の妥当性の検査(ARGCHK機能)”を参照してください。

argchk

引数および関数結果に関する手続引用の妥当性の検査を行います。誤りを検出した場合は、診断メッセージを出力します。

noargchk

引数および関数結果に関する手続引用の妥当性の検査を行いません。

{ subchk | nosubchk }

部分配列、配列要素または部分列の宣言と引用において、宣言した大きさと引用した範囲の正当性を検査するかどうかを指示します。

本検査はプログラムの翻訳時および実行時に行われます。本検査機能については、“8.2.1.2 添字式および部分列範囲の検査(SUBCHK機能)”を参照してください。

subchk

添字式および部分列範囲の検査を行います。誤りを検出した場合は、診断メッセージを出力します。

nosubchk

添字式および部分列範囲の検査を行いません。

{ undef | undefnan | noundef }

モジュールまたはサブモジュールの宣言部および共通ブロックに属さない変数の引用において、その変数に値が定義されていることを検査するかどうかを指示します。

本検査はプログラムの実行時に行われます。本検査機能については、“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照してください。

undef

未定義データの引用の検査を行います。誤りを検出した場合は、診断メッセージを出力します。

undefnan

未定義データの引用の検査を行います。本オプションの指定時は、実数型および複素数型の誤りは、無効演算例外として検出されます。誤りを検出した場合は、変数の型に応じて診断メッセージまたは無効演算例外メッセージが出力されます。

noundef

未定義データの引用の検査を行いません。

{ inf_detail | inf_simple }

本検査機能で検出された誤りにおいて、診断メッセージに出力する情報を指示します。

本機能については、“8.2.1 デバッグを行うための検査機能”を参照してください。

inf_detail

エラーの原因を識別するためのメッセージとエラーが発生した行番号に加えて、エラー発生の原因となった変数名、手続名、要素位置などの情報が、診断メッセージとして出力されます。

ただし、-Nquickdbg=undefnanオプションが有効な場合に無効演算例外として検出された誤りに関しては、エラーが発生したことを通知するメッセージとエラーが発生した行番号のみが、診断メッセージとして出力されます。

inf_simple

エラーが発生したことを通知するメッセージとエラーが発生した行番号が、診断メッセージとして出力されます。

{ recursive | norecursive }

サブルーチン副プログラムまたは関数副プログラムにRECURSIVEキーワードを付加するかどうかを指示します。デフォルトは、-Nnorecursiveです。

recursive

要素別処理手続でないサブルーチン副プログラムまたは関数副プログラムにRECURSIVEキーワードを付加します。

-Nrecursiveオプションが有効な場合、Fortran規格に合致しないプログラムになります。原始プログラムを修正し、再帰的呼出しを行う手続にRECURSIVEのキーワードを付加することを推奨します。

norecursive

RECURSIVEキーワードを付加しません。

{ reordered_variable_stack | noreordered_variable_stack }

自動割付け変数をスタック領域に割り付ける順序をコンパイラに指示します。デフォルトは、-Nnoreordered_variable_stackです。-Nnoline、-g0、または-Koclオプションが有効な場合、割付け順序は保証されません。本機能の詳細については、“E.4 スタック領域へのデータ割付け”を参照してください。

reordered_variable_stack

以下の順序で、自動割付け変数の割付け順を決めます。

1. アライメントの降順
2. アライメントが等しい場合は、データサイズの降順
3. アライメントおよびデータサイズが等しい場合は、ソースプログラム内の宣言文の記載順

アライメントの降順に自動割付け変数を割り付けると、プログラム全体のスタック領域を減らすことができます。

noreordered_variable_stack

ソースプログラム内の宣言文の記載順に、自動割付け変数を割り付けます。

{ rt_tune | rt_notune }

実行時情報を出力するかどうかを指示します。デフォルトは、-Nrt_notuneです。

実行時情報はプログラムのチューニングに利用することができます。詳細については、“付録H 実行時情報出力機能”を参照してください。

rt_tune

実行時情報を出力します。

rt_notune

実行時情報を出力しません。

rt_tune_func

-Nrt_tuneの出力に加えて、利用者定義の手続ごとの実行時情報を出力します。

本オプションは、-Nrt_tuneオプションを誘導します。-Nrt_tune_funcオプションのあとに-Nrt_notuneオプションが指定された場合、実行時情報は出力されません。

実行時情報出力機能については、“付録H 実行時情報出力機能”を参照してください。

rt_tune_loop[=*kind*]　　*kind*: { all | innermost }

-Nrt_tuneの出力に加えて、ループごとの実行時情報を出力します。*kind*にはallおよびinnermostが指定できます。*kind*の省略値はallです。

本オプションは、-Nrt_tuneオプションを誘導します。-Nrt_tune_loopオプションのあとに-Nrt_notuneオプションが指定された場合、実行時情報は出力されません。

実行時情報出力機能については、“付録H 実行時情報出力機能”を参照してください。

rt_tune_loop=all

すべてのループの実行時情報を出力します。

rt_tune_loop=innermost

最内ループの実行時情報を出力します。

{ save | nosave }

主プログラムを除くプログラムに保存要素並びを省略したSAVE文を付加するかどうかを指示します。デフォルトは、-Nnosaveです。

save

主プログラムを除くプログラムに保存要素並びを省略したSAVE文を付加します。本オプションは、-Nrecursiveおよび-Kautoオプションと同時には指定できません。

nosave

主プログラムを除くプログラムに保存要素並びを省略したSAVE文を付加しません。

{ setvalue[=*set_arg*] | nosetvalue }
set_arg: { { heap | noheap } | { stack | nostack } | { scalar | noscalar } | { array | noarray } | { struct | nostruct } }

手続の入口およびALLOCATE文で、スタック領域またはヒープ領域に割り付けられた変数にゼロ値を自動的に設定するかを指示します。デフォルトは、-Nnosetvalueです。

ゼロ値設定処理が動作することにより、実行時間が増加する場合がありますので注意が必要です。

-Eg、-H、-Nquickdbgオプションのいずれかが有効な場合、-Nsetvalueオプションは有効になりません。

setvalue[=*set_arg*]

変数にゼロ値を自動的に設定します。

*set_arg*には、heap、noheap、stack、nostack、scalar、noscalar、array、noarray、struct、またはnostructのいずれか1つを指定できます。これらの指定は、-Nsetvalue=*set_arg*オプションを複数個指定することで、組み合わせて利用することができます。

*set_arg*の指定を省略した場合、以下と等価です。

-Nsetvalue=heap,setvalue=stack,setvalue=scalar,setvalue=array,setvalue=struct

{ heap | noheap }

手続の入口およびALLOCATE文で、ヒープ領域に割り付けられた変数にゼロ値を設定するかを指示します。

サービスサブルーチンMALLOCおよび組込み関数MALLOCで獲得した領域には、ゼロ値は設定されません。OpenMPのPRIVATE指示節またはLASTPRIVATE指示節で指定した割付け変数には、OpenMP構文のブロックの先頭でゼロ値は設定されません。

次の変数が対象となります。

- 自動割付け変数、かつ-Knoautoobjstackオプションが有効
- 割付け変数
- ポインタ変数

setvalue=heap

手続の入口およびALLOCATE文で、ヒープ領域に割り付けられた変数にゼロ値を設定することを指示します。

setvalue=heap を指定すると、-Nsetvalue=scalar、-Nsetvalue=arrayおよび-Nsetvalue=structを誘導します。

setvalue=noheap

手続の入口およびALLOCATE文で、ヒープ領域に割り付けられた変数にゼロ値を設定しないことを指示します。

{ stack | nostack }

手続の入口で、スタック領域に割り付けられた変数にゼロ値を設定するかを指示します。

次の変数が対象となります。

- 自動割付け変数、かつ-Kautoobjstackオプションが有効
- SAVE属性をもたない局所変数、かつ-Kautoオプションが有効
- RECURSIVEまたはPUREを指定したプログラム単位の局所変数

- AUTOMATIC属性またはAUTOMATIC文で指定した局所変数
- OpenMPのPRIVATE指示節またはLASTPRIVATE指示節で指定した局所変数

setvalue=stack

手続の入口で、スタック領域に割り付けられた変数にゼロ値を設定することを指示します。

ただし、OpenMPのPRIVATE指示節またはLASTPRIVATE指示節で指定した局所変数は、OpenMP構文のブロックの先頭でゼロ値が設定されます。

setvalue=stack を指定すると、-Nsetvalue=scalar、-Nsetvalue=arrayおよび-Nsetvalue=structを誘導します。

setvalue=nostack

手続の入口で、スタック領域に割り付けられた変数にゼロ値を設定しないことを指示します。

{ scalar | noscalar }

数値型および論理型のスカラ変数にゼロ値を設定するかを指示します。

setvalue=scalar

数値型および論理型のスカラ変数にゼロ値を設定することを指示します。

setvalue=noscalar

数値型および論理型のスカラ変数にゼロ値を設定しないことを指示します。

{ array | noarray }

数値型および論理型の配列変数にゼロ値を設定するかを指示します。

setvalue=array

数値型および論理型の配列変数にゼロ値を設定することを指示します。

setvalue=noarray

数値型および論理型の配列変数にゼロ値を設定しないことを指示します。

{ struct | nostruct }

派生型のスカラ変数および配列変数、文字型のスカラ変数および配列変数にゼロ値を設定するかを指示します。

setvalue=struct

派生型のスカラ変数および配列変数、文字型のスカラ変数および配列変数にゼロ値を設定することを指示します。

長さ型パラメタをもつ派生型は対象にはなりません。

setvalue=nostruct

派生型のスカラ変数および配列変数、文字型のスカラ変数および配列変数にゼロ値を設定しないことを指示します。

nosetvalue

割り付けられた変数にゼロ値を自動的に設定しないことを指示します。

本オプションは、

-Nsetvalue=noheap,setvalue=nostack,setvalue=noscalar,setvalue=noarray,setvalue=nostruct

を指定した場合と等価です。

以下の注意が必要です。

1. -Nsetvalueオプションは、以下の条件を満たす場合に有効になります。
   - -Nsetvalue=heapまたは-Nsetvalue=stackのいずれかが有効である。かつ、
   - -Nsetvalue=scalar、-Nsetvalue=array、または-Nsetvalue=structのいずれかが有効である。
2. -Nsetvalue=heapまたは-Nsetvalue=stackから、-Nsetvalue=scalar、-Nsetvalue=array、および-Nsetvalue=structが誘導されます。そのため、-Nsetvalue=noscalar、-Nsetvalue=noarray、または-Nsetvalue=nostructを有効にする場合には、-Nsetvalue=heap または-Nsetvalue=stackのあとに指定する必要があります。

3. スタック領域の数値型および論理型のスカラ変数だけをゼロ値設定の対象とする場合、以下の指定が必要となります。
   -Nsetvalue=stack,setvalue=noarray,setvalue=nostruct

   -Nsetvalue=stackから-Nsetvalue=scalar、-Nsetvalue=array、-Nsetvalue=structが誘導されるため、
   -Nsetvalue=stack,setvalue=scalarの指定では、スカラ変数だけに限定されません。

{ use_rodata | nouse_rodata }

実引数の定数が結合する仮引数を通して定義される可能性がある場合、その実引数の定数を書込み禁止領域に割り付けるかどうかを指示します。デフォルトは、-Nuse_rodataです。

use_rodata

実引数の定数を書込み禁止領域に割り付けます。

nouse_rodata

実引数の定数を書込み禁止領域に割り付けません。

{ Rtrap | Rnotrap }

組込み演算のエラーと浮動小数点演算の割込み事象を検出し、実行時の診断メッセージを出力するかどうかを指示します。翻訳時オプション-NRtrapは、プログラムの翻訳時およびリンク時に指定する必要があります。本オプションは、翻訳時に演算される定数式に対しては有効になりません。デフォルトは、翻訳時オプション-NRnotrapです。

組込み演算のエラーについては、“8.1.5 組込み関数エラーの処理”を参照してください。

浮動小数点演算の割込み事象については、“8.1.7 例外ハンドリングエラーの処理”および“8.2.2.1 異常終了の原因”を参照してください。

なお、Armアーキテクチャである富士通製CPU A64FXでは、除数が0の場合の整数除算例外は検出されません。詳細は、“A.2.3 除数が0の場合の整数除算例外について”を参照してください。

Rtrap

主プログラム単位の翻訳時に翻訳時オプション-NRtrapが有効な場合、以下のエラーを検出します。

・組込み演算エラー(jwe0259i-e～jwe0282i-e、jwe1397i-e、jwe1398i-e、jwe1413i-e、jwe1414i-e、jwe1416i-e、およびjwe1417i-e):
実行時に引数の値をチェックし、エラーである場合にメッセージを出力します。

・浮動小数点演算の割込み事象(jwe1017i-u):
検出するシグナルに対応するトラップを有効にしてハンドリングし、割込み事象が検出された場合、メッセージを出力します。浮動小数点アンダフローによるプログラム割込みを検出したい場合、実行時に環境変数FLIB_EXCEPT=uを指定してください。

命令の投機実行を伴う翻訳時オプション-Kpreexおよび-Ksimd=2を同時に指定した場合、本来は発生しない例外が発生する可能性があります。

また、翻訳時オプション-Kfp_relaxedが指定された場合でも、翻訳時オプション-NRtrapが有効、かつ翻訳時オプション-Knosimdまたは-KNOSVEのいずれかが有効である場合、SQRT関数を逆数近似演算に変換する最適化が抑止されます。したがって、翻訳時オプション-NRnotrapが有効な場合に比べ、実行性能が低下する場合があります。

出力される実行時の診断メッセージの詳細については、“Fortran/C/C++実行時メッセージ”を参照してください。

Rnotrap

主プログラム単位の翻訳時に翻訳時オプション-NRnotrapが有効な場合、組込み演算エラーおよび浮動小数点演算の割込み事象は検出されません。また、診断メッセージも出力されません。

-O [*opt_lvl*]　　*opt_lvl* : { 0 | 1 | 2 | 3 }

最適化レベルを指示する機能です。

最適化レベルには0、1、2、または3があり、-Oオプションの引数として指定します。-Oオプションの引数は省略可能であり、*opt_lvl*を省略した場合は、引数として3を指定したものと解釈します。デフォルトは、-O2です。

最適化については、“第9章 最適化機能”を参照してください。

さらに最適化機能として、-Kオプション指定により不変式の先行評価と演算評価方法の変更、-xオプション指定により利用者定義の手続のインライン展開を指定することができます。

0

-O0オプションが有効な場合、最適化は行いません。しかし、翻訳時間や翻訳作業領域は、最小となりますのでコンパイルエラーのチェックなどプログラミングの初期段階で利用してください。

1

-O1オプションが有効な場合、基本的な最適化を行います。-O0オプションが有効な場合に比べてオブジェクトプログラムの大きさが縮小され、実行時間も大幅に短縮されます。

2

-O2オプションが有効な場合、-O1オプションが有効な場合の基本的な最適化に加えて以下の最適化が行われます(括弧内のオプションが有効となります)。

- ループアンローリング(-Kunroll)
- ソフトウェアパイプライニング(-Kswp)
- ループブロッキング(-Kloop_blocking)
- ループ融合(-Kloop_fusion)
- ループ分割(-Kloop_fission)
- ループ交換(-Kloop_interchange)
- prefetch命令の生成(-Kprefetch_sequential,prefetch_cache_level=all)
- SIMD化(-Ksimd=auto)
- ループの先頭アライメント調整(-Kalign_loops)
- 末尾呼出しの最適化(-Ksibling_calls)
- 最適化機能の繰返し実施

  最適化機能の繰返し実施は、最適化レベル1で行われる最適化機能を最適化の余地がなくなるまで繰り返して実施します。

-O1オプションが有効な場合に比べ、翻訳時間やオブジェクトプログラムサイズが増加する場合があります。また、利用者は-Koptmsg=2オプションを有効にすることにより、各最適化が行われたかどうかを翻訳時の診断メッセージで知ることができます。

3

-O3オプションが有効な場合、-O2オプションが有効な場合の最適化に加えて以下の最適化が行われます。

- 多重ループのアンローリング
- ループ交換などの最適化を促進するための完全多重ループ化(-Kloop_perfect_nest)
- ループアンスイッチング
- CLONE最適化

-O2オプションが有効な場合に比べ、翻訳時間やオブジェクトプログラムサイズが増加する場合があります。また、利用者は-Koptmsg=2オプションを有効にすることにより、各最適化が行われたかどうかを翻訳時の診断メッセージで知ることができます。

-P

-Pオプションは、プリプロセッサ(前処理)の結果をファイルに出力することを指示します。

原始プログラムファイルのサフィックスが.F、.FOR、.F90、.F95、.F03、および.F08でない場合、-Cppオプションを指定しなければなりません。

-Pオプションを指定した場合、プリプロセッサ(前処理)だけが実行されます。

本オプションは、-Eオプションと同時には指定できません。

プリプロセッサの結果は、原始プログラムファイルに対応して、以下のように生成します。

原始プログラムファイル	プリプロセッサの結果ファイル
file.F	*file*.cpp.f

原始プログラムファイル	プリプロセッサの結果ファイル
file.f	
file.FOR	*file*.cpp.for
file.for	
file.F90	*file*.cpp.f90
file.f90	
file.F95	*file*.cpp.f95
file.f95	
file.F03	*file*.cpp.f03
file.f03	
file.F08	*file*.cpp.f08
file.f08	

-S

アセンブラ原始プログラム(サフィックス.sのファイル)を生成することを指示します。

-Sオプションが有効な場合、オブジェクトプログラムの作成およびリンカの呼出しは行いません。また、入力ファイルにアセンブラファイルまたはオブジェクトファイルだけが指定された場合は処理を行いません。

{ -SSL2 | -SSL2BLAMP }

数学ライブラリ(SSL II、BLAS、LAPACK)の結合を指示します。詳細については、“SSL IIオンラインマニュアル”、“SSL IIスレッド並列機能オンラインマニュアル”、または “BLAS,LAPACK, ScaLAPACKオンラインマニュアル”を参照してください。

-SSL2

SSL II、SSL IIスレッド並列機能、BLAS／LAPACKを結合します。

-SSL2BLAMP

SSL II、SSL IIスレッド並列機能、BLAS／LAPACKスレッド並列版を結合します(-SSL2オプションとの違いは、BLAS／LAPACKにおいてスレッド並列版が結合されることです)。

-U *name*

#undef前処理命令と同様に*name*の定義を無効にします。-Dオプションと-Uオプションに同一の*name*を指定した場合は、その順序によらず*name*を定義しません。

-V

起動された各コマンドについて、そのバージョン情報を表示します。

-W *tool*,*arg1*[,*arg2*]...

-Wオプションは、指定された*arg1*[,*arg2*]...をそれぞれ別の引数として*tool*に渡します。各引数は、直前の引数とコンマのみで区切られていなければなりません(コンマを引数の一部としたい場合は、直前にバックスラッシュ(¥)を置きます。このとき、バックスラッシュ自体は、引数には含まれません)。*tool*には、以下のいずれかの文字を指定します。

文字	意味
p	プリプロセッサ
0	コンパイラ
a	アセンブラ
l	リンカ

例えば、-Wa,-oobjfileは、-oとobjfileをその順序でアセンブラに渡します。また、-Wl,-Inameにより、リンク段階のダイナミックリンカの省略時名を変更することができます。他の指定したコマンド行オプションに関して、引数がツールに渡される順序は変わることがあります。

-X *lan_lvl*　　*lan_lvl* : { 6 | 7 | 9 | 03 | 08 | d7 }

言語仕様のレベルを指定する機能です。-Xオプションの引数として6、7、9、03、08、またはd7が指定できます。

各言語仕様の間では、解釈が異なる部分があります。利用者は、翻訳するFortran原始プログラムがどの言語仕様で記述されているかをFortranコンパイラに指示します。これにより、正しい解釈でFortran原始プログラムを翻訳・実行することができます。

言語仕様レベルによる解釈の違いについては“6.1 言語仕様レベル”を参照してください。

6

-X6オプションが有効な場合、解釈が異なる部分に関してFortran 66言語仕様を採用します。

7

-X7オプションが有効な場合、解釈が異なる部分に関してFortran 77言語仕様を採用します。

9

-X9オプションが有効な場合、解釈が異なる部分に関してFortran 95言語仕様を採用します。-Nf90moveオプションも同時に有効になります。

03

-X03オプションが有効な場合、解釈が異なる部分に関してFortran 2003言語仕様を採用します。-X08オプションが有効な場合と同じ解釈になります。-Nf90moveオプションも同時に有効になります。-X03オプションが有効な場合、-Nalloc_assignオプションがデフォルトになります。

08

-X08オプションが有効な場合、解釈が異なる部分に関してFortran 2008言語仕様を採用します。-Nf90moveオプションも同時に有効になります。-X08オプションが有効な場合、-Nalloc_assignオプションがデフォルトになります。

d7

-Xd7オプションが有効で、Fortran原始プログラムおよび前処理指令Fortran原始プログラムのファイルサフィックスが.f、.for、.F、または.FORの場合、-X7オプションが有効になります。

-#

-#オプションは、frtpxコマンドによって実行されるパスおよびオプションを表示します。ただし、実行は行いません。

-###

-###オプションは、frtpxコマンドによって実行されるパスおよびオプションを表示します。

--linkstl=*stl_kind*　　*stl_kind* : { libfjc++ | libc++ | libstdc++ }

C++プログラムとの結合で必要な標準テンプレートライブラリ(STL)の検索を行うよう指示します。

言語間結合の詳細については、“第11章 言語間結合”を参照してください。

本オプションは、リンク時に指定する必要があります。

libfjc++

C++コンパイラのtradモードのSTLを検索します。

libc++

C++コンパイラのclangモードのSTLであるlibc++を検索します。

libc++はC++コンパイラのclangモードにおいて、-stdlib=libc++オプションを指定したときに使用されるSTLです。

libstdc++

C++コンパイラのclangモードのSTLであるlibstdc++を検索します。

libstdc++はC++コンパイラのclangモードにおいて、デフォルトで使用されるSTLです。または、-stdlib=libstdc++オプションを指定したときに使用されるSTLです。

結合するC++プログラムが使用しているSTLと、本オプションのサブオプションに指定したSTLが一致しない場合、以下のいずれかの現象が発生する可能性があります。

― リンク時のリンクエラー

－ 実行時の動作不定

2.3 翻訳コマンドの環境変数

翻訳コマンドが認識する環境変数について説明します。

FORT90CPX
FORT90C

翻訳時オプション設定用の環境変数です。環境変数FORT90CPXはクロスコンパイラ用、環境変数FORT90Cはネイティブコンパイラ用です。

環境変数の値として、翻訳時オプションを設定することができます。常に指定する翻訳時オプションは環境変数に設定しておくと便利です。

翻訳時オプションの優先順位は、“2.4 翻訳時プロフィルファイル”を参照してください。

例

例1と例2は等価です。

例1: 翻訳時オプションを環境変数FORT90CPXに設定(shの場合)

```
$ export FORT90CPX="-fw -I/usr/prv/usri"
$ frtpx a.f90
```

例2: 翻訳時オプションを翻訳コマンドのオペランドに指定

```
$ frtpx -fw -I/usr/prv/usri a.f90
```

FCOMP_LINK_FJOBJ

本処理系では、通常、リンク時に本処理系独自オブジェクトを結合しますが、以下の条件のときは結合しません。

－ 本処理系がリンカに渡している-Lオプションを、ユーザが本処理系の翻訳コマンドに直接指定してリンクを行う。

これにより、リンクエラー(undefined reference to)になる場合がありますが、環境変数FCOMP_LINK_FJOBJを設定することで回避できます。

環境変数FCOMP_LINK_FJOBJに設定する値は任意です。環境変数FCOMP_LINK_FJOBJが設定されていると、リンク時に本処理系独自オブジェクトを結合します。

リンクエラーの例は、“A.2.4 リンクエラー(undefined reference to)について”を参照してください。

例

環境変数FCOMP_LINK_FJOBJの設定例

```
$ export FCOMP_LINK_FJOBJ=true
```

FCOMP_UNRECOGNIZED_OPTION

環境変数FCOMP_UNRECOGNIZED_OPTIONを設定することにより、翻訳コマンドが識別不可能な翻訳時オプションに対する動作を変更することができます。

環境変数FCOMP_UNRECOGNIZED_OPTIONには、warningまたはerrorのいずれかを設定します。環境変数FCOMP_UNRECOGNIZED_OPTIONを設定しない場合、または無効な値が設定された場合は、warningが適用されます。

設定値	説明
warning	識別不可能な翻訳時オプションに対して警告メッセージを出力し、翻訳を継続します。
error	識別不可能な翻訳時オプションに対してエラーメッセージを出力し、翻訳を中断します。

例

例1: 環境変数FCOMP_UNRECOGNIZED_OPTIONにwarningを設定(shの場合)

```
$ FCOMP_UNRECOGNIZED_OPTION=warning
$ export FCOMP_UNRECOGNIZED_OPTION
$ frtpx -unrecognized_option a.f90
frtpx: 警告: -unrecognized_optionは認識できないオプションです。
$ echo $?
0
```

例2: 環境変数FCOMP_UNRECOGNIZED_OPTIONにerrorを設定(shの場合)

```
$ FCOMP_UNRECOGNIZED_OPTION=error
$ export FCOMP_UNRECOGNIZED_OPTION
$ frtpx -unrecognized_option a.f90
frtpx: エラー: -unrecognized_optionは認識できないオプションです。
$ echo $?
1
```

LIBRARY_PATH

環境変数LIBRARY_PATHを設定することにより、リンク時にライブラリを検索するディレクトリを追加することができます。

コロン":"を区切りとして、複数のディレクトリを設定することができます。

ライブラリの検索は以下の順序で行われます。

1. -Lオプションの引数に指定されたディレクトリ
2. 本処理系が提供するライブラリを格納するディレクトリ
3. 標準ライブラリのディレクトリ
4. 環境変数LIBRARY_PATHに指定されたディレクトリ

例

環境変数LIBRARY_PATHの使用例

```
$ export LIBRARY_PATH="/usr/local/lib64:/usr/local_2/lib64"
```

TMPDIR

環境変数TMPDIRを設定することにより、翻訳コマンドが使用するテンポラリディレクトリを変更することができます。

環境変数 TMPDIRを設定しない場合、/tmpが使用されます。共通領域へ出力したくない場合は、テンポラリディレクトリを環境変数TMPDIRで変更してください。

例

環境変数 TMPDIRの使用例

```
$ export TMPDIR=/usr/local/tmp
```

2.4 翻訳時プロフィルファイル

翻訳時プロフィルファイル(/etc/opt/FJSVxtclanga/jwd_prof)を設定することにより、翻訳時オプションのデフォルトを変更することができます。

注意

翻訳時プロフィルファイルの設定内容については、システム管理者にお問い合わせください。

翻訳時プロフィルファイルの形式は以下のとおりです。

- 二重引用符(")または一重引用符(')で囲まれていない空白文字は、意味を持ちません。
- 文字列の開始および終了を表す二重引用符および一重引用符は、すべて等価です。ただし、一連の文字列は必ず開始を表す二重引用符および一重引用符と同一の文字または行末によって終了します。
- 文字列に含まれていない“#”から行末まではコメントと解釈します。

例

翻訳時プロフィルファイルの指定例

```
#Default options
-fw -Kocl
```

翻訳時オプションは、以下の優先順位に従って決まります。

1. 翻訳指示行(-Koptions指定時のみ)
2. 翻訳コマンドのオペランド
3. 翻訳時オプション設定用の環境変数
4. 翻訳時プロフィルファイル
5. デフォルト

2.5 翻訳コマンドの復帰値

以下に、frtpxコマンドが設定する復帰コードを示します。

復帰コード	意味
0	正常に終了しました。
1	翻訳エラーまたはリンカでエラーが発生しました。
上記以外	リンカでエラーが発生しました。

2.6 翻訳指示行

原始プログラムの手続の先頭に!options行を記述し、翻訳時オプション-Koptionsが指定されたとき、そこに記述されたオプションを有効にすることができます。

翻訳指示行の形式

```
!options opt [,opt] ....
```

opt: オプション

オプションには-O[1-3]が指定できます。

例:

```
1---5----10---15---20---25
!options -O2
subroutine sub
```

注意事項

モジュール手続および内部副プログラムの先頭には記述できません。

第3章 Fortranプログラムの実行

この章では、Fortran原始プログラムを実行するための手続について説明します。

3.1 実行コマンド

利用者は、翻訳コマンドfrtpxで作成した実行可能プログラムのファイルを実行コマンドとして起動します。これにより、Fortran原始プログラムが実行されます。

3.2 実行コマンドの形式

利用者は、実行コマンド名に続いて、Fortranライブラリで定義される実行時オプションまたは利用者が定義したオプションを指定することができます。

実行時オプションの説明については“3.3 実行時オプション”を参照してください。

以下に、実行コマンドの形式を示します。

コマンド名	オペランド
ファイル名	[□-Wl,実行時オプション[,実行時オプション]...][□利用者定義オプション[,利用者定義オプション]...]

□: 1つ以上の空白が必要なことを意味します。

備考1. ファイル名は、実行可能プログラムが格納されているファイル名であり、翻訳コマンドfrtpxの -oオプションの引数として指定されたものです。-oオプションが省略されたときのファイル名は a.outです。

備考2. オペランドに実行時オプションを指定するときは、先頭に“-Wl,”(lは英小文字)を指定しなければなりません。また、実行時オプション間の区切り子は、コンマ(,)でなければなりません。

備考3. 利用者定義オプションは、本処理系で提供されているサービスサブルーチンGETPARMまたはGETARGを使用することで取り出すことができます。サービスルーチンについては、オンラインマニュアルで提供されている“Fortran文法書”を参照してください。

備考4. オペランドに同じ実行時オプションが指定された場合は、あとで指定された実行時オプションが有効になります。

コマンド名a.outに続き、Fortranライブラリの実行時オプション“-Wl,-i”と利用者が定義したオプション“-K”および“-L”を指定した例を以下に示します。

例:

```
$ ./a.out -Wl,-i -K,-L
```

3.3 実行時オプション

実行時オプションは、実行コマンド名に続いて指定することができます。また、環境変数に指定することもできます。

実行時オプションの形式

-Wl[,-a] [,-d*num*] [,-e*num*] [,-g*num*] [,-i] [,-l*elvl*] [,-m*u_no*] [,-n] [,-p*u_no*] [,-q] [,-r*u_no*] [,-t*sec*] [,-x] [,-Lb] [,-Li] [,-Lr] [,-Lu] [,-Q] [,-Re] [,-Rp] [,-Ry] [,-T*u_no*]

実行時オプションは先頭に“-Wl”(lは英小文字)を指定し、続いて、オプションを指定します。実行時オプション間の区切り子は、コンマ(,)でなければなりません。同じオプションが指定された場合、あとで指定されたオプションが有効となります。

例: 実行時オプションの指定

```
$ ./a.out -Wl,-a,-p10,-x
```

以下に各オプションの意味を説明します。

-a

Fortranプログラムの終了時に、強制的に異常終了させることを指示します。

-aオプションを指定した場合、強制的な異常終了処理が行われます。この異常終了処理は、使用中のファイルのクローズ処理の直前で行われます。同時に動作していたプログラムの仮想記憶領域の内容が coreファイルに出力されます。

例: 実行時オプション -a

```
$ ./a.out -Wl,-a
```

-d*num*　　1 ≦ *num* ≦ 32767

直接探査入出力文が実行されたときに、使用される入出力バッファの大きさを指定します。直接探査入出力で使用される入出力域の大きさはFortran記録を複数ブロッキングすることによって、管理されます。*num*には、Fortran記録の数を表す1～32767の整数値を指定します。numの値が1～32767以外の値である場合、指定は無効になります。

-dオプションを指定した場合、*num*個の Fortran記録が1個のブロッキングレコードとして扱われます。このとき、使用される入出力バッファの大きさは、OPEN文のRECL指定子の値の*num*倍の大きさです。

-dオプションを指定しない場合、使用される入出力バッファの大きさは、8Mバイトとなります。

複数プロセスでファイルが共有される場合は、-d1を指定してください。

-dオプションは、連続した記録番号で入出力する場合に有効です。ただし、-dオプションで指定した値は、実行時に使用されるすべての装置番号に対して有効となります。このため、無駄なメモリが消費されることがあるため、注意してください。装置番号ごとに入出力バッファの大きさを指定する方法として、環境変数で指定する方法があります。環境変数で指定する方法については、“3.8 実行時の環境変数”を参照してください。

-dオプションと環境変数を同時に指定した場合、環境変数が有効となります。

例: 実行時オプション -d

```
$ ./a.out -Wl,-d10
```

-e*num*　　0 ≦ *num* ≦ 32767

実行時に発生したエラーの総数によって、プログラムの実行の終了を制御します。引数*num*は、エラーの総発生許容回数を指示する0～32767の整数値です。*num*≧1のとき、発生したエラーの合計が*num*回に達するとプログラムの実行が終了されます。
-e*num*オプションを指定しない場合または引数*num*に0を指定した場合は、エラーの発生回数によってプログラムの実行が終了されることはありません。ただし、個々のエラーの発生に対して、本処理系で規定される標準的なエラー打切り回数に達した場合は、プログラムの実行は終了されます。例えば、入力データのエラー(jwe0171i-e)が10回起こった場合、-eの値として0や32767を指定していても、プログラムの実行は打ち切られます。

例: 実行時オプション -e

```
$ ./a.out -Wl,-e10
```

-g*num*　　1 ≦ *num* ≦ 2097151

順番探査入出力文または流れ探査入出力文が実行されたときに、使用される入出力バッファの大きさを指定します。順番探査入出力および流れ探査入出力で使用される入出力バッファの大きさは、装置番号ごとにKバイト単位で管理されます。*num*は、1Kバイト単位で指定します。指定可能な値は、1～2097151の整数値です。1～2097151以外の値を指定した場合、指定は無効になります。

-gオプションを指定しない場合、使用される入出力バッファの大きさは、8Mバイトとなります。

-gオプションは、書式なし順番探査入出力文または書式なし流れ探査入出力文で大量データを入出力する場合に有効です。ただし、-gオプションで指定した値は、実行時に使用されるすべての装置番号に対して有効となります。このため、無駄なメモリが消費されることがあるので、注意してください。装置番号ごとに入出力バッファの大きさを指定する方法として、環境変数で指定する方法またはOPEN文のBLOCKSIZE指定子で指定する方法があります。環境変数で指定する方法については、“3.8 実行時の環境変数”を参照してください。
-gオプションと環境変数を同時に指定した場合、環境変数が有効となります。

例: 実行時オプション -g

```
$ ./a.out -Wl,-g10
```

-i

実行時に割込み事象が発生した場合の処理をFortranライブラリで行うかどうかを指示します。

-iオプションを指定した場合、実行時に割込み事象が発生した場合の処理は行われません。

-iオプションを指定しない場合、実行時に割込み事象が発生した場合の処理が行われ、診断メッセージが出力されます。捕捉されるシグナルについては、“8.2.2 異常終了プログラムのデバッグ”を参照してください。また、割込み事象が発生した場合の処理については、“8.1.7 例外ハンドリングエラーの処理”を参照してください。

例: 実行時オプション -i

```
$ ./a.out -Wl,-i
```

-l*elvl*　　*elvl* : { i | w | e | s }

実行時の診断メッセージの出力を制御します。引数*elvl*には、出力する最低のエラーレベルi、w、eまたはsを指定します。-lオプションを指定しない場合は、エラーレベルがw、eまたはsの診断メッセージが出力されます。ただし、診断メッセージの最大印刷回数を超えたものは印刷されません。

例: 実行時オプション -l

```
$ ./a.out -Wl,-le
```

i

-liオプションを指定した場合は、すべてのエラーレベルの診断メッセージが出力されます。

w

-lwオプションを指定した場合は、エラーレベルがw、e、sおよびuの診断メッセージが出力されます。

e

-leオプションを指定した場合は、エラーレベルがe、sおよびuの診断メッセージが出力されます。

s

-lsオプションを指定した場合は、エラーレベルがsおよびuの診断メッセージが出力されます。

-m*u_no*　　0 ≦ *u_no* ≦ 2147483647

標準エラー出力ファイルと接続する装置番号を指定します。引数*u_no*は、装置番号を表す0～2147483647の整数値です。

-m*u_no*オプションを指定した場合、引数*u_no*で指定した装置番号が標準エラー出力ファイルと接続されます。

-m*u_no*オプションを指定しない場合、システムの標準値である装置番号0が標準エラー出力ファイルと接続されます。

*u_no*で指定した装置番号(システム標準値は0)は、標準エラーファイルに結合されるため、環境変数によってファイルと結合できません。環境変数に指定しても、無効となりますので、ご注意ください。なお、端末以外に出力したいときは、リダイレクション機能をご使用ください。

装置番号とファイルの接続については、“3.8 実行時の環境変数”または“7.2 装置番号とファイルの接続”を参照してください。

例: 実行時オプション -m

```
$ ./a.out -Wl,-m10
```

-n

端末に対する促進メッセージの出力を制御します。

-nオプションを指定した場合、書式付き順番探査READ文、並びによるREAD文および変数群READ文を用いて標準入力ファイルからデータを入力するとき、端末に入力促進メッセージが出力されます。

-nオプションを指定しない場合、書式付き順番探査READ文、並びによるREAD文および変数群READ文を用いて標準入力ファイルからデータを入力するとき、端末に入力促進メッセージは出力されません。

例: 実行時オプション -n

```
$ ./a.out -Wl,-n
```

-p*u_no*　　0 ≦ *u_no* ≦ 2147483647

標準出力ファイルと接続する装置番号を指定します。引数*u_no*は、装置番号を表す0～2147483647の整数値です。

-p*u_no*オプションを指定した場合、引数*u_no*で指定する装置番号が標準出力ファイルと接続されます。

-p*u_no*オプションを指定しない場合、システム標準値である装置番号 6が標準出力ファイルと接続されます。

*u_no*で指定した装置番号(システム標準値は6)は、標準出力ファイルに接続されるため、環境変数によってファイルと接続できません。環境変数に指定しても、無効となりますので、ご注意ください。なお、端末以外に出力したいときは、リダイレクション機能をご使用ください。

装置番号とファイルの接続については、"3.8 実行時の環境変数"または"7.2 装置番号とファイルの接続"を参照してください。

例: 実行時オプション -p

```
$ ./a.out -Wl,-p10
```

-q

-qオプションは、入出力文の出力文字を大文字にするか小文字にするかを指定します。書式付き出力文のE形編集、EN形編集、ES形編集、D形編集、Q形編集、G形編集、L形編集およびZ形編集で出力した文字、INQUIRE文の問合せ指定子に返却される文字定数(NAME指定子は除く)の英字の出力文字が変更されます。

-qオプションを指定した場合、英大文字で出力されます。

-qオプションを指定しない場合、英小文字で出力されます。

なお、Fortran 95またはFortran 2003言語仕様では、書式付き出力文のE形編集、EN形編集、ES形編集、D形編集、Q形編集、G形編集、L形編集およびZ形編集で出力した文字、INQUIRE文の問合せ指定子に返却される文字定数(NAME指定子は除く)は、大文字で出力されます。このため、-qオプションは意味をもちません。

例: 実行時オプション -q

```
$ ./a.out -Wl,-q
```

-r*u_no*　　0 ≦ *u_no* ≦ 2147483647

標準入力ファイルと接続する装置番号を指定します。引数*u_no*は、装置番号を表す0～2147483647の整数値です。

-r*u_no*オプションを指定した場合、引数*u_no*で指定する装置番号が標準入力ファイルと接続されます。

-r*u_no*オプションを指定しない場合、システム標準値である装置番号 5が標準入力ファイルと接続されます。

*u_no*で指定した装置番号(システム標準値は5)は、標準入力ファイルに接続されるため、環境変数によってファイルと接続できません。環境変数に指定しても、無効となりますので、ご注意ください。なお、端末以外から入力したいときは、リダイレクション機能をご使用ください。

装置番号とファイルの接続については、"3.8 実行時の環境変数"または"7.2 装置番号とファイルの接続"を参照してください。

例: 実行時オプション -r

```
$ ./a.out -Wl,-r10
```

-t*sec*　　1 ≦ *sec* ≦ 9999

プログラムの実行開始から打ち切るまでの時間を秒数で指定します。この時間は、CPU時間です。引数*sec*は、打切り時間を秒単位で表す1～9999の整数値です。

例えば、-t20を指定した場合、20秒(CPU時間)が経過したら実行が打ち切られます。

例: 実行時オプション -t

```
$ ./a.out -Wl,-t20
```

-x

書式付き入力文で数値編集を行うときに、入力欄の空白を無視するかまたは0として扱うかを指定します。

-xオプションを指定した場合、事前にOPEN文が実行されていない書式付き順番探査入力文で数値編集を行うときに、入力欄の空白は0として扱われます。これは、OPEN文のBLANK指定子で、ZEROを指定した場合と同じです。

-xオプションを指定しない場合、入力欄の空白はNULLとして解釈され、無視されます。これは、OPEN文のBLANK指定子で、NULLを指定するか、省略した場合と同じです。

例: 実行時オプション -x

```
$ ./a.out -Wl,-x
```

-Lb

-Lbオプションを指定すると、サービスルーチンの4バイトの論理型または論理型の引数および復帰値が8バイトの論理型として扱われます。

例: 実行時オプション -Lb

```
$ ./a.out -Wl,-Lb
```

-Li

-Liオプションを指定すると、サービスルーチンの4バイトの整数型または整数型の引数および復帰値が8バイトの整数型として扱われます。

例: 実行時オプション -Li

```
$ ./a.out -Wl,-Li
```

-Lr

-Lrオプションを指定すると、サービスルーチンの4バイトの実数型または実数型の引数および復帰値が8バイトの実数型として扱われます。

例: 実行時オプション -Lr

```
$ ./a.out -Wl,-Lr
```

-Lu

書式なし順番探査入出力文で転送する書式なしFortran記録の長さが2Gバイト以上である場合、複数のFortran記録に分割して入出力がされます。-Luオプションを指定した場合、論理記録の先頭と最後に存在する、Fortran記録の長さを設定するための領域を4バイトから8バイトに拡張し、Fortran記録の長さが2Gバイト以上である場合でも1つのFortran記録として入出力することができます。

書式なし順番探査READ文の実行において、-Luオプションを指定して作成したファイルを入力する場合、実行時オプション-Luを指定してください。また、-Luオプションを指定せずに作成したファイルを入力する場合、実行時オプション-Luは指定しないでください。

ファイルの作成時と入力時で、-Luオプションの有無が異なる場合、書式なし順番探査READ文の実行動作は保証されません。

例: 実行時オプション -Lu

```
$ ./a.out -Wl,-Lu
```

-Q

書式付き入力文の1つのFortran記録を構成する欄の幅が、入力されるFortran記録の長さより長い場合、そのFortran記録のうしろに入力欄の幅が必要とするだけの空白を論理的に詰めるかどうかを指示します。

-Qオプションを指定した場合、事前にOPEN文が実行されていない書式付き順番探査入力文において、空白は詰められません。これは、OPEN文のPAD指定子でNOを指定した場合と同じです。

-Qオプションを指定しない場合は、論理的に空白が詰められます。これは、OPEN文のPAD指定子でYESを指定するか、省略した場合と同じです。

例: 実行時オプション -Q

```
$ ./a.out -Wl,-Q
```

-Re

このオプションを指定することにより、以下のエラー機能が抑止されます。

- 診断メッセージ、トレースバックマップ、エラー集計情報の出力
- ERRSET/ERRSTRサービスサブルーチンによるエラー項目の修正
- 利用者修正出口処理ルーチンの実行

エラーが発生した場合、Fortranシステムの標準修正が行われ、実行が継続されます。

本実行時オプションは、LLVM OpenMPライブラリのエラー機能に対しては無効となります。LLVM OpenMPライブラリのエラー機能については“12.4 実行時メッセージ”を参照してください。

例: 実行時オプション -Re

```
$ ./a.out -Wl,-Re
```

-Rp

並列でプログラムが実行されたかどうかを通知する実行時の診断メッセージの出力を指示します。

-Rpオプションを指定した場合、翻訳時オプション-Kparallelを指定したプログラムが一度も並列で実行されないとき、実行時の診断メッセージ(jwe1034i-w)が出力されます。

-Rpオプションを指定しない場合、実行時の診断メッセージ(jwe1034i-w)は出力されません。

本実行時オプションは、LLVM OpenMPライブラリを使用する場合、無効となります。

例: 実行時オプション -Rp

```
$ ./a.out -Wl,-Rp
```

-Ry

-Ryオプションを指定した場合、西暦年を下2けたで表示するサービスルーチン引用に対して、実行時に診断メッセージが出力されます。西暦年を下2けたで表示するルーチンには、以下のものがあります。

- DATEサービスサブルーチン
- JDATEサービス関数

ユーザプログラムで、返却値を単純に大小比較していると、誤動作する可能性がありますので、プログラムでの考慮が必要になります。実行時オプション-Ry,-liを指定することにより、これらのルーチンを実行した場合に診断メッセージを出力することができます。なお、下4けたで西暦年を通知するルーチンには、以下のものがあります。

- CTIMEサービス関数
- FDATEサービスサブルーチン
- IDATEサービスサブルーチン
- GETDATサービスサブルーチン
- DATE_AND_TIME組込みサブルーチン

例: 実行時オプション -Ry

```
$ ./a.out -Wl,-Ry
```

-Tまたは-T*u_no*　　0 ≦ *u_no* ≦ 2147483647

書式なし入出力文において、ビッグエンディアンデータのファイルを入出力するときに指定します。引数*u_no*は、装置番号を表す0～2147483647の整数値です。引数*u_no*を省略した場合、書式なしファイルと接続しているすべての装置番号に対して有効となります。

引数*u_no*を指定した場合、*u_no*と接続している装置番号に対して有効となります。*u_no*が書式付きのファイルと接続している場合は効果がありません。

-Tオプションを指定した場合、書式なし入出力文において論理型データ、整数型データ、IEEE 754浮動小数点形式データがエンディアン変換されます。

-Tと-T*u_no*を同時に指定した場合、-Tが有効となります。

ビッグエンディアンデータの入出力変換については、“7.11 エンディアン入出力変換”を参照してください。

例: 実行時オプション -T

```
$ ./a.out -Wl,-T10
```

3.4 実行コマンドの環境変数

利用者は、環境変数FORT90Lの値として実行時オプションを設定することができます。環境変数 FORT90Lの中に設定された実行時オプションは、実行コマンドの起動時に有効となります。常に指定する実行時オプションをあらかじめ環境変数FORT90Lの中に設定しておくと便利です。

ただし、環境変数FORT90Lに設定した実行時オプションは、実行コマンド行で無効にすることができません。環境変数FORT90Lに設定した実行時オプションを無効にしたい場合は、環境変数FORT90Lの再設定が必要なので注意してください。

環境変数FORT90Lの中に設定された値は、実行コマンド行に指定されている実行時オプションの最初に付加され処理されます。

環境変数FORT90Lの使用例を以下に示します。

例1:

a. FORT90Lを使用したとき

```
$ export FORT90L='-Wl,-e99,-le'
$ ./a.out -Wl,-m99 -k
```

b. FORT90Lを使用しないとき

```
$ ./a.out -Wl,-e99,-le,-m99 -k
```

a. とb. は等価です。実行コマンドa.outを実行したとき、実行時オプション-e99,-le,-m99および利用者定義のオプション -kが有効になります。

例2:

```
$ export FORT90L='-Wl,-e10'
$ ./a.out -Wl,-e99
```

実行コマンドa.outを実行したとき、実行時オプション-e99が有効になります。

3.5 実行時プロフィルファイル

実行時プロフィルファイル(/etc/opt/FJSVxtclanga/jwe_prof)を設定することにより、実行時オプションのデフォルトを変更することができます。

注意

実行時プロフィルファイルの設定内容については、システム管理者にお問い合わせください。

実行時プロフィルファイルの形式は、以下のとおりです。

- 先頭から“#”で始まる行はコメント行とみなされるため、何を記述してもかまいません。
- 1つの実行時オプションを複数行に渡って記述することはできません。
- 指定できる実行時オプションの形式は、“3.3 実行時オプション”で説明されている記述に従わなければなりません。

例

実行時プロフィルファイルの指定例

```
#Default options
-Wl,-x,-le
```

実行時オプションは、以下の優先順位に従って決まります。

1. 実行コマンドのオペランド
2. 環境変数 FORT90L
3. 実行時プロフィルファイル
4. 標準値

3.6 実行コマンドの復帰値

以下に実行コマンドが設定する復帰コードを示します。

復帰コード	意味
0	プログラムは正常に終了しました。または、iレベル(警告)のエラーが発生しました。
4	wレベル(軽度)のエラーが発生しました。
8	eレベル(中度)のエラーが発生しました。
12	sレベル(重度)のエラーが発生しました。
16	wレベル、eレベルまたは sレベルのエラーの発生回数が打切り回数に達しました。または、uレベル(異常)のエラーが発生しました。
240	プログラムが異常終了しました。
上記以外	実行時オプション -a指定により、プログラムが強制終了しました。

3.7 実行コマンドの標準入出力および標準エラー出力

実行コマンドの標準入力、標準出力および標準エラー出力はそれぞれFortran原始プログラムで扱う装置番号と対応しています。以下に、対応を記述します。

標準入力	:	装置番号5
標準出力	:	装置番号6
標準エラー出力	:	装置番号0

3.8 実行時の環境変数

実行時に指定する環境変数について説明します。

fu*xx filen*

装置番号とファイルを接続するための環境変数です。*xx*は装置番号です。*filen*には、接続するファイル名を指定します。

装置番号とファイルを接続する方法については、“7.2 装置番号とファイルの接続”を参照してください。

この環境変数を使用して、標準入力、標準出力および標準エラー出力と接続される装置番号(実行時オプション-r、-p、-mで指定した装置番号、システム標準である場合は5、6、0)とは接続できませんので、ご注意ください。これらの装置番号とファイルを結合したいときは、リダイレクション機能をご使用ください。

以下に、装置番号10とdata.fileというファイルを指定する例を示します。

例:

```
$ export fu10='data.file'
```

この指定により、実行時に装置番号 10と data.fileが接続されます。

fu*xx*bf *size*

順番探査入出力文、直接探査入出力文、または流れ探査入出力文を実行したときに使用される入出力バッファの大きさを、装置番号ごとに指定するための環境変数です。

*xx*は装置番号です。*size*は、順番探査入出力または流れ探査入出力ではKバイト単位、直接探査入出力ではFortran記録数を単位として指定します。有効となる値は、順番探査入出力または流れ探査入出力では1≦size≦2097151、直接探査入出力では1≦*size*≦2147483647/OPEN文のRECL指定子の値です。*size*の値が無効である場合、入出力バッファの大きさは Fortranシステムの省略値となります。省略値は、8Mバイトです。

実行時オプション-dまたは-gと同時に指定した場合、環境変数が有効となります。但し、*size*の値が無効である場合、実行時オプション-dまたは-gで指定した値が有効となります。

以下に、順番探査入出力または流れ探査入出力で使用する入出力バッファの大きさと、直接探査入出力で使用する入出力バッファの大きさを指定する例を記述します。

例1: 順番探査入出力または流れ探査入出力の場合

装置番号10に対して、順番探査入出力文または流れ探査入出力文が実行されるとき、64Kバイトの入出力バッファが使用されます。

```
$ export fu10bf=64
```

例2: 直接探査入出力の場合

装置番号10に対して、直接探査入出力文が実行されるとき、50個のFortran記録が1個のブロッキングとして扱われ、OPEN文のRECL指定子の値の50倍の大きさの入出力バッファが使用されます。

```
$ export fu10bf=50
```

FLIB_ALLOC_ALIGN *size*

ALLOCATE文で割り付けられる領域を*size*バイト境界にアライメントします。*size*には、16以上の2のべき乗数を指定します。

以下に、環境変数FLIB_ALLOC_ALIGNの指定例を示します。

例:

```
$ export FLIB_ALLOC_ALIGN=128
```

この指定により、ALLOCATE文で割り付けられる領域が128バイト境界にアライメントされます。

FLIB_EXCEPT u

浮動小数点アンダフローによる割込み事象を検出します。

翻訳時に-NRtrapオプションを指定し、実行時に本環境変数を指定した場合、浮動小数点アンダフローによるプログラム割込みが検出され、実行時の診断メッセージ(jwe0012i-e)が出力されます。

翻訳時に-NRtrapオプションを指定しない場合、または、本環境変数を指定しない場合、浮動小数点アンダフローによるプログラム割込みは検出されません。

以下に、環境変数FLIB_EXCEPTの指定例を示します。

```
$ export FLIB_EXCEPT=u
```

FLIB_HOOK_TIME *time*

一定時間間隔呼び出しによるフック機能において、ユーザ定義関数の呼び出し間隔を指定します。環境変数FLIB_HOOK_TIMEに*time*を指定したとき、*time*ミリ秒ごとにユーザ定義関数が呼び出されます。

*time*の単位はミリ秒で、有効となる値は0≦*time*≦2147483647です。*time*に0が指定された場合、一定時間間隔での呼び出しは無効となります。環境変数が指定されていない場合、Fortranシステムの省略値が有効となります。省略値は60000ms(1分)です。

フック機能については、“8.2.3 フック機能”を参照してください。

以下に環境変数FLIB_HOOK_TIMEの指定例を示します。

```
$ export FLIB_HOOK_TIME=600000
```

この指定により、10分間間隔でユーザ定義関数が呼び出されます。

FLIB_RTINFO

実行時情報を出力します。

実行時情報出力機能については、“付録H 実行時情報出力機能”を参照してください。

FLIB_RTINFO_CSV *file*

実行時情報出力機能において、取得した情報を、CSV形式でファイルに出力します。*file*には任意のファイル名が指定可能です。*file*を省略した場合は、flib_rtinfo.csvというファイルに出力します。

実行時情報出力機能については、“付録H 実行時情報出力機能”を参照してください。

FLIB_RTINFO_NOFUNC

実行時情報出力機能において、翻訳時オプション-Nrt_tune_funcを指定した場合でも、利用者定義の手続ごとの情報出力を抑止します。

実行時情報出力機能については、“付録H 実行時情報出力機能”を参照してください。

FLIB_RTINFO_NOLOOP

実行時情報出力機能において、翻訳時オプション-Nrt_tune_loopを指定した場合でも、ループごとの情報出力を抑止します。

実行時情報出力機能については、“付録H 実行時情報出力機能”を参照してください。

FLIB_TRACEBACK_MEM_SIZE *size*

トレースバックマップに出力されるデバッグ情報を保持するヒープ領域の大きさを変更します。

*size*の単位はMiBで、有効となる値は1<=*size*<=128の整数値です。

環境変数が指定されていない場合、または*size*の値が無効である場合、ヒープ領域の大きさはFortranシステムの省略値となります。省略値は16MiBです。

トレースバックマップについては、“4.2.2 トレースバックマップ”を参照してください。

FLIB_UNDEF_VALUE *hex*

未定義データの引用の検査における検査値を変更します。*hex*には16進表記の値が指定可能です。有効となる値は00≦*hex*≦FFの範囲の値になります。環境変数が指定されていない場合、または、*hex*の値が無効である場合、Fortranシステムの省略値が有効となります。省略値は8Bです。

未定義データの引用の検査については、“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照してください。

以下に環境変数FLIB_UNDEF_VALUEの指定例を示します。

```
$ export FLIB_UNDEF_VALUE=8C
```

この指定により、検査値の各バイトには、8Cの値が適用されます。

FLIB_USE_STOPCODE *n*

STOP/ERROR STOP文の終了符号を復帰コードに反映するか否かを指定します。*n*が1の場合は、終了符号の下位1バイト(0から255)を復帰コードに反映します。*n*に1以外の値を指定した場合および本環境変数を指定しない場合には、終了符号を復帰コードに反映しません。

以下に、環境変数FLIB_USE_STOPCODEの指定例を示します。

```
$ export FLIB_USE_STOPCODE=1
```

この指定により、STOP/ERROR STOP文の終了符号が復帰コードに反映されるようになります。

TMPDIR *dir*

名前なしファイルの生成ディレクトリを指定します。環境変数が指定されていない場合、実行時のカレントディレクトリにファイルが生成されます。

以下に環境変数 TMPDIRの指定例を示します。

```
$ export TMPDIR='/tmp'
```

この指定により、名前なしファイルは/tmpに生成されます。

第4章 Fortranシステムの出力情報

この章では、Fortran言語で記述されたプログラムの翻訳および実行において、本処理系が出力する情報について説明します。

4.1 翻訳時の出力情報

ここでは、翻訳時の出力情報について説明します。

4.1.1 翻訳時の診断メッセージ

Fortranプログラムの翻訳時にfrtpxコマンドおよびFortranコンパイラが出力する情報として診断メッセージがあります。診断メッセージは、frtpxコマンド行に指定された各種のオペランドに誤りがある場合、入力原始プログラム中に誤りがある場合、利用者に通知すべき有用な情報がある場合、および誤りではないが注意すべき事項がある場合に出力されます。

frtpxコマンドが出力する診断メッセージの形式を以下に示します。

```
frtpx：メッセージ本文
```

Fortranコンパイラが出力する診断メッセージの形式を以下に示します。

```
jwdxxxxz-y　ファイル名　行番号　桁位置　ループ識別番号　メッセージ本文
```

jwd*xxxxz-y*

jwd : Fortranコンパイラのメッセージを表します。

xxxx : メッセージの通し番号を表します。

z : i,o,sまたはpの英字のうちどれかであり、メッセージ種別を表します。

メッセージ種別	説明
i	最適化以外のエラーや注意を促すメッセージ
o	最適化の状況を示すメッセージ
s	SIMD化の状況を示すメッセージ
p	自動並列化の状況を示すメッセージ

y: i, w, sまたはuの英字のうちどれかであり、メッセージのエラーレベルおよび復帰コードを表します。

エラーレベル	復帰コード	説明
i	0	エラーではないが注意を促すメッセージです。
w		軽度のエラーを示すメッセージであり、システムは処理を続行します。
s	1	重度のエラーを示すメッセージであり、システムはエラーの文を無視し、その他の文については処理を続行します。
u		致命的なエラーを示すメッセージであり、システムは処理を打ち切ります。

ファイル名

エラーが発生した原始プログラムのファイル名を表示します。

行番号

エラーが発生した原始プログラムの行番号を表示します。

注意

最適化の影響を受け、行番号がずれる場合があります。

桁位置

必要に応じて、エラーが発生した原始プログラムの行内の桁位置を表示します。

ループ識別番号

必要に応じて、最適化(ループ分割など)によりコンパイラが内部的に生成したループの識別番号を表示します。

注意

ループの識別番号は、ループの実行順序を保証するものではありません。

メッセージ本文

利用者環境でのメッセージ種別に従い、英語または日本語のどちらかでメッセージ本文を出力します。

4.1.2 翻訳時のガイダンスメッセージ

翻訳時オプション-Koptmsg=guideを指定すると、以下の最適化が適用できなかった場合の阻害要因と対処方法を示すガイダンスメッセージを出力することができます。

- SIMD化
- 自動並列化
- ソフトウェアパイプライニング
- インライン展開

ガイダンスメッセージは、各診断メッセージに続けて出力します。

出力形式を以下に示します。

```
jwdxxxxz-y 診断メッセージ
  [ガイダンス]
  ガイダンスメッセージ本文
```

診断メッセージの詳細は、“4.1.1 翻訳時の診断メッセージ”を参照してください。

ガイダンスメッセージの出力例を以下に示します。

```
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: ループをフルアンローリングしました。
  jwd6204s-i  "guide.f", line 4: 変数'n2'と'n1'の大小関係が分からないため、定義引用順序が逐次実行と変わる可能性があり、このDOループはSIMD化できません。
    [ガイダンス]
     ループをSIMD化した時のデータの定義引用順序が逐次実行と同じであることが明確な場合は、以下のいずれかの対処をする。
       - データが回転をまたいで定義引用されない場合、最適化指示子NORECURRENCEを指定する。
       - 回転をまたいで定義引用される場合、最適化指示子NOVRECを指定する。
```

4.1.3 翻訳情報出力

ここでは、翻訳情報について説明します。

4.1.3.1 翻訳情報出力のオプション

翻訳時オプション-Nlstまたは-Nlst_out=*file*を指定することにより、翻訳情報を得ることができます。翻訳情報は.lstファイルに出力されます。Fortran原始プログラムを複数ファイル指定した場合には、最初のファイル名.lstファイルに出力されます。また、サブオプションを指定することにより、翻訳情報を制御することができます。以下にオプションの指定とその意味を示します。翻訳情報の出力例については、“4.1.3.2 翻訳情報の出力形式”を参照してください。

-Nlst

原始プログラムリスト、エラーメッセージを出力します。

-Nlst=a

翻訳時オプション-Nlstでの出力に加えて、名前の属性情報も出力します。

-Nlst=d

翻訳時オプション-Nlstでの出力に加えて、派生型の構成情報も出力します。

-Nlst=i

翻訳時オプション-Nlstでの出力に加えて、インクルードされたファイルのプログラムリスト、インクルードファイル名一覧も出力します。

-Nlst=m

翻訳時オプション-Nlst=pでの出力に加えて、自動並列化の状況をOpenMP指示文および指示節によって表現した原始プログラムを出力します。本機能によって出力されるOpenMP指示文および指示節には、入力ファイルにすでに記述されていた指示文および指示節と区別するために !FRTが最後に付加されます。なお、自動並列化の状況をOpenMP指示文および指示節で正確に表現できない場合、“!$OMP”ではなく“!!!!! PARALLEL INFO:”で始まる行が出力されます。その行には、自動並列化と同時に行われた最適化の情報が付加されることがあります。

1. 本機能によって出力されるOpenMP指示文および指示節の種類を以下に示します。

   - 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. “!!!!! PARALLEL INFO:”で始まる行に出力される情報の種類を以下に示します。なお、これらの情報を参考にして、利用者がプログラムを書き換えることにより、該当ループを容易にOpenMP化することができます。

   - LOOP INTERCHANGED
     ループ交換が行われたことを意味します。
   - LOOP FUSED
     ループ融合が行われたことを意味します。
   - LOOP FISSION
     ループ分割が行われたことを意味します。

― COLLAPSED
ループが一重化されたことを意味します。

― OTHER METHODS
上記以外の要因により、自動並列化の状況をOpenMP指示文および指示節で正確に表現できないことを意味します。
主な要因を以下に示します。

- DOループ以外である。
- 誘導変数(ループの繰返しに伴って一定値ずつ増加または減少する、DO変数以外の変数)を含むループである。
- 派生型の変数を含むループである。
- ループ内で添字が不変な配列要素への定義を含むループである。
- リダクション演算の対象変数が、ループ内で添字が不変な配列要素である。
- ループの初期値、終値、増分値が、配列要素や関数引用などを含んでいる。

― UNKNOWN VARIABLE(*var_name*)

OpenMP指示文および指示節の種別を正確に判断できなかった変数を意味します。

主な要因を以下に示します。

- 最適化の対象変数のため、OpenMP指示文および指示節の種別を正確に判断できない変数
- 複素数型の変数

― OpenMPの指示節 (*var_name*)
OpenMPの指示文を正確に判断できなかった場合、参考情報としてOpenMPの指示節の情報が出力されることがあります。指示節の種類を以下に示します。

- 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. 使用する際の注意事項を以下に示します。

― -Pオプションによって生成されたファイルに対して、-Nlst=mオプションを指定して翻訳した場合、前処理を行う前のファイル名に対して.ompが付加されたファイルが作成されます。

例:

```
$ frtpx -P file.F95
```

file.cpp.f95が生成されます。

```
$ frtpx -Nlst=m -Kparallel file.cpp.f95
```

OpenMP指示文が挿入されたファイルとしてfile.omp.F95が生成されます。

― MAXやMINという名前をもつ変数やプログラム単位が存在する原始プログラムの場合、REDUCTION([MAX | MIN]:*var_name*)というOpenMP指示文が生成されると、その原始プログラムはSレベルのエラーとなり翻訳することができません。その際には、MAXやMINという名前を使用しないように原始プログラムを修正してください。

ー 最適化の影響により、自動並列化された場合でも、OpenMP指示文や“!!!!! PARALLEL INFO:”で始まる行が出力されない場合があります。

-Nlst=p

翻訳時オプション-Nlstでの出力に加えて、自動並列化および最適化情報も出力します。以下に表示記号についての意味を説明します。なお、OpenMP仕様によって並列化を指示された文についての出力は、“12.3.1.2 OpenMPプログラムの最適化情報を出力するための翻訳時オプション”を参照してください。

1. 自動並列化情報
   DO文または配列演算に対する表示記号の意味は以下のとおりです。これらはDOループ全体または配列演算の並列化状況を表します。

p	DOループまたは配列演算が並列化されたことを示します。
m	DOループまたは配列演算が並列化された部分とされなかった部分があることを示します。
s	DOループまたは配列演算が並列化されなかったことを示します。
空白	DOループまたは配列演算が並列化対象でないことを示します。

   DO文に対して上記のp、mまたはsが表示された場合、DOループ中の実行文に対してもp、mまたはsが表示されます。実行文に対する表示記号の意味は以下のとおりです。

p	並列化可能な実行文であることを示します。
m	並列化可能な部分と不可能な部分が含まれる実行文であることを示します。
s	並列化不可能な実行文であることを示します。

   並列化の対象となった文のうち、非実行文および実行の制御のみを行う実行文（ELSE文、END IF文、CONTINUE文、ENDDO文、ELSEWHERE文など）は、前後の実行文の並列化情報に依存した記号で表示されます。また、これらの文の並列化情報は空白となることがあります。
   DOループまたは配列演算が並列化可能であっても、性能を考慮して、本処理系が並列化しない場合があります。この場合、DO文または配列演算に対してsを表示して、DOループ内の文に対しては解析結果をそのまま表示します。

2. 最適化情報
   最適化の表示記号の意味は以下のとおりです。

i	インライン展開されたことを示します。
f	フルアンローリングされたことを示します。
数字	アンローリング展開数を示します。

-Nlst=t

翻訳時オプション-Nlst=pでの出力に加えて、より詳細な最適化情報および統計情報を出力します。本機能で出力する内容は、ループ単位に出力する最適化情報や文単位で出力する並列化次元情報および手続ごとに、その手続の1回の引用で消費されるスタックの大きさを大まかに出力する統計情報があります。

1. ループごとに出力する情報を以下に示します。

   a. 自動並列化に関する情報を表示

      - Standard iteration count: *N*
        ループの繰返し数が、*N*以上のときは並列実行し、*N*未満のときは逐次実行することを意味します。

   b. ループに対して行った最適化情報を表示

      - INTERCHANGED(nest: *nest-no*)
        ループ交換されたことを意味します。

        - *nest-no*は、ループ交換の最適化によって移動されたループの入れ子の深さを表します。

        - (nest:)は、一重化などのほかの最適化が行われた場合は表示されないことがあります。

- FUSED(lines: *num1*,*num2*[,*num3*]...)
ループ融合が行われたことを意味します。

  - *num1*の行番号のループに、*num2*以降のループが融合されたことを表します。先行ループに吸収された*num2*以降の行番号のループについては、(lines:)は表示されません。

- FISSION(num: *N*)
ループ分割が行われたことを意味します。
*N*は、ループ分割後のループ数を表します。

- COLLAPSED
ループが一重化されたことを意味します。

- SOFTWARE PIPELINING(IPC: *ipc*, ITR: *itr*, MVE: *mve*, POL: *pol*)
ループに対しソフトウェアパイプラインされたことを意味します。

  - *ipc*は、ソフトウェアパイプライニングが適用されたループの、サイクル当たりの命令数 (Instructions Per Cycle)の予測値です。

  - *itr*は、ソフトウェアパイプライニングが適用されたループが実行時に選択されるために、必要なループの繰返し数です。翻訳時メッセージjwd8205o-iで出力される値と同じです。

  - *mve*は、ソフトウェアパイプライニングによるループの展開数です。

  - *pol*は、使用された命令スケジューリングアルゴリズムを示します。`S`は小さなループに、`L`は大きなループに適したアルゴリズムが使用されたことを意味します。それぞれ、-Kswp_policy=small、-Kswp_policy=largeオプションを指定したときに使用されるアルゴリズムです。

- SIMD
SIMD化されたことを意味します。以下のいずれかで表示されます。

  - SIMD(VL: *length*[,*length*]...)
SIMD化されたことを意味します。*length*は、1つのSIMD命令で処理する配列の要素数を表します。ループ分割された各ループで*length*が異なる場合、*length*が複数表示されます。

  - SIMD(VL: AGNOSTIC; VL: *length*[,*length*]... in 128-bit)
SVEのベクトルレジスタを特定のサイズとみなさずSIMD化されたことを意味します。*length*は、SVEのベクトルレジスタサイズを128ビットとみなして、1つのSIMD命令で処理する配列の要素数を表します。ループ分割された各ループで*length*が異なる場合、*length*が複数表示されます。

- STRIPING
ループストライピングされたことを意味します。

- MULTI-OPERATION FUNCTION
マルチ関数化された関数を含むことを意味します。

- PATTERN MATCHING(matmul)
ループをライブラリ呼出し（matmul）に変換したことを意味します。

- UNSWITCHING
ループアンスイッチングされたことを意味します。

- FULL UNROLLING
ループがフルアンローリングされたことを意味します。

- CLONE
ループがCLONE最適化されたことを意味します。

- LOOP VERSIONING
ループバージョニングされたことを意味します。

c. プリフェッチに関する情報を表示

- ハードウェアプリフェッチ

- PREFETCH(HARD) Expected by compiler:

  ループ内で使用される連続的にアクセスされる配列データに対して、ハードウェアプリフェッチを利用することを期待しprefetch命令を生成しないことを示します。

- 配列名,...

  ハードウェアプリフェッチを利用することを期待した配列名を示します。コンパイラが内部的に生成した配列は"(unknown)"と表示します。

- prefetch命令

  - PREFETCH(SOFT) : *N*

    ループ中に存在するすべてのprefetch命令の個数を示します。
    また、本表示に続けて、プリフェッチのアクセス種別ごとに、以下の形式でprefetch命令の個数と配列名を示します。

    アクセス種別 : *N*
    配列名 : *N*,...

    - アクセス種別

      - SEQUENTIAL : *N*
      ループ内で使用される連続的にアクセスされる配列データに対するprefetch命令の個数を示します。

      - STRIDE : *N*
      ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対するprefetch命令の個数を示します。

      - INDIRECT : *N*
      ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対するprefetch命令の個数を示します。

      - SPECIFIED : *N*
      最適化指示子PREFETCH_READまたはPREFETCH_WRITEの指定によって生成したprefetch命令の個数を示します。

    - 配列名

      - 配列名 : *N*,...
      配列ごとのprefetch命令の個数を示します。コンパイラが内部的に生成した配列は"(unknown)"と表示します。

d. zfillの最適化に関する情報を表示

  - ZFILL
  zfillの最適化が適用されたことを示します。

  - 配列名,...
  zfillの最適化が適用された配列名を示します。コンパイラが内部的に生成した配列は"(unknown)"と表示します。

e. レジスタに関する情報を表示

  - SPILLS:
  最内ループ中に存在するレジスタの退避・復元命令の個数をレジスタ種別ごとに示します。

  - GENERAL : SPILL *N* FILL *N*
  汎用レジスタのメモリへの退避・復元命令の個数を示します。

  - SIMD&FP : SPILL *N* FILL *N*
  SIMDと浮動小数点レジスタのメモリへの退避・復元命令の個数を示します。

  - SCALABLE : SPILL *N* FILL *N*
  拡張レジスタのメモリへの退避・復元命令の個数を示します。

  - PREDICATE : SPILL *N* FILL *N*
  プレディケートレジスタのメモリへの退避・復元命令の個数を示します。

2. DO文または配列演算に対する表示記号を以下に示します。

p	並列化された範囲の先頭を示します。(-Nlst=pのpと同時に出力され、ppとなります)

3. 統計情報に出力する詳細情報を以下に示します。

a. 手続ごとに、その手続の1回の引用で消費されるスタックの大きさを出力します。注意事項を以下に示します。

  - 内部手続で使用するスタックの大きさは、親手続に含めて表示されます。

- 以下のデータは、スタックに割付けられますが、その大きさは、本機能の出力には含まれません。

  - THREADPRIVATE指示文に指定されたデータ

  - -Kautoobjstackオプションが有効な場合の自動割付けデータ

b. 手続ごとのprefetch命令の個数を示します。

4. SIMD化情報

DO文および配列演算に対する表示記号の意味は、以下のとおりです。これらは、DOループ全体および配列演算のSIMD化状況を表します。

v	DOループおよび配列演算がSIMD化されたことを示します。
m	DOループおよび配列演算がSIMD化された部分とされなかった部分があることを示します。
s	DOループおよび配列演算がSIMD化されなかったことを示します。
空白	DOループおよび配列演算がSIMD化対象でないことを示します。

DO文に対して上記のv、mまたはsが表示された場合、DOループ中の実行文に対してもv、mまたはsが表示されます。実行文に対する表示記号の意味は、以下のとおりです。

v	SIMD化可能な実行文であることを示します。(注)
m	SIMD化可能な部分と不可能な部分が含まれる実行文であることを示します。(注)
s	SIMD化不可能な実行文であることを示します。

SIMD化の対象となった文のうち、非実行文および実行の制御のみを行う実行文(ELSE文、END IF文、CONTINUE文、ENDDO文、ELSEWHERE文など)は、前後の実行文のSIMD化情報に依存した記号で表示されます。また、これらの文のSIMD化情報は空白となることがあります。

DOループおよび配列演算がSIMD化可能であっても、性能を考慮し、本処理系がSIMD化しない場合があります。この場合、DO文および配列演算に対してsを表示し、DOループ内の文に対しては、解析結果をそのまま表示します。

(注) DOループがSIMD化された場合(DOループにvまたはmが表示された場合)、SIMD命令が使用されない実行文に対してもvが表示されることがあります。DOループをSIMD化できない要因が含まれている場合には、sが表示されますので、チューニングの指針としてください。

-Nlst=x

翻訳時オプション-Nlstでの出力に加えて、名前および文番号の相互参照情報も出力されます。

これらのサブオプションは、-Nlst=a,lst_out=*file*,lst=xのようにコンマで区切って指定することができます。

-Nlst_out=*file*

*file*に指定されたファイル名に翻訳情報を出力します。

4.1.3.2 翻訳情報の出力形式

翻訳時オプション-Nlstを指定した場合の出力例を以下に示します。

```
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
```

診断メッセージが出力された場合の出力例を以下に示します。

```
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    END DO
         6     1        END PROGRAM

 Diagnostic messages: program name(ERROR)
   jwd1027i-s  "error.f", line 3: 文番号'    20'は定義されていません。
   jwd1131i-s  "error.f", line 3, column 9: DO構文、IF構文、CASE構文、SELECT TYPE構文、ASSOCIATE構文、CRITICAL構文、またはBLOCK構文の入れ子が正しくありません。

 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: この仮引数'I'は、副プログラム中で使用されていません。
   jwd1771i-s  "error.f", line 10, column 17: 'I'は、すでにINTENT属性をもつので指定できません。

 Procedure information
   Lines      : 5
   Statements : 4

 Total information
   Procedures       : 2
```

```
  Total lines      : 11
  Total statements : 10
```

翻訳時オプション-Nlst=iを指定した場合の出力例を以下に示します。

```
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 "INC1"
   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 "inc1"
   1         1        use mod,only:cc
   2         2
            13              CC = "FUJITSU FORTRAN"
            14              END SUBROUTINE

Procedure information
  Lines      : 8
  Statements : 4

Total information
  Procedures       : 3
  Total lines      : 18
  Total statements : 12
Include file name list
   1 : ./inc1
   2 : ./inc2
```

翻訳時オプション-Kparallel -x- -Nlst=pを指定した場合の出力例を以下に示します。

```
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               END DO
         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
```

翻訳時オプション-Nlst=xを指定した場合の出力例を以下に示します。

```
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
```

翻訳時オプション-Nlst=a,lst=xを指定した場合の出力例を以下に示します。

```
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
```

翻訳時オプション-Nlst=dを指定した場合の出力例を以下に示します。

```
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
```

翻訳時オプション-Kparallel,reduction -Nlst=mを指定した場合に出力される(.omp.ファイル)例

```
      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
```

翻訳時オプション-Kfast -Kparallel -Nlst=tを指定した場合の出力例を以下に示します。

```
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 翻訳情報の注意事項

翻訳情報には、以下の注意事項があります。

- 翻訳時オプション-Nlst=pまたは-Nlst=tで出力される翻訳情報(最適化情報)は、以下のようなケースで正しく出力されない場合があります。
  - ー インライン展開が行われた場合、最適化の動作はインライン展開された場所によって異なる可能性があります。このようなケースでは、以下のように出力される可能性があります。
    - 1つのループに対して、複数の最適化メッセージが出力される。
    - 翻訳情報(最適化情報)と反対の意味の最適化メッセージが出力される。
    - 翻訳情報(最適化情報)が出力されない。

    また、インライン展開された関数では、prefetch数は累計されて出力されます。
  - ー SIMD化、自動並列化、ループ融合、ループ分割など詳細な最適化情報として出力される最適化が、1つのループに対して複数適用されるケースでも、以下のように出力される可能性があります。
    - 行番号に対する最適化情報がずれる。
    - 翻訳情報(最適化情報)と反対の意味の最適化メッセージが出力される。
  - ー ループアンスイッチング最適化は、IF構文の条件が成立した場合のループと成立しない場合のループを生成しますが、その複数のループのうちいずれかの翻訳情報と最適化メッセージが出力されます。また、行番号に対する最適化情報が出力されない場合があります。
  - ー 1行に複数のループを記述した場合、複数ループのうちいずれかの最適化情報が出力されることになります。最適化情報を出力したいループは可能な限り、同一行に記述することは避けてください。
  - ー 条件文とGOTO文で構成されるループでは詳細な最適化情報は出力されません。行番号に対する最適化情報もずれる可能性があります。
- 以下のいずれかの条件を満たす場合、コンパイラはループを生成することがあります。翻訳時オプション-Nlst=pまたは-Nlst=tで出力される翻訳情報(最適化情報)および最適化メッセージが、その生成ループに対して出力されることがあります。
  - ー 実引数がポインタ配列、形状引継ぎ配列、または部分配列であり、対応する仮引数がポインタでも形状引継ぎ配列でもない。
  - ー 翻訳時オプション-Nquickdbg=undef、-Nquickdbg=undefnan、または-Nsetvalue=array が有効である。
  - ー OpenMPのFIRSTPRIVATE、LASTPRIVATE、REDUCTION、COPYIN、またはCOPYPRIVATE指示節に配列の変数名が現れる。
  - ー 翻訳時オプション-Karray_privateまたは最適化指示子ARRAY_PRIVATE、FIRST_PRIVATE、LAST_PRIVATEにより、自動並列化されたループがある。
- リンク時最適化が適用された場合、翻訳情報で表示される最適化とは異なった最適化が実行時に適用されることがあります。
- ソースプログラムに#line指令が含まれている場合、翻訳時オプション-Nlst=pまたは-Nlst=tで出力される翻訳情報(最適化情報)は、#line指令に指定した行番号を元に出力されます。例えば、ソースプログラムの10行目のDO文に対し、#line指令で3行目と指定していた場合、DO文の翻訳情報(最適化情報)はソースプログラムの3行目に対して出力されます。

4.2 実行時の出力情報

Fortranプログラムを実行すると、以下の情報が出力されます。

- 出力文による印刷情報
- PAUSE文またはSTOP/ERROR STOP文によるメッセージ
- 診断メッセージ
- トレースバックマップ
- エラー集計情報
- デバッグ機能の出力情報

・ 実行時情報出力機能の出力情報

ここでは、Fortranプログラムの実行時にエラーが起こったときに出力される診断メッセージ、トレースバックマップおよびエラー集計情報について説明します。その他の出力情報については、それぞれの項目を参照してください。出力文については“7.7 入出力文の使用方法”、PAUSE文およびSTOP/ERROR STOP文については“6.5.6 PAUSE文”および“6.5.7 STOP/ERROR STOP文”、デバッグ機能については“8.2 デバッグのための機能”、実行時情報出力機能については“付録H 実行時情報出力機能”を参照してください。

4.2.1 実行時の診断メッセージ

診断メッセージは、Fortranプログラムの実行中に、PAUSE文およびSTOP/ERROR STOP文が実行されたときまたはエラーが検出されたときに出力されます。

診断メッセージは以下の形式で出力されます。

```
jwexxxxt-y 付加情報 メッセージ本文
```

jwe*xxxxt-y*

jwe : Fortranプログラムの実行時に出力されるメッセージを示します

xxxx : メッセージ番号を表します

t : iまたはaが出力されます。iは応答不要なメッセージを表します。また、aは応答が必要なメッセージを表し、PAUSE文が実行された場合にだけ出力されます。PAUSE文については“6.5.6 PAUSE文”を参照してください。

y : i、w、e、sまたは uのどちらかの英字であり、診断メッセージのレベルを表します。診断メッセージのレベルは以下の意味を持っています。

y	診断メッセージのレベルの意味
i	エラーではありません。注意を促しています。
w	軽度のエラーが発生したことを示します。実行は続行されます。
e	中度のエラーが発生したことを示します。エラーが発生した文は無視されます。標準は10回エラーが発生すると、実行が打ち切られます。
s	重度のエラーが発生したことを示します。エラーが発生した文は無視されます。標準は1回エラーが発生すると、実行が打ち切られます。
u	実行が続行できないような致命的なエラーが発生したことを示します。実行は打ち切られます。

付加情報

必要に応じてエラーが発生した原始プログラムの行番号を表示します。

メッセージ本文

エラーの内容を表す英文が出力されます。

4.2.2 トレースバックマップ

トレースバックマップは、エラー制御表にトレースバックマップを印刷するという指定がある場合に、診断メッセージの直後に出力されます。トレースバックマップの情報は、主プログラムからエラーが発生したプログラム単位までの呼出し関係を出力します。ただし、例外ハンドリングの処理が行われた場合、トレースバックマップの情報が正しく出力されない場合がありますのでご注意ください。

また、トレースバックマップの情報は、スタートアップで獲得されたヒープ領域に保持されているデバッグ情報から取得します。情報が出力されない場合は、環境変数FLIB_TRACEBACK_MEM_SIZEでヒープ領域の大きさをFortranシステムの省略値より大きくすると、出力されることがあります。なお、ヒープ領域の不足が原因で情報が取得できない場合は、診断メッセージjwe1655i-iが出力されます。この診断メッセージはエラー処理中に出力されるため、診断メッセージが出力される順序が正しくない場合があります。

エラー制御表については“8.1 エラー制御”、例外ハンドリングについては“8.1.7 例外ハンドリングエラーの処理”および“8.2.2 異常終了プログラムのデバッグ”、翻訳時オプション-Nnolineについては“2.2 翻訳時オプション”、環境変数FLIB_TRACEBACK_MEM_SIZEについては“3.8 実行時の環境変数”を参照してください。

```
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 : エラーが発生したプログラム単位の入口名または組込み関数名です。各プログラム名が不定の場合、'????????'が表示されます。

name2 : name1を呼び出したプログラム単位の入口名または組込み関数名です。各プログラム名が不定の場合、'????????'が表示されます。

name3 : 主プログラムの名前です。o.sは、制御プログラムを表わしています。各プログラム名が不定の場合、'????????'が表示されます。

s1 : エラーが発生した文の行番号です。翻訳時オプション-Nnolineを指定した場合、出力されません。

s2 : 各プログラム単位を呼び出している文の行番号です。翻訳時オプション-Nnolineを指定した場合、出力されません。

wwwwwwww : エラーが発生した文の絶対番地が16進数で表示されています。

xxxxxxxx : エラーが発生した文のプログラム単位の先頭からの相対番地が16進数で表示されています。不定の場合は、表示されません。

yyyyyyyy : 各プログラム単位の先頭の絶対番地が 16進数で表示されています。不定の場合は、'0000000000000000'が表示されます。

zzzzzzzz : 各プログラム単位を呼び出している文の絶対番地が16進数で表示されています。不定の場合は、'0000000000000000'が表示されます。

name1、*name2*、および*name3*は、原始プログラムで記述した手続名を以下の加工した名前で表示されます。

手続の種類	加工名
主プログラム	MAIN__
主プログラムの内部副プログラム	主プログラム名.内部副プログラム名_
外部副プログラム	外部副プログラム名_
外部副プログラムの内部副プログラム	外部副プログラム名.内部副プログラム名_
モジュールのモジュール副プログラム、またはモジュールで分離引用仕様を宣言した分離モジュール手続	モジュール名.モジュール副プログラム名_
モジュールのモジュール副プログラムの内部副プログラム、またはモジュールで分離引用仕様を宣言した分離モジュール手続の内部副プログラム	モジュール名.モジュール副プログラム名.内部副プログラム名_
サブモジュールのモジュール副プログラム	_祖先モジュール名.サブモジュール名.モジュール副プログラム名_
サブモジュールで分離引用仕様を宣言した分離モジュール手続	_祖先モジュール名.サブモジュール名.モジュール副プログラム名_(注)
サブモジュールのモジュール副プログラムの内部副プログラム	_祖先モジュール名.サブモジュール名.モジュール副プログラム名.内部副プログラム名_
サブモジュールで分離引用仕様を宣言した分離モジュール手続の内部副プログラム	_祖先モジュール名.サブモジュール名.モジュール副プログラム名.内部副プログラム名_(注)

注) サブモジュール名は、分離引用仕様を宣言しているサブモジュール名です。

備考1. 翻訳時オプション-AUが指定されていないとき、主プログラムを除く加工名は、すべて小文字に統一されます。

備考2. 共有オブジェクト内のトレースバックマップ情報は、保証されません。

4.2.3 エラー集計情報

Fortranプログラムの実行の終了時に、以下のようなエラーの集計情報が出力されます。

ただし、エラー識別番号900から1000までのシステムエラーは、エラー集計情報として出力されません。

```
error summary (Fortran)
error number error level  error count
  jwennnni          y          zzz
total error count = xxx
```

nnnn : 診断メッセージの番号です。

y : yレベル(y :i、w、e、s、u)のエラーが発生したことを表しています。

zzz : エラーの発生回数を表しています。

xxx : それぞれの診断メッセージの合計回数を表しています。

第5章 データの型、種別、および内部表現

この章では、データの型、種別、および内部表現について説明します。

5.1 データの表現

データは、型により、値の表現範囲、内部表現および占有記憶領域が異なります。

5.1.1 整数型データ

1バイトの整数型(one-byte integer type)は、整数値を正確に表現し、-128から127までの値をとり、1バイトの記憶領域を占めます。種別値1が対応します。

2バイトの整数型(two-byte integer type)は、整数値を正確に表現し、-32768から32767までの値をとり、2バイトの記憶領域を占めます。種別値2が対応します。

4バイトの整数型(four-byte integer type)は、整数値を正確に表現し、-2147483648から2147483647までの値をとり、4バイトの記憶領域を占めます。種別値4が対応し、Fortran規格の基本整数型に対応します。

8バイトの整数型(eight-byte integer type)は、整数値を正確に表現し、-9223372036854775808から9223372036854775807までの値をとり、8バイトの記憶領域を占めます。種別値8が対応します。

各整数型の先頭のビットは、符号を示します。正の整数は、符号ビットがゼロで、残りのビットが真数表示の2進数として表現されます。負の整数は、符号ビットに1をもつ2の補数として表現されます。ゼロの整数は、すべてのビットがゼロとなります。

5.1.2 論理型データ

1バイトの論理型(one-byte logical type)は、真または偽の値だけをとり、1バイトの記憶領域を占めます。種別値1が対応します。

2バイトの論理型(two-byte logical type)は、真または偽の値だけをとり、2バイトの記憶領域を占めます。種別値2が対応します。

4バイトの論理型(four-byte logical type)は、真または偽の値だけをとり、4バイトの記憶領域を占めます。種別値4が対応し、Fortran規格の基本論理型に対応します。

8バイトの論理型(eight-byte logical type)は、真または偽の値だけをとり、8バイトの記憶領域を占めます。種別値8が対応します。

論理型は2進数で表現され、真の値は、最後のビットが1であり、かつ、残りのビットはすべてゼロです。偽の値は、すべてのビットがゼロです。

5.1.3 実数型および複素数型データ

実数型データの内部表現は、IEEE 754に準拠しています。

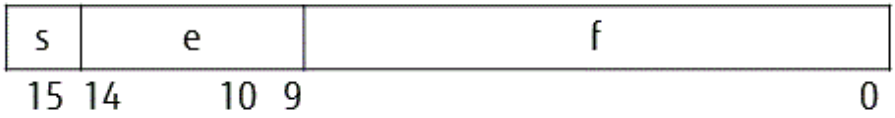
複素数型データの実部と虚部の内部表現は、同じ種別の実数型の内部表現と同じです。実部と虚部は、連続した記憶域を占めるので、実数の倍の領域で1つのデータとなります。

以下に、半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型のデータフォーマットの内部表現を示します。

半精度実数型

半精度実数型(half precision real type)は、実数値を近似的に表現し、精度が2進で11けた(10進で約3けた)であり、絶対値がゼロまたは2^{-14}(約10^{-4})から2^{15}(約10^{4})までの値をとり、2バイトの記憶領域を占めます。種別値2が対応します。

以下は単精度実数型における符号(s)、指数部(e)、仮数部(f)のビット構成を示しています。



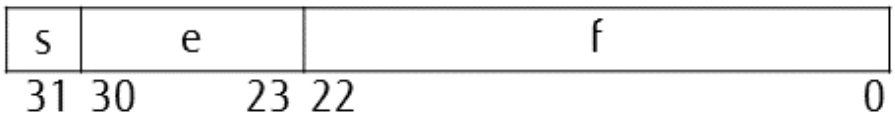
以下に特殊データの内部表現とそれに対応する値を示します。

名称	16進定数値	10進定数値
0	0000	0.0E+00
＋正規化された最大値	7BFF	65504
－正規化された最大値	FBFF	-65504
＋正規化された最小値	0400	6.10351562E-05
－正規化された最小値	8400	-6.10947609E-05
＋正規化されていない最大値	03FF	6.09755516E-05
－正規化されていない最大値	83FF	-6.09755516E-05
＋正規化されていない最小値	0001	5.96046448E-08
－正規化されていない最小値	8001	-5.96046448E-08
＋無限大	7C00	Inf（Infinity）
－無限大	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

単精度実数型

単精度実数型（single precision real type）は、実数値を近似的に表現し、精度が2進で24けた（10進で約6けた）であり、絶対値がゼロまたは2^{-126}（約10^{-37}）から2^{127}（約10^{38}）までの値をとり、4バイトの記憶領域を占めます。種別値4が対応し、Fortran規格の基本実数型に対応します。

以下は単精度実数型における符号（s）、指数部（e）、仮数部（f）のビット構成を示しています。



以下に特殊データの内部表現とそれに対応する値を示します。

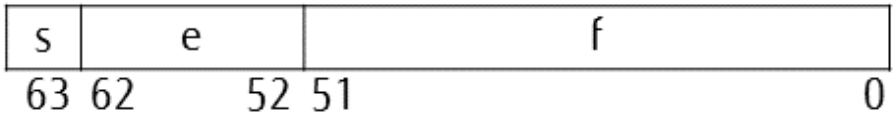
名称	16進定数値	10進定数値
0	00000000	0.0E+00
＋正規化された最大値	7F7FFFFF	3.40282347E+38
－正規化された最大値	FF7FFFFF	-3.40282347E+38
＋正規化された最小値	00800000	1.17549435E-38
－正規化された最小値	80800000	-1.17549435E-38
＋正規化されていない最大値	007FFFFF	1.17549421E-38
－正規化されていない最大値	807FFFFF	-1.17759421E-38
＋正規化されていない最小値	00000001	1.40129846E-45
－正規化されていない最小値	80000001	-1.40129846E-45
＋無限大	7F800000	Inf（Infinity）
－無限大	FF800000	-Inf
+quiet NaN	7FC00000～ 7FFFFFFF	NaN（Not a Number）

名称	16進定数値	10進定数値
-quiet NaN	FFC00000～ FFFFFFFF	-NaN
+signaling NaN	7F800001～ 7FBFFFFF	NaN
-signaling NaN	FF800001～ FFBFFFFF	-NaN

倍精度実数型

倍精度実数型（double precision real type）は、実数値を近似的に表現し、精度が2進で53けた（10進で約15けた）であり、絶対値がゼロまたは2^{-1022}（約10^{-307}）から2^{1023}（約10^{308}）までの値をとり、8バイトの記憶領域を占めます。種別値8が対応します。

以下は倍精度実数型における符号（s）、指数部（e）、仮数部（f）のビット構成を示しています。



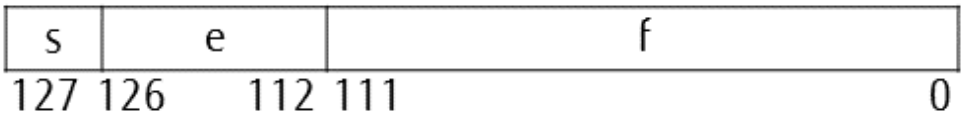
以下に特殊データの内部表現とそれに対応する値を示します。

名称	16進定数値	10進定数値
0	00000000 00000000	0.0E+00
＋正規化された最大値	7FEFFFFF FFFFFFFF	1.797693134862316D+308
－正規化された最大値	FFEFFFFF FFFFFFFF	-1.797693134862316D+308
＋正規化された最小値	00100000 00000000	2.225073858507201D-308
－正規化された最小値	80100000 00000000	-2.225073858507201D-308
＋正規化されていない最大値	000FFFFF FFFFFFFF	2.225073858507201D-308
－正規化されていない最大値	800FFFFF FFFFFFFF	-2.225073858507201D-308
＋正規化されていない最小値	00000000 00000001	4.940656458412465D-324
－正規化されていない最小値	80000000 00000001	-4.940656458412465D-324
＋無限大	7FF00000 00000000	Inf
－無限大	FFF00000 00000000	-Inf
+quiet NaN	7FF80000 00000000 ～ 7FFFFFFF FFFFFFFF	NaN
-quiet NaN	FFF80000 00000000 ～ FFFFFFFF FFFFFFFF	-NaN
+signaling NaN	7FF00000 00000001 ～ 7FF7FFFF FFFFFFFF	NaN
-signaling NaN	FFF00000 00000001 ～ FFF7FFFF FFFFFFFF	-NaN

4倍精度実数型

4倍精度実数型（quadruple precision real type）は、実数値を近似的に表現し、精度が2進で113けた（10進で約33けた）であり、絶対値がゼロまたは2^{-16382}（約10^{-4931}）から2^{16383}（約10^{4932}）までの値をとり、16バイトの記憶領域を占めます。種別値16が対応します。

以下は4倍精度実数型における符号（s）、指数部（e）、仮数部（f）のビット構成を示しています。



以下に特殊データの内部表現とそれに対応する値を示します。

名称	16進定数値	10進定数値
0	00000000 00000000 00000000 00000000	0.0E+00
＋正規化された 最大値	7FFEFFFF FFFFFFFF FFFFFFFF FFFFFFFF	1.189731495357231765085 7593266280070Q+4932
－正規化された 最大値	FFFEFFFF FFFFFFFF FFFFFFFF FFFFFFFF	-1.189731495357231765085 7593266280070Q+4932
＋正規化された 最小値	00010000 00000000 00000000 00000000	3.362103143112093506262 6778173217526Q-4932
－正規化された 最小値	80010000 00000000 00000000 00000000	-3.362103143112093506262 6778173217526Q-4932
＋正規化されていない 最大値	0000FFFF FFFFFFFF FFFFFFFF FFFFFFFF	3.362103143112093506262 6778173217519Q-4932
－正規化されていない 最大値	8000FFFF FFFFFFFF FFFFFFFF FFFFFFFF	-3.362103143112093506262 6778173217519Q-4932
＋正規化されていない 最小値	00000000 00000000 00000000 00000001	6.475175119438025110924 4389582276465Q-4966
－正規化されていない 最小値	80000000 00000000 00000000 00000001	-6.475175119438025110924 4389582276465Q-4966
＋無限大	7FFF0000 00000000 00000000 00000000	Inf
－無限大	FFFF0000 00000000 00000000 00000000	-Inf
+quiet NaN	7FFF8000 00000000 00000000 00000000 ～ 7FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	NaN
-quiet NaN	FFFF8000 00000000 00000000 00000000 ～ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	-NaN
+signaling NaN	7FFF0000 00000000 00000000 00000001 ～ 7FFF7FFF FFFFFFFF FFFFFFFF FFFFFFFF	NaN

名称	16進定数値	10進定数値
-signaling NaN	FFFF0000 00000000 00000000 00000001 ～ FFFF7FFF FFFFFFFF FFFFFFFF FFFFFFFF	-NaN

複素数型

半精度複素数型（half precision complex type）は、複素数値を近似的に表現し、一対の半精度実数型のデータで表現します。半精度複素数型のデータは、4バイトの記憶領域を占有し、最初の2バイトが実部、あとの2バイトが虚部を表します。その実部と虚部の精度および絶対値は、半精度実数型と同じです。

単精度複素数型（single precision complex type）は、複素数値を近似的に表現し、一対の単精度実数型のデータで表現します。単精度複素数型のデータは、8バイトの記憶領域を占有し、最初の4バイトが実部、あとの4バイトが虚部を表します。その実部と虚部の精度および絶対値は、単精度実数型と同じです。Fortran規格の基本複素数型に対応します。

倍精度複素数型（double precision complex type）は、複素数値を近似的に表現し、一対の倍精度実数型のデータで表現します。倍精度複素数型のデータは、16バイトの記憶領域を占有し、最初の8バイトが実部、あとの8バイトが虚部を表します。その実部と虚部の精度および絶対値は、倍精度実数型と同じです。

4倍精度複素数型（quadruple precision complex type）は、複素数値を近似的に表現し、一対の4倍精度実数型のデータで表現します。4倍精度複素数型のデータは、32バイトの記憶領域を占有し、最初の16バイトが実部、あとの16バイトが虚部を表します。その実部と虚部の精度および絶対値は、4倍精度実数型と同じです。

5.1.4 文字型データ

文字型（character type）のデータは、0から65000個までの任意の文字の列であり、1バイトの記憶域に一つの文字が入ります。したがって、文字型のデータの占有する記憶領域のバイト数は、列の中の文字数と等しくなります。

文字型のデータの中の各文字は、連続的に1、2、3、...と番号付けられた文字位置をもちます。番号は、文字の列の左からの文字の位置を示します。種別値1が対応します。

文字型データの内部表現は、ASCIIコード系で表され、1個の文字は、パリティビットなしの8ビットの2進数で表現されます。

5.1.5 派生型データ

派生型は、組込み型から構成される型であり、末端の成分はすべて組込み型です。派生型のデータは、組込み型のデータの集まりです。派生型のスカラデータであるこのデータの集まりを構造体（structure）といいます。

派生型が連続型でない場合、派生型のデータの記憶領域列の順序は、不定です。

5.2 データの正しい境界

コンパイラは、通常、原始プログラム内の各データに対して正しい境界を割り当てます。

下表に、データの型による正しい境界を示します。

表5.1 データの型に対する正しい境界

データの型	正しい境界
1バイトの整数型	任意の番地
2バイトの整数型	2の倍数の番地
4バイトの整数型	4の倍数の番地
8バイトの整数型	8の倍数の番地
1バイトの論理型	任意の番地
2バイトの論理型	2の倍数の番地
4バイトの論理型	4の倍数の番地

データの型	正しい境界
8バイトの論理型	8の倍数の番地
半精度実数型	2の倍数の番地
単精度実数型	4の倍数の番地
倍精度実数型	8の倍数の番地
4倍精度実数型	16の倍数の番地
半精度複素数型	2の倍数の番地
単精度複素数型	4の倍数の番地
倍精度複素数型	8の倍数の番地
4倍精度複素数型	16の倍数の番地
文字型	任意の番地
派生型	成分の正しい境界の最大値の番地

共通ブロックに属する変数、派生型の要素はコンパイラが自動的に境界を割り当てます。ただし、翻訳時オプション-AAが有効な場合は、この機能は抑止されます。

EQUIVALENCE文により記憶域を共有している変数の場合は、利用者が指定したとおりの境界となるので注意が必要です。また、派生型の定義において、派生型の最後の要素の次の境界は、その派生型の正しい境界でなければなりません。

例：翻訳時オプション -AAが有効な場合の派生型定義

```
TYPE TAG
  REAL(8)     R
  INTEGER(4)  I
END TYPE
```

この派生型（TAG）の正しい境界は、8の倍数の番地です。しかし、構成要素である変数Iの次の境界は、4の倍数の番地となるため、この派生型の定義は誤りです。

実数型配列のデータ境界について

コンパイラは、8バイト以下の型をもつ配列の先頭は8バイト境界、8バイトより大きい型をもつ配列の先頭は、型のサイズの境界に割り当てます。

ただし、引数で渡された配列、COMMONの途中の配列およびポインタ配列など、それに該当しない場合もあります。

5.3 精度の変換

Fortranで扱う算術データの精度には、ハードウェア上の限界があります。ここでは、データの精度を考慮する際に、本処理系で用意されている精度拡張機能および精度縮小機能を使用する場合の方法について説明します。

5.3.1 精度の拡張

作成したプログラムの演算精度を上げたい場合に精度拡張機能を使用します。

精度拡張機能は、指定された型の定数、変数および関数に対して、原始プログラム上で定義された型から精度を一段高い型に変換するものです。

5.3.1.1 精度拡張のオプション

精度拡張は、半精度を除く実数型および複素数型の定数、変数または関数に対して行うことができます。

精度拡張を指示する翻訳時オプション -Adを指定した場合、データの型は次のように変換されます。

単精度実数型	→	倍精度実数型
単精度複素数型	→	倍精度複素数型

精度拡張を指示する翻訳時オプション-Aqを指定した場合、データの型は次のように変換されます。

倍精度実数型	→	4倍精度実数型
倍精度複素数型	→	4倍精度複素数型

以下に、翻訳時オプション-Adを指定した場合の定数、変数または関数の精度拡張の例を示します。

例:翻訳時オプション -Adによる精度拡張

```
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)
```

上記のプログラムを翻訳時オプション-Adを指定して翻訳した場合、下記のプログラムと同等になります。

```
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)
```

関数が精度拡張された場合、組込み関数に対しては、より高い精度の型をもつ関数名に変えられます。ただし、EXTERNAL文での宣言などにより、利用者定義の関数とされた場合、関数名は変わりません。

利用者が定義した外部関数の引用の場合、その関数値が精度拡張の対象となります。したがって、関数引用が精度拡張された場合は、関数副プログラムも精度拡張機能を働かせて翻訳しなければなりません。

組込み関数の引用の場合、その関数名、引数および関数値の型が、精度拡張により影響を受けます。

組込み関数が個別名の場合、適当な関数名に変えられます。

組込み関数名が総称名の場合、その引数に対して精度拡張機能が働いて、適当な関数名が選択されます。

組込み関数名が実引数として引用された場合、精度拡張された結果の関数名に変えられます。

5.3.2 精度縮小と誤差の影響分析

精度縮小機能は、浮動小数点数で表されるデータの精度に限界があるために生じる計算誤差(丸め誤差)が、利用者が意図した計算の結果に、どの程度の影響を及ぼしているかを計測するための機能です。

これを応用して、使用しているアルゴリズムが、誤差に対して信頼できるかを調べることで、数値計算プログラムのデバッグを行うことができます。

例えば、連立一次方程式を解くプログラムまたは行列の固有値・固有ベクトルを求めるプログラムでは、行列が不適当なとき、行列に対するわずかな入力誤差が、非常に大きい誤差に結びつき、求めた結果が正しい結果とは似ても似つかぬことがあります。

このような誤差に敏感な部分を発見することにより、それに見合う解法の選択、入力誤差の影響を安定化する工夫などの数値計算プログラムの改良・デバッグを行うために精度縮小機能があります。

一般に、演算けた数と解の精度について、次のような関係があることが認められています。

$$d = (p - a) / m$$

d: 解の精度(けた)
p: 演算けた数
a: 損失けた
m: 問題と解法に固有な値

*m*は通常1であり、代数方程式の根を求めるときなどはその重根度に対応しています。この式は、問題の条件数をより適確に表現するものですが、次のような場合には成立しません。

- 入力データや定数などに誤差がある場合
- 問題解法の計算式に誤差がある場合
- 反復計算などを適当に打ち切る場合

すなわち、計算誤差(丸め誤差)が主要な誤差である範囲でのみ成立します。このような場合、演算けた数を1けた増減すれば、解の精度が1/*m*けた増減することがわかります。したがって、演算けた数を少し動かせば、解の精度もそれについて動くはずです。精度縮小機能は、この原理を利用して、解の精度を推測しようとするものです。精度縮小機能は、翻訳時オプション-C*l*の指定によって動作します。

5.3.2.1 精度縮小機能の定義

精度縮小機能は、次の翻訳時オプションの指定によって動作します。

```
-Cl    0≦l≦15
```

精度縮小機能は、単精度実数型、倍精度実数型、4倍精度実数型、単精度複素数型、倍精度複素数型および4倍精度複素数型のデータに対して適用されます。

この機能が動作すると、精度縮小機能の対象のデータが代入文で定義される場合に、そのデータの値のハードウェア上の表現による下位*l*ビットを0にします。

この下位*l*ビットが0になった値が実際のデータ値となります。

*l*は精度縮小数と呼ばれ、0≦*l*≦15でなければなりません。

*l*が0の場合の動作は、翻訳時オプション-C*l*を指定しない場合と等価です。しかし、この*l*はPRNSETサービスサブルーチンによって、実行時に動的に変えることができます。したがって、動的に変更された*l*の値がゼロ以外であれば精度縮小機能が働きます。

精度縮小数*l*の値は、実行可能プログラムに対して一つ定まるものです。

このため、パラレルリージョンからPRNSETサービスサブルーチンを実行した場合には、特定のスレッドにおける動的な変更が、他のスレッドの動作にも影響を与える可能性があります。

*l*の初期値は、主プログラムの翻訳時に指定されたものです。

副プログラムの翻訳時の-C*l*の指定は、精度縮小機能を動作させるためだけであり、そのときの精度縮小数*l*の値は意味をもちません。

5.3.2.2 精度縮小機能の使用方法

浮動小数点数の計算は、精度縮小数を変え、同じ計算を2回以上実行し、その結果を比較してください。

1回目:　　-C0で翻訳して実行
2回目:　　-C1で翻訳して実行

2回の計算結果の一致する部分が必ずしも正しいとは限りませんが、少なくとも、一致しない部分は正しくありません。確実さを増すためには、実行回数を増やしてください。すなわち、-C*l*の*l*を2から10程度に変えて実行回数を増やすことにより確実に解の精度を判定できます。ただし、一般的には、2回実行すれば十分です。

何回も計算機で実行するのは不便なので、1回の実行によって精度を調べる方が能率的です。

このように1回の実行で調べるには、精度判定の目的をもつ原始プログラムを作る必要があります。

このプログラムは、翻訳時オプション-C*l*を付けて翻訳し、プログラムの中でPRNSETサービスサブルーチンを使用するものです。

このプログラムは、以下の手順に従って作成してください。

1. 調査プログラム部分に必要な変更を受けるデータを退避します。
2. CALL PRNSET(*l*)を実行します。
3. 退避したデータを使って計算し、*l*に対応する所に結果を保存します。
4. *l*を変更し、所定の回数に達していなければ2)に分岐します。
5. *l*に対するそれぞれの結果を比較して、一致する情報を印刷します。

以下に精度縮小機能を利用したプログラムの例を示します。

例:精度縮小機能を利用したプログラム

```
      REAL,DIMENSION(10,11) :: A,B
      REAL,DIMENSION(10,10) :: X
1     READ (*,FMT='(I2,I2)',END=99) N,M          ! データの入力
      PRINT '(I2,I2)', N,M                       ! データの印刷
      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)
      ENDDO
      DO L = 1,M
        CALL PRNSET(L-1)
        CALL SWEEP(A,B,N)                        ! 精度縮小を変えて計算
        X(:N,L) = B(:N,N+1)
      ENDDO
      WRITE(UNIT=*,FMT='(//,A,5X,3(A,10X),A)') &
      &   ' -CI  ','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)                 ! 計算結果の印刷
      ENDDO
      WRITE(6,'(/,A,7X,3(A,14X),A)') ' -CI:-C0','X1','X2','X3','X4'
      DO L = 2,M
          X(:N,L) = X(:N,L)-X(:N,1)
      ENDDO
      DO L = 2,M                                 ! -CIの結果と-C0の結果
          WRITE(6,'(A,I1,A,1P8E16.7)') ' -C',L-1,':-C0',X(:N,L)
      ENDDO
      GOTO 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                               ! 連立一次方程式を掃出し法で解く
          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)
          ENDDO
      ENDDO
      END SUBROUTINE SWEEP
```

入力データの印刷1行目の内容は元数とテスト個数で2から5行目の内容はそれぞれ4個の係数と1個の定数項です。

```
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
```

計算結果

```
 -CI       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
```

-C0の結果との差

-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

備考1. この例題は、性質(条件)の悪い連立一次方程式を掃出し法で解くものです。

備考2. 縮小ビット数を0～8に指定した場合、解および-C0の場合の結果との差が印刷されます。

-C0との差がほぼ誤差を示しており、デバッグの目的を十分果していることがわかります。

第6章 プログラミング上の注意

この章では、本処理系を用いてプログラミングを行う場合の注意事項について説明します。

入出力処理については、“第7章 入出力処理”で説明します。

6.1 言語仕様レベル

Fortran言語仕様には、4つのレベルがあります。

1. Fortran 66言語仕様

   Fortran 66言語仕様と呼び、これに基づいて記述されたプログラムをFortran 66プログラムと呼びます。翻訳時オプション-X6を指定した場合、Fortran 66プログラムとして解釈されます。

2. Fortran 77言語仕様

   Fortran 77言語仕様と呼び、これに基づいて記述されたプログラムをFortran 77プログラムと呼びます。以下のどちらかの場合、Fortran 77プログラムとして解釈されます。

   ― 翻訳時オプション-X7を指定

   ― ファイルのサフィックスが“.f”、“.for”、“.F”、または“.FOR”であり、翻訳時オプション-Xd7を指定

   例: Fortran 77プログラムと解釈される場合

   ```
   $ frtpx a.f90 -X7
   ```

   ```
   $ frtpx b.f -Xd7
   ```

3. Fortran 95言語仕様(Fortran 90言語仕様を含む)

   Fortran 95言語仕様と呼び、これに基づいて記述されたプログラムをFortran 95プログラムと呼びます。以下のどちらかの場合、Fortran 95プログラムとして解釈されます。

   ― 翻訳時オプション-X9を指定

   ― ファイルのサフィックスが“.f90”、“.f95”、“.F90”、または“.F95”であり、翻訳オプション-Xを指定していない

   例: Fortran 95プログラムと解釈される場合

   ```
   $ frtpx b.f -X9
   ```

   ```
   $ frtpx a.f90
   ```

   ```
   $ frtpx a.f95
   ```

4. Fortran 2008言語仕様(Fortran 2003言語仕様を含む)

   本処理系は、Fortran 2003言語仕様とFortran 2008言語仕様で同じ動作をします。Fortran 2008言語仕様と呼び、これに基づいて記述されたプログラムをFortran 2008プログラムと呼びます。以下のいずれかの場合、Fortran 2008プログラムとして解釈されます。

   ― 翻訳時オプション-X03または-X08を指定

   ― ファイルのサフィックスが“.f03”、“.F03”、“.f08”、または“.F08”であり、翻訳オプション-Xを指定していない

   ― ファイルのサフィックスが“.f”、“.for”、“.F”、または“.FOR”であり、翻訳時オプション-Xを指定していない

   例: Fortran 2008プログラムと解釈される場合

   ```
   $ frtpx b.f
   ```

   ```
   $ frtpx a.f90 -X03
   ```

   ```
   $ frtpx a.f08
   ```

Fortranでは言語仕様によりプログラムの解釈が異なる項目があります。これらの解釈が異なる項目は、次のいずれかの言語仕様レベルに従って解釈が行われ、プログラムが実行されます。

- 主プログラム単位を翻訳した言語仕様レベル
- 仕様が現れるプログラム単位を翻訳した言語仕様レベル

本処理系は、Fortran 66、Fortran 77、Fortran 95、およびFortran 2008プログラムを翻訳するコンパイラですが、言語仕様の相違からプログラムに対する指定を誤ると正しい結果が得られない場合があります。

プログラムの解釈が異なる項目および解釈方法について以下に示します。

言語仕様によりプログラムの解釈が異なる項目	仕様が現れるプログラム単位を翻訳した言語仕様レベルで解釈される項目	主プログラム単位を翻訳した言語仕様レベルで解釈される項目
組込み関数名が指定されたEXTERNAL文 (“6.6.2 EXTERNAL属性が指定された組込み関数名”参照)	○	
DOループの繰返し数(“6.5.4.1 DOループの繰返し数”参照)	○	
X形編集記述子の効果(“7.16.5 X形編集記述子の効果”参照)		○
総称名としてのREALおよびCMPLX (“6.6.3 総称名としてのREALおよびCMPLX”参照)	○	
組込み手続名(“6.6.4 組込み手続名”参照)	○	
型宣言された組込み関数名(“6.6.5 型宣言された組込み関数名”参照)	○	
重なりのある文字代入(“6.4.1 重なりのある文字代入”参照)	○	
割付け代入(“6.4.2 割付け代入”参照)	○	
初期値をもつ変数に対するSAVE属性 (“6.2.5 初期値をもつ変数に対するSAVE属性”参照)	○	
RECL指定子があり、ACCESS指定子がないOPEN文 (“7.4.4 RECL指定子があり、ACCESS指定子がないOPEN文”参照)	○	○
INQUIRE文(“7.6.2 言語仕様レベルによるINQUIRE文の動作”参照)		○
整数型のG形編集(“7.16.1 整数型のG形編集”参照)		○
書式付き出力文の文字出力(“7.16.2 書式付き出力文の文字出力”参照)		○
G形編集記述子の編集結果(“7.16.3 G形編集記述子の編集結果”参照)		○
変数群入出力(“7.7.4.3 変数群入出力の言語仕様レベルによる違い”参照)		○
文字列編集記述子の診断メッセージ (“7.16.4 文字列編集記述子の診断メッセージ”参照)		○
並び出力における囲みなし文字定数の出力形式 (“7.7.5.2.1 並び出力における囲みなし文字定数の出力形式”参照)		○
再結合時のOPEN文の指定子の値 (“7.4.2 再結合時のOPEN文の指定子の値”参照)		○
書式付き入出力と書式なし入出力の混在 (“7.4.3 書式付き入出力と書式なし入出力の混在”参照)		○
直接探査入出力におけるデータ転送エラー		○

言語仕様によりプログラムの解釈が異なる項目	仕様が現れるプログラム単位を翻訳した言語仕様レベルで解釈される項目	主プログラム単位を翻訳した言語仕様レベルで解釈される項目
(“7.7.3.3 直接探査入出力におけるデータ転送エラー”参照)		
論理型の入力編集(“7.16.6 L形編集記述子”参照)		○
括弧で囲まれた変数(“6.3.7 括弧で囲まれた変数”参照)	○	
括弧で囲まれた単純出力並びの出力形式 (“6.5.9 括弧で囲まれた単純出力並び”参照)	○	
キーワード TYPEで始まる出力文(“6.5.10 キーワード TYPEで始まる出力文”参照)	○	
並びによる入出力(“7.7.5.3 並びによる入出力の言語仕様レベルによる違い” 参照)		○
引数に指定された実数型の値が負のゼロである場合の組込み関数ATAN2、ATAN2D、ATAN2Qの結果の値 (“6.6.7 組込み関数の結果値” 参照)		○
引数に指定された複素数型の虚部の値が負のゼロである場合の組込み関数LOG、SQRTの結果の値 (“6.6.7 組込み関数の結果値” 参照)		○

6.2 データの宣言および指定

ここでは、データの宣言および指定について説明します。

6.2.1 COMMON文

COMMON文を使用するには、以下の注意が必要です。

6.2.1.1 共通ブロックに属する変数の境界

共通ブロックに属する変数は、正しい境界(“5.2 データの正しい境界”参照)に並べなければなりません。

正しい境界に合っていない場合、コンパイラは、自動的に空き領域を挿入するので、変数は正しい境界に合わせられます。ただし、以下の例外があります。

- 翻訳時オプション -AAが有効な場合、コンパイラは空き領域を挿入しません。
- BIND属性が指定された共通ブロックの変数は、相互利用可能な境界に合わせられます。
- 倍精度実数型、4倍精度実数型、倍精度複素数型または4倍精度複素数型の変数が8の倍数の番地にない場合、4の倍数の番地に合わせられます。ただし、翻訳時オプション-Kalign_commonsが有効な場合、8の倍数の番地に合わせられます。

なお、共通ブロックの先頭は、必ず8の倍数の番地に合わせられています。

6.2.1.2 初期値をもつ共通ブロック

以下の条件のどちらかの場合、リンク時にエラーになります。

1. 異なるファイルにまたがって、同じ名前の共通ブロックまたは無名共通ブロックに2度以上初期値を設定している。

例1: リンク時にエラーになる場合

```
!  ファイル名  :  a.f
PROGRAM  MAIN
COMMON  /COM/A,B
DATA  A/1.0/
CALL  SUB
END
```

```
!  ファイル名  :  b.f
SUBROUTINE  SUB
COMMON  /COM/A,B
DATA  B/2.0/
PRINT  *,A,B
END SUBROUTINE
```

2. 異なるファイルにまたがって、大きさの異なる同じ名前の共通ブロックまたは無名共通ブロックを引用し、同じ共通ブロックの中で大きさが最大でない共通ブロックに初期値を設定している。

例2: リンク時にエラーになる場合2

```
!  ファイル名  :  a.f
PROGRAM  MAIN
COMMON  /COM/A,B
PRINT  *,A
END
```

```
!  ファイル名  :  b.f
BLOCK  DATA  BLK
COMMON  /COM/A
DATA  A/1.0/
END
```

6.2.2 EQUIVALENCE文

各結合実体は、それぞれ正しい境界に置かなければなりません。

6.2.3 初期値の設定

ここでは、初期値の設定に対する注意事項について説明します。

6.2.3.1 文字定数による初期値設定

文字定数は、文字型の変数に対する初期値として指定することができます。

文字型以外のデータに対して、初期値として文字定数を指定するとwレベルの診断メッセージが出力されます。

配列名に対する初期値の指定は、その要素数と同数の文字定数が必要です。

要素の長さよりも長い文字定数を指定しても、後続の配列要素の初期値とはなりません。

例1: 配列名に対する文字定数の初期値

```
CHARACTER(LEN=3)  A(2)
DATA  A/'ABCDEF'/
```

この場合の初期値は以下のようになります。

A(1)　'ABC'

A(2)　初期値なし(不定)

また、配列要素または文字部分列の初期値として、要素または部分列の長さよりも長い文字定数を指定しても、残りの部分が無視されるだけです。

例2: 配列要素に対する文字定数の初期値

```
CHARACTER(LEN=2)  B(4)
DATA  B(2)/'ABCDE'/
```

この場合の初期値は以下のようになります。

B(1)　初期値なし(不定)

B(2)　'AB'

B(3)　初期値なし(不定)

B(4)　初期値なし(不定)

B(3)、B(4)に、初期値として'CD'、'E'を与えたい場合、次のどちらかの方法があります。

```
CHARACTER(LEN=2)  B(4)
DATA  B(2), B(3), B(4)/'AB', 'CD', 'E'/
```

または

```
CHARACTER(LEN=2)  B(4)
CHARACTER(LEN=6)  BB
EQUIVALENCE  (B(2),BB)
DATA  BB/'ABCDE'/
```

6.2.3.2 非10進定数表現による初期値設定

配列名に対して初期値を設定する場合は、その配列の要素の数と同じ数だけ16進定数、8進定数または2進定数を記述しないと、不足分は不定となります。

すなわち、1つの配列要素に1つの16進定数、8進定数または2進定数が対応し、16進定数、8進定数または2進定数の長さがその配列要素の大きさを超えても、あふれは生じません。

例: 配列名に対する16進定数の初期値

```
INTEGER  A(3)
DATA  A/Z'F0F0A100F2F2A202'/
```

A(1)の初期値	F2F2A202
A(2)の初期値	初期値なし(不定)
A(3)の初期値	初期値なし(不定)

1つのスカラ変数の占めるバイト数より長い16進定数、8進定数または2進定数を指定した場合は、定数の左側の超過部分が無視されます。

1つのスカラ変数の占めるバイト数より短い16進定数、8進定数または2進定数を指定した場合は、データの不足分の領域(左側)にゼロ(0)が補われます。

6.2.3.3 重複する初期値設定

同一の変数に対して、初期値の設定を2回以上指定した場合、wレベルの診断メッセージが出力され、原始プログラム上で物理的に最後の指定が有効となります。

ただし、このようなプログラムは修正してください。

6.2.4 CRAY仕様のポインタ変数の引用

CRAY仕様のポインタ変数を引用する場合には、以下の注意が必要です。

a. 手続呼出しの仮引数のアドレスを副プログラム内で保存してはいけません。

例1: 誤ったポインタ変数の引用1

```
SUBROUTINE  SUB(A)
POINTER  (IP,IPP)
INTEGER  A(10),IPP(10)
SAVE  IP
  :
IP  =LOC(A(2))
  :
END SUBROUTINE
```

b. 関数は、実引数のアドレスを結果として返却してはいけません。

例2: 誤ったポインタ変数の引用2

```
FUNCTION  ADDR(I)
INTEGER  ADDR,I(5)
ADDR  =  LOC(I(2))
END
POINTER  (IP,PTR)
INTEGER  PTR  ,DATA(5),ADDR
IP  =  ADDR(DATA)
```

```
PTR  =  1
END
```

c. ポインタ変数で参照されるデータは、組込み関数LOCでアドレスを明示的にプログラム単位内で確定した場合を除き、他のデータと重なってはいけません。
代入文などによりアドレスを複写して引用してはいけません。

例3: 誤ったポインタ変数の引用3

```
POINTER  (IP1,ARRAY1)
POINTER  (IP2,ARRAY2)
INTEGER  DATA(5),ARRAY1(5),ARRAY2(5)
IP1  =  LOC(DATA)
ARRAY1  =  1
IP2  =  IP1
ARRAY2  =  2              !  代入文により確定したアドレスでは引用できません
END
```

6.2.5 初期値をもつ変数に対するSAVE属性

Fortran 95以降の言語仕様では、初期値をもつ変数はSAVE属性をもちます。

Fortran 66およびFortran 77言語仕様では、SAVE文またはSAVE属性をもつ型宣言文で指定しない限りSAVE属性はもちません。

例: 初期値をもつ変数に対するSAVE属性

```
SUBROUTINE  S
INTEGER  ::  I = 1
I = 1
END SUBROUTINE
```

Fortran 95言語仕様では上記プログラムは、以下のプログラムと同等です。

```
SUBROUTINE  S
INTEGER  ::  I = 1
SAVE  I
I = 1
END SUBROUTINE
```

6.2.6 定数式の実数型または複素数型演算

非実行文の定数式の実数型または複素数型演算では、実行文と同じ演算であっても、丸め誤差程度の違いが生じることがあります。

6.3 式

ここでは、式について説明します。

6.3.1 整数型データの演算

整数型のデータを代入する場合、値の大きさのチェックは行われません。

整数型データの演算結果が、許される値の範囲を超えたとき、結果は不定になります。

このことは特に、整数型データの比較のときに影響します。

例:

```
IF(I-J) 1,2,3
```

この例で"I-J"の結果がオーバフローする可能性があるときには、論理IF文の関係演算子による比較に変えるべきです。オーバフローした場合、次にどの文番号をもつ文が実行されるか不定です。

PARAMETER文およびPARAMETER属性をもつ型宣言文の定数式において、オーバフローした場合、名前付き定数の値は不定になります。

6.3.2 実数型データの演算

各精度の実数型のデータは、できるだけ正確に扱われますが、機械語で完全に一致するように表現できるとは限りません。

例: 許容値を認めていない判定

```
      IF(A-0.1)      1,2,1
 3    IF(B.EQ.C)   GO   TO   4
```

この例で、Aが正確に計算できれば0.1となるはずでも、計算機上では必ずしも文番号2の文に分岐するとは限りません。

BとCとが等しくなるはずでも、文番号4の文に分岐するとは限りません。

実数型の関係演算では、等値比較を行うべきでありません。

翻訳時オプション-Ec(“2.2 翻訳時オプション”参照)を指定すれば、許容値を認めていない関係式に対してiレベルの診断メッセージが出力されます。

ここで、許容値を認めていない関係式とは、関係式e1 relop e2において、relop(関係演算子)が“==”、“/=”、“.EQ.”または“.NE.”であり、e1またはe2が半精度実数型、単精度実数型、倍精度実数型、4倍精度実数型、半精度複素数型、単精度複素数型、倍精度複素数型または4倍精度複素数型のものです。判定の例を以下に示します。

例: 望ましい判定

```
      IF(ABS(A-0.1)-0.00001)   2,1,1
 3    IF(ABS(B-C)<D)   GO   TO   4
```

PARAMETER文およびPARAMETER属性をもつ型宣言文の定数式において、演算結果が正規化されていない値(“5.1.3 実数型および複素数型データ”参照)となる定数式は記述できません。

6.3.3 論理型データの演算

論理型データに真(.TRUE.)または偽(.FALSE.)以外の値を設定し、その値を論理値として扱うプログラムを作成してはいけません。

例えば、論理型の変数に16進定数で初期値を設定し論理値として扱うプログラムまたはEQUIVALENCE文で論理型の変数と整数型の変数を同じ領域に割り付け、代入された整数値を論理値として扱うようなプログラムを作成してはいけません。

例: 論理値以外での論理演算

```
LOGICAL :: A = Z'00000002',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
```

この場合、論理IF文でGO TO文が実行されることを期待してはいけません。

論理式の結果は、真または偽であり、ビットごとの論理演算ではありません。

例:

```
LOGICAL(1) :: L1,L2
DATA L1/Z'FF'/
L2 = .NOT.L1
```

この例で、L2の結果は偽とはなりません。

6.3.4 要素の評価順序

要素が演算子によって結合される場合、どちらの要素を先に評価するかは、規定されません。例えば、Fを関数とするとき、式F(X)+F(Y)において、F(X)とF(Y)のどちらが先に評価されるかを規定したプログラムを作成してはなりません。

6.3.5 式の一部の評価

式の値を得る場合、式の一部が評価されないときがあります。

例えば、論理式L1.AND.L2において、L1が偽であれば、L2を評価しなくても式の値は偽であることがわかります。

したがって、このような評価されない部分に関数の引用を書く場合は、十分注意しなければなりません。

6.3.6 関数引用による値の更新

文の中で、関数の引用がある場合、その文の中のほかの要素の値を変更してはなりません。

例えば、F(I).AND.I==1の式において、関数引用のF(I)でIの値を変更してはいけません。

ただし、論理IF文の論理式の中の関数の引用は、論理式が真のとき実行される実行文の中の要素を変更してもかまいません。

また、共通ブロック中の要素の値が、文中の関数の引用の値に影響がある場合、その文中のほかの関数の引用の実行は、共通ブロック中のその要素の値を変更してはいけません。

例えば、以下のプログラムにおけるFA( )+FB( )は、誤りです。

例:

```
FUNCTION FA( )
COMMON /C/A
FA = A
END
FUNCTION FB( )
COMMON /C/A
A = 1
FB = A
END
```

6.3.7 括弧で囲まれた変数

使用者手続の実引数に括弧で囲まれた変数が出現した場合、Fortran言語仕様により解釈が異なります。

Fortran 66およびFortran 77言語仕様では、記述された変数が実引数となります。

Fortran 95以降の言語仕様では、変数の値を一時的な領域へ複写し、その一時領域が実引数となります。

6.4 代入文

6.4.1 重なりのある文字代入

Fortran 66およびFortran 77言語仕様では、翻訳時オプション-Nf90moveを指定しない場合、文字代入文の右辺と左辺で同じデータ(記憶領域が同じ)を使用し、かつ、その定義の範囲と参照の範囲が重なっているプログラムの動作は不定です。翻訳時オプション-Nf90moveを指定した場合、Fortran 95以降の言語仕様と同じ結果になります。

Fortran 95以降の言語仕様では、文字代入文の右辺と左辺の重なりの有無にはプログラムの動作は影響しません。

例: Fortran 95の重なりのある文字代入

```
CHARACTER(LEN=5) C
C = '12345'
C(1:4) = C(2:5)
PRINT *,C! 23455が出力されます。
```

Fortran 66およびFortran 77言語仕様では、翻訳時オプション-Nf90moveを指定しない場合、重なりのある文字列代入文の動作は不定です。

6.4.2 割付け代入

翻訳時オプション-Nalloc_assignが有効で、代入文の左辺が割付け変数である場合、翻訳時に診断メッセージjwd2881i-iが出力され、実行時にFortran 2003規格以降の割付け代入の動作が行われます。

翻訳時オプション-Nnoalloc_assignが有効で、代入文の左辺が割付け変数である場合、翻訳時に診断メッセージjwd2754i-iが出力され、実行時にFortran 95規格の代入の動作が行われます。

Fortran 95言語仕様のソースプログラムに対して、翻訳時オプション-Nalloc_assignを指定した場合、実行性能が低下します。

翻訳時オプション-Nnoalloc_assignを指定するか、または、次のようにプログラムを修正することにより、左辺が割付け変数にはならず、実行性能が低下することはありません。

例: 左辺を割付け変数でなくするため部分配列添字を指定

修正前: 左辺は割付け変数

```
INTEGER,ALLOCATABLE::ARRAY(:)
ALLOCATE(ARRAY(2))
ARRAY=[1,2]
END
```

修正後: 左辺は割付け変数ではない

```
INTEGER,ALLOCATABLE::ARRAY(:)
ALLOCATE(ARRAY(2))
ARRAY(:)=[1,2]
END
```

6.5 実行文

ここでは、実行文について説明します。

6.5.1 GO TO文

計算形GO TO文および割当て形GO TO文の文番号並びに書かれた文番号の中に、1つでも未定義の文番号があると翻訳時にsレベルのエラーとなります。

したがって、あとで処理を追加する予定の位置へは、必ず文番号を定義するようにしなければなりません。

割当て形GO TO文の分岐先が、文番号並びと異なる文番号への分岐の場合、翻訳時にはエラーが出力されないので、間違った動作をすることがあります。

割当て形GO TO文の記述で、文番号の並びを指定した方が実行時の効率が良くなります。

このとき、この割当て形GO TO文の次に実行される可能性のある文の文番号は、すべて指定しなければなりません。

6.5.2 算術IF文

算術IF文の算術式の型が整数型で、その算術式の絶対値がその型の表現できる値を超えた場合には、次にどの文番号をもつ文が実行されるか不定です。

例えば、次のような算術IF文を実行した結果、次に10、20または30のいずれの文番号をもつ文が実行されるか不定です。

例:

```
I = 3
IF(2147483647+I) 10,20,30
```

6.5.3 IF文

IF文の論理式の評価が論理型の内部表現(“5.1.2 論理型データ”参照)に該当しない値をもつ場合の動作は不定です。論理式の評価については“6.3.3 論理型データの演算”を参照してください。

例:

```
LOGICAL TR
EQUIVALENCE(TR,JR)
DATA JR/16/
IF(TR) CALL S
```

CALL文が実行されることを期待してはいけません。

6.5.4 DO文

DOループの範囲の繰返し数が、初期値、限界値および増分値の値によっては、0になる場合があります。

DOループの範囲の実行中、DO変数や増分値または限界値を変更して繰返し数を変えることはできません。

DOループの範囲の外側からDOループの範囲の内側へ入る文がある場合、注意を促す診断メッセージが出力されます。

このようなプログラムの動作は、不定です。

備考. Fortran 66プログラムで定義しているDOの拡張範囲については、注意を促す診断メッセージが出力され正常に動作しますが、将来の互換性を考え修正してください。

6.5.4.1 DOループの繰返し数

Fortran 77以降の言語仕様では、DOループの繰返し数*r7*は、以下の式で定義されます。

```
r7 = MAX ( INT ( ( m2 - m1 + m3 ) / m3 ) , 0 )
```

ここで、*m1*、*m2*および*m3*は、それぞれDO文の初期値、限界値および増分値です。

Fortran 66言語仕様では、DOループの繰返し数*r6*は、以下の式で定義されます。

```
r6 = MAX ( ( m2 - m1 ) / m3 + 1 , 1 )
```

Fortran 77以降の言語仕様では、DO文で定義されるDOの範囲が必ずしも実行されるとは限りませんが、Fortran 66言語仕様では、DOの範囲が少なくとも1回は実行されることを意味しています。

6.5.4.2 DO CONCURRENT

DO制御がCONCURRENTをもつ形式のDO構文(DO CONCURRENT)は、最適化指示子NORECURRENCEを指定したDO構文と同等の解釈がされます。最適化指示子NORECURRENCEについては“9.10.4 最適化指示子”を参照してください。

6.5.5 CASE構文

場合式がスカラ整数式の場合、場合値は、スカラ整数式の型で表現できる範囲でなければなりません。

例: CASE構文の誤り

```
INTEGER(2) :: I
   :
SELECT CASE(I)
  CASE(40000)      !場合式が2バイトの整数型なので、場合値は
                   ! -32768から32767までの値でなければなりません。
   :
END SELECT
```

6.5.6 PAUSE文

PAUSE文は、標準エラー出力ファイルに対して診断メッセージ(jwe0001a)を出力します。

メッセージが端末に出力され、かつ、標準入力が端末の場合は、応答待ちになり、プログラムの実行を中断します。

応答待ちプログラムの実行を再開させるには、標準入力に対して任意のデータを入力してください。上記以外の場合は、PAUSE文を無視し、そのままプログラムの実行を継続します。

例: PAUSE文の診断メッセージ

```
PAUSE
   jwe0001a pause
```

```
PAUSE n
   jwe0001a pause n
```

n: 1から5けたの10進数字列

```
PAUSE c
   jwe0001a pause c
```

c: アポストロフィで囲まれた文字定数で、長さは255文字以下

6.5.7 STOP/ERROR STOP文

STOP/ERROR STOP文は、プログラムの実行を終了し、標準エラー出力ファイルに対して以下のように診断メッセージを出力します。

文の種類	終了符号の指定	出力される診断メッセージ
STOP	なし	-
	あり	jwe0002i
ERRO STOP	なし	jwe0003i
	あり	

-: メッセージ出力なし。

また、プログラムの実行時、環境変数FLIB_USE_STOPCODEに値1が設定されている場合、STOP/ERROR STOP文の終了符号が以下のように復帰コードに反映されます。

文の種類	終了符号	復帰コード
STOP	(指定なし)	0
	スカラ整数型の定数式	終了符号で指定した値の下位1バイト(0から255)
	スカラ基本文字型の定数式	0
ERROR STOP	(指定なし)	1
	スカラ整数型の定数式	終了符号で指定した値の下位1バイト(0から255)
	スカラ基本文字型の定数式	1

例: STOP/ERROR STOP文の診断メッセージ

- STOP *スカラ整数型の定数式*

```
STOP 123456
   jwe0002i stop 123456
```

```
STOP -3 + 4
   jwe0002i stop 1
```

- STOP *スカラ基本文字型の定数式*

```
STOP 'NORMAL'
   jwe0002i stop NORMAL
```

```
STOP ('AB'//'CD')//'EF'
   jwe0002i stop ABCDEF
```

ー ERROR STOP *スカラ整数型の定数式*

```
ERROR STOP 123456
   jwe0003i error stop 123456
```

```
ERROR STOP -3 + 2
   jwe0003i error stop -1
```

ー ERROR STOP *スカラ基本文字型の定数式*

```
ERROR STOP 'ERROR'
   jwe0003i stop ERROR
```

```
ERROR STOP ('AB'//'CD')//'EF'
   jwe0003i stop ABCDEF
```

例: STOP/ERROR STOP文の復帰コード

文の種類	終了符号	例文	診断メッセージ	復帰コード
STOP	(指定なし)	STOP	-	0
	スカラ整数型の定数式	STOP -3 + 4	jwe0002i stop 1	1
		STOP 400	jwe0002i stop 400	144
	スカラ基本文字型の定数式	STOP 'ABC'	jwe0002i stop ABC	0
		STOP ('AB'//'CD')//'EF'	jwe0002i stop ABCDEF	0
ERROR STOP	(指定なし)	ERROR STOP	jwe0003i error stop	1
	スカラ整数型の定数式	ERROR STOP 222	jwe0003i error stop 222	222
		ERROR STOP -3 + 2	jwe0003i error stop -1	255
	スカラ基本文字型の定数式	ERROR STOP 'ERROR'	jwe0003i error stop ERROR	1
		ERROR STOP ('AB'//'CD')//'EF'	jwe0003i error stop ABCDEF	1

-: メッセージ出力なし。

6.5.8 ALLOCATE／DEALLOCATE文

ALLOCATE文およびDEALLOCATE文にSTAT指定子を指定した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
912	ALLOCATE文を実行した時に、領域が不足しています。
1001	ALLOCATE文で指定した割付け変数が、すでに割り付けられています。
1003	DEALLOCATE文で指定した割付け変数が、割り付けられていません。
1004	DEALLOCATE文で指定したポインタが、未結合状態です。
1005	DEALLOCATE文で指定したポインタと結合している実体が、ALLOCATE文で生成されていません。

6.5.9 括弧で囲まれた単純出力並び

名前付き定数を含む括弧で囲んだ形で、単純出力並びに指定した場合、Fortran 66言語仕様では、括弧で囲まれた単純出力並びとして出力されます。Fortran 77以降の言語仕様では、複素定数として出力されます。

例:

```
REAL A,B
PARAMETER (A=1.0,B=2.0)
WRITE (*,*) (A,B)
END
```

上記のFortranプログラムを実行すると、Fortran 66言語仕様では、以下のように、括弧で囲まれた単純出力並びとして2つの実定数が出力されます。

```
$ ./a.out
 1.00000000 2.00000000
$
```

Fortran 77以降の言語仕様では、以下のように、複素定数として出力されます。

```
$ ./a.out
 (1.00000000,2.00000000)
$
```

6.5.10 キーワード TYPEで始まる出力文

キーワード TYPEで始まる出力文は、Fortran 66およびFortran 77言語仕様で翻訳および実行することができます。

Fortran 95以降の言語仕様では、派生型定義のTYPE文またはTYPE型宣言文と解釈され、sレベルの診断メッセージが出力されます。キーワードTYPEをPRINTに修正してください。

例: キーワードTYPEで始まる出力文

```
A=1.0
TYPE *, A! Fortran 95以降の言語仕様ではsレベルの診断メッセージが
END !出力されます。
```

6.6 手続

ここでは、手続について説明します。

6.6.1 文関数の引用

文関数の引用において、仮引数の型と実引数の型が異なる場合、診断メッセージが出力され、その評価は、実引数の型によって行われます。

ただし、その実引数に算術要素または論理要素を使用する場合、その実引数の型は、式の変換規則で許されるものでなければなりません。

次の例では、文関数SFの引用で、実引数の型は倍精度実数型となります。

しかし、文関数ESFの引用は、エラーです。

例: 文関数の引用

```
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)              ! SFの実引数の型は倍精度実数型 :
L = X == Y .AND. ESF(X,Y,Z)       !式の変換規則で許されない引数です。
```

6.6.2 EXTERNAL属性が指定された組込み関数名

Fortran 77以降の言語仕様では、EXTERNAL属性が指定された組込み関数名は、その名前は組込み関数としての性格を失い、利用者が定義する外部手続または初期値設定プログラム単位として扱われます。

Fortran 66言語仕様では、基本外部関数を表す名前がEXTERNAL文に指定された場合、その名前は基本外部関数として扱われます。

備考. Fortran 77以降の言語仕様での組込み関数とは、Fortran 66言語仕様での基本外部関数および組込み関数の両者を総称したものです。

6.6.3 総称名としてのREALおよびCMPLX

Fortran 77以降の言語仕様では、REALおよびCMPLXの総称関数は、それぞれ実数型および複素数型への型変換の機能をもっています。

Fortran 95以降の言語仕様では、REAL総称関数の第1引数が複素数型で第2引数を省略した場合、結果の種別型パラメタは、第1引数の種別型パラメタになります。

Fortran 66言語仕様では、REALおよびCMPLXの総称関数は、それぞれ複素数型の実数部の取出しおよび複素数の形成の機能をもっています。

6.6.4 組込み手続名

次の名前は、Fortran 77以降の言語仕様では、組込み関数として解釈されますが、Fortran 66言語仕様では組込み関数(基本外部関数)としては解釈されず、ユーザ定義の関数として扱われます。

```
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
```

次の名前は、Fortran 95以降の言語仕様では、組込み手続として解釈されますが、Fortran 66およびFortran 77言語仕様では、ユーザ定義の手続として取り扱われます。

```
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、RRSPACING、SCALE、SCAN、
SELECTED_INT_KIND、SELECTED_REAL_KIND、SET_EXPONENT、SHAPE、SIZE、SPACING、SPREAD、SUM、TINY、SYSTEM_CLOCK、TRANSFER、
TRANSPOSE、TRIM、UBOUND、UNPACK、VERIFY
```

以下の名前は、翻訳時オプション-Nobsfunが指定されたとき、組込み関数として解釈されます。

```
AIMAX0、AJMAX0、I2MAX0、IMAX0、JMAX0、IMAX1、JMAX1、AIMIN0、AJMIN0、I2MIN0、 IMIN0、JMIN0、IMIN1、JMIN1、FLOATI、FLOATJ、
DFLOTI、DFLOTJ、IIABS、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、IIAND、JIAND、IIEOR、JIEOR、IIOR、JIOR、INOT、JNOT、
IISHFT、JISHFT、IISHFTC、JISHFTC、IZEXT、JZEXT、IZEXT2、JZEXT2、JZEXT4、VAL
```

6.6.5 型宣言された組込み関数名

Fortran 77以降の言語仕様では、組込み関数名が本来もっている型と異なる型で型宣言されても、組込み関数としての性格を失うことはありません。

Fortran 66言語仕様では、組込み関数名が本来もっている型と異なる型で型宣言されると、組込み関数としての性格を失い、関数引用として使用されるときはユーザ定義の関数とみなされます。

6.6.6 サービスルーチン

翻訳時オプション-AUを指定した場合、サービスルーチン名は英小文字でなければなりません。

各サービスルーチンの明示的引用仕様を定義したモジュールSERVICE_ROUTINESを引用することにより、引数の数、引数の型などの正当性の検査が実施されます。モジュールSERVICE_ROUTINESを引用する場合には、以下の注意が必要です。

- SERVICE_ROUTINESにはすべてのサービスサブルーチンおよびサービス関数の明示的引用仕様が含まれます。引用する場合は以下のようにONLY句を指定し、実際に引用するサービスルーチンだけを参照結合することをお勧めします。

```
USE SERVICE_ROUTINES ,ONLY:ACCESS
```

- 翻訳時オプション-AUを指定した場合、モジュール名SERVICE_ROUTINESおよび各サービスルーチン名は小文字でなければなりません。
- 翻訳時オプション-CcdII8、-CcI4I8、-CcdRR8、-CcR4R8、-CcdLL8、-CcL4L8、-Ccd4d8、-Cca4a8、-Adまたは-Aqを指定した場合、サービスルーチンの引数および結果の対応する型が変換されます。実行時に対応する実行時オプション-Lb、-Li、-Lrを指定する必要があります。

モジュールSERVICE_ROUTINESを引用しない場合には、以下の注意が必要です。

- サービスルーチン引用の正当性の検査は実施されません。
- 翻訳時オプション-mldefault=cdeclを指定した場合、以下のようにプログラムが正しく動作しないことがあります。
  - ー リンク時にエラーになります。

    例: GETTODサービスサブルーチン
  - ー システムコールおよびシステム関数と同一名のサービスルーチンを使用している場合、システムコールおよびシステム関数が呼び出されます。

    例: FSTATサービス関数

モジュールの引用については、“第10章 モジュール、サブモジュール、およびモジュール引用の注意事項”を参照してください。

翻訳時オプション-Nmallocfreeを指定した場合、malloc関数およびfreeサブルーチンは組込み手続として解釈されます。

6.6.7 組込み関数の結果値

負の符号を持つ0が引数に指定された場合に、Fortran 2003以降の言語仕様とFortran 95以前の言語仕様とで、組込み関数の結果値が異なることがあります。

以下に結果値が異なるケース、および、それぞれの言語仕様での結果値を示します。

- ATAN2(Y,X):X＜0かつYの値が0で符号が負のとき、結果値はFortran 95以前の言語仕様ではπ、Fortran 2003以降の言語仕様では-πとなる。
- ATAN2D(Y,X):X＜0かつYの値が0で符号が負のとき、結果値はFortran 95以前の言語仕様では180.0、Fortran 2003以降の言語仕様では-180.0となる。
- ATAN2Q(Y,X):X＜0かつYの値が0で符号が負のとき、結果値はFortran 95以前の言語仕様では2.0、Fortran 2003以降の言語仕様では-2.0となる。
- LOG(X):XがREAL(X)<0となる複素数であってXの虚部が0で符号が負のとき、虚部の結果値はFortran 95以前の言語仕様ではπ、Fortran 2003以降の言語仕様では-πとなる。
- SQRT(X):XがREAL(X)<0となる複素数であってXの虚部が0で符号が負のとき、虚部の結果値はFortran 95以前の言語仕様では正の値、Fortran 2003以降の言語仕様では負の値となる。

6.6.8 組込み関数の精度

本処理系の組込み関数は、ロジックエンハンスによる高速化、アーキテクチャ固有の最適化などの要因により、コンパイラのバージョンやアーキテクチャの違いで、精度が丸め誤差程度で異なることがあります。そのため、組込み関数の引用においては、精度誤差を考慮してください。

6.6.9 EXECUTE_COMMAND_LINE組込みサブルーチン

EXECUTE_COMMAND_LINE組込みサブルーチンで、CMDSTATを指定した場合、以下の値が返却されます。

1307は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
-2	WAITに.FALSE.が指定されましたが、コマンドの実行は同期して行われました。本処理系ではコマンドの非同期実行はサポートされません。
1307	コマンド実行のために利用されるシステムコールでエラーが発生しました。

第7章 入出力処理

この章では、原始プログラムに記述した入出力文がどのように実行されるか、利用者が目的に合った実行を行うのにどのような手続が必要か、および入出力文で扱うファイルについて説明します。入出力文で扱うファイルには、内部ファイルと外部ファイルがありますが、特にことわりのない限り、“ファイル ”と記述している場合は外部ファイルを意味します。

7.1 ファイルの基本事項

入出力文で扱うファイルには、内部ファイルと外部ファイルがあります。

内部ファイルは、内部記憶領域にあり、スカラ文字型変数、文字型配列要素、文字型配列、または文字部分列のうちのどれかです。内部ファイルは内部ファイル入出力文で入出力することができます。

外部ファイルは、Fortranプログラムの外部媒体上にあり、Fortran における探査法の順番探査、直接探査、または流れ探査に接続されるファイルです。Fortranが扱うことができるファイルを次に示します。

- 標準入力ファイル(stdin)
- 標準出力ファイル(stdout)
- 標準エラー出力ファイル(stderr)
- 通常ファイル

なお、標準ファイル以外を通常ファイルと呼ぶことにします。通常ファイルは、順番探査、直接探査、または流れ探査のいずれかで入出力できます。標準ファイルは、順番探査でだけ入出力できます。ただし、書式なし入出力で探査することはできません。

7.1.1 ファイルの存在

ファイルは、実行可能プログラムの実行開始時に存在しているものと、実行可能プログラムの実行中に新しく存在するものとがあります。

存在しているファイルまたは新たに存在するファイルは、入出力文に指定した装置番号と結びつけられることにより、入出力することができます。

この結びつきについては、“7.2 装置番号とファイルの接続”を参照してください。

7.1.2 ファイルの接続

入出力文を実行するためには、ファイルと装置番号が結びつけられていなければなりません。これをファイルの接続といいます。また、CLOSE文の実行でファイルと装置番号の結びつきが解除されます。これを、ファイルの接続の解除といいます。1つの装置番号は同時に2つ以上のファイルと接続できません。また、1つのファイルは同時に2つ以上の装置番号と接続できません。

7.1.2.1 接続

ファイルの接続には、次のものがあります。

- 動的な接続
- 事前の接続

動的な接続は、実行可能プログラムの実行中に、OPEN文、WRITE文、READ文、BACKSPACE文、ENDFILE文、またはPRINT文の実行によって行われます。ただし、直接探査入出力文または流れ探査入出力文を実行するファイルに対しては、OPEN文の実行によって行われます。

事前の接続は、実行可能プログラムの実行前に、Fortranプログラムの外部からの指定によって行われます。

事前の接続が行われているファイルに対して、OPEN文が実行された場合、事前に接続されているファイル名とOPEN文のFILE指定子で指定されたファイル名が同じか、またはOPEN文のFILE指定子が省略された場合には、事前の接続が有効となります。また、事前に接続されているファイル名とOPEN文のFILE指定子で指定されたファイル名が異なる場合は、事前の接続が解除され、OPEN文で指定されたファイルが動的に接続されます。

システムで定義されている標準入力ファイル(stdin)、標準出力ファイル(stdout)および標準エラー出力ファイル(stderr)については、事前の接続が行われているのと同じ状態です。FILE指定子の指定があるOPEN文が実行されると、システムで定義されている標準ファイルの接続が解除されるので注意してください。

以下は、装置番号とファイルの対応を示したものです。

装置番号	ファイル	意味
0	標準エラー出力ファイル	診断メッセージを出力するファイル。 Fortranシステムが装置番号として0を使用します。
5	標準入力ファイル	装置番号5の入力文が対応します。
6	標準出力ファイル	装置番号6の出力文が対応します。
1～4 7～2147483647	利用者指定の名前付きファイル または名前なしファイル	各装置番号の入出力文が対応します。

7.1.2.2 接続の解除

ファイルの接続の解除は、次の場合に行われます。

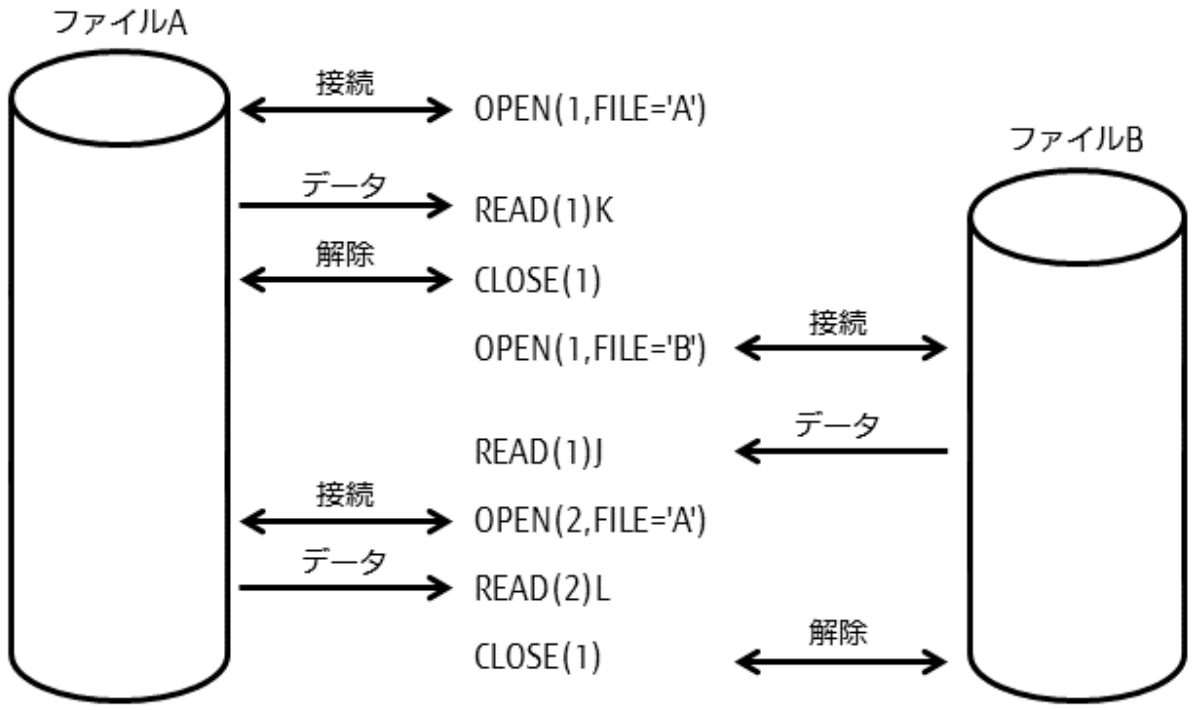
- CLOSE文の実行
- 実行可能プログラムの終了
- 同一装置番号に対しファイル名が異なるOPEN文の実行

接続の解除では、以下の処理を行います。

- ファイルと装置番号の接続解除
- ファイルの保存または削除の処理

このため、一度接続を解除された装置番号に対してファイルを接続するには、OPEN文の実行によって、動的な接続を行わなければなりません。接続を解除されたファイルは、同じ実行可能プログラム内で、同じ装置番号または別の装置番号と再接続できます。しかし、一度接続を解除されたファイルを再接続するには、ファイル名を指定したOPEN文を実行しなければなりません。

すなわち、一度接続を解除された名前なしファイルは、再び接続することはできません。標準入力ファイル、標準出力ファイル、標準エラー出力ファイルは名前なしファイルと同じ扱いです。以下に、ファイルの接続関係の変遷例を示します。



7.2 装置番号とファイルの接続

装置番号とファイルを接続するには以下の3つの方法があります。

- プログラムの中でファイル名指定のOPEN文を実行することにより接続します。
- リダイレクション機能を使うことにより標準入力、標準出力、標準エラー出力をプログラム外で接続します。
- 環境変数を使用して装置番号とファイルをプログラム外で接続します。

7.2.1 装置番号とファイルの接続の優先順位

装置番号とファイルを接続する場合、その接続方法により、優先順位があります。

標準入力、標準出力および標準エラー出力ファイルの場合の接続方法とその優先順位を、以下の表に示します。

指定方法	説明	優先順位
ファイル名指定のOPEN文	OPEN文のFILE指定子による指定	高い
実行時オプション	標準入出力、標準エラー出力と結合する装置番号の設定。 -p*u_no*(標準出力) -r*u_no*(標準入力) -m*u_no*(標準エラー出力)	↑ ↓
Fortranシステムの標準値	標準入出力、標準エラー出力と結合する装置番号の設定。 標準出力(装置番号6) 標準入力(装置番号5) 標準エラー出力(装置番号0)	低い

利用者指定の名前付きファイルおよび名前なしファイルの場合の接続方法とその優先順位を、以下の表に示します。

指定方法	説明	優先順位
ファイル名指定の OPEN文	OPEN文の FILE指定子による指定	高い
環境変数	環境変数 fuxx(xxは装置番号)による割当て	低い

7.2.2 環境変数による装置番号とファイルの接続

環境変数によって、装置番号とファイルの対応づけをとることができます。以下に環境変数によるファイル名の指定方法を示します。

環境変数名	設定値
fu*xx*	*ファイル名*

fu: 固定です。

xx: 装置番号(00～2147483647)を示します。

ファイル名: (フル)パス名であっても、単なるファイル名であってもかまいません。

実行時オプション-m、-r、-pによって表される装置番号(標準は、それぞれ0、5、6)は、それぞれ固定的に標準エラー出力、標準入力、標準出力に接続されるため、環境変数によってファイルと接続することはできません。これらのファイルは、リダイレクション機能を使用することにより端末以外に接続することができます。

環境変数でのファイルの接続はファイルの接続が解除されたときに無効になり、それ以降も無効になります。ファイルの接続が解除されるときの入出力文は以下のどちらかです(標準入力ファイル、標準出力ファイルおよび標準エラー出力ファイルに関しても同様です)。

- CLOSE文
- ファイル名指定のOPEN文

例1: 装置番号1とファイル名test.dataの接続

環境変数の設定:

```
$ export fu01=test.data
```

例2: 無効な環境変数による接続

Fortranプログラム

```
OPEN(1,FILE='b.file')
   ...
```

コマンド列

```
$ export fu01=a.file
$ ./a.out
```

a.outを実行すると環境変数による接続(a.fileへの接続)は無効となり、OPEN文で指定したファイル名(この場合は、b.file)が有効となります。

7.3 Fortran記録

Fortran記録は、値の列または文字の列であり、1つのFortran記録は必ずしも1つの物理記録に対応するものではありません。Fortran記録は、次の7つがあります。

- 書式付きFortran記録(“7.3.1 書式付きFortran記録”参照)
- 書式なしFortran記録(“7.3.2 書式なしFortran記録”参照)
- 並びによるFortran記録(“7.3.3 並びによるFortran記録”参照)
- 変数群Fortran記録(“7.3.4 変数群Fortran記録”参照)
- ファイル終了記録(“7.3.5 ファイル終了記録”参照)
- BINARY Fortran記録(“7.3.6 BINARY Fortran記録”参照)
- STREAM Fortran記録(“7.3.7 STREAM Fortran記録”参照)

Fortran記録の構成を、以下に示します。

Fortran記録の種類	入出力文の種類	記録形式	1つの Fortran記録
書式付きFortran記録	書式付き順番探査入出力文	不定長記録	1つの論理記録として存在します。
	書式付き直接探査入出力文	固定長記録	1つの論理記録として存在します。
	内部ファイル入出力文	固定長記録	1つの文字列スカラ変数として存在します。
書式なしFortran記録	書式なし順番探査入出力文	可変長記録	1つの論理記録として存在します。
	書式なし直接探査入出力文	固定長記録	1つの論理記録として存在します。
並びによるFortran記録	並びによる入出力文 PRINT文	不定長記録	1つの論理記録として存在します。
	内部ファイル入出力文	固定長記録	1つの文字列スカラ変数として存在します。
変数群Fortran記録	変数群入出力文	不定長記録	&変数名から/または&endまでの1つの論理記録として存在します。
	内部ファイル入出力文	固定長記録	1つの文字型スカラ変数として存在します。
BINARY Fortran記録	書式なし順番探査入出力文	固定長記録 (注1)	1つの論理記録として存在します。

Fortran記録の種類	入出力文の種類	記録形式	1つの Fortran記録
	書式なし直接探査入出力文	固定長記録 (注1)	1つの論理記録として存在します。
STREAM Fortran記録	書式付き流れ探査入出力文	不定長記録	1つの論理記録として存在します。
	書式なし流れ探査入出力文	固定長記録 (注1)	1つの論理記録として存在します。
ファイル終了記録	順番探査入出力文	なし	長さの属性をもたない最後のFortran記録として存在します。

注1) 入出力するデータの大きさによって、Fortran記録の長さが決まります。

備考. Fortran記録は外部媒体上では、不定長、固定長、または可変長の論理記録という形態で存在します。不定長記録では、Fortran記録と改行文字(¥n)を含めたもの、固定長記録では、Fortran記録そのもの、可変長記録では、Fortran記録に8バイト(Fortran記録の長さを格納した前後の4バイト)を加えたものです。

7.3.1 書式付きFortran記録

書式付きFortran記録は、任意の文字列で構成され、書式仕様によって定義されます。

この書式付きFortran記録は、書式付き順番探査入出力文、書式付き直接探査入出力文および内部ファイル入出力文で扱うことができます。

7.3.1.1 書式付き順番探査入出力文で扱うFortran記録

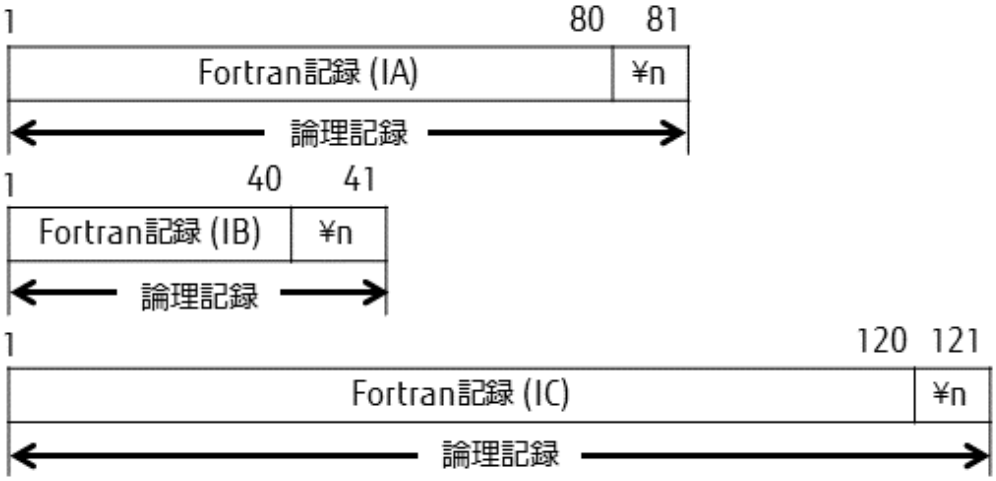
書式付き順番探査入出力文で扱うファイルの記録形式は不定長記録であり、1つのFortran記録が1つの論理記録に対応します。不定長記録の記録長は、扱うFortran記録の長さにより可変です。記録の最大の長さは、OPEN文のRECL指定子で指定することができます。論理記録の終端には改行文字(¥n)が入っています。書式付き順番探査WRITE文における書式仕様に$編集記述子または¥編集記述子を指定した場合、改行文字を含まないFortran記録を出力することができます。

以下に書式付きFortran記録と不定長記録形式との関係を示します。

[原始プログラム]

```
      INTEGER,DIMENSION(20) :: IA
      INTEGER,DIMENSION(10) :: IB
      INTEGER,DIMENSION(30) :: IC
      WRITE(2,10) IA,IB
10    FORMAT(2014,/,1014)
      WRITE(2,20) IC
20    FORMAT(3014)
```

[Fortran記録]



7.3.1.2 書式付き直接探査入出力文で扱うFortran記録

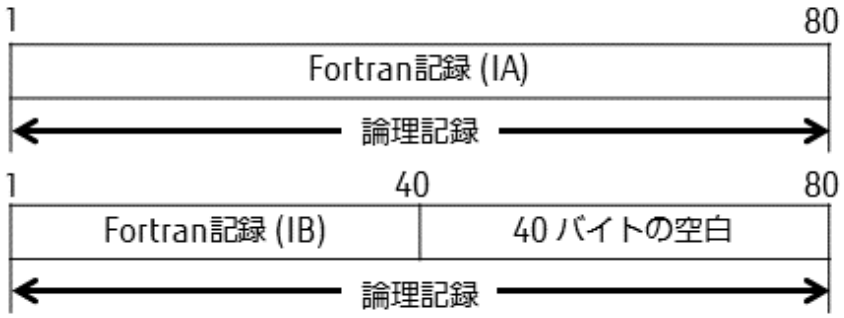
書式付き直接探査入出力文で扱うファイルの記録形式は固定長記録です。1つのFortran記録は1つの論理記録を形成します。論理記録の長さはOPEN文のRECL指定子で与えなければなりません。出力では、論理記録よりFortran記録が短いときは残りの部分に空白が詰められます。ただし、論理記録よりFortran記録が長くてはなりません。この固定長記録形式はFortran固有の形式です。

以下に書式付きFortran記録と固定長記録形式との関係を示します。

[原始プログラム]

```
    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(2014)
```

[Fortran記録]



7.3.1.3 内部ファイル入出力文で扱うFortran記録

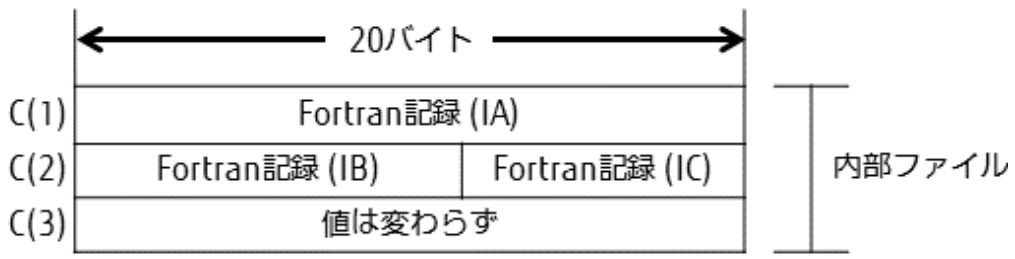
内部ファイル入出力文で扱うファイルの形式は固定長記録であり、内部ファイルに存在します。内部ファイルがスカラ文字型変数、文字型配列要素、または文字部分列のとき、それはただ1つのFortran記録を含みます。Fortran記録の長さはスカラ文字型変数、文字型配列要素、または文字部分列の長さを超えてはなりません。内部ファイルが文字型配列のとき、1つのFortran記録の長さは配列要素の長さを超えてはなりません。

出力で、Fortran記録の長さがスカラ文字型変数、文字型配列要素、または文字部分列の長さより短いときには、それぞれの残りの部分に空白が詰められます。入力では、Fortran記録の長さはそれぞれの長さに等しくなります。以下に書式付きFortran記録と内部ファイルの関係を示します。

[原始プログラム]

```
    CHARACTER(LEN=20),DIMENSION(3) :: C
    INTEGER,DIMENSION(5) :: IA
    INTEGER,DIMENSION(3) :: IB
    INTEGER,DIMENSION(2) :: IC
    WRITE(C,10) IA,IB,IC
10  FORMAT(5I4)
```

[Fortran記録(内部ファイル:C)]



7.3.2 書式なしFortran記録

書式なしFortran記録は、値(文字および文字以外のデータをともに含んでもよいし、データを含まなくてもよい)の列で構成され、長さは原始プログラムの入出力並びに依存(長さはゼロであってもよい)します。この書式なしFortran記録は、書式なし順番探査入出力文および書式なし直接探査入出力文で扱うことができます。

7.3.2.1 書式なし順番探査入出力文で扱うFortran記録

書式なし順番探査入出力文で扱うファイルの記録形式は可変長記録であり、1つのFortran記録が1つの論理記録に対応します。可変長記録形式の記録の長さは、扱うFortran記録の長さにより可変であり、論理記録の先頭および最後の各4バイト(実行時オプション-Luが有効な場合は、8バイト)には、1つのFortran記録の長さが格納されています。この最大の長さは、OPEN文のRECL指定子で指定することができます。

先頭の長さ域は、書式なし入力文が実行されるときに使用され、最後の長さ域は、BACKSPACE文が実行されるときに使用されます。この可変長記録は、Fortran固有の形式です。

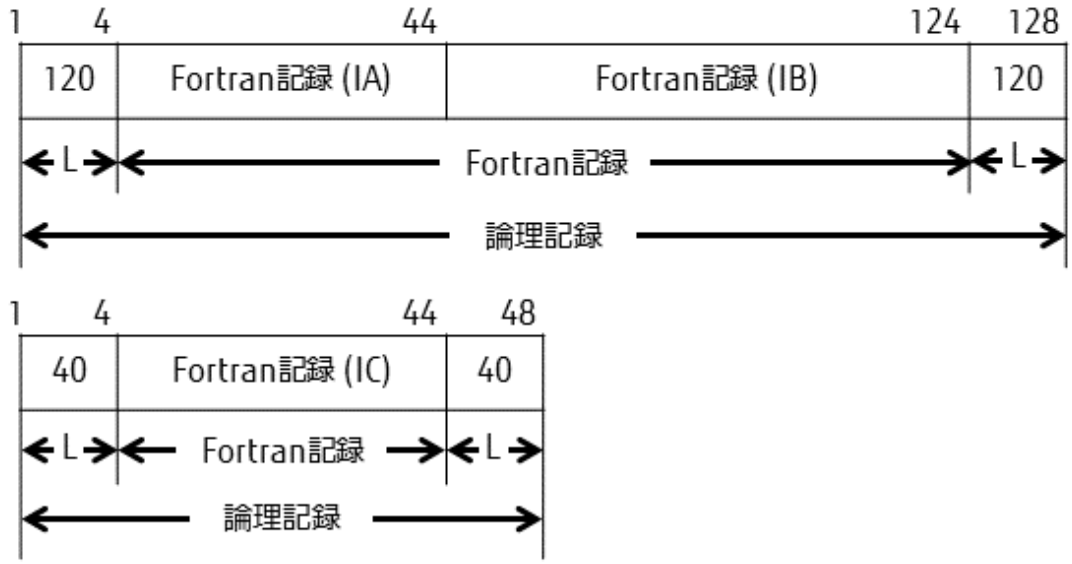
以下に書式なしFortran記録と可変長記録形式との関係を示します。

[原始プログラム]

```
INTEGER,DIMENSION(10) :: IA,IC
INTEGER,DIMENSION(20) :: IB
WRITE(2) IA,IB
WRITE(2) IC
```

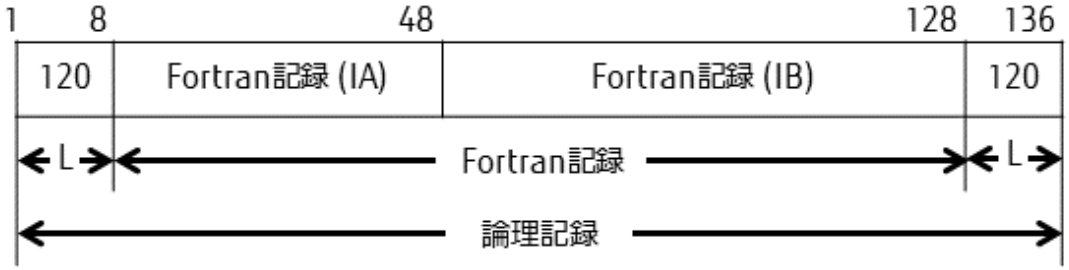
[Fortran記録]

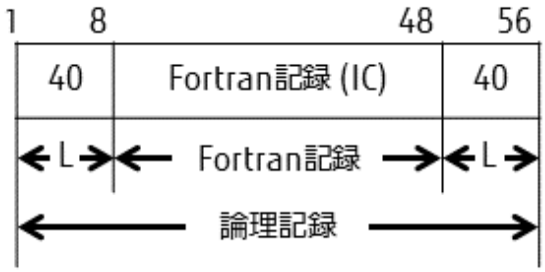
実行時オプション-Luを指定していない場合



L:それぞれのFortran記録の長さを示す長さ情報

実行時オプション-Luが指定されている場合





L:それぞれのFortran記録の長さを示す長さ情報

長さが2Gバイト(2147483648バイト)以上となるFortran記録については、Fortran記録を分割して入出力されます。分割されたそれぞれのFortran記録の長さは、1から2147483647バイトまでです。

分割されたFortran記録が存在する場合、これらを1つのFortran記録とみなすために、論理記録の先頭と末尾に設定される値が以下となります。

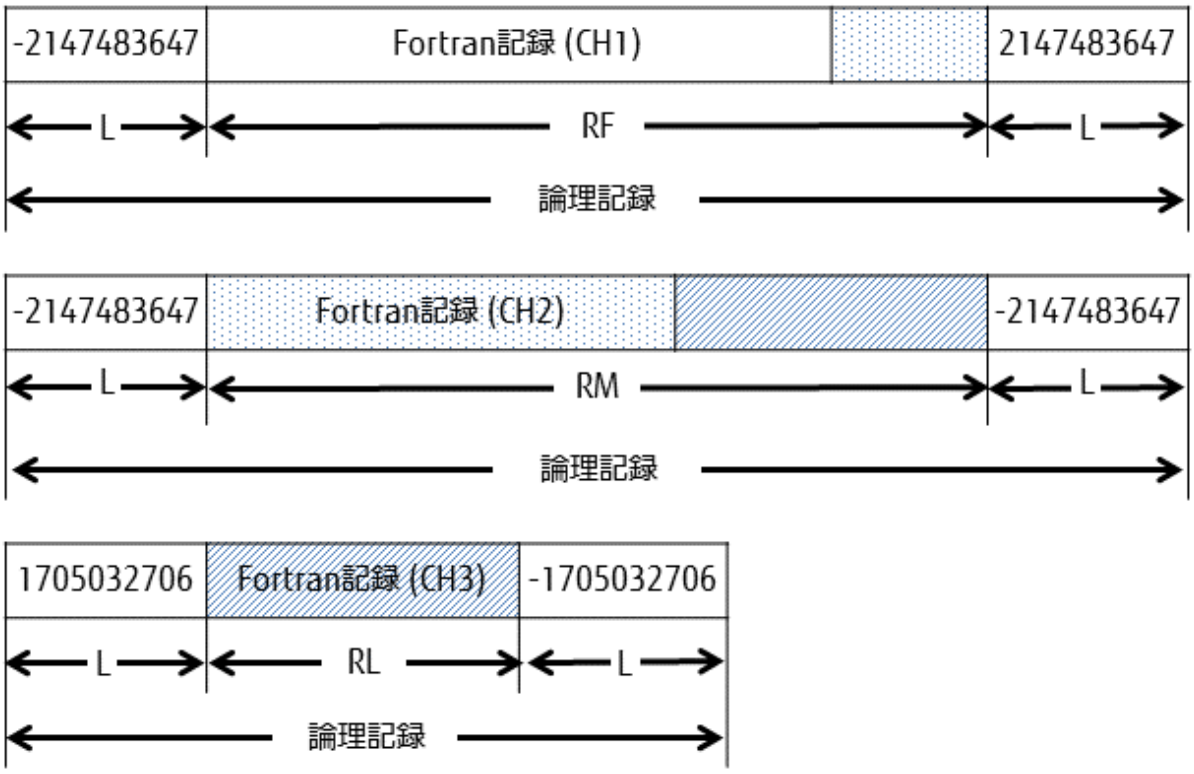
分割されたFortran記録	論理記録の先頭に設定される値	論理記録の末尾に設定される値
最初	分割されたFortran記録の長さと絶対値の等しい負の値が設定されます。 (-2147483647が設定されます)	分割されたFortran記録の長さが設定されます。 (2147483647が設定されます)
途中	分割されたFortran記録の長さと絶対値の等しい負の値が設定されます。 (-2147483647が設定されます)	分割されたFortran記録の長さと絶対値の等しい負の値が設定されます。 (-2147483647が設定されます)
最後	分割されたFortran記録の長さが設定されます。	分割されたFortran記録の長さと絶対値の等しい負の値が設定されます。

以下に、長さが2Gバイト以上となる書式なしFortran記録と可変長記録形式との関係を示します。

[原始プログラム]

```
CHARACTER(LEN=2000000000) :: CH1,CH2,CH3
OPEN(10,FILE="work.data",FORM="UNFORMATTED")
WRITE(10) CH1,CH2,CH3
```

[Fortran記録]



L: それぞれのFortran記録の長さを示す長さ情報
RF: 分割されたFortran記録(最初)
RM: 分割されたFortran記録(途中)
RL: 分割されたFortran記録(最後)

7.3.2.2 書式なし直接探査入出力文で扱うFortran記録

書式なし直接探査入出力文で扱うファイルの記録の形式は固定長記録であり、1つ以上の論理記録が対応します。固定長記録の形式については“7.3.1.2 書式付き直接探査入出力文で扱うFortran記録”を参照してください。

記録長はOPEN文のRECL指定子で与えなければなりません。1つのFortran記録は1つ以上の論理記録を形成します。出力では、論理記録の途中でFortran記録が終了したとき、残りの部分にバイナリゼロが詰められます。また、Fortran記録の長さが論理記録長より長いときは、出力並びの残りの部分がFortran記録として次の記録番号に出力されます。この固定長記録形式はFortran固有の形式です。

7.3.3 並びによるFortran記録

並びによるFortran記録は、並びによる入出力文、PRINT文および内部ファイル入出力文で扱うことができ、データ項目と値区切り子で構成されます。データ項目とは、入出力並び項目に割当てられる文字の列です。出力では、出力並び項目の個数および型によってFortran記録の長さが決定されます。入力では、論理記録の先頭から入力並び項目の処理を終了するまでのデータ項目を1つのFortran記録として扱います。

並びによる入出力文で扱うファイルの記録形式は不定長記録および固定長記録です。不定長記録形式については“7.3.1.1 書式付き順番探査入出力文で扱うFortran記録”と同じです。固定長記録の形式については“7.3.1.3 内部ファイル入出力文で扱うFortran記録”と同じです。

7.3.4 変数群Fortran記録

変数群Fortran記録は、変数群入出力文および内部ファイル入出力文で扱うことができ、&変数群名から/または&endまでのデータ項目(変数群名によって指定された要素に割当てられた文字の列)で構成されます。変数群Fortran記録と論理記録との対応は、並びによる入出力文で扱うFortran記録と同じです。

変数群入出力文で扱うファイルの記録形式は不定長記録です。不定長記録の形式については“7.3.1.1 書式付き順番探査入出力文で扱うFortran記録”と同じです。また、内部ファイル入出力文で扱うファイルの記録形式は固定長記録です。固定長記録の形式については“7.3.1.3 内部ファイル入出力文で扱うFortran記録”と同じです。

7.3.5 ファイル終了記録

ファイル終了記録は、1つのファイルの最後の記録としてだけ存在することができ、長さに関する属性はもちません。

ファイル終了記録はENDFILE文によって書かれます。ファイルは順番探査として接続されていなければなりません。ファイル終了記録は、ファイルが順番探査として接続されていて、かつ、最後の処理がENDFILE文以外で、WRITE文実行後の以下のどれかの条件のときにも暗黙に出力されます。

- REWIND文が実行されています。
- BACKSPACE文が実行されています。
- CLOSE文が実行されています。

7.3.6 BINARY Fortran記録

BINARY Fortran記録は、値(文字および文字以外のデータをともに含んでもよいし、データを含まなくてもよい)の列で構成され、長さは原始プログラムの入出力並びに依存(長さはゼロであってもよい)します。このFortran記録は、書式なし順番探査入出力文および書式なし直接探査入出力文で扱うことができます。

BINARY Fortran記録は固定長形式です。

7.3.7 STREAM Fortran記録

STREAM Fortran 記録は、記憶単位で構成され、正の整数によって記憶位置が一意に識別できます。このFortran 記録は、流れ探査入出力文で扱うことができます。

7.3.7.1 書式付き流れ探査入出力文で扱うFortran記録

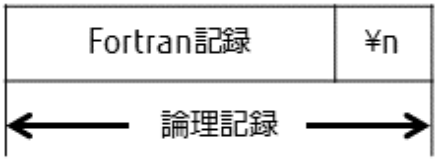
書式付き流れ探査入出力文で扱うファイルの記録形式は不定長記録であり、1つのFortran記録が1つの論理記録に対応します。不定長記録の記録長は、扱うFortran記録の長さにより可変です。論理記録の終端には改行文字(¥n)が入っています。書式付き流れ探査WRITE文における書式仕様に$編集記述子または¥編集記述子を指定した場合、改行文字を含まないFortran記録を出力することができます。

書式付き流れ探査入出力文は、端末装置に対しては実行できません。

[原始プログラム]

```
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記録]



7.3.7.2 書式なし流れ探査入出力文で扱うFortran記録

書式なし流れ探査入出力文で扱うファイルの記録形式は固定長記録であり、1つのFortran 記録が1つの論理記録に対応します。固定長記録形式の記録の長さは、扱うFortran 記録の長さにより可変です。

書式なし流れ探査入出力文は、端末装置に対しては実行できません。

以下に書式なしFortran 記録と固定長記録形式との関係を示します。

[原始プログラム]

```
OPEN(10, FILE='x', ACCESS='stream', FORM='unformatted')
DO I=1,10
   WRITE(10,POS=4*I ) I
   ENDDO
CLOSE(10)
```

[Fortran記録]

7.4 OPEN文

ここではOPEN文について説明します。

7.4.1 OPEN文の指定子

ここでは、OPEN文に指定できる以下の指定子について説明します。他の指定子については“7.7.1 入出力文の制御情報”を参照してください。

- FILE指定子(“7.4.1.1 FILE指定子”参照)
- STATUS指定子(“7.4.1.2 STATUS指定子”参照)
- RECL指定子(“7.4.1.3 RECL指定子”参照)
- ACTION指定子(“7.4.1.4 ACTION指定子”参照)
- BLOCKSIZE指定子(“7.4.1.5 BLOCKSIZE指定子”参照)
- CONVERT指定子(“7.4.1.6 CONVERT指定子”参照)
- NEWUNIT指定子(“7.4.1.7 NEWUNIT指定子”参照)

STATUS 指定子、ACCESS 指定子、FORM 指定子、BLANK 指定子、ACTION 指定子、PAD 指定子、POSITION 指定子、DELIM 指定子、ASYNCHRONOUS 指定子、DECIMAL 指定子、ROUND 指定子、およびSIGN 指定子に指定する文字式の値は、各指定子によって異なります。各指定子に指定された文字式は、文字式の先頭より、各指定子で指定可能な文字定数の長さだけ比較されます(ただし、先頭の空白は比較されません)。

比較した結果が等しくない、または文字式の長さが比較される文字定数の長さに満たない場合、診断メッセージ(jwe0086i-w)が出力され、その指定子は指定されないものとみなされます。文字式の長さが、比較される文字定数の長さより長い場合、残りは空白でなければなりません。

以下に、OPEN文のACCESS指定子で指定する文字式の扱いを示します。他の指定子も同様に扱われます。

例: 文字列"SEQUENTIAL"として扱われる場合

```
OPEN(10,ACCESS='SEQUENTIAL')
OPEN(10,ACCESS='sequential')
OPEN(10,ACCESS='SeQuEnTiAl')
OPEN(10,ACCESS='SEQUENTIAL     ')
OPEN(10,ACCESS='      SEQUENTIAL')
```

例: 指定子が未指定として扱われる場合

```
OPEN(10,ACCESS='SEQUENTIALAB')
OPEN(10,ACCESS='SEQUEN')
OPEN(10,ACCESS='S EQUENTIAL')
OPEN(10,ACCESS='              ')
```

7.4.1.1 FILE指定子

この指定子には、ファイル名を指定します。

FILE指定子に指定する文字式の値で、後続に空白があったときは、その空白が無視されます。

以下に、現在のディレクトリを'/home/v1954'とした場合の、OPEN文のFILE指定子に指定する文字式の扱いを示します。

例: /home/v1954/file.data1と解釈される場合

```
OPEN(10,FILE='/home/v1954/file.data1')
OPEN(10,FILE='file.data1')
OPEN(10,FILE=' file.data1')
```

例: /home/c1121/file.data1と解釈される場合

```
OPEN(10,FILE='../c1121/file.data1')
```

FILE指定子で指定したファイルの名前は、STATUS指定子の値によって決定されます。

FILE指定子	STATUS指定子		
	NEW、OLD、SHR、REPLACE	SCRATCH	UNKNOWN
FILE指定子あり	FILE指定子に指定された値の名前付きファイルとして扱います。	エラーです。	ファイルの状態により NEWまたはOLDと同様に扱われます。
FILE指定子なし 環境変数割当てなし	名前付きファイル (fort.nn)として扱われます。	名前なしファイルとして扱われます。	名前付きファイル (fort.nn)として扱われます。
FILE指定子なし 環境変数割当てあり	環境変数に指定されたファイル名の名前付きファイルとして扱われます。	エラーです。	環境変数に指定されたファイル名の名前付きファイルとして扱われます。

備考1. FILE指定子なしの場合、環境変数による装置番号とファイルの接続(“7.2.2 環境変数による装置番号とファイルの接続”参照)があるときと、ないときで扱いが異なります。

備考2. 名前なしファイルの名前は、tF*nnn*(*nnn*は装置番号)を頭文字とする任意の文字列です。詳細については、tempnam関数を参照してください。

7.4.1.2 STATUS指定子

STATUS指定子に指定する値は、ファイルの状態に対応しなければなりません。すなわち、すでに存在しているファイルに対して、NEWまたはSCRATCHを指定してはなりません。UNKNOWNが指定されているときは、本処理系によって、ファイルの状態が決定されます。この指定子を省略した場合、UNKNOWNを指定したものとみなされます。

以下に、STATUS指定子とファイルの関係を記述します。

STATUS指定子	ファイル
NEW	プログラムの実行を開始するまでに、存在していないファイルを指定します。
OLD SHR	プログラムの実行を開始するときに既存のファイルまたはプログラムの実行中に新たに生成されたファイルを指定します。
SCRATCH	プログラムの実行を開始するまでに、存在していないファイルであり、かつ、プログラムの終了またはファイルがクローズされるときに、ファイルは削除されます。

STATUS指定子	ファイル
UNKNOWN	ファイルの状態が分からず、本処理系にゆだねる場合に指定します。本処理系では以下の処理を行います。 ・ FILE指定子あり プログラムの実行を開始するまでに存在していないときはNEW、存在しているときはOLD ・ FILE指定子なし 環境変数による割当てがないときはSCRATCH、割当てがあるときはOLD
REPLACE	ファイルの状態にかかわらず、新たにファイルを生成する場合に指定します。本処理系では以下の処理を行います。 ・ ファイルあり FILE指定子または環境変数により指定されたファイルを削除した後、指定されたファイルを生成します。 ・ ファイルなし FILE指定子または環境変数により指定されたファイルを生成します。

7.4.1.3 RECL指定子

この指定子は、直接探査として接続されるファイルの場合は、ファイル上の各Fortran記録の長さを指定します。直接探査入出力文を実行する前に、この指定子が指定されているOPEN文を実行しなければなりません。存在するファイルに対して指定した値は、ファイルを構成するFortran記録の長さと同じでなければなりません。

流れ探査として接続されるファイルの場合は、この指定子を指定してはいけません。

この指定子の値は、正の値でなければなりません。

以下に、ACCESS指定子とRECL指定子の関係を記述します。

ACCESS指定子	RECL指定子あり		RECL指定子なし
	Fortran 95以降の言語仕様	Fortran 66 および Fortran 77 言語仕様	
SEQUENTIAL	ファイルは順番探査として接続されます。	実行時の診断メッセージ(jwe0097i-w)が出力され、ファイルは順番探査として接続されます。	ファイルは順番探査として接続されます。
DIRECT	ファイルは直接探査として接続されます。	ファイルは直接探査として接続されます。	エラー
STREAM	エラー	エラー	ファイルは流れ探査として接続されます。
なし	ファイルは順番探査として接続されます。	翻訳時の診断メッセージ(jwd1449i-i)が出力され、ファイルは直接探査として接続されます。	ファイルは順番探査として接続されます。

7.4.1.4 ACTION指定子

READと指定したときは、指定された装置番号と接続されたファイルをオープンするときのオープンモードに、'r'を指定します。すなわち、接続されたファイルに対してWRITE文およびPRINT文を実行することはできません。

WRITEと指定したときは、順番探査および流れ探査に対しては指定された装置番号と接続されたファイルをオープンするときのオープンモードに'w'を指定します。すなわち、接続されたファイルに対してREAD文、BACKSPACE文およびREWIND文を実行することはできません。直接探査に対してはWRITEを指定してもREADWRITEとみなされます。

READWRITEまたはBOTHと指定したときは、指定された装置番号と接続されたファイルをオープンするときのオープンモードに'r+'または'w+'を指定します。すなわち、接続されたファイルに対して、許されるすべての入出力文の実行ができます。ただし、read権だけが与え

られているファイルをオープンするときは、オープンモードに 'r'を指定します。したがって、READWRITEまたはBOTHを指定していても、WRITE文を実行したときに、実行時のエラーとなります。

この指定子を省略した場合、READWRITEを指定したものとみなされます。

以下にACTION指定子とオープンモードの関係を示します。

アクセス方法	ファイルの STATUS	ACTION指定子	OPENモード	備考
順番探査	NEW	READ	エラー	
		WRITE	w	
		READWRITE/ BOTH	w+	
	OLD/SHR	READ	r	
		WRITE	w	
		READWRITE/ BOTH	r+ (注1)	
	SCRATCH	READ	r	名前なしファイルに対する ACTION='READ'はエラーです。 存在しないファイルに対する、 ACTION='READ'はエラーです。
		WRITE	w	
		READWRITE/ BOTH	w+ (注1)	
	REPLACE	READ	r	
		WRITE	w	
		READWRITE/ BOTH	w+ (注2)	
直接探査	NEW	READ	エラー	
		WRITE	w	
		READWRITE/ BOTH	w+	
	OLD/SHR	READ	r	
		WRITE	r+	
		READWRITE/ BOTH	r+ (注2)	
	SCRATCH	READ	r	名前なしファイルに対する ACTION='READ'はエラーです。 存在しないファイルに対する ACTION='READ'はエラーです。
		WRITE	w+	
		READWRITE/ BOTH	r+ (注2)	
	REPLACE	READ	r	
		WRITE	w+	
		READWRITE/ BOTH	w+ (注2)	
流れ探査	NEW	READ	エラー	
		WRITE	w	

アクセス方法	ファイルの STATUS	ACTION指定子	OPENモード	備考
		READWRITE/ BOTH	w+	
	OLD/SHR	READ	r	
		WRITE	w	
		READWRITE/ BOTH	r+ (注1)	
	SCRATCH	READ	r	名前なしファイルに対する ACTION='READ'はエラーです。 存在しないファイルに対する ACTION='READ'はエラーです。
		WRITE	w	
		READWRITE/ BOTH	w+ (注1)	
	REPLACE	READ	r	
		WRITE	w	
		READWRITE/ BOTH	w+ (注2)	

注1) read 権だけが与えられているファイルをオープンしようとした場合、'r' モードでオープンされます。

注2) read 権だけが与えられているファイルをオープンしようとした場合、エラーとなります。

7.4.1.5 BLOCKSIZE指定子

この指定子は、順番探査入出力または流れ探査入出力のバッファサイズをバイト数で指定します。BLOCKSIZE指定子を省略した場合、バッファサイズは、8Mバイトです。

7.4.1.6 CONVERT指定子

この指定子に、BIG_ENDIANを指定した場合は、その装置で非同期入出力文を実行した際の非同期データ転送は行いません。

7.4.1.7 NEWUNIT指定子

NEWUNIT指定子に指定した変数に定義される値は、以下になります。

- NEWUNIT指定子に指定した変数の型が1バイト整数型の場合、-10から-128が割り当てられます。NEWUNIT指定子を指定したOPEN文の実行によって、同時に120以上のファイルに接続することはできません。
- NEWUNIT指定子に指定した変数の型が、2バイト整数型、4バイト整数型、および8バイト整数型の場合、-10から-16384が割り当てられます。NEWUNIT指定子を指定したOPEN文の実行によって、同時に16376以上のファイルに接続することはできません。

NEWUNIT指定子を指定した場合、FILE指定子を指定するか、STATUS指定子の値にSCRATCHを指定しなければなりません。

7.4.2 再結合時のOPEN文の指定子の値

CLOSE文の実行によって外部装置を解除しないで、その装置を同じ実行可能プログラム中で同じファイルに再び接続した場合、OPEN文で指定したBLANK指定子、DELIM指定子、PAD指定子、DECIMAL 指定子、ROUND 指定子、SIGN 指定子、ERR指定子、およびIOSTAT指定子以外の指定子の値が、直前に接続された外部装置の状態と異なるとき、OPEN文の実行時の動作が異なります。

Fortran 66およびFortran 77言語仕様では、直前に接続された外部装置の状態を有効にし、正常終了します。

Fortran 95以降の言語仕様では、診断メッセージ(jwe1115i-w)が出力され、直前に接続された外部装置の状態を有効にします。

また、STATUS 指定子の値がOLD以外であった場合、Fortran 2003以降の言語仕様では、診断メッセージ(jwe1115i-w)が出力され、直前に接続された外部装置の状態を有効にします。

例:

プログラム ap.f

```
OPEN(10,FILE='XX.DAT',POSITION='APPEND')
WRITE(10,*) 123
OPEN(10,FILE='XX.DAT',POSITION='REWIND')
WRITE(10,*) 456
END
```

実行結果

Fortran 66およびFortran 77言語仕様の場合

```
$ ./a.out
$ more XX.DAT
 123
 456
$
```

Fortran 95以降の言語仕様の場合

```
$ ./a.out
jwe1115i-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
  jwe1115i         w           1
total error count = 1
$ more XX.DAT
 123
 456
$
```

7.4.3 書式付き入出力と書式なし入出力の混在

書式付き入出力として接続されているファイルに書式なしデータ転送を行ったとき、または、書式なし入出力として接続されているファイルに書式付きデータ転送を行ったとき、入出力文の実行動作が異なります。

Fortran 66およびFortran 77言語仕様では、入出力文の書式に従い、データ転送を行い、正常終了します。

Fortran 95以降の言語仕様では、診断メッセージ(jwe0113i-e)を出力し、すでに接続されている書式に従いデータ転送を行います。

例:

プログラム fm.f

```
OPEN(10,FILE='XX.DAT',FORM='UNFORMATTED')
WRITE(10,*)123
END
```

実行結果

Fortran 66およびFortran 77言語仕様の場合

```
$ ./a.out
$ more XX.DAT
 123
$
```

Fortran 95以降の言語仕様の場合

```
$ ./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 RECL指定子があり、ACCESS指定子がないOPEN文

Fortran 66およびFortran 77言語仕様では、RECL指定子があり、ACCESS指定子がないOPEN文は、ACCESS指定子としてACCESS='DIRECT'が省略値となります。

Fortran 95以降の言語仕様では、RECL指定子があり、ACCESS指定子がないOPEN文は、ACCESS指定子としてACCESS='SEQUENTIAL'が省略値となります。

例: RECL指定子があり、ACCESS指定子がないOPEN文

```
OPEN(1,RECL=80)
```

Fortran 66およびFortran 77言語仕様では、ACCESS='DIRECT'が省略値になります。

7.5 CLOSE文の指定子

ここでは、CLOSE文に指定できるSTATUS指定子について説明します。他の指定子は、“7.7.1 入出力文の制御情報”を参照してください。

7.5.1 STATUS指定子

この指定子は、装置番号と接続されているファイルを存在させておくか削除するかの指定を行います。

ファイルを存在させておくか削除するかは、この指定子とOPEN文のSTATUS指定子により決定されます。

以下に、OPEN文のSTATUS指定子とCLOSE文のSTATUS指定子の関係について示します。

OPEN文	CLOSE文		
	KEEP/FSYNC	DELETE	指定なし
NEW OLD SHR REPLACE	CLOSE文の実行後も存在します。	CLOSE文の実行後は存在しません。	CLOSE文の実行後も存在します。
SCRATCH	CLOSE文の実行後は 存在しません。	CLOSE文の実行後は 存在しません。	CLOSE文の実行後は 存在しません。

備考. OPEN文でUNKNOWNを指定したときは、ファイルの状態として、NEW、OLD、SCRATCHのどれかが決定されます。

CLOSE文にFSYNCを指定すると、以下の条件をすべて満たす場合のみ、fsync(2)システムコールを実行して、メモリ上にあるファイルの内容を外部記憶装置上のものと同期します。

1. 接続しているファイルが名前付き通常ファイルである。
2. 上記1.のファイルが存在している。
3. 上記1.のファイルが使用中である。
4. ファイルシステムがfsync(2)システムコールの同期操作をサポートしている。

上記1.の条件を満たさない場合は、CLOSE文にFSYNCを指定するとエラーになります。

上記2.または3.の条件を満たさない場合は、fsync(2)システムコールは実行されません。処理は続行されます。

上記4.の条件を満たさない場合は、エラーになります。

7.6 INQUIRE文

ここでは INQUIRE文について説明します。

7.6.1 INQUIRE文の問合せ指定子

INQUIRE文の指定子に指定する文字型のスカラ変数名または配列要素名の長さが、設定する文字定数より小さい場合は、NAME指定子と他の指定子とで扱いが異なります。

以下に、INQUIRE文で設定される文字定数の扱いについて示します。

問い合わせ指定子の長さ≧文字定数の長さ		問合せ指定子の長さ＜文字定数の長さ	
NAME指定子	他の指定子	NAME指定子	他の指定子
文字定数の長さだけ先頭から設定します。残りには、空白を詰めます。		実行時の診断メッセージを出力し、空白を設定します。	実行時の診断メッセージを出力し、出力の長さだけ先頭から設定します。

各問合せ指定子に設定される値は、ファイル名によるINQUIRE文と装置番号によるINQUIRE文で異なります。“表7.1 ファイル名によるINQUIRE文で設定される値”および“表7.2 装置番号によるINQUIRE文で設定される値”は、それぞれの文で設定される値を示したものです。

表7.1 ファイル名によるINQUIRE文で設定される値

問合せ指定子	ファイルが存在している		ファイルが存在していない
	使用中 (注1)	未使用 (注2)	
EXIST	真	真	偽
OPENED	真	偽	偽
NUMBER	○	-1	-1
NAMED	真	真	真
NAME	○	○	○
FLEN	○	0	0
ACCESS	○	"UNDEFINED"	"UNDEFINED" (注8)
SEQUENTIAL	○	"UNKNOWN"	"UNKNOWN" (注8)
DIRECT	○	"UNKNOWN"	"UNKNOWN" (注8)
ACTION	○	"UNDEFINED"	"UNDEFINED" (注8)
READ	○	"UNKNOWN"	"UNKNOWN" (注8)
WRITE	○	"UNKNOWN"	"UNKNOWN" (注8)
READWRITE	○	"UNKNOWN"	"UNKNOWN" (注8)

問合せ指定子	ファイルが存在している		ファイルが存在していない
	使用中 (注1)	未使用 (注2)	
FORM	○	"UNDEFINED"	"UNDEFINED" (注8)
FORMATTED	○	"UNKNOWN"	"UNKNOWN" (注8)
UNFORMATTED	○	"UNKNOWN"	"UNKNOWN" (注8)
BINARY	○	"UNKNOWN"	"UNKNOWN" (注8)
RECL	○ (注3)	0	0
NEXTREC	○ (注4)	0	0
BLANK	○ (注5)	"UNDEFINED"	"UNDEFINED" (注8)
PAD	○ (注5)	"UNDEFINED" (注7)	"UNDEFINED" (注13)
POSITION	○ (注6)	"UNDEFINED"	"UNDEFINED" (注8)
DELIM	○ (注5)	"UNDEFINED"	"UNDEFINED" (注8)
BLOCKSIZE	○ (注9)	0	0
CONVERT	○ (注10)	"UNKNOWN"	"UNKNOWN" (注8)
ASYNCHRONOUS	○ (注11)	"UNDEFINED"	"UNDEFINED"
DECIMAL	○	"UNDEFINED"	"UNDEFINED"
ENCODING	○ (注5)	"UNKNOWN"	"UNKNOWN"
ROUND	○	"UNDEFINED"	"UNDEFINED"
SIGN	○	"UNDEFINED"	"UNDEFINED"
ID	○	0	0
PENDING	○	偽	偽
POS	○ (注12)	-1	-1
SIZE	○	○	-1
STREAM	○	"UNKNOWN"	"UNKNOWN"

真/偽: 4バイトの論理値です。

○: 確定した値が設定されます。

注1) 使用中とは、該当する装置番号に対して入出力文がすでに実行されていることです。

注2) 未使用とは、該当する装置番号に対して入出力文が実行されていないことです。

注3) 直接探査の場合、OPEN文で指定した値が設定されます。また、順番探査の場合、OPEN文のRECL指定子に指定した値、2147483647(RECL指定子を省略した場合)または0(Fortran 66およびFortran 77言語仕様の場合)が設定されます。

注4) 直接探査の場合だけ設定され、他の場合は0が設定されます。

注5) 書式付き入出力の場合だけ設定され、他の場合は"UNDEFINED"が設定されます。

注6) 順番探査の場合だけ設定され、他の場合は"UNDEFINED"が設定されます。

注7) Fortran95 言語仕様の場合、"YES" が設定されます。

注8) Fortran 66 およびFortran 77 言語仕様の場合、空白が設定されます。

注9) 順番探査および流れ探査の場合だけ設定され、他の場合は0が設定されます。

注10) 書式なし入出力の場合だけ設定され、他の場合は"NATIVE"が設定されます。

注11) 書式なし順番探査以外が指定された装置/ファイルである場合、または特殊ファイルが接続されている場合、常にNOが設定されます。

注12) 流れ探査の場合だけ設定され、他の場合は-1が設定されます。

注13) Fortran95 言語仕様の場合、"YES" が設定されます。Fortran 66 およびFortran 77 言語仕様の場合、空白が設定されます。

備考1. Fortran 66およびFortran 77言語仕様の場合、文字型の返却値については、小文字で返却されます。

備考2. 不定値の設定は、文字型の指定子については空白、整数型の指定子については0を設定します。

備考3. Fortranシステムとして確定値が設定できない場合は、-1、0、空白、"UNKNOWN"、または"UNDEFINED"を設定します。

表7.2 装置番号によるINQUIRE文で設定される値

問合せ指定子	装置番号が範囲内 (注1)			装置番号が範囲外 (注2)
	ファイルと結合されている (注3)		ファイルと結合されていない (注4)	
	使用中 (注5)	未使用 (注6)		
EXIST	真	真	真	偽
OPENED	真	真	偽	偽
NUMBER	○	-1	-1	-1
NAMED	真 (注7)	偽	偽	偽
NAME	○ (注8)	""	""	""
FLEN	○	0	0	0
ACCESS	○	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (注14)
SEQUENTIAL	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
DIRECT	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
ACTION	○	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (注14)
READ	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
WRITE	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
READWRITE	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
FORM	○	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (注14)

問合せ指定子	装置番号が範囲内 (注1)			装置番号が範囲外 (注2)
	ファイルと結合されている (注3)		ファイルと結合されていない (注4)	
	使用中 (注5)	未使用 (注6)		
FORMATTED	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
UNFORMATTED	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
BINARY	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
RECL	○ (注9)	0	0	0 (注14)
NEXTREC	○ (注10)	0	0	0 (注14)
BLANK	○ (注11)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (注14)
PAD	○ (注11)	"UNDEFINED" (注18)	"UNDEFINED" (注18)	"UNDEFINED" (注19)
POSITION	○ (注12)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (注14)
DELIM	○ (注11)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (注14)
BLOCKSIZE	○ (注13)	0	0	0
CONVERT	○ (注15)	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (注14)
ASYNCHRONOUS	○ (注16)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
DECIMAL	○	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
ENCODING	○ (注11)	"UNKNOWN"	"UNKNOWN"	"UNKNOWN"
ROUND	○	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
SIGN	○	"UNDEFINED"	"UNDEFINED"	"UNDEFINED"
ID	○	0	0	0
PENDING	○	偽	偽	偽
POS	○ (注17)	-1	-1	-1
SIZE	○	○	-1	-1
STREAM	○	"UNKNOWN"	"UNKNOWN"	"UNKNOWN"

真/偽: 4バイトの論理値です。

○: 確定した値が設定されます。

" ": 空白を示します。

注1) 装置番号が範囲内とは、装置番号が0から2147483647までの範囲内にあることです。

注2) 装置番号が範囲外とは、装置番号が0から2147483647までの範囲外にあることです。

注3) 装置番号とファイルが接続されている状態です。

注4) 装置番号とファイルが接続されていない状態です。

注5) 該当する装置番号に対して入出力文がすでに実行されていることです。

注6) 該当する装置番号に対して入出力文が実行されていないことです。

注7) 名前なしファイルの場合、偽が設定されます。

注8) 名前なしファイルの場合、空白が設定されます。

注9) 直接探査の場合、OPEN文で指定した値が設定されます。また、順番探査の場合、OPEN文のRECL指定子に指定した値、2147483647(RECL指定子を省略した場合)または0(Fortran 66およびFortran 77言語仕様の場合)が設定されます。

注10) 直接探査の場合だけ設定され、他の場合は 0が設定されます。

注11) 書式付き入出力の場合だけ設定され、他の場合は"UNDEFINED" が設定されます。

注12) 順番探査の場合だけ設定され、他の場合は"UNDEFINED"が設定されます。

注13) 順番探査および流れ探査の場合だけ設定され、他の場合は0が設定されます。

注14) Fortran 66 およびFortran 77 言語仕様の場合、空白が設定されます。

注15) 書式なし入出力の場合だけ設定され、他の場合は"NATIVE"が設定されます。

注16) 書式なし順番探査以外が指定された装置/ファイルである場合、または特殊ファイルが接続されている場合、常にNOが設定されます。

注17) 流れ探査の場合だけ設定され、他の場合は-1が設定されます。

注18) Fortran95 言語仕様の場合、"YES" が設定されます。

注19) Fortran 66 およびFortran 77 言語仕様の場合、空白が設定されます。Fortran95 言語仕様の場合、"YES" が設定されます。

備考1. Fortran 66およびFortran 77言語仕様の場合、文字型の返却値については、小文字で返却されます。

備考2. 不定値の設定は、文字型の指定子については空白、整数型の指定子については0を設定します。

備考3. Fortranシステムとして確定値が設定できない場合は、-1、0、空白、"UNKNOWN"、または"UNDEFINED"を設定します。

7.6.2 言語仕様レベルによるINQUIRE文の動作

INQUIRE文で返却される値についての詳細は、“7.6.1 INQUIRE文の問合せ指定子”を参照してください。

7.6.2.1 ACTION指定子の返却値

ACTION='READWRITE'またはACTION='BOTH'で扱われているファイルに対して、INQUIRE文のACTION指定子を問い合わせた場合、Fortran 66およびFortran 77言語仕様では、"BOTH"が返却されます。

Fortran 95以降の言語仕様では、"READWRITE"が返却されます。

7.6.2.2 INQUIRE文の問合せ指定子に返却される文字定数

NAME指定子を除く問合せ指定子に返却される文字定数は、Fortran 66およびFortran 77言語仕様では、小文字で返却されます。ただし、実行時オプション-qを指定した場合、大文字で返却されます。

Fortran 95以降の言語仕様では、実行時オプション-qの指定に関係なく大文字で返却されます。

例:

```
CHARACTER(LEN=10) CH
OPEN(10,ACCESS='SEQUENTIAL')
INQUIRE(10,ACCESS=CH)
```

上記のFortranプログラムを実行すると、Fortran 66およびFortran 77言語仕様では、CHに文字列"sequential"が返却されます。Fortran 95以降の言語仕様では、CHに文字列"SEQUENTIAL"が返却されます。

7.6.2.3 PAD指定子の返却値

Fortran 66およびFortran 77言語仕様では、ファイルが使用中でない場合、"undefined"または空白が返却されます。Fortran 95言語仕様では、"YES"が返却されます。

Fortran 2003以降の言語仕様では、"UNDEFINED"が返却されます。

7.6.2.4 ファイルINQUIRE文の問合せ指定子の返却値

Fortran 66およびFortran 77言語仕様では、ファイルが存在していない場合、ACCESS指定子、SEQUENTIAL指定子、DIRECT指定子、ACTION指定子、READ指定子、WRITE指定子、READWRITE指定子、FORM指定子、FORMATTED指定子、UNFORMATTED指定子、BLANK指定子、POSITION指定子、DELIM指定子、ASYNCHRONOUS 指定子、DECIMAL 指定子、ROUND 指定子、SIGN 指定子、およびSTREAM 指定子には、空白、"undefined"、または"unknown"が返却されます。

Fortran 95以降の言語仕様では、"UNDEFINED"または"UNKNOWN"が返却されます。

7.6.2.5 装置INQUIRE文の問合せ指定子の返却値

Fortran 66およびFortran 77言語仕様では、指定した装置番号が0から2147483647以外の場合、ACCESS指定子、SEQUENTIAL指定子、DIRECT指定子、ACTION指定子、READ指定子、WRITE指定子、READWRITE指定子、FORM指定子、FORMATTED指定子、UNFORMATTED指定子、BLANK指定子、POSITION指定子、DELIM指定子、ASYNCHRONOUS 指定子、DECIMAL 指定子、ROUND 指定子、SIGN 指定子、およびSTREAM 指定子には、空白、"undefined"、または"unknown"が返却されます。

Fortran 95以降の言語仕様では、"UNDEFINED"または"UNKNOWN"が返却されます。

7.7 入出力文の使用方法

ここでは、利用者が目的とする入出力を行うための、入出力文の使用方法について説明します。ファイル位置付け入出力文については、“7.9 ファイル位置付け文”を参照してください。

入出力文は、すべての入出力装置に対して適用できるわけではありません。以下に、入出力文と適用できる入出力装置の関係を示します。

<table>
<tr><th rowspan="2">入出力文</th><th colspan="2">装置</th></tr>
<tr><th>端末装置</th><th>直接探査記憶装置</th></tr>
<tr><td>書式付き順番探査入出力文</td><td colspan="2">可</td></tr>
<tr><td>書式なし順番探査入出力文</td><td rowspan="3">不可</td><td rowspan="3">可</td></tr>
<tr><td>直接探査入出力文</td></tr>
<tr><td>流れ探査入出力文</td></tr>
<tr><td>並びによる入出力文</td><td colspan="2" rowspan="2">可</td></tr>
<tr><td>変数群入出力文</td></tr>
</table>

7.7.1 入出力文の制御情報

ここでは、入出力文の構成要素である制御情報について説明します。

UNIT指定子、FMT指定子、IOSTAT指定子、ERR指定子、END指定子、EOR指定子、およびIOMSG指定子について説明します。

7.7.1.1 UNIT指定子

UNIT指定子に*を指定したとき、または装置番号を省略した入出力文における装置番号は、本処理系の標準値または実行時オプションの指示に従います。

以下の表に、このときに決定される装置番号について示します。

入出力文	実行時オプション	標準値
Fortranシステムで出力する診断メッセージ	-m*u_no*で指定した*u_no*の値	0
*指定の READ文	-r*u_no*で指定した*u_no*の値	5

入出力文	実行時オプション	標準値
装置番号を省略した READ文		
*指定の WRITE文 PRINT文	-p*u_no*で指定した*u_no*の値	6

備考. 実行時オプションで指定した値が標準値より優先されます。

7.7.1.2 FMT指定子

入出力データは、編集記述子によって、数値データ、論理型データ、文字型データ、2進定数表現されているデータ、8進定数表現されているデータ、または16進定数表現されているデータとして扱われ、編集されます。例えば、実数型の出力並びに対してI形編集記述子を指定すれば、整数型の数値データとして編集出力します。

以下の表に、編集記述子と入出力並びの型について示します。

入出力並びの型	編集記述子							
	L	I	F,E,EN,ES,D,Q	A	G	B	Z	O
1バイトの論理型 2バイトの論理型 4バイトの論理型 8バイトの論理型	○	△	△	○	○	○	○	○
1バイトの整数型 2バイトの整数型 4バイトの整数型 8バイトの整数型	△	○	△	○	○	○	○	○
実数型 倍精度実数型 4倍精度実数型	△	△	○	○	○	○	○	○
複素数型 倍精度複素数型 4倍精度複素数型	△	△	○	○	○	○	○	○
文字型	×	×	×	○	○	○	○	○

○: 許される組合せです。

△: 矛盾した組合せです。ただし、警告メッセージを出力し、編集記述子に合った編集が行われます。

×: 許されない組合せなので、メッセージを出力し、入出力処理を終了します。

7.7.1.3 IOSTAT指定子

IOSTAT指定子には、入出力文の実行によって、次のどれかの条件を識別する値が代入されます。

- 誤り条件が発生しています。
- ファイル終了条件が発生しています。
- 記録終了条件が発生しています。
- 誤り条件、ファイル終了条件および記録終了条件のいずれも発生していません。

“表7.3 誤り条件、ファイル終了条件および記録終了条件”に、誤り条件、ファイル終了条件および記録終了条件が発生した場合、IOSTAT指定子に代入される整数値について示します。誤り条件、ファイル終了条件および記録終了条件が発生しない場合、IOSTAT指定子には0が代入されます。

IOSTAT指定子が指定されている入出力文で誤り条件、ファイル終了条件、記録終了条件が発生したとき、その旨の診断メッセージは出力されません。また、標準修正および利用者の修正処理は行われません。詳細は“8.1 エラー制御”を参照してください。

表7.3 誤り条件、ファイル終了条件および記録終了条件

条件	内容	対応する入出力文	IOSTAT指定子に代入される値
誤り条件(回復不可能エラー)	ファイルの入出力操作中に回復不可能なエラーが発生しました。	各入出力文	1
誤り条件(その他)	入出力文でエラーが発生しています。	各入出力文	エラー識別番号
ファイル終了条件	ファイル終了記録を検出しました。	順番探査READ文 並びによるREAD文 変数群READ文 流れ探査READ文	-1
ファイル終了条件	内部ファイルの終わりを超えて Fortran 記録を読み込みました。	内部ファイルREAD文	-1
記録終了条件	停留入力文で入力並びの長さがFortran記録の長さを超えています。	停留入力文	-2

7.7.1.4 ERR指定子

入出力文の実行中に、誤り条件が発生し、プログラムの実行を制御する必要がある場合にERR指定子を指定します。誤り条件は、“表7.3 誤り条件、ファイル終了条件および記録終了条件”に示した内容です。

ERR指定子が指定されている入出力文で誤り条件が発生した場合、その旨の診断メッセージは出力されません。また、標準修正および利用者の修正処理は行われません。

7.7.1.5 END指定子

順番探査入出力文の実行中に、ファイル終了条件が発生し、プログラムの実行を制御する必要がある場合にEND指定子を指定します。ファイル終了条件は、“表7.3 誤り条件、ファイル終了条件および記録終了条件”に示した内容です。

END指定子が指定されている入力文でファイル終了条件が発生したとき、その旨の診断メッセージは出力されません。また、標準修正および利用者の修正処理は行われません。

7.7.1.6 EOR指定子

停留入力文の実行中に、記録終了条件が発生し、プログラムの実行を制御する必要がある場合にEOR指定子を指定します。記録終了条件は、“表7.3 誤り条件、ファイル終了条件および記録終了条件”に示した内容です。

EOR指定子が指定されている入力文で記録終了条件が発生したとき、その旨の診断メッセージは出力されません。また、標準修正および利用者の修正処理は行われません。

7.7.1.7 IOMSG指定子

誤り条件、ファイル終了条件および記録終了条件が発生した場合に、エラーメッセージが設定されます。

ただし、IOSTAT指定子またはERR指定子を同時に指定していない場合には、メッセージの設定は行われません。なお、誤り条件、ファイル終了条件および記録終了条件が発生していない場合には、指定子に指定された値は変更されません。

7.7.2 順番探査入出力文

順番探査入出力文は、書式付きFortran記録および書式なしFortran記録を入出力します。

書式付きFortran記録は、書式付き順番探査入出力文で入出力できます。また、書式なしFortran記録は、書式なし順番探査入出力文で入出力できます。

7.7.2.1 書式付き順番探査入出力文

書式付き順番探査入出力文は、文字コードからなるデータを書式仕様に従って順番に入出力するものです。一般に、この入出力文は、その内容を目で見る必要があるとき使用します。

書式付き順番探査READ文は、UNIT指定子で指定されたファイルの、現在位置付けられているFortran記録を、書式仕様に従って入力します。このREAD文に入力並びが指定されている場合、Fortran記録を、書式仕様の編集記述子によって内部コードに編集し、入力並びの各項目に入力します。入力並びの指定されていない書式付き順番探査READ文ではFortran記録を読み飛ばす処理を行います。

書式付き順番探査WRITE文は、書式仕様で編集されたFortran記録を、UNIT指定子で指定されたファイルの、現在位置付けられている位置に出力します。このWRITE文に出力並びが指定されている場合、各出力並び項目を、書式仕様の編集記述子によって文字コードに編集し、Fortran記録として出力します。出力並びの指定されていない書式付き順番探査WRITE文は、改行文字だけからなるFortran記録を出力します。書式仕様に$編集記述子が指定されている場合は、改行文字を含まないFortran記録を出力します。

以下に書式付き順番探査入出力文で入出力されるデータ例を示します。

```
      REAL A,B
      INTEGER I,J
      A = 1.5                   ! 3fc00000(16 進数値表現) がA に代入されます。
      I = 100                   ! 00000064(16 進数値表現) がI に代入されます。
      WRITE(1,10) I,A           ! 2031303020312e35(16 進数値表現) が
                                !     Fortran 記録として代入されます。
      REWIND 1                  ! ファイルの始点に位置付けられます。
      READ(1,10)J,B             ! J およびB に以下の値が入力されます。
                                !     J : 00000064(16 進数値表現)
                                !     B : 3fc00000(16 進数値表現)
10    FORMAT(1X,I3,1X,F3.1)
      STOP
      END
```

7.7.2.2 書式なし順番探査入出力文

書式なし順番探査入出力文は、内部コードからなるFortran記録を順番に入出力します。一般に、この入出力文は、その内容を目で見る必要がないとき使用します。

書式なし順番探査入出力文は、端末装置に対して実行できません。

書式なし入力文で入力することができるFortran記録は、富士通のFortran処理系によって出力したFortran記録です。

書式なし順番探査READ文は、UNIT指定子で指定されたファイルの、現在位置付けられているFortran記録を入力します。このREAD文に入力並びが指定されている場合、それぞれの入力並び項目へ入力並びで指定された順番に、Fortran記録のデータをそのまま入力します。このときFortran記録のデータのバイト数が入力並び項目のバイト数の総計より少ないならば、エラーとなり、多いならば、残りのデータを読み捨てます。入力並びの指定されていない書式なし順番探査READ文ではFortran記録を読み飛ばす処理を行います。

書式なし順番探査WRITE文は、出力並び項目を、出力並びで指定された順番に、UNIT指定子で指定されたファイルの現在位置付けられている位置に、Fortran記録として出力します。出力並びの指定されていない書式なし順番探査WRITE文は、記録長0のFortran記録を出力します。

Fortran記録の長さは、出力並び項目のバイト数の総計です。

以下に書式なし順番探査入出力文で入出力されるデータの例を示します。

```
REAL A,B
INTEGER I,J
A = 1.5                   ! 3fc00000(16 進数値表現) がA に代入されます。
I = 100                   ! 00000064(16 進数値表現) がI が代入されます。
WRITE(1) A,I              ! 3fc0000000000064(16 進数値表現) が
                          ! Fortran 記録として代入されます。
REWIND 1                  ! ファイルの始点に位置付けられます。
READ(1) B,J               ! B およびJ へ以下の値が入力されます。
                          ! B : 3fc00000(16 進数値表現)
                          ! J : 00000064(16 進数値表現)
STOP
END
```

7.7.3 直接探査入出力文

直接探査入出力文は、直接探査として接続されたファイルの任意の位置に書式付きFortran記録または書式なしFortran記録を入出力するときに使用します。

直接探査入出力文を実行するためには、それ以前に同一ファイルに対して、OPEN文が実行されていなければなりません。

OPEN文のTOTALREC指定子が指定されている場合、指定子で指定された記録数の範囲で入出力できます。なお、直接探査として接続されたファイルに対して順番探査入出力文によりFortran記録の追加をしてはなりません。

7.7.3.1 直接探査READ文

直接探査READ文は、FMT指定子が指定されている場合とされていない場合では、データの入力方法が異なります。

書式付き直接探査READ文は、REC指定子で指定された記録番号のFortran記録を書式仕様に従って、UNIT指定子で指定された装置番号に接続されているファイルから入力します。このREAD文に入力並びが指定されている場合、Fortran記録を書式仕様の編集記述子によって内部コードに編集し、入力並びの各項目に入力します。入力並びの指定されていない書式付き直接探査READ文ではFortran記録を読み飛ばす処理を行います。

書式なし直接探査READ文は、REC指定子で指定された記録番号のFortran記録をUNIT指定子で指定された装置番号に接続されているファイルから入力します。このREAD文に入力並びが指定されている場合、すべての入力並び項目へデータが入力されるまで、Fortran記録の読込みを行います。入力並びの指定されていない書式なし直接探査READ文ではFortran記録を読み飛ばす処理を行います。

7.7.3.2 直接探査WRITE文

直接探査WRITE文は、FMT指定子が指定されている場合とされていない場合では、データの出力方法が異なります。

書式付き直接探査WRITE文は、書式仕様で編集されたFortran記録をREC指定子で指定された記録番号の位置へ出力します。このWRITE文に出力並びが指定されている場合、各出力並び項目を書式仕様の編集記述子によって文字データに編集し、Fortran記録として出力します。出力並びが指定されていない書式付き直接探査WRITE文では、空白だけのFortran記録を出力します。

書式なし直接探査WRITE文は、各出力並び項目を出力並び項目で指定された順番に、REC指定子で指定された記録番号の位置へ出力します。出力並びが指定されていない書式なし直接探査WRITE文では、ゼロだけのFortran記録を出力します。1つのFortran記録は1つの論理記録を形成し、各出力並び項目のバイト数の総計が、1つの論理記録の長さより長い場合、出力並び項目のデータを複数のFortran記録として出力します。

7.7.3.3 直接探査入出力におけるデータ転送エラー

直接探査として接続されているファイルへの書式なしデータ転送において、出力項目並びの大きさがFortran記録の大きさよりも大きい場合または入力項目並びの大きさがFortran記録の大きさよりも小さい場合、Fortran 66およびFortran 77言語仕様では、次のFortran記録に書き出し、正常終了します。

Fortran 95以降の言語仕様では、実行時の診断メッセージ(jwe1162i-w)を出力し、次のFortran記録に転送します。

例:

プログラム di.f

```
OPEN(10,FILE='XX.DAT',FORM='UNFORMATTED',ACCESS='DIRECT',RECL=4)
WRITE(10,REC=1)1.0_8
INQUIRE(10,NEXTREC=i)
PRINT*,'NEXTREC=',i
END
```

実行結果

Fortran 66およびFortran 77言語仕様の場合

```
$ ./a.out
 NEXTREC=3
$
```

Fortran 95以降の言語仕様の場合

```
$ ./a.out
jwe1162i-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
  jwe1162i         w           1
total error count = 1
$
```

7.7.4 変数群入出力文

変数群入出力文は、あらかじめNAMELIST文によって宣言された変数群名とそれに属するスカラ変数名および配列変数名に対して、順番探査または流れ探査として接続したファイルおよび内部ファイル上の文字列からなる変数群Fortran記録との間でデータ転送を行う入出力文です。

直接探査で接続されているファイルの記録を変数群書式を用いて読んだり書いたりしてはいけません。

変数群書式による入出力において、内部ファイルの記録を読んだり書いたりすることができます。

内部ファイル入出力文に変数群書式を指定した場合も含めて変数群入出力文として説明します。

7.7.4.1 変数群READ文

変数群READ文は、入力される変数群名によって指定された要素の型と、入力される変数群Fortran記録の定数の型が一致していなければなりません。以下に、変数群READ文における要素の型と定数の型との対応を示します。

要素の型	定数の型					
	論理型	整数型	実数型	複素数型	文字型	16進型
1バイトの論理型 2バイトの論理型 4バイトの論理型 8バイトの論理型	○	△ (注1)	△ (注1)	×	△ (注2)	△ (注7) (注8)
1バイトの整数型 2バイトの整数型 4バイトの整数型 8バイトの整数型	△ (注3)	○	△ (注3)	×	△ (注2)	○ (注8)
実数型 倍精度実数型 4倍精度実数型	△ (注4)	○ (注5)	○	×	△ (注2)	△ (注7) (注8)
複素数型 倍精度複素数型 4倍精度複素数型	△ (注4) (注6) (注9)	△ (注5) (注6) (注9)	△ (注6) (注9)	○	△ (注2)	△ (注7) (注8)
文字型	×	×	×	×	○	× (注7) (注8)

○: 正常に動作することを示します。

△: 許されない対応ですが、修正解釈を行って動作することを示します。

×: 許されない対応なので、入力処理を終了することを示します。

注1) 論理型の編集を行います。

注2) 文字型の編集を行います。

注3) 整数型の編集を行います。

注4) 実数型の編集を行います。

注5) 定数の右端に小数点があるものとして、実数型の編集を行います。

注6) 実部に編集結果を入力し、虚部には値0を設定します。

注7) Fortran 66およびFortran 77言語仕様の場合、16進型はすべて“○”とします。

注8) Fortran 95以降の言語仕様の場合、16進型は文字データとして扱います。したがって、要素の型が文字型のときのみ“○”、そのほかの型は“△”とします。翻訳時オプションについては、“2.2 翻訳時オプション”を参照してください。

注9) Fortran 95以降の言語仕様の場合、Wレベルの実行時メッセージが出力されます。

備考. 入力処理は、要素の型に合った編集を行いますが、定数の型が16進型および文字型の場合、定数の型に合った編集を行います。

NAMELIST配列変数名に対する文字定数の扱い

配列変数名に対する文字定数の扱いは、要素の型が文字型か文字型以外かによって異なります。

要素の型が文字型の場合、1つの配列要素に対して1つの文字定数を1対1に対応するように入力し、その文字定数の長さが配列要素の長さを超えていても、次の配列要素にあふれを及ぼすことはありません。以下に、要素の型が文字型の場合の文字定数の扱いについて示します。

例: 文字型の配列変数名に対する文字定数

```
CHARACTER(LEN=4),DIMENSION(5) :: HA
NAMELIST/NAMEH/HA
READ(1,NML=NAMEH)       !(1) 変数群名 NAMEHに対する READ文
```

装置番号1に対する変数群Fortran記録が次のように設定されているものとします。

```
&NAMEH HA = 2*'NAMELIST READ STATEMENT' /
```

(1) で入力した結果は次のとおりです。(文字表現)

HA(1)	'NAME '
HA(2)	'NAME '
HA(3)	値は変わらず
HA(4)	値は変わらず
HA(5)	値は変わらず

要素の型が文字型以外の場合、最初の配列要素に対応する文字定数を配列全体に対して入力します。

すなわち、文字定数の長さが1つの要素より長い場合、指定した文字定数分を要素を超えて入力します。文字定数の長さが配列全体の長さより短い場合、残りの要素の値は変わりません。ただし、1つの要素の途中まで入力されている場合、要素内の残りには空白が入力されます。

2番目以降の配列要素に対しては、各配列要素と1対1に対応し、その文字定数の長さが配列要素の長さを超えていても、次の配列要素にあふれを及ぼすことはありません。以下に、要素の型が文字型以外の場合の文字定数の扱いについて示します。

例: 文字型を除く配列変数名に対する文字定数

```
INTEGER,DIMENSION(5) :: IA
NAMELIST/NAMEI/IA
READ(1,NML=NAMEI)  !(1)変数群名 NAMEIに対する READ文
```

装置番号1に対する変数群Fortran記録が次のように設定されているものとします。

```
&NAMEI IA = 2*'NAMELIST READ STATEMENT' /
```

(1)で入力した結果は次のとおりです。(文字表現)

IA(1)	'NAME'
IA(2)	'NAME'
IA(3)	' REA'
IA(4)	'D ST'
IA(5)	'ATEM'

lA(1)の処理で配列 IA全体に文字定数を入力します。

NAMELIST入力編集動作変数群

READ文は、入力編集を行うときの変数群書式を、要素の型、定数の型および定数の大きさから次のように定めます。

定数の型が文字型以外、かつ、要素の型が文字型以外の場合、要素の型に合わせて編集を行い、その編集記述子はG*w*です。ここで*w*は、入力対象となる定数の大きさ(区切りから次の区切りまでの大きさ)です。

定数の型が文字型の場合、文字型の編集を行い、その編集記述子はA*w*です。ここで*w*は、文字定数を表すアポストロフィからこれに対応するアポストロフィまでの大きさ、引用符からこれに対応する引用符までの大きさまたは文字定数の文字の長さです。

定数の型が文字型以外、かつ、要素の型が文字型の場合、文字型の編集を行い、その編集記述子はA*w*です。

7.7.4.1.1 変数群入力における変数群記録

変数群入力における変数群記録(namelist record)は、次のどちらかの形です。

```
&name [v=c[,c]...[,v=c[,c]...]...] /
```

```
&name [v=c[,c]...[,v=c[,c]...]...] &end
```

name: 対応する変数群入出力文に指定した変数群名です。

v: 変数群名*name*に対応するNAMELIST文に指定した変数名またはそれに準ずる配列要素、部分配列、または文字部分列です。

c: 空値または次のいずれかの形です。

```
const
```

```
r*const
```

```
r*
```

const: ホレリス定数、2進定数、8進定数、および16進定数を除く定数表現です。

r: 繰返し数であり、符号なし整定数です。

*r***const*: *const*を*r*個連続して書いたものと同じです。

*r**: 空値を*r*個連続して書いたものと同じです。

入力する項目が文字型であり、以下の条件をすべて満たしていれば、アポストロフィまたは引用符で文字定数を囲む必要はありません。

- 文字定数の中に値区切り子とみなされる空白、コンマ、斜線、またはアンド記号を含んでいない。
- 小数点記号がコンマであるとき、文字定数の中に値区切り子とみなされるセミコロンを含んでいない。
- 文字定数の中に名前とみなされる左括弧、右括弧、パーセント記号を含んでいない。
- 文字定数が複数のFortran記録にまたがっていない。
- 空白でない先頭の文字がアポストロフィまたは引用符以外の文字である。
- 入力する項目の形が、*r***const*でない。

文字定数の囲み文字としてのアポストロフィまたは引用符を省略した場合、その文字定数は、最初の空白、コンマ、斜線の終わりで終了するものとみなされます。また、文字列中のアポストロフィまたは引用符は、2連アポストロフィまたは2連引用符としては扱われません。

7.7.4.2 変数群WRITE文

変数群WRITE文は、変数群名に属するスカラ変数名および配列変数名に対して、それぞれの型に合った出力編集を行い、変数群Fortran記録を出力します。

ここでは、出力編集を行うときの出力形式(編集記述子)について説明します。

スカラ変数名に対する出力形式は、スカラ変数名に続いて等号、続いて編集結果、更に定数の区切りとしてコンマを出力します。

配列変数名に対する出力形式は、配列変数名に続いて等号、続いて各配列要素に対する編集結果をコンマで区切って出力します。

以下に、要素の型に対する出力形式を示します。

要素の型	出力形式(編集記述子)
1バイトの論理型	G1,1H, (注1)
2バイトの論理型	G1,1H, (注1)
4バイトの論理型	G1,1H, (注1)
8バイトの論理型	G1,1H, (注1)
1バイトの整数型	G4,1H, (注1)
2バイトの整数型	G6,1H, (注1)
4バイトの整数型	G11,1H, (注1)
8バイトの整数型	G20,1H, (注1)
実数型	1P,E15.8,1H, (注1)
	0P,G16.9,1H, (注1)
倍精度実数型	1P,E22.15,1H, (注1)
	0P,G23.16,1H, (注1)
4倍精度実数型	1P,E43.34E4,1H, (注1)
	0P,G44.35E4,1H, (注1)
複素数型	1H(,1P,E15.8,1H,,E15.8,2H), (注2)
	1H(,0P,G16.9,1H,,G16.9,2H), (注2)
倍精度複素数型	1H(,1P,E22.15,1H,,E22.15,2H), (注2)
	1H(,0P,G23.16,1H,,G23.16,2H), (注2)
4倍精度複素数型	1H(,1P,E43.34E4,1H,,E43.34E4,2H), (注2)
	1H(,0P,G44.35E4,1H,,G44.35E4,2H), (注2)
長さ*n*バイトの文字型	1H',要素の内容,2H', (注3)

要素の型	出力形式(編集記述子)
	1H",要素の内容,2H", (注4)
	要素の内容,1H, (注5)

注1) G形編集記述子またはE形編集記述子で出力された値の直後が終わりを示す/または&endの場合、1H,は省略されます。

注2) G形編集記述子または E形編集記述子で出力された値の直後が終わりを示す/または&endの場合、2H),は1H)となります。

注3) 囲み記号の引用符をもつ文字定数の直後が変数群出力の終わりを示す/または&endの場合、2H',は1H'となります。

注4) 囲み記号のアポストロフィをもつ文字定数の直後が変数群出力の終わりを示す/または&endの場合、2H",は1H"となります。

注5) 囲みなし文字定数の直後が変数群出力の終わりを示す/または&endの場合、1H,は省略されます。

備考1. 文字型以外の出力編集の結果で意味のない空白が現われたならばその空白を削除して出力するので、実際の出力長は編集記述子の欄の幅の合計より小さくなることがあります。

備考2. 実数型および複素数型において、“1P,Ew1.d1”となるか“0P,Gw2.d2”となるかは、データの絶対値xが0または0.1≦x＜10**d2ならば、“0P,Gw2.d2”となり、そのほかならば“1P,Ew1.d1”となります。

7.7.4.3 変数群入出力の言語仕様レベルによる違い

文字型の出力

Fortran 66およびFortran 77言語仕様では、文字列の前後に囲み記号のアポストロフィをもった文字定数が出力されます。

Fortran 95以降の言語仕様では、OPEN文でDELIM指定子を省略するか、またはOPEN文を実行しない場合、DELIM指定子の省略値はNONEと解釈されます。

この場合、文字列の前後に囲み記号のアポストロフィまたは引用符をもたない文字定数が出力されます。

例:

```
CHARACTER(LEN=5) :: CH='XXXXX'
NAMELIST /X/CH
OPEN(10)
WRITE(10,NML=X)
```

上記のFortranプログラムを実行すると、Fortran 66およびFortran 77言語仕様では、以下のように出力されます。

```
&X
CH='XXXXX'
&end
```

Fortran 95以降の言語仕様では、以下のように出力されます。

```
&X CH=XXXXX/
```

変数群出力文の終わりを示す形式

Fortran 66およびFortran 77言語仕様では、変数群出力文の終わりは&endで出力されます。

Fortran 95以降の言語仕様では、変数群出力文の終わりは、/で出力されます。

例:

```
INTEGER :: IH=123
NAMELIST /X/IH
WRITE(10,NML=X)
```

上記のFortranプログラムを実行すると、Fortran 66およびFortran 77言語仕様では、以下のように出力されます。

```
&X
IH=123
&end
```

Fortran 95以降の言語仕様では、以下のように出力されます。

```
&X IH=123/
```

変数群入力の空白文字の扱い

Fortran 66およびFortran 77言語仕様では、空白文字は入力データの一部として扱われます。

Fortran 95以降の言語仕様では、空白文字は値区切り子として扱われます。ただし、囲み記号でくくられている場合、値区切り子ではなく、入力データとして扱われます。

例:

```
INTEGER,DIMENSION(3) :: IH
NAMELIST /X/IH
READ(*,NML=X)
```

入力データをIH =1□2□3,とすると(□: 空白を示します。)、Fortran 66およびFortran 77言語仕様では、以下のように扱われます。

```
IH(1) = 10203
IH(2) = 0
IH(3) = 0
```

Fortran 95以降の言語仕様では、以下のように扱われます。

```
IH(1) = 1
IH(2) = 2
IH(3) = 3
```

変数群入力の入力データに指定された16進定数の扱い

Fortran 66およびFortran 77言語仕様では、16進定数として扱われます。

Fortran 95以降の言語仕様では、文字定数として扱われます。

例:

```
CHARACTER(LEN=5) CH
NAMELIST /X/CH
READ(*,NML=X)
```

入力データをCH=Z3132333435とすると、Fortran 66およびFortran 77言語仕様では、以下のように扱われます。

```
CH = '12345'
```

Fortran 95以降の言語仕様では、以下のように扱われます。

```
CH = 'Z3132'
```

変数群入力の入力データに指定されたホレリス定数の扱い

Fortran 66およびFortran 77言語仕様では、ホレリス定数として扱われます。

Fortran 95以降の言語仕様では、文字定数として扱われます。

例:

```
CHARACTER(LEN=5) CH
NAMELIST /X/CH
READ(*,NML=X)
```

入力データをCH=3H123とすると、Fortran 66およびFortran 77言語仕様では、以下のように扱われます。

```
CH = '123  '
```

Fortran 95以降の言語仕様では、以下のように扱われます。

```
CH = '3H123'
```

囲まれた文字定数が複数のFortran記録にまたがる場合の2レコード目以降の先頭1バイトの出力

Fortran 66およびFortran 77言語仕様では、空白が出力されます。

Fortran 95以降の言語仕様では、空白は出力されず、2レコード目以降の先頭1バイトに値が出力されます。

例:

```
CHARACTER(LEN=1000) CH
NAMELIST /X/CH
WRITE(10,NML=X)
```

Fortran 66およびFortran 77言語仕様では、以下のように出力されます。

CH='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ← 1レコード

□XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' ← 2レコード

↑1バイト目に空白が出力されます。

Fortran 95以降の言語仕様では、以下のように扱われます。

CH='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ← 1レコード

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' ← 2レコード

↑1バイト目に値が出力されます。

変数群入力の繰返し形式のデータの扱い

Fortran 66およびFortran 77言語仕様では、*r**の形で入力データを与えた場合、空値に続く値が *r*個分連続したデータとして扱われます。また、*r**または*r***c*の形で入力データを与え、*がFortran記録の最後の文字の場合、次のFortran記録に最初に現れる値が *r*個分連続したデータとして扱われます。

Fortran 95以降の言語仕様では、空値が繰返し数の個数分連続したデータとして扱われます。

例:

```
INTEGER :: I(4)=(/1,2,3,4/)
NAMELIST /X/I
READ(*,NML=X)
```

入力データをI =2*□30,40,とする(□: 空白を示します)と、Fortran 66およびFortran 77言語仕様では、以下のように扱われます。

```
I(1) = 30
I(2) = 30
I(3) = 40
I(4) = 4
```

Fortran 95以降の言語仕様では、以下のように扱われます。

```
I(1) = 1
I(2) = 2
I(3) = 30
I(4) = 40
```

変数群出力文の変数群名および変数名の出力

Fortran 66およびFortran 77言語仕様では、変数群名および変数名はプログラム中で記述された文字で出力されます。

Fortran 95以降の言語仕様では、変数群名および変数名は英大文字で出力されます。

例:

```
INTEGER :: Iu = 1
NAMELIST /Xu/Iu
WRITE(10,NML=Xu)
```

上記のFortranプログラムを実行すると、Fortran 66およびFortran 77言語仕様では、以下のように出力されます。

```
&Xu
Iu=1
&end
```

Fortran 95以降の言語仕様では、以下のように出力されます。

```
&XU IU=1/
```

変数群Fortran記録の出力

Fortran 66およびFortran 77言語仕様では、以下の形式で出力されます。

第1レコード	□ &*name*
第2レコード	□ *v*=*c*[,*c*]...[,*v*=*c*[,*c*]...]...
第3レコード	□ &end

Fortran 95以降の言語仕様では、以下の形式で出力されます。

第1レコード以降	□ &*name* *v*=*c*[, *c*]...[, *v*=*c*[, *c*]...].../

ここで、□は空白、*name*は変数群名、*v*および*c*は変数群要素並びと値を表します。

実数ゼロの出力

値が0である実数型の変数群出力を行った結果が、言語仕様レベルによって異なります。

Fortran 66、Fortran 77およびFortran 95言語仕様では、E 形編集で出力した結果と同等です。

Fortran 2003以降の言語仕様では、F 形編集で出力した結果と同等です。

例:

```
REAL :: R = 0
NAMELIST /DATA/R
WRITE(*, NML=DATA)
```

上記のFortran プログラムを実行すると、Fortran 95言語仕様では、&DATA R=0.00000000E+00/ と出力されます。

Fortran 2003以降の言語仕様では、&DATA R=0.00000000/ と出力されます。

7.7.5 並びによる入出力文

並びによる入出力文は、並びによる書式に従って、順番探査または流れ検査として接続したファイルおよび内部ファイル上の文字列からなる並びによるFortran記録との間でデータ転送を行う入出力文です。

内部ファイル入出力文に並びによる書式を指定した場合も含めて並びによる入出力文として説明します。

7.7.5.1 並びによるREAD文

並びによるREAD文は、並びによる書式に従って、外部ファイルおよび内部ファイル上の並びによるFortran記録を入力並びの要素に転送します。

入力並びの要素の型と対応するデータ項目の型は同じでなければなりません。

以下に入力並びの要素の型とデータ項目の型の対応を示します。

要素の型	定数の型				
	論理型	整数型	実数型	複素数型	文字型
1バイトの論理型 2バイトの論理型 4バイトの論理型 8バイトの論理型	○	△ (注1)	△ (注1)	×	△ (注2)
1バイトの整数型 2バイトの整数型 4バイトの整数型 8バイトの整数型	△ (注3)	○	△ (注3)	×	△ (注2)
実数型 倍精度実数型 4倍精度実数型	△ (注4)	○ (注5)	○	×	△ (注2)
複素数型 倍精度複素数型 4倍精度複素数型	△ (注4) (注6) (注7)	△ (注5) (注6) (注7)	△ (注6) (注7)	○	△ (注2)
文字型	△ (注2)	△ (注2)	△ (注2)	△ (注2)	○

○: 正常に動作することを示します。

△: 許されない対応ですが、修正解釈を行って動作することを示します。

×: 許されない対応なので、入力処理を終了することを示します。

注1) 論理型の編集を行います。

注2) 文字型の編集を行います。

注3) 整数型の編集を行います。

注4) 実数型の編集を行います。

注5) 定数の右端に小数点があるものとして、実数型の編集を行います。

注6) 実部に編集結果を入力し、虚部には値0を設定します。

注7) Fortran 95以降の言語仕様の場合、Wレベルの実行時メッセージを出力します。

備考. 入力処理は、要素の型に合った編集を行いますが、定数の型が文字型の場合、定数の型に合った編集を行います。

並び入力の配列変数名に対する文字定数の扱い

配列変数名に対する文字定数の扱いは、要素の型が文字型か文字型以外かによって異なります。

要素の型が文字型の場合、1つの配列要素に対して1つの文字定数が1対1に対応するように入力され、その文字定数の長さが1つの配列要素の長さを超えていても、次の配列要素にあふれを及ぼすことはありません。

要素の型が文字型以外の場合、1番目の配列要素に対応する文字定数が、配列全体に対して入力されます。すなわち、文字定数の長さが1つの要素より長い場合、指定した文字定数分を要素を超えて入力します。文字定数の長さが配列全体の長さより短い場合は残りの要素の値は変わりません。ただし、1つの要素の途中まで入力されている場合、要素内の残りには空白が入力されます。2番目以降の配列要素に対しては、それぞれの配列要素と1対1に対応し、文字定数の長さが1つの配列要素の長さを超えていても、次の配列要素にあふれを及ぼすことはありません。

例: 配列変数名に対する文字型データの入力

```
CHARACTER(LEN=4),DIMENSION(5):: CHAR5
INTEGER,DIMENSION(5):: I4
```

```
   ...
READ(5,*)CHAR5 !文字型の配列要素に対する並びによるREAD文
READ(5,*)I4    !文字型以外の配列要素に対する並びによるREAD文
```

装置番号5に対する並びによるFortran記録は以下の形式で与えられたものとします。

```
3*'ABCDEFGHIJKLMNOPQ'/
3*'ABCDEFGHIJKLMNOPQ'/
```

CHAR5およびI4への入力結果は次のとおりです。

CHAR5(1)	'ABCD'	I4(1)	'ABCD'
CHAR5(2)	'ABCD'	I4(2)	'ABCD'
CHAR5(3)	'ABCD'	I4(3)	'ABCD'
CHAR5(4)	値は変わらず	I4(4)	'MNOP'
CHAR5(5)	値は変わらず	I4(5)	'Q '

I4(1)の処理で配列I4全体に文字定数を入力します。

並びによる入力編集動作

並びによるREAD文は、入力編集を行うときの書式仕様を、要素の型、定数の型および定数の大きさから次のように定めます。

定数の型が文字型以外、かつ、要素の型が文字型以外の場合、要素の型に合わせて編集を行い、その編集記述子はG*w*です。ここで*w*は、入力対象となる定数の大きさ(区切りから次の区切りまでの大きさ)です。

定数の型が文字型の場合、文字型の編集を行い、その編集記述子はA*w*です。ここで*w*は、文字定数を表すアポストロフィからこれに対応するアポストロフィまでの大きさ、文字定数を表す引用符からこれに対応する引用符までの大きさ、または文字定数を表す文字の長さです。

定数の型が文字型以外、かつ、要素の型が文字型の場合、文字型の編集を行い、その編集記述子はA*w*です。

7.7.5.2 並びによるWRITE文

並びによるWRITE文は、並びによる書式に従って、編集された並びによるFortran記録を出力します。

以下に並びによるWRITE文が、出力編集を行うときの編集記述子について示します。

要素の型	出力形式 (編集記述子)
1バイトの論理型	G1,1H□ (注1)
2バイトの論理型	G1,1H□ (注1)
4バイトの論理型	G1,1H□ (注1)
8バイトの論理型	G1,1H□ (注1)
1バイトの整数型	G4,1H□ (注1)
2バイトの整数型	G6,1H□ (注1)
4バイトの整数型	G11,1H□ (注1)
8バイトの整数型	G20,1H□ (注1)
実数型	1P,E15.8,1H□ (注1)、 0P,G16.9,1H□ (注1)
倍精度実数型	1P,E22.15,1H□ (注1)、 0P,G23.16,1H□ (注1)

要素の型	出力形式 (編集記述子)
4倍精度実数型	1P,E43.34E4,1H□ (注1)、 0P,G44.35E4,1H□ (注1)
複素数型	1H(,1P,E15.8,1H,,E15.8,2H)□ (注2)、 1H(,0P,G16.9,1H,,G16.9,2H)□ (注2)
倍精度複素数型	1H(,1P,E22.15,1H,,E22.15,2H)□ (注2)、 1H(,0P,G23.16,1H,,G23.16,2H)□ (注2)
4倍精度複素数型	1H(,1P,E43.34E4,1H,,E43.34E4,2H)□ (注2)、 1H(,0P,G44.35E4,1H,,G44.35E4,2H)□ (注2)
長さ nバイトの文字型	Gn,1H□ (注3)(注6)、 1H',Gn,2H'□ (注4)(注6)、 1H",Gn,2H"□ (注5)(注6)

□: 空白を示します。

注1) G形編集記述子またはE形編集記述子で出力された値の直後がFortran記録の終わりの場合、1H□は省略されます。

注2) G形編集記述子または E形編集記述子で出力された値の1バイトあとがFortran記録の終わりの場合、2H)□は1H)となります。

注3) 囲みなし文字定数の直後がFortran記録の終わりの場合、1H□は省略されます。

注4) 囲み記号の引用符をもつ文字定数の直後がFortran記録の終わりの場合、2H'□は1H'となります。

注5) 囲み記号のアポストロフィをもつ文字定数の直後がFortran記録の終わりの場合、2H"□は1H"となります。

注6) Fortran 66およびFortran 77言語仕様の場合、最後の空白は出力されません。

備考1. 文字型以外の出力編集の結果で意味のない空白が現われたならばその空白を削除して出力するので、実際の出力長は編集記述子の欄の幅の合計より小さくなることがあります。

備考2. 実数型および複素数型において、“1P,Ew1.d1”となるか“0P,Gw2.d2”となるかは、データの絶対値xが0または0.1≦x＜10**d2ならば、“0P,Gw2.d2”となり、そのほかならば“1P,Ew1.d1”となります。

例: 並びによる WRITE文

```
INTEGER :: YY = 2002, MM = 09, DD = 10
CHARACTER(LEN=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
```

上記Fortranプログラムを実行すると、以下のように出力されます。

```
$ ./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 並び出力における囲みなし文字定数の出力形式

Fortran 66およびFortran 77言語仕様では、値は区切られません。

Fortran 95以降の言語仕様では、値は1個の空白で区切られます。

例:

```
CHARACTER(LEN=10) :: CH = ' '
WRITE(*,*)'Year',2002
WRITE(CH,*)'Month',9
WRITE(*,'(A10)')CH
END
```

上記のFortranプログラムを実行すると、Fortran 66およびFortran 77言語仕様では、以下のように、続けて出力されます。

```
$ ./a.out
 Year2002
 Month9
$
```

Fortran 95以降の言語仕様では、以下のように、空白で区切られて出力されます。

```
$ ./a.out
 Year 2002
 Month 9
$
```

7.7.5.3 並びによる入出力の言語仕様レベルによる違い

実数ゼロの出力

値が0である実数型の並び出力を行った結果が、言語仕様レベルによって異なります。

Fortran 66、Fortran 77およびFortran 95言語仕様では、E 形編集で出力した結果と同等です。

Fortran 2003以降の言語仕様では、F 形編集で出力した結果と同等です。

例:

```
REAL :: R = 0
WRITE(*,*) R
```

上記のFortran プログラムを実行すると、Fortran 95言語仕様では、0.00000000E+00 またはと出力されます。Fortran 2003以降の言語仕様では、0.00000000 と出力されます。

7.7.6 内部ファイル入出力文

内部ファイル入出力文は、内部ファイル上の文字列から、指定された書式に対応するFortran記録との間でデータ転送を行う入出力文です。

内部ファイル入出力文の説明については、指定する書式仕様に対応する入出力文の説明を参照してください。

7.7.7 停留入出力文

前進入出力文を実行すると、ファイル位置を入力または出力した直後のFortran記録に進めますが、停留入出力文では、ファイル位置を直後のFortran記録に進めずに、1つのFortran記録内のデータを続けて入出力することができます。

このため、停留入出力文を実行すると、ファイル位置がFortran記録の途中になることがあります。停留入力文には、ファイル位置により、そのあとの処理を制御するために、SIZE指定子とEOR指定子を指定することができます。SIZE指定子は、実際に入力したデータの文字数を通知します。また、EOR指定子を使用することにより、現在のFortran記録の終わりを超えて入力したときに、指定した文番号へ分岐することができます。

以下の例は、ADVANCE='NO'をREAD文に指定して入力し、SIZE、EOR指定子で、データの入力を制御するものです。

例:

```
     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',EOR=10,SIZE=IS,END=20,IOSTAT=IO) NO
         IF (IS < 8) CYCLE
         GO TO 30
   10    CYCLE
   30    PRINT*,NAME,' ',NO,' ',IS
      ENDDO
      GO TO 40
   20 PRINT*,'END OF DATA '
   40 END
```

7.8 入出力文の組合せ

入出力文が実行されるときは、その前にどのような入出力文が実行されているかによって処理が異なります。ここでは、書式なし順番探査入出力文、書式付き順番探査入出力文(並びによる入出力文または変数群入出力文を含む)、直接探査入出力文、ファイル位置付け文、OPEN文、CLOSE文およびFLUSH文に対する組合せの前後関係処理について説明します。ここで説明する入出力文の組合せは、同一装置番号に対する前後関係処理です。

以下の2つの表に、入出力文の前後関係による処理を示します。

現在の入出力文	直前の入出力文											
	書式付き順番探査READ文	書式付き順番探査WRITE文	書式なし順番探査READ文	書式なし順番探査WRITE文	書式付き直接探査READ文	書式付き直接探査WRITE文	書式なし直接探査READ文	書式なし直接探査WRITE文	書式付き流れ探査READ文	書式付き流れ探査WRITE文	書式なし流れ探査READ文	書式なし流れ探査WRITE文
書式付き順番探査READ文	○	× (注1)	×	×	×	×	×	×	×	×	×	×
書式付き順番探査WRITE文	○ (注2)	○	×	×	×	×	×	×	×	×	×	×
書式なし順番探査READ文	×	×	○	× (注1)	×	×	×	×	×	×	×	×
書式なし順番探査WRITE文	×	×	○ (注2)	○	×	×	×	×	×	×	×	×
書式付き直接探査入出力文	×	×	×	×	○	○	× (注5)	× (注5)	×	×	×	×
書式なし直接探査入出力文	×	×	×	×	× (注5)	× (注5)	○	○	×	×	×	×

現在の入出力文	直前の入出力文											
	書式付き順番探査READ文	書式付き順番探査WRITE文	書式なし順番探査READ文	書式なし順番探査WRITE文	書式付き直接探査READ文	書式付き直接探査WRITE文	書式なし直接探査READ文	書式なし直接探査WRITE文	書式付き流れ探査READ文	書式付き流れ探査WRITE文	書式なし流れ探査READ文	書式なし流れ探査WRITE文
書式付き流れ探査READ文	×	×	×	×	×	×	×	×	○	○ (注6)	×	×
書式付き流れ探査WRITE文	×	×	×	×	×	×	×	×	○	○	×	×
書式なし流れ探査READ文	×	×	×	×	×	×	×	×	×	×	○	○ (注6)
書式なし流れ探査WRITE文	×	×	×	×	×	×	×	×	×	×	○	○
BACKSPACE文	○	○ (注3)	○	○ (注3)	×	×	×	×	○	○	×	×
ENDFILE文	○ (注4)	○	○ (注4)	○	×	×	×	×	○	○	○	○
REWIND文	○	○	○	○	×	×	×	×	○	○	○	○
OPEN文	○	○	○	○	○	○	○	○	○	○	○	○
CLOSE文	○	○	○	○	○	○	○	○	○	○	○	○
FLUSH文	無操作	○	無操作	○	無操作	○	無操作	○	無操作	○	無操作	○
WAIT文	○	○	○	○	○	○	○	○	○	○	○	○

○: 正常に動作することを示します。

×: エラーメッセージを出力し、何らかの修正処理を行うことを示します。

無操作: 何も動作しないので、無意味なことを示します。

注1) 直前のWRITE文で書いたFortran記録よりあとのFortran記録は不定なので、READ文を実行してはなりません。

注2) 現在のWRITE文で書いたFortran記録よりあとのFortran記録は不定となります。

注3) ファイル終了記録を出力したあとでBACKSPACE動作が行われます。

注4) READ文が読み込んだ次のFortran記録にEOFが書き出されます。

注5) Fortran 66およびFortran 77言語仕様の場合、正常に動作します。

注6) POS=指定子を省略した流れ探査READ文との組合せはエラーとなります。

備考. ここで、書式付きREAD文および書式付きWRITE文には、並びによる入出力文および変数群入出力文も含まれます。

現在の入出力文	直前の入出力文								
	BACKSPACE文	ENDFILE文	REWIND文	OPEN文	CLOSE文	FLUSH文	WAIT文	EOF検出(注1)	なし
書式付き順番探査READ文	○	○(注7)	○(注2)	○	○(注6)	○	○	○(注7)	○(注8)
書式付き順番探査WRITE文	○	○(注7)	○(注2)	○	○(注6)	○	○	○(注7)	○(注8)
書式なし順番探査READ文	○	○(注7)	○(注2)	○	○(注6)	○	○	○(注7)	○(注8)
書式なし順番探査WRITE文	○	○(注6)	○(注2)	○(注6)	○(注6)	○	○	○(注11)	○(注8)
書式付き直接探査入出力文	×	×	×	○	×	○	○	×	×(注3)
書式なし直接探査入出力文	×	×	×	○	×	○	○	×	×(注3)
書式付き流れ探査READ文	○	○	○(注2)	○	×	○	○	○	○(注3)
書式付き流れ探査WRITE文	○	○	○(注2)	○	×	○	○	○	○(注3)
書式なし流れ探査READ文	×	○	○(注2)	○	×	○	○	○	○(注3)
書式なし流れ探査WRITE文	×	○	○(注2)	○	×	○	○	○	○(注3)
BACKSPACE文	○	○(注5)	無操作	○	○(注6)	○	○	○(注4)	○(注10)
ENDFILE文	○	無操作	○(注5)	○(注5)	○(注6)	○	○	無操作(注9)	○(注5)
REWIND文	○	○	無操作	○	無操作(注9)	○	○	○	無操作
OPEN文	○	○	○	○	○	○	○	○	○
CLOSE文	○	○	○	○	無操作	○	○	○	○
FLUSE文	無操作	無操作	無操作	無操作	無操作	無操作	無操作	無操作	無操作
WAIT文	○	○	○	○	○	○	○	○	○

○: 正常に動作することを示します。

×: エラーメッセージを出力し、何らかの修正処理を行うことを示します。

無操作: 何も動作しないので、無意味なことを示します。

注1) EOF検出とは、入出力動作によってファイル終了記録が検出されたことを示します。

注2) REWIND文の前後で書式付き入出力文と書式なし入出力文の組合せは許されません。

注3) すでに生成されているファイルに対しても、必ず、OPEN文を実行しなければなりません。

注4) ファイル終了記録に対してBACKSPACE動作が行われます(実際は、すでにファイル終了記録の直前に位置付けられているので無操作となります)。

注5) データのないファイルが生成されます。

注6) ファイル終了記録からの入出力動作となります。Fortran 95以降の言語仕様の場合、実行時エラーメッセージ(jwe0111i-e)が出力されます。Fortran 66およびFortran 77言語仕様の場合、実行時エラーメッセージは、出力されません。

注7) ファイル終了記録からの入出力動作となります。

注8) ファイル名'fort.*nn*'(*nn*:装置参照番号)のファイルに入出力動作が行われます。

注9) エラーとなります。

注10) ファイル終了記録に対してBACKSPACE動作が行われます(実際は、すでにファイル終了記録の直前に位置付けられているので無操作となります)。

注11) ファイル終了記録からの入出力動作となります。Fortran 95以降の言語仕様の場合、実行時エラーメッセージ(jwe0115i-e)が出力されます。Fortran 66およびFortran 77言語仕様の場合、実行時エラーメッセージは、出力されません。

備考. ここで、書式付きREAD文および書式付きWRITE文には、並びによる入出力文および変数群入出力文も含まれます。

7.8.1 入出力文の許されない組合せ

入出力文の組合せには、探査方法およびファイルの状態により、許されない組合せがあります。

許されない組合せが生じた場合は、エラー処理を行い、それぞれに定められた修正処理を行って実行を継続します。

7.8.1.1 書式付き順番探査入出力文との組合せ

現在の入出力文が書式付き順番探査入出力文、並びによる入出力文および変数群入出力文の場合、直前の入出力文が次に示すものであってはなりません。

- 書式なし順番探査入出力文
- 直接探査入出力文
- 流れ探査入出力文

また、現在の入出力文がREAD文の場合、直前の入出力文がWRITE文であってはなりません。ただし、入出力装置が端末装置のときはこの限りではありません。

現在の文がWRITE文の場合、直前の文がENDFILE文であってはなりません。ただし、Fortran 66およびFortran 77言語仕様の場合は、エラーではありません。

7.8.1.2 書式なし順番探査入出力文との組合せ

現在の入出力文が書式なし順番探査入出力文の場合、直前の入出力文が次に示すものであってはなりません。

- 書式付き順番探査入出力文
- 並びによる入出力文
- 変数群入出力文
- 直接探査入出力文
- 流れ探査入出力文

また、現在の入出力文がREAD文の場合、直前の入出力文がWRITE文であってはなりません。

現在の文がWRITE文の場合、直前の文がENDFILE文であってはいけません。ただし、Fortran 66およびFortran 77言語仕様の場合は、エラーではありません。

7.8.1.3 直接探査入出力文との組合せ

現在の入出力文が直接探査入出力文の場合、直前の入出力文が次に示すものであってはなりません。

- 順番探査入出力文
- 流れ探査入出力文
- 並びによる入出力文
- 変数群入出力文

- ファイル位置付け文
- CLOSE文

7.8.1.4 流れ探査入出力文との組合せ

現在の入出力文が流れ探査入出力文の場合、直前の入出力文が次に示すものであってはなりません。

- 順番探査入出力文
- 直接探査入出力文
- 並びによる入出力文
- 変数群入出力文
- ファイル位置付け文
- CLOSE 文

7.8.1.5 ファイル位置付け文との組合せ

順番探査入出力文は、ファイル位置付け文との組合せにより、入出力動作の対象となるファイルに対して位置付け制御を行うことができます。

現在の入出力文がファイル位置付け文の場合、直前の入出力文が次に示すものであってはなりません。各ファイル位置付け文について以下に示します。

現在の文がBACKSPACE文の場合

- 直接探査入出力文
- 書式なし流れ探査入出力文

現在の文がREWIND文の場合

- 直接探査入出力文

現在の文が ENDFILE文の場合

- 直接探査入出力文

7.9 ファイル位置付け文

順番探査入出力文は、ファイル位置付け文との組合せにより、入出力動作の対象となるファイルに対して位置付け制御を行うことができます。ここでは、ファイル位置付け文による位置付け制御について説明します。

ファイル位置付け文には、以下の3つがあります。

- BACKSPACE文(“7.9.1 BACKSPACE文”参照)
- ENDFILE文(“7.9.2 ENDFILE文”参照)
- REWIND文(“7.9.3 REWIND文”参照)

7.9.1 BACKSPACE文

BACKSPACE文を実行すると、指定された装置に接続しているファイルは、そのファイルに現在記録が存在する場合、現在の記録の前に位置付けられます。そのファイルに現在記録が存在しない場合、直前の記録の前に位置付けられます。そのファイルに現在記録も直前記録もない場合、ファイル位置は変わりません。BACKSPACE文が暗黙的にファイル終了記録を書いた場合には、ファイルは、ファイル終了記録の直前記録の前に位置付けられます。

直前の入出力文が順番探査READ文の場合

ファイルの位置付けを直前の Fortran記録になるように設定します。

例: READ文実行後のBACKSPACE文の動作

```
DIMENSION IDATA(5), NDATA(5)
   ...          !(1)のREAD文の実行の前は、Fortran記録2の直前に
   ...          !    位置付けられていたものとする
READ(1)IDATA    !(1)Fortran記録2の読込み
BACKSPACE 1     !(2)Fortran記録2の直前に位置付ける
READ(1)NDATA  !(3)Fortran記録 2の読込み
   ...
   ...
```

<ファイルの状況>

結果として、IDATAとNDATAには同一のデータが読み込まれたことになります。

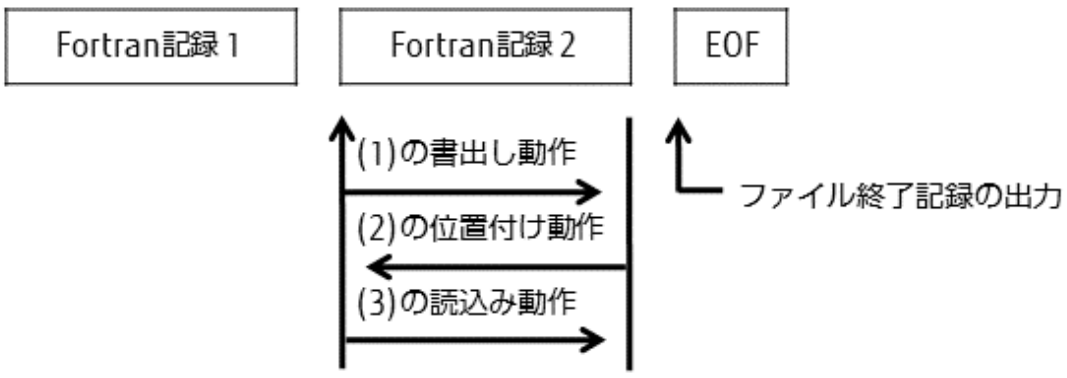
直前の入出力文が順番探査WRITE文の場合

ファイル終了記録を出力した後、ファイルの位置付けを直前のFortran記録となるように位置付けます。

例: WRITE文実行後のBACKSPACE文の動作

```
DIMENSION IDATA(5), NDATA(5)
   ...          !(1)のWRITE文の実行の前は、Fortran記録1の直後に
   ...          !    位置付けられていたものとする
WRITE(1)IDATA   !(1)Fortran記録2の書出し
BACKSPACE 1     !(2)ファイル終了記録を出力したあとで、Fortran記録2の
   ...          !    直前に位置付ける
READ(1)NDATA    !(3)Fortran記録2の読込み
   ...
   ...
```

<ファイルの状況>



結果として、IDATAで出力したデータがNDATAに読み込まれたことになります。

直前の入出力文がENDFILE文の場合

ファイル終了記録の直前に位置付けます。

7.9.2 ENDFILE文

ENDFILE文を実行すると、そのファイルの直後記録として、ファイル終了記録が出力され、ファイルは、そのファイルの最後の記録となるファイル終了記録の後ろに位置付けられます。

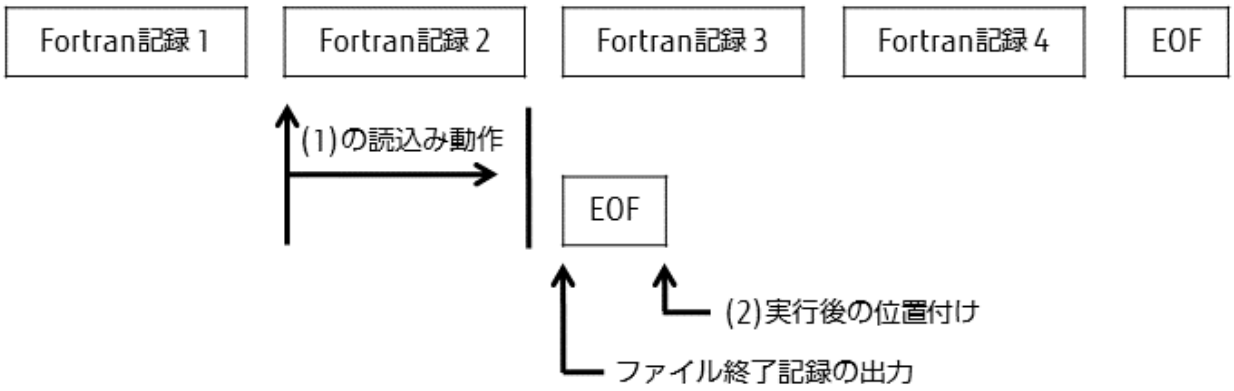
直前の入出力が READ文の場合

ENDFILE文は順番探査または流れ探査として接続したファイルの場合だけ許されます。

例: READ文実行後のENDFILE文の動作

```
DIMENSION IDATA(5)
   ...          !(1)のREAD文の実行の前は、Fortran記録2の直前に
   ...          !   位置付けられていたものとする
READ(1)IDATA    !(1)Fortran記録2の読込み後、このFortran記録2の
                !   最後に位置付ける
ENDFILE 1       !(2)Fortran記録3の先頭にEOFを書き出す
   ...
```

<ファイルの状況>



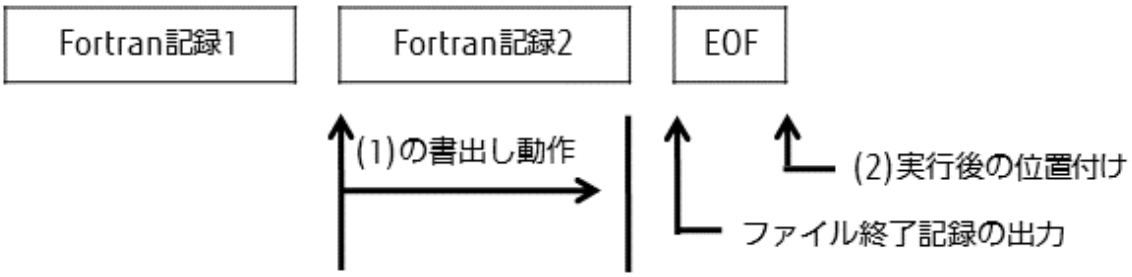
直前の入出力文がWRITE文の場合

現在のFortran記録の直後にファイル終了記録を出力し、ファイルの位置付けをファイル終了記録の直前になるように設定します。

例: WRITE文実行後のENDFILE文の動作

```
DIMENSION IDATA(5)
   ...          !(1)のWRITE文の実行前は、Fortran記録1の直後に
   ...          !   位置付けられていたものとする
WRITE(1)IDATA   !(1)Fortran記録2の書出し
ENDFILE 1       !(2)ファイル終了記録を出力し、ファイル終了記録の
                !   直後に位置付ける
   ...
   ...
```

<ファイルの状況>



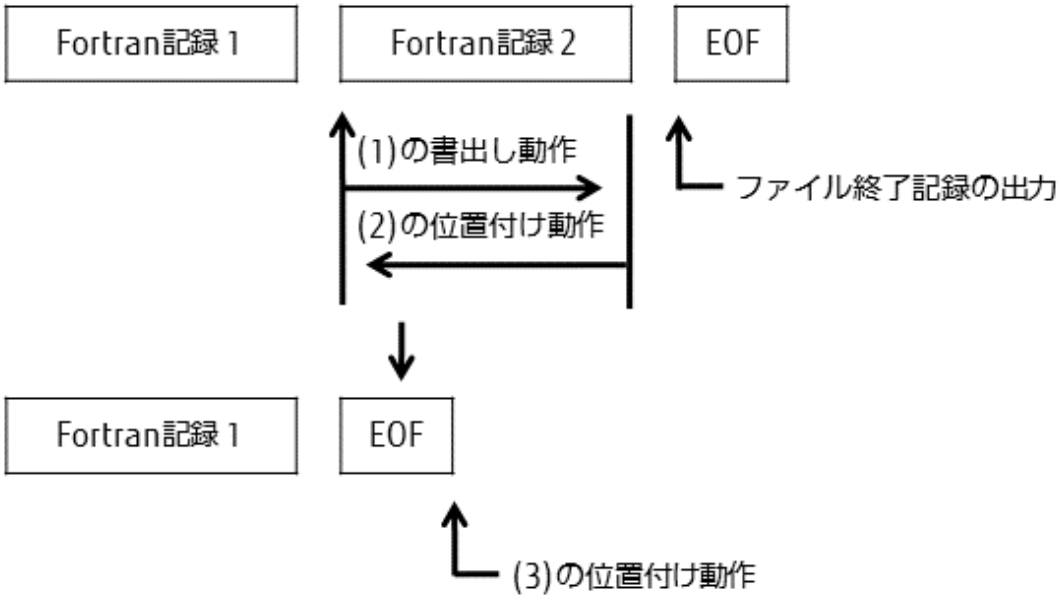
直前の文がBACKSPACE文の場合

現在のFortran記録の直後にファイル終了記録を出力します。

例: BACKSPACE文実行後のENDFILE文の動作

```
DIMENSION IDATA(5)
   ...          !(1)のWRITE文実行前は、Fortran記録1の直後に
   ...          !    位置付けられていたものとする
WRITE(1)IDATA   !(1)Fortran記録2の書出し
BACKSPACE 1     !(2)ファイル終了記録を出力した後で、
                !    Fortran記録2の直前に位置付ける
ENDFILE 1       !(3)Fortran記録2を消去したあとで、
                !    ファイル終了記録の直後に位置付ける
   ...
   ...
```

<ファイルの状況>



直前の入出力文がREWIND文の場合

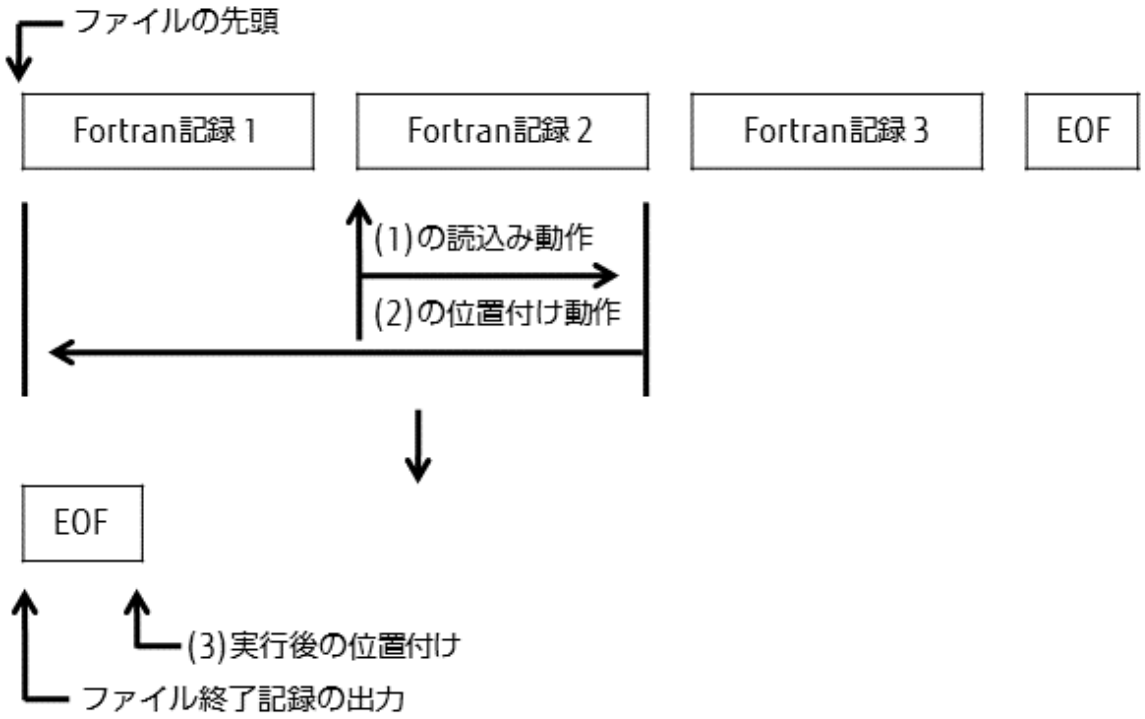
ファイルの先頭にファイル終了記録を出力し、ファイルの位置付けをファイル終了記録の直後になるように設定します。

例: REWIND文実行後のENDFILE文の動作

```
DIMENSION IDATA(5)
   ...          !(1)のREAD文実行前は、Fortran記録2の直前に
   ...          !    位置付けられていたものとする
READ(1)IDATA    !(1)Fortran記録2の直後に位置付ける
```

```
REWIND 1        !(2)ファイルの先頭(最初のFortran記録1の直前)に位置付ける
ENDFILE 1       !(3)すべてのFortran記録を消去したあと、
                !    ファイル終了記録を出力し、ファイル終了記録の直後に位置付ける
  ...
  ...
```

<ファイルの状況>



結果として、ファイル終了記録だけが存在するファイルが作られる。

7.9.3 REWIND文

ファイルが順番探査または流れ探査として接続されている場合、ファイルの位置付けをファイルの先頭のFortran記録の直前になるように設定します。

7.10 FLUSH文

FLUSH文を実行すると、外部ファイルに書き込まれたデータを他の手続で利用したり、Fortran以外の方法で外部ファイルに置かれたデータをREAD文で利用することができます。接続されているが存在していないファイルに対してFLUSH文を実行した場合、または、直前の入出力文がWRITE文でない場合、FLUSH文は無操作となります。

FLUSH文で指定できる指定子は、UNIT指定子、IOSTAT指定子、ERR指定子です。これらの指定子については“7.7.1 入出力文の制御情報”を参照してください。

7.11 エンディアン入出力変換

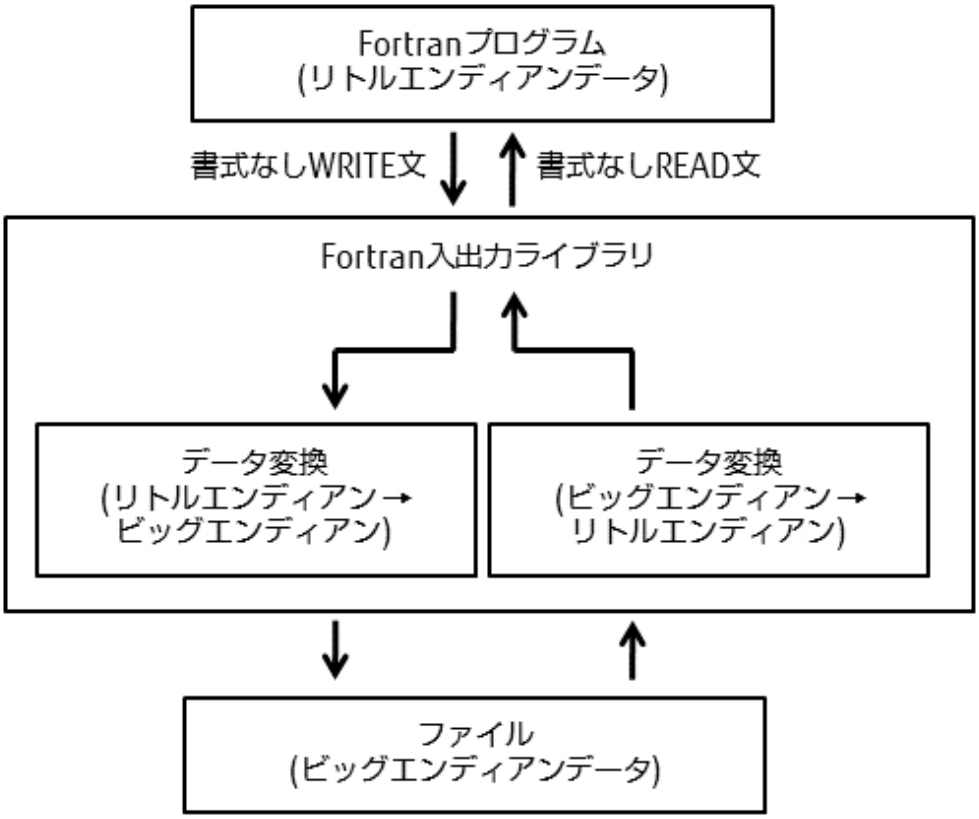
ここでは、ビッグエンディアンデータとリトルエンディアンデータの変換について説明します。

7.11.1 ビッグエンディアンデータと入出力文との関係

実行時オプション-Tを指定するか、またはOPEN文のCONVERT指定子に'BIG_ENDIAN'を指定することにより、ビッグエンディアンデータを入出力することができます。実行時オプションについては、“3.3 実行時オプション”を参照してください。

- 書式なしREAD文で、ビッグエンディアンデータで作成されている書式なしFortran記録を入力し、リトルエンディアンデータに変換して、入力並びに設定することができます。
- 書式なしWRITE文で、出力並びに設定されているリトルエンディアンデータをビッグエンディアンデータに変換して、書式なしFortran記録へ出力することができます。

以下はビッグエンディアンデータの流れを示しています。



対象となる入出力文は、以下の3つです。

- 書式なし順番探査入出力文
- 書式なし直接探査入出力文
- 書式なし流れ探査入出力文

対象となるFortran記録は、富士通のFortran処理系で出力した書式なしFortran記録です。

書式なしFortran記録については、“7.3.2 書式なしFortran記録”を参照してください。

変換対象となる入出力並びの型は、以下の型です。

- 1バイトの整数型
- 2バイトの整数型
- 4バイトの整数型
- 8バイトの整数型
- 実数型
- 倍精度実数型
- 4倍精度実数型
- 複素数型
- 倍精度複素数型

- 4倍精度複素数型
- 1バイトの論理型
- 2バイトの論理型
- 4バイトの論理型
- 8バイトの論理型

7.12 標準ファイル

ここでは、入出力文で使用する標準ファイルについて説明します。

7.12.1 標準ファイルとオープンモード

標準ファイルとオープンモードの関係を以下に示します。

入出力ファイル	指定方法(指定がないと端末)	オープンモード
標準入力ファイル	command<file-name	r
	command<<eof-sym	r
標準出力ファイル	command>file-name	w
	command>>file-name	a
標準エラー出力ファイル	command2>file-name	w
	command2>>file-name	a

7.12.2 標準ファイルに対する入出力文の制限

標準入力ファイル、標準出力ファイルおよび標準エラー出力ファイルに対しては、順番探査入出力文、補助入出力文のみ扱うことができます。

標準入力ファイルは、ACTION='READ'の扱いです。また、標準出力ファイルおよび標準エラー出力ファイルは、ACTION='WRITE'の扱いです。

標準入力ファイル、標準出力ファイルおよび標準エラー出力ファイルに対して、FILE指定子を指定したOPEN文を実行すると、今までの接続は解除され、新たにファイル(一般ファイルまたは一時ファイル)がオープンされます。これ以降は、標準入力、標準出力および標準エラー出力の装置番号に対する扱いは、通常のファイルとして扱われます。

7.12.3 入出力文とシーク可能／不可能ファイルの関係

シーク可能ファイルとは、シーク動作可能なファイルのことを示し、以下のように区別します。

シーク可/不可	ファイルタイプ
シーク可能ファイル	通常ファイル
シーク不可能ファイル	ブロック型特殊ファイル
	文字型特殊ファイル
	FIFO特殊ファイル

備考. ファイルタイプとは、statおよびfstatシステムコールで得られる情報です。

次に示す入出力文および編集記述子は、シーク可能ファイルでなければなりません。シーク不可能ファイルに対して実行すると、診断メッセージが出力され、文は無視されます。

- REWIND文
- BACKSPACE文

- ENDFILE文(INPUT系)
- 直接探査入出力文
- 流れ探査入出力文
- 書式なし順番探査入出力文
- TL形編集記述子およびT形編集記述子(結果的に、TL形と同じ動作をする場合のみ)

7.13 ACTION指定子による入出力文の制限

ACTION指定子で指定された値に対して、使用できる入出力文に制限があります。

入出力文	ACTION指定子		
	READ	WRITE	READWRITE
READ文	○	×	○
WRITE文	×	○	○
PRINT文	×	○	○
REWIND文	○	×	○
BACKSPACE文	○	×	○
OPEN文	○	○	○
CLOSE文	○	○	○
ENDFILE文	－	○	○

○: 正常に動作することを示します。

×: エラーメッセージを出力し、何らかの修正処理を行うことを示します。

－: 無操作であることを示します。

7.14 実数値の有効けた数

本処理系における実数値の有効けた数を以下に示します。この有効けた数を超えて、書式付き入出力文でデータ転送を行った場合、誤差が生じることがあります。

入出力並びの型	有効けた数
実数型	9
倍精度実数型	16
4倍精度実数型	35
複素数型	9
倍精度複素数型	16
4倍精度複素数型	35

7.15 入出力バッファ

順番探査入出力および流れ探査入出力実行時に使用される入出力バッファを変更する方法を、以下に示します。これらを同時に指定した場合、その優先順位は、OPEN文のBLOCKSIZE指定子、環境変数、実行時オプションの順番です。

指定	バッファサイズ	優先順位
OPEN文のBLOCKSIZE指定子	指定した値	最も高い

指定	バッファサイズ	優先順位
環境変数(fuxxbf)	指定した値	↑
実行時オプション(-g)	指定した値	↓
なし	8Mバイト	最も低い

7.16 編集記述子

ここでは、編集記述子について説明します。

7.16.1 整数型のG形編集

Fortran 66およびFortran 77言語仕様では、整数型の入出力並びに対応する編集記述子がG*w.d*形またはG*w.d*E*e*形の場合、I*w.m*形編集として入出力されます。Fortran 95以降の言語仕様では、I*w*形編集として入出力されます。

例:

```
WRITE(*,'(G10.5)')123
```

上記のWRITE文を実行すると、Fortran 66およびFortran 77言語仕様では、以下のように出力されます。

```
□□□□□00123(□は空白を表します)
```

Fortran 95以降の言語仕様では、以下のように出力されます。

```
□□□□□□□123(□は空白を表します)
```

7.16.2 書式付き出力文の文字出力

書式付き出力文のE形編集、EN形編集、ES形編集、D形編集、Q形編集、G形編集、L形編集およびZ形編集で出力した場合、文字E、D、Qの出力は言語レベルにより異なります。Fortran 66およびFortran 77言語仕様では、小文字で出力されます。ただし、実行時オプション-qを指定すると、大文字で出力することができます。Fortran 95以降の言語仕様では、実行時オプション-qの指定に関係なく大文字で出力されます。

例:

```
WRITE(*,FMT='(1H E15.5)')6.5432
```

上記のFortranプログラムを実行すると、Fortran 66およびFortran 77言語仕様では、0.65432e+01と出力されます。Fortran 95以降の言語仕様では、0.65432E+01と出力されます。

7.16.3 G形編集記述子の編集結果

値0または値が0の変数をG形編集で編集した場合、Fortran 66およびFortran 77言語仕様では、E形編集記述子で編集されます。

Fortran 95以降の言語仕様では、F形編集記述子で編集されます。

7.16.4 文字列編集記述子の診断メッセージ

Fortran 66およびFortran 77言語仕様では、実行時にwレベルの診断メッセージ(jwe0159i-w)が出力され、システムの処理として、入力データが書式中の文字列に置き換えられます。

Fortran 95以降の言語仕様では、実行時にeレベルの診断メッセージ(jwe1181i-e)が出力され、システムの処理として、入出力並びが無視されます。

7.16.5 X形編集記述子の効果

X形編集記述子(*n*X)が、出力における書式仕様として使用された場合には、Fortran 77以降のプログラムとFortran 66プログラムでは、その動作が異なります。

Fortran 77以降の言語仕様では、Fortran記録の文字の位置を*n*文字分だけ進んだ位置に設定するだけで、空白は挿入されません。

Fortran 66言語仕様では、*n*個の空白がFortran記録に挿入されます。

次のような場合にだけその動作が異なります。

出力の書式仕様が次の形式の場合に、

```
FORMAT(5H FFFF,I4,T2,4X,I2)
```

Fortran 77以降のプログラムでは、　　□Ｆ Ｆ Ｆ Ｆ ｊ ｊ ｉ ｉ

Fortran 66プログラムでは、　　□□□□□ｊ ｊ ｉ ｉ

となります(iiはI4に対応する変数の数値の下位2けた、jjはI2に対応する変数の数値、□は空白とします)。

ただし、Fortran 77以降のプログラムにおいても、最初に用意されるFortran記録は、空白で埋められているので、一般的な使用法では問題ありません。

7.16.6 L形編集記述子

L*w*形編集記述子に対応する入力の型が論理定数でない場合、Fortran 66およびFortran 77言語仕様では.FALSE.を設定し、正常終了します。Fortran 95以降の言語仕様では、実行時の診断メッセージ(jwe1202i-w)が出力されます。

例:

プログラム l.f

```
LOGICAL :: L=.TRUE.
OPEN (10,FILE="a.file")
READ (10,"(L1)") L
PRINT *,L
END
```

ファイル a.file

```
1
```

実行結果

Fortran 66およびFortran 77言語仕様の場合

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

Fortran 95以降の言語仕様の場合

```
$ ./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、およびZ編集記述子のw、dの省略値

I、L、F、E、D、G、B、O、およびZ編集記述子において、欄幅*w*、小数部のけた数*d*を省略することができます。

*w*を省略した場合、以下のけた数が補われて、編集されます。

編集記述子	変数の型													
	I1	I2	I4	I8	R4	R8	R16	C8	C16	C32	L1	L2	L4	L8
I形編集	4	6	11	20	11	11	11	11	11	11	4	6	11	20
L形編集	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F形編集	15	15	15	22	15	22	43	15	22	43	15	15	15	22
E形編集	15	15	15	22	15	22	43	15	22	43	15	15	15	22
D形編集	15	15	15	22	15	22	43	15	22	43	15	15	15	22
G形編集	4	6	11	20	15	22	43	15	22	43	2	2	2	2
B形編集	9	17	33	65	33	65	129	33	65	129	9	17	33	65
O形編集	4	7	12	23	12	23	44	12	23	44	4	7	12	23
Z形編集	3	5	9	17	9	17	33	9	17	33	3	5	9	17

I1 :1バイトの整数型
I2 :2バイトの整数型
I4 :4バイトの整数型
I8 :8バイトの整数型
R4 :実数型
R8 :倍精度実数型
R16 :4倍精度実数型
C8 :複素数型
C16 :倍精度複素数型
C32 :4倍精度複素数型
L1 :1バイトの論理型
L2 :2バイトの論理型
L4 :4バイトの論理型
L8 :8バイトの論理型

7.16.8 実数型特殊データの編集

E形編集記述子、D形編集記述子、Q形編集記述子、G形編集記述子、F形編集記述子、EN形編集記述子、およびES形編集記述子のフォーマットを、*Xw.d*(*X*: E、D、Q、G、F、EN、ESのどれか)とした場合、Fortran 2003以降の言語仕様においては、出力幅wけたに対して右詰めでInf、NaNを出力し、残ったけたを空白で埋めます。

Fortran95以前の言語仕様においては、左詰めでInf、NaNを出力し、残ったけたを空白で埋めます。負の値の場合は、-Inf、-NaNを出力します。

また、S形編集記述子、SP形編集記述子、またはSS形編集記述子により符号を制御することもできます。

文字列(Inf、-Inf、Infinity、-Infinity、NaN、-NaN、NaN()、-NaN())は、実数型の変数の入力データとしても使用できます。

NaN()の括弧内はゼロ個以上の英数字です。また、符号“+”は、付けても付けなくてもかまいません。

以下に、それぞれの文字列に対応した、実際に変数に設定される値を示します。

文字列	単精度	倍精度	4倍精度
+NaN または	0x7fffffff	0x7fffffff ffffffff	0x7fffffffffffffff ffffffffffffffff

文字列	単精度	倍精度	4倍精度
+NaN()			
-NaN または -NaN()	0xffffffff	0xffffffff ffffffff	0xffffffffffffffff ffffffffffffffff
+Inf または +Infinity	0x7f800000	0x7ff00000 00000000	0x7fff000000000000 0000000000000000
-Inf または -Infinity	0xff800000	0xfff00000 00000000	0xffff000000000000 0000000000000000

7.17 非同期入出力文

ここでは、非同期入出力文について説明します。

7.17.1 非同期入出力文の実行

OPEN文に値がYESであるASYNCHRONOUS 指定子を指定して接続したファイルに対して、値がYES であるASYNCHRONOUS 指定子を指定した入出力文は、非同期入出力文として実行されます。

非同期入出力文は、システム内で非同期転送が可能なデータ転送を非同期化して処理されます。

ただし、書式なし順番探査以外の入出力文であるか、以下の翻訳時オプションまたは実行時オプションを指定している場合には、データ転送の非同期化がされません。

- 翻訳時オプション -H (詳細は“2.2 翻訳時オプション”を参照してください)
- 翻訳時オプション -Eg(詳細は“2.2 翻訳時オプション”を参照してください)
- 実行時オプション -G (詳細は“3.3 実行時オプション”を参照してください)
- 実行時オプション -C (詳細は“3.3 実行時オプション”を参照してください)
- 実行時オプション -T (詳細は“3.3 実行時オプション”を参照してください)

また、入出力並びに指定された項目の数、大きさ、種類、型、または属性によっても、データ転送の非同期化がされない場合があります。

7.17.2 データ転送入出力文のID指定子

WRITE文またはREAD文のID指定子には、プログラム中で実行される非同期入出力文を一意に区別する0以上9223372036854775808未満の整数値が代入されます。ID指定子に代入される値が、指定子に代入可能な値の範囲を超えている場合には、その値は保証されません。

7.18 利用者定義の派生型入出力

7.18.1 利用者定義の入出力手続

- 親データ転送文に内部ファイルを指定した場合、利用者定義の派生型入出力手続には、装置番号の値として-1が渡されます。
- 親データ転送文の文字書式仕様にDT編集記述子の値指定がある場合、値指定に指定可能な値は、-2147483648～2147483647です。
- 利用者定義の派生型入出力手続で、入出力に関するエラーが発生した場合、親データ転送文は終了します。入出力に関するエラーについては、“8.1.4 入出力エラーの処理”を参照してください。
- 子データ転送文には、ASYNCHRONOUS='YES'は指定できません。

・ 利用者定義派生型入出力手続では、以下の入出力文およびサービスルーチンを実行してはいけません。

  － OPEN文

  － CLOSE文

  － BACKSPACE文

  － ENDFILE文

  － REWIND文

  － WAIT文

  － FGETCサービス関数

  － FPUTCサービス関数

  － FSEEKサービス関数

  － FSEEKO64サービス関数

  － FTELLサービス関数

  － FTELLO64サービス関数

  － GETCサービス関数

  － PERRORサービスサブルーチン

  － PUTCサービス関数

・ 親データ転送文の装置番号が外部装置と接続しているとき、利用者定義の派生型入出力手続で、FLUSH文またはFLUSHサービスサブルーチンを実行した場合、無操作となります。

・ 親データ転送文の装置番号が標準出力エラーファイルと接続しているとき、利用者定義の派生型入出力手続の呼出し中に、実行時のエラーメッセージが出力された場合、転送した順番にデータが出力されないことがあります。

・ 親データ転送文と子データ転送文が同じ内部ファイルを参照している場合、子データ転送文が出力したデータは内部ファイルに転送されないことがあります。

・ 親データ転送文と子データ転送文が内部ファイルを参照し、子データ転送文をLLVM OpenMPにより並列化している場合、実行が終了しないなど、正しく実行されないことがあります。

7.18.2 データ転送入出力項目並び

・ 派生型の並び項目が多相的である、または、割付け可能な末端成分またはポインタである末端成分をもつ場合、その並び項目は利用者定義の派生型入出力手続で処理しなければなりません。親データ入出力文に書式仕様の指定がある場合、これらの並び項目に対応する編集記述子はDT形編集記述子でなければなりません。

7.18.3 変数群入出力

・ 親データ転送文または子データ転送文が変数群入力である場合、変数群入力データにおいて、変数群要素名に部分配列は指定できません。

・ 変数群要素が多相的である、または、割付け可能な末端成分またはポインタである末端成分をもつ場合、その実体は利用者定義の派生型入出力手続で処理しなければなりません。

7.18.4 INQUIRE文

・ 親データ転送文が内部ファイルを指定した場合、利用者定義の派生型入出力手続に渡された装置番号に対して、INQUIRE文を実行すると、エラーが発生します。IOSTAT指定子が指定されている場合、1124が設定されます。

第8章 プログラムのデバッグ

実行可能プログラムでエラーが発生したとき、異常終了したとき、または利用者が目的とした結果を得られないときは、その原因を追求し、原始プログラムの修正を行う必要があります。

本章では、これらの原因を見い出す手段として、本処理系が用意している各種のデバッグ機能について説明します。

8.1 エラー制御

Fortranプログラムの実行中にエラーが発生した場合、次のような処置が行われます。

- 診断メッセージが出力されます。
- エラーの種類により実行が継続されます。

これらのエラー発生時の処置は、エラーモニタによって制御されます。

利用者は、実行時にエラーモニタの機能を使って、個々のエラーに対する処置の制御を自分で行うことができます。その処置の制御を、次に示します。

- 実行可能プログラムの実行を打ち切る回数の制御
- エラーに対するメッセージの最大出力回数の制御
- トレースバックマップを出力するかどうかの制御
- 利用者が用意する(または定義する)エラー処理ルーチンを呼ぶかどうかの制御

利用者が、これらの制御を必要としない場合、本処理系で用意されている標準値(“表8.1 エラー制御表の標準値”参照)が使用されます。

エラーモニタは、LLVM OpenMPライブラリのエラーに対しては無効となります。LLVM OpenMPライブラリのエラーについては、“12.4 実行時メッセージ”を参照してください。

8.1.1 エラーモニタ

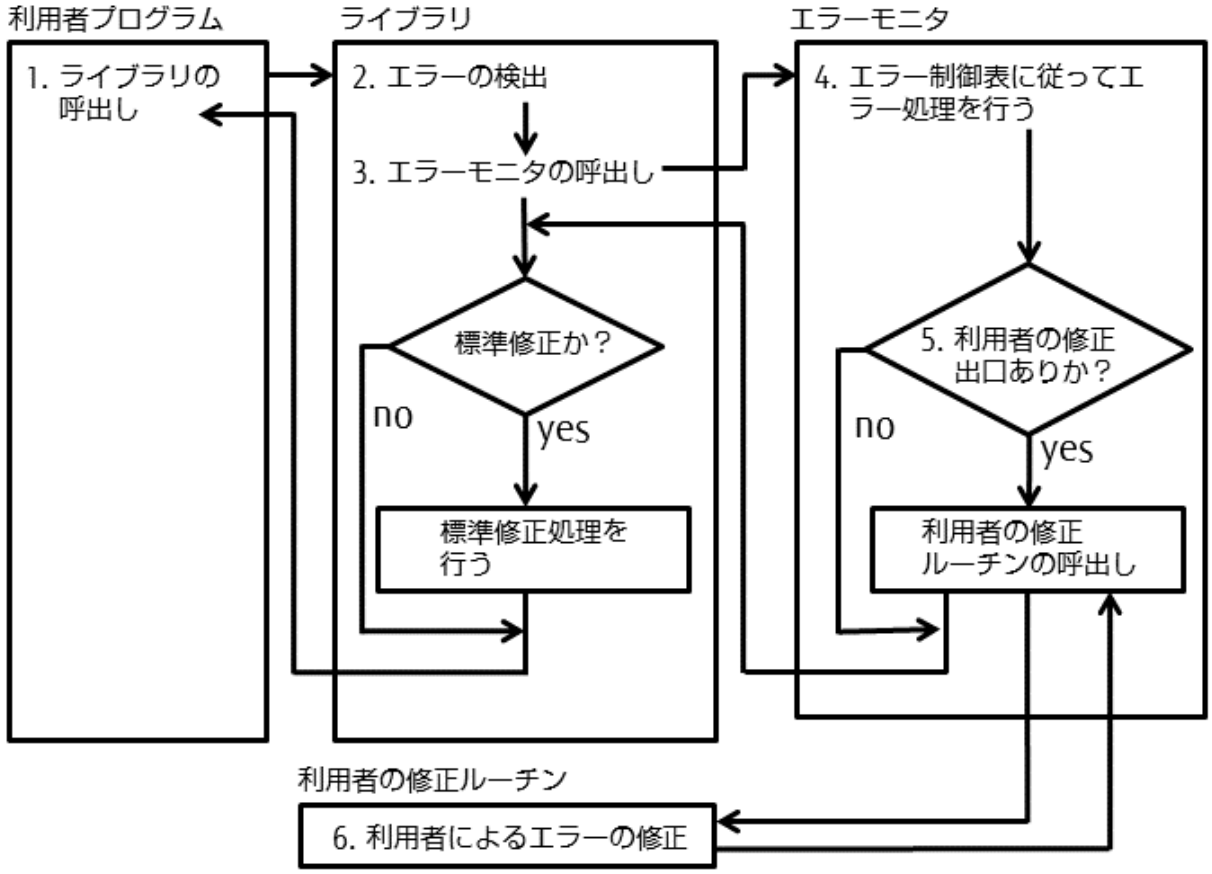
Fortranプログラムの実行中にエラーが発生した場合、エラーモニタが呼び出されます。エラーモニタは、発生したエラーに対応するエラー制御表内のエラー項目の情報によって、各種のエラー処理を行います。

エラーモニタは次のことを行います。

- エラーの発生が打切り回数に達しているか調べ、打切り回数に達していればその旨を通知します。
- エラーに対するメッセージ出力回数を調べ、最大出力回数を超えていなければ診断メッセージを出力します。
- トレースバックマップの出力の指定があればその出力を行います。ただし、他言語のプログラムと結合されているときは、出力されないトレースバックマップ情報があります。
- エラーの発生が打切り回数に達した場合、実行可能プログラムの終了処理を行います。打切り回数に達していない場合は、エラーの修正処置を行ったあとに実行を継続させます。

エラーの修正処置には、本処理系で用意されている標準修正処理と利用者が用意する修正処理があります。利用者が修正処理を用意する場合、その処置をするプログラムを作成しなければなりません。エラーが発生してからのエラーモニタの呼出し関係を以下に示します。

図8.1 エラーモニタの呼出し関係

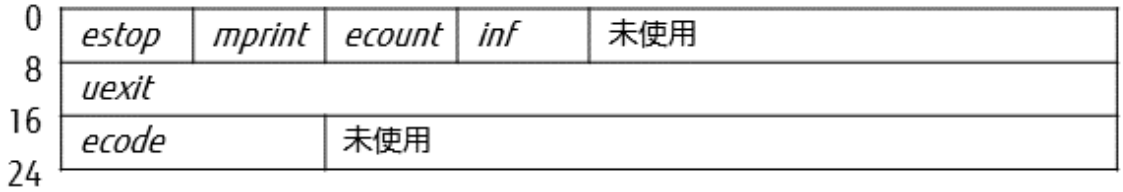


1～6：ライブラリでエラーを検出したときの呼出し関係

エラーモニタの動作は、エラー制御表内のエラー項目によって制御されます。エラー制御表は、エラー識別番号に対応するエラー項目の集まりからなっています。

エラー項目の構成は、以下のようになっています。これらの項目は、エラー処理ルーチンで参照および設定することができます。

図8.2 エラー項目の構成



estop

エラー打切り回数です。0～255の値が設定されています。0は、無制限を示します。

mprint

メッセージの最大出力回数です。0～255の値が設定されています。0は出力しないことを示します。

ecount

プログラムの実行中に発生したエラーの回数が記録されます。0～255の値が設定されています。

inf

ビットでエラーの処理のための制御情報を示します。

ビット0

診断メッセージの先頭に制御文字が与えられているとき1、与えられていないとき0(“8.1.2.3 ERRSETサービスサブルーチン”参照)。

ビット1

利用者がこのエラー項目を修正可能なとき1、修正不可能なとき0。

ビット2

*ecount*が255を超えたとき1、255以下のとき0。

ビット3

診断メッセージとともにバッファの内容を出力するとき1、バッファの内容を出力しないとき0。

ビット4

未使用。

ビット5

メッセージを出力するとき1、メッセージを*mprint*の値によって出力するとき0。

ビット6

トレースバックマップを出力するとき1、出力しないとき0。

ビット7

未使用。

uexit

利用者が用意した修正ルーチンの番地が格納されます。標準修正の場合、最後のビットが1となります。

ecode

エラーのレベルを示します。

= 0: iレベルのエラーを示します。

メッセージを出力したあとにプログラムの実行を続けます。

= 4: wレベルのエラーを示します。

エラーの発生した文を一部修正したあとにプログラムの実行を続けます。

= 8: eレベルのエラーを示します。

エラーの発生した文を無視します。実行は打切り回数(標準は10回)によって、続行または中断します。

= 12: sレベルのエラーを示します。

エラーの発生した文を無視します。実行は打切り回数(標準は1回)によって、続行または中断します。

= 16: uレベルのエラーを示します。

プログラムの実行を打ち切ります。

エラー制御表内の各エラー項目は、エラー項目の修正が不可能なものを除いて、実行時にエラー処理サービスサブルーチンERRSETを使用して、その内容を変えることが可能です。

エラー識別番号に対応するエラー項目の標準値は、本処理系で、下表に示すように定められています。

表8.1 エラー制御表の標準値

番号	エラー打切り回数	メッセージ最大出力回数	エラー項目の修正	バッファの内容出力	トレースバックマップの出力	標準修正	エラーレベル
11～14	1	1	不可能	出力しません	出力しません	なし	u
17～18	1	1	不可能	出力しません	出力しません	なし	u
19	1	1	不可能	出力しません	出力します	なし	u
20	1	1	不可能	出力しません	出力しません	なし	u

番号	エラー打切り回数	メッセージ最大出力回数	エラー項目の修正	バッファの内容出力	トレースバックマップの出力	標準修正	エラーレベル
21～22	1	1	可能	出力しません	出力します	あり	s
28	1	1	可能	出力しません	出力します	あり	s
42	10	5	可能	出力しません	出力します	あり	e
55	10	5	可能	出力しません	出力します	あり	e
62	10	5	可能	出力しません	出力します	あり	e
64～65	10	5	可能	出力しません	出力します	あり	e
71～72	10	5	可能	出力しません	出力します	あり	e
74	10	5	可能	出力しません	出力します	あり	e
80～82	10	5	可能	出力しません	出力します	あり	e
83～86	無制限	5	可能	出力しません	出力します	あり	w
87～89	10	5	可能	出力しません	出力します	あり	e
97	無制限	5	可能	出力しません	出力します	あり	w
102～103	10	5	可能	出力しません	出力します	あり	e
104	無制限	5	可能	出力しません	出力します	あり	w
105	10	5	可能	出力しません	出力します	あり	e
109	無制限	5	可能	出力しません	出力します	あり	w
111～115	10	5	可能	出力しません	出力します	あり	e
120	10	5	可能	出力しません	出力します	あり	e
122	10	5	可能	出力しません	出力します	あり	e
131～134	10	5	可能	出力しません	出力します	あり	e
151	10	5	可能	出力しません	出力します	あり	e
152	無制限	1	可能	出力しません	出力しません	あり	w
153～154	無制限	5	可能	出力しません	出力します	あり	w
155～157	10	5	可能	出力しません	出力します	あり	e
158～159	無制限	1	可能	出力しません	出力しません	あり	w
162	無制限	5	可能	出力しません	出力します	あり	w
171～172	10	5	可能	出力しません	出力します	あり	e
173～175	無制限	5	可能	出力します	出力します	あり	w
176	10	5	可能	出力しません	出力します	あり	e
182～183	無制限	1	可能	出力しません	出力しません	あり	w
184～189	10	5	可能	出力します	出力します	あり	e
190	無制限	1	可能	出力しません	出力しません	あり	w
202～255	10	5	可能	出力しません	出力します	あり	e
256	無制限	1	可能	出力しません	出力します	あり	w
257	10	5	可能	出力しません	出力します	あり	s
259～282	10	5	可能	出力しません	出力します	あり	e
291	1	1	不可能	出力しません	出力しません	なし	u
292	1	1	不可能	出力しません	出力しません	なし	u

番号	エラー打切り回数	メッセージ最大出力回数	エラー項目の修正	バッファの内容出力	トレースバックマップの出力	標準修正	エラーレベル
301	10	5	可能	出力しません	出力します	あり	w
304	無制限	5	可能	出力しません	出力します	あり	w
306	1	1	不可能	出力しません	出力しません	なし	u
308	1	1	可能	出力しません	出力します	あり	s
311～318	無制限	無制限	可能	出力しません	出力します	あり	w
320	10	5	可能	出力しません	出力します	あり	w
322	10	5	可能	出力しません	出力します	あり	w
323～324	無制限	5	可能	出力しません	出力します	あり	w
329	1	1	可能	出力しません	出力します	あり	s
330	無制限	無制限	可能	出力しません	出力します	あり	w
1001	1	1	可能	出力しません	出力します	あり	s
1003～1008	1	1	可能	出力しません	出力します	あり	s
1017	1	1	不可能	出力しません	出力しません	なし	u
1031	1	1	可能	出力しません	出力します	あり	s
1032	無制限	5	可能	出力しません	出力しません	あり	w
1033	無制限	5	可能	出力しません	出力します	あり	i
1034	無制限	5	可能	出力しません	出力します	あり	w
1035	1	1	不可能	出力しません	出力します	なし	u
1036	無制限	5	可能	出力しません	出力します	あり	w
1038	1	1	可能	出力しません	出力します	あり	s
1039	1	1	不可能	出力しません	出力します	なし	u
1040～1041	1	1	可能	出力しません	出力します	あり	s
1042	無制限	5	可能	出力しません	出力しません	あり	w
1043	無制限	5	可能	出力しません	出力しません	あり	i
1044～1046	1	1	不可能	出力しません	出力します	なし	u
1047	無制限	5	可能	出力しません	出力しません	あり	w
1048	1	1	不可能	出力しません	出力しません	なし	u
1049	無制限	5	可能	出力しません	出力しません	あり	w
1050	無制限	5	可能	出力しません	出力しません	あり	w
1051	無制限	5	可能	出力しません	出力しません	あり	w
1052	1	1	不可能	出力しません	出力します	なし	u
1071～1072	10	5	可能	出力しません	出力しません	あり	e
1073	10	5	可能	出力しません	出力します	あり	e
1111～1113	10	5	可能	出力しません	出力します	あり	e
1114～1115	無制限	5	可能	出力しません	出力します	あり	w
1117～1127	10	5	可能	出力しません	出力します	あり	e
1141～1143	10	5	可能	出力しません	出力します	あり	e
1146	10	5	可能	出力しません	出力します	あり	e

番号	エラー打切り回数	メッセージ最大出力回数	エラー項目の修正	バッファの内容出力	トレースバックマップの出力	標準修正	エラーレベル
1147	10	5	可能	出力しません	出力します	あり	e
1148	10	5	可能	出力しません	出力します	あり	e
1149	無制限	5	可能	出力しません	出力します	あり	w
1150～1151	10	5	可能	出力しません	出力します	あり	e
1152～1153	1	1	可能	出力しません	出力します	あり	s
1161	1	1	可能	出力しません	出力します	あり	s
1162～1163	無制限	5	可能	出力しません	出力します	あり	w
1181～1184	10	5	可能	出力しません	出力します	あり	e
1201～1203	無制限	5	可能	出力します	出力します	あり	w
1204	10	5	可能	出力しません	出力します	あり	e
1231	無制限	5	可能	出力しません	出力します	あり	w
1232	1	1	可能	出力しません	出力します	あり	s
1301	10	5	可能	出力しません	出力します	あり	e
1302	無制限	5	可能	出力しません	出力します	あり	i
1303～1304	10	5	可能	出力しません	出力します	あり	e
1305～1307	1	1	可能	出力しません	出力します	あり	s
1355～1370	10	5	可能	出力しません	出力します	あり	e
1371	無制限	5	可能	出力しません	出力します	あり	w
1372	10	5	可能	出力しません	出力します	あり	e
1373	無制限	5	可能	出力しません	出力します	あり	w
1375～1378	1	1	可能	出力しません	出力します	あり	s
1381～1387	1	1	可能	出力しません	出力します	あり	s
1390～1392	1	1	可能	出力しません	出力します	あり	s
1393	10	5	可能	出力しません	出力します	あり	e
1394～1395	1	1	可能	出力しません	出力します	あり	s
1396～1414	10	5	可能	出力しません	出力します	あり	e
1415	1	1	可能	出力しません	出力します	あり	s
1416～1417	10	5	可能	出力しません	出力します	あり	e
1418～1419	1	1	可能	出力しません	出力します	あり	s
1420	無制限	5	可能	出力しません	出力します	あり	w
1421	10	5	可能	出力しません	出力します	あり	e
1422～1423	無制限	5	可能	出力しません	出力します	あり	w
1425～1454	10	5	可能	出力しません	出力します	あり	e
1455	無制限	5	可能	出力しません	出力します	あり	w
1456～1461	10	5	可能	出力しません	出力します	あり	e
1462	1	1	不可能	出力しません	出力します	あり	s
1463～1474	10	5	可能	出力しません	出力します	あり	e
1475	1	1	不可能	出力しません	出力します	あり	s

番号	エラー打切り回数	メッセージ最大出力回数	エラー項目の修正	バッファの内容出力	トレースバックマップの出力	標準修正	エラーレベル
1501	1	1	不可能	出力しません	出力します	なし	u
1551	10	5	可能	出力しません	出力します	あり	e
1561	無制限	無制限	可能	出力しません	出力します	あり	w
1563～1564	1	1	不可能	出力しません	出力しません	なし	u
1565～1566	無制限	無制限	可能	出力しません	出力します	あり	w
1569～1571	無制限	無制限	可能	出力しません	出力します	あり	w
1572～1575	1	1	可能	出力しません	出力します	あり	s
1576～1577	無制限	5	可能	出力しません	出力します	あり	w
1578	1	1	可能	出力しません	出力します	あり	s
1579	無制限	5	可能	出力しません	出力します	あり	w
1582	無制限	5	可能	出力しません	出力します	あり	i
1606	1	1	不可能	出力しません	出力します	なし	u
1652	無制限	無制限	可能	出力しません	出力しません	あり	w
1653～1654	無制限	1	不可能	出力しません	出力します	なし	w
1655	無制限	1	不可能	出力しません	出力しません	なし	i

エラーモニタは、エラーが発生したときの修正状況を、次の形式でトレースバックマップのあとに出力します。

```
taken to (corrector) corrective action, execution continuing.
```

corrector: { standard | user }

standard: 標準修正されたことを示します。

user: 利用者によって修正されたことを示します。

8.1.2 エラー処理サービスサブルーチン

実行時のエラー発生に対して、その処理を利用者が制御するためにいくつかのサービスサブルーチンが用意されています。

利用者は、これらのサービスサブルーチンを使ってエラーモニタの機能を最大限に活用することができます。

これらのサービスサブルーチンは、エラー制御表内のエラー項目をプログラム実行中の間だけ、動的に制御したり、トレースバックマップを出力したり、利用者が用意したエラー処理を実行したりするものです。

8.1.2.1 ERRSAVサービスサブルーチン

ERRSAVサービスサブルーチンは、指定されたエラー項目の先頭16バイトを、利用者が用意した16バイトの領域に退避させたいときに使用します。

呼出し形式は、以下の形式です。

CALL ERRSAV(*errno* , *darea*)

errno

基本整数型スカラ。エラー制御表内のエラー項目に対応するエラー識別番号。

darea

16バイトの文字型スカラ。指定されたエラー項目の退避領域。*darea* の型が文字型でない場合、結果は不定となります。

8.1.2.2 ERRSTRサービスサブルーチン

ERRSTRサービスサブルーチンは、利用者が用意した領域にあるエラー項目を、指定されたエラー識別番号に対応するエラー制御表内のエラー項目に格納するものです。これによって、そのエラーが発生した場合、新しいエラー項目の情報により、エラーの処理が制御されます。

呼出し形式は、以下の形式です。

CALL ERRSTR(*errno* , *darea*)

errno

基本整数型スカラ。格納すべきエラー制御表内のエラー項目に対応するエラー識別番号。

darea

16バイトの文字型スカラ。利用者が用意したエラー項目領域。*darea* の型が文字型でない場合、結果は不定となります。

例: ERRSAVおよびERRSTRサービスサブルーチンの使用例

```
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サービスサブルーチン

ERRSETサービスサブルーチンは、指定されたエラー識別番号に対応するエラー制御表内のエラー項目の個々の制御情報を変更したいときに使用します。ここで変更できる制御情報は、エラー打切り回数、メッセージの最大出力回数、トレースバックマップの出力の有無およびエラーの修正処置です。

呼出し形式は、以下の形式です。

CALL ERRSET(*errno* , *estop* , *mprint* , *trace* , *uexit* , *r*)

errno

基本整数型スカラ。変更すべきエラー項目に対応するエラー識別番号。

estop

基本整数型スカラ。エラー打切り回数。指定された値が、0以下であれば変更されません。255より大きければエラー打切り回数は無制限となります。

mprint

基本整数型スカラ。メッセージの最大出力回数。指定された値が0ならば変更されません。負ならば最大出力回数は0回となります。255より大きければ最大出力回数は無制限となります。

trace

基本整数型スカラ。値とトレースバックマップの出力の制御情報の関係を以下に示します。

=0: トレースバックマップの出力の制御情報は変更されません。

=1: トレースバックマップを出力しないように変更されます。

=2: トレースバックマップを出力するように変更されます。

1、2以外の場合は0とみなします。

uexit

基本整数型スカラ。エラーの修正処置として0、1、または利用者定義の修正サブルーチンの名前を指定します。0の場合は変更されません。1の場合は本処理系の標準修正を行うようにすることを意味します。利用者定義の修正サブルーチンの名前を指定する場合、この名前は外部手続名として宣言されていなければなりません。

r

基本整数型スカラ。*errno*が132以外の場合はエラー識別番号を指定します。ここで、*r*＞*errno*であれば、*errno*から*r*までの範囲のエラー識別番号に対応するエラー項目群を*estop*、*mprint*、*trace*、*uexit*に従って変更することを意味します。*r*≦*errno*であれば、*r*を無視します。必要のない場合は、*r*≦*errno*の値を指定します。*errno*が132の場合、*r*=1であれば、データを出力するとき、書式仕様で定められたFortran記録がファイルのレコードの長さより大きい場合、次のFortran記録の先頭に1つの空白を付加して新しいレコードとして、出力されます。*r*≠1であれば、次のFortran記録の先頭に、空白は付加されません。例2を参照してください。

ERRSETサービスサブルーチンの各引数のうち、*mprint*、*trace*、および*uexit*を変更しない場合は、0を指定します。

注意

エラーレベルがsであるメッセージ識別番号のエラー打ち切り回数を変更し、プログラムの実行を継続した場合、プログラムが異常終了するなど、プログラムが正しく動作しない場合があります。プログラムが正しく動作しなかった場合、estopに0以下の値を指定してください。

例1:

```
EXTERNAL FIXUP
CALL ERRSET(202,50,-1,0,FIXUP,205)
```

この例は、エラー識別番号202から205までに対応するエラー項目群を次のように変更するものです。

- エラー打ち切り回数を50回とします。
- エラーが発生してもメッセージを出力しません。
- トレースバックマップの出力制御情報は変更しません。
- エラーの修正処理を利用者定義のFIXUPルーチンで行うようにします。

例2:

```
CALL ERRSET(132,0,0,0,0,1)
```

この例は、出力操作のときに書式仕様で定められたFortran記録がファイルのレコードの長さより大きい場合、次のFortran記録の先頭に1つの空白をつけて新しいレコードとするよう指示するものです。

8.1.2.4 ERRTRAサービスサブルーチン

ERRTRAサービスサブルーチンは、現在実行中のプログラム単位までのトレースバックマップを出力したいときに使用します。このトレースバックマップの形式は、“4.2.2トレースバックマップ”を参照してください。なお、このサービスサブルーチンが呼ばれたあとは、そのまま実行の継続が行われます。

呼出し形式は、以下の形式です。

CALL ERRTRA

例:

```
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 エラーモニタの使用方法

利用者は、エラー処理サービスサブルーチンおよび利用者定義のエラー修正サブルーチンを用意して、エラーに関する各種のサービス機能を活用することができます。ここでは、これらの機能の活用の方法およびその際の注意事項を説明します。

8.1.3.1 利用者が用意するエラー修正サブルーチン

利用者が用意するエラー修正サブルーチンは、ERRSETサービスサブルーチンによってエラー項目にそのサブルーチン名を指定しなければなりません。

エラーモニタは、該当するエラー項目に利用者のエラー修正サブルーチンの指定があれば、次のような形式でそのエラー修正サブルーチンを呼び出します。

CALL *user*(*ret* , *errno* [, *data1* , … , *datan*])

user

利用者のエラー修正サブルーチンの名前。

ret

復帰コードの入る4バイトの整数型スカラ。利用者のエラー修正サブルーチンで値を設定しなければなりません。

errno

検出されたエラーのエラー識別番号。

data1,…,*datan*

エラー修正サブルーチンで使用するスカラ変数名または配列要素名の並び。利用者のエラー修正サブルーチンに渡される上記の引数は、以下のようになります。

- 入出力のエラー
  “8.1.4 入出力エラーの処理”で示す引数が渡されます。
- 組込み関数のエラー
  “8.1.5 組込み関数エラーの処理”で示す引数が渡されます。
- 組込みサブルーチンのエラー
  “8.1.6 組込みサブルーチンエラーの処理”で示す引数が渡されます。
- 例外ハンドリングのエラー
  “8.1.7 例外ハンドリングエラーの処理”で示す引数が渡されます。
- 入出力、組込み手続、割込みを除くエラー
  “8.1.8 そのほかのエラーの処理”で示す引数が渡されます。

利用者が、エラー修正サブルーチンを作成する場合には、以下のような制限があるので注意が必要です。

- 入出力文の使用に関する制限
  入出力のエラーが発生した場合、その原因となった入出力文と同じ装置番号に対する入出力文は使用できません。例えば、装置番号10に対する入出力文を実行中のエラーであれば、装置番号10に対する入出力文は使用できません。
- 組込み関数の使用に関する制限
  組込み関数のエラーが発生した場合、利用者のエラー修正サブルーチン内でその原因となった組込み関数を使用できません。例えば、SQRTの引用で引数の値が誤りであったとすれば、利用者のエラー修正サブルーチン内でSQRTの引用はできません。また、べき乗演算子を含む式の使用などの場合に暗黙に引用される組込み関数も同様となります。
- 組込みサブルーチンの使用に関する制限
  組込みサブルーチンのエラーが発生した場合、利用者のエラー修正サブルーチン内でその原因となった組込みサブルーチンを使用できません。
- 復帰コード
  利用者のエラー修正サブルーチンで渡された引数*data1*、*data2*、…を変更しなかった場合、復帰コード*ret*に0を設定しなければなりません。また、渡された引数*data1*、*data2*、…を変更した場合、復帰コード*ret*に1を設定しなければなりません。ただし、エラーモニタから呼び出されたときは*ret*に1が設定されているので、利用者のエラー修正サブルーチンで渡された引数を変更した場合は*ret*に1を設定する必要はありません。これらの処理を誤った場合、結果は保証されません。

  復帰コードの値として0または1だけが設定できます。0、1以外の値を設定しても結果は保証されません。
- 引数の型
  エラー修正サブルーチンをFortran言語で記述する場合、引数の型を呼出し元のCALL文の実引数の型と一致させなければなりません。

8.1.3.2 組込み関数で発生するエラーの注意

組込み関数で引数に誤りがあった場合、その引数を組込み関数内の局所的な変数に代入し、エラーモニタを呼び出します。エラーモニタに渡されるのは、この局所的な変数です。したがって、利用者のエラー修正サブルーチンでは、この局所的な変数を修正する必要があります。誤った引数で組込み関数を呼び出しているプログラム単位内の引数の値を修正しても意味がありません。

8.1.3.3 利用者によるエラー処理の例

利用者によるエラー処理の例を、以下に示します。この例は、誤った入力データ(1234567890123)を入力して、エラー識別番号171を出力し、利用者のエラー修正サブルーチンFIXUPで、入力データを(9999)に修正して処理を継続するものです。

例: 利用者によるエラー処理

実行プログラム

```
      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
```

利用者のエラー修正サブルーチン

```
SUBROUTINE FIXUP(IRET, INO, I4CONV)
IF (INO.EQ.171) THEN
  I4CONV = 9999
  IRET = 1
END IF
RETURN
END SUBROUTINE
```

(入力データ)

```
1234567890123
```

(実行結果)

```
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 入出力エラーの処理

IOSTAT指定子、ERR指定子、END指定子、またはEOR指定子が指定されている入出力文の実行中に各指定子の対象となる誤り条件が発生した場合、エラーモニタによるエラー制御は行われず、各指定子に合った処理が行われます。各指定子については、“7.7.1.3 IOSTAT指定子”、“7.7.1.4 ERR指定子”、“7.7.1.5 END指定子”、“7.7.1.6 EOR指定子”を参照してください。

誤り条件が発生した場合、一般的に実行中のファイル内の位置や入力並びの値は、不定となります。

入出力文の実行中に発生するエラーに対するシステムの標準修正処理および利用者が修正可能な処理などについては、“表8.2 入出力に関するエラー処理”に示します。利用者修正プログラムに渡される引数は次のとおりです。

- 復帰コード
- エラー識別番号
- “表8.2 入出力に関するエラー処理”に示されているデータ

表8.2 入出力に関するエラー処理

番号	標準修正	利用者の修正処理	利用者修正プログラムに渡すデータ	
21～22	プログラムを終了します	修正は不可能	UNIT	なし
28	プログラムを終了します	修正は不可能	UNIT	なし
42	文を無視します	修正は不可能	UNIT	なし
55	文を無視します	修正は不可能	UNIT	なし
62	文を無視します	修正は不可能	UNIT	なし
64～65	文を無視します	修正は不可能	UNIT	なし
71～72	文を無視します	修正は不可能	UNIT	なし
74	文を無視します	修正は不可能	UNIT	なし
80	文を無視します	修正は不可能	UNIT	REC
81	文を無視します	修正は不可能	なし	
82	文を無視します	修正は不可能	UNIT	なし
83～85	指定子に値を設定しないで、処理を続行します	修正は不可能	なし	
86	指定子に値を設定しないで、処理を続行します	修正は不可能	UNIT	なし
87～89	文を無視します	修正は不可能	UNIT	なし
97	ACCESS指定子にDIRECTが指定されたものとみなします	修正は不可能	UNIT	なし
102～103	文を無視します	修正は不可能	UNIT	なし
104	POSITION指定子を無視して、処理を続行します	修正は不可能	UNIT	なし
105	文を無視します	修正は不可能	UNIT	なし
109	指定子に指定された型の最大値を設定して、処理を続行します	修正は不可能	UNIT	なし
111～115	文を無視します	修正は不可能	UNIT	なし
120	文を無視します	修正は不可能	UNIT	なし
122	文を無視します	修正は不可能	UNIT	なし
131	以降の入出力並びを無視します	修正は不可能	なし	
132	入力の場合、レコードを超えたデータを無視します 出力の場合、レコードを超えたデータを次の論理レコードに出力します	修正は不可能	UNIT	なし
133	以降の要素を無視します	修正は不可能	UNIT	なし
134	以降の入出力並びを無視します	修正は不可能	UNIT	なし
151	以降の入出力並びを無視します	修正は不可能	UNIT	なし
152	編集記述しに合った編集を行います	修正は不可能	UNIT	なし
153	開始の左括弧を補います	修正は不可能	UNIT	なし
154	書式仕様中の30を超えた入れ子を無視します	修正は不可能	UNIT	なし
155	編集記述子を無視して、書式仕様の最後の右括弧とみなします	修正は不可能	UNIT	なし

番号	標準修正	利用者の修正処理	利用者修正プログラムに渡すデータ	
156	反復子を無視して、書式仕様の最後の右括弧とみなします	修正は不可能	UNIT	なし
157	正しくない文字を無視して、書式仕様の最後の右括弧とみなします	書式仕様を示す文字をFCODEに代入します	FCODE	なし
158	右括弧を補い書式制御を続行します	修正は不可能	UNIT	なし
159	書式中の文字列を入力データに置き換えます	修正は不可能	UNIT	なし
162	入出力並びを無視して、書式制御を続行します	修正は不可能	UNIT	なし
171	1バイト、2バイト、または4バイトの入力項目が負の場合、指定できる最小の値とみなします 1バイト、2バイト、または4バイトの入力項目が正の場合、指定できる最大の値とみなします	I4CNVを修正します	I4CNV	なし
172	8バイトの入力項目のデータが負の場合、指定できる最小の値とみなします 8バイトの入力項目のデータが正の場合、指定できる最大の値とみなします	I8CNVを修正します	I8CNV	なし
173～175	正しくない文字を0とみなします	修正した文字をIDATAに代入します	IDATA	なし
176	データの絶対値が、$2^{-126}*2^{-23}$より小さい(入力項目が実数型)、$2^{-1022}*2^{-52}$より小さい(入力項目が倍精度実数型)、または$2^{-16382}*2^{-112}$より小さい(入力項目が4倍精度実数型)場合、真の0を設定します データの絶対値が、$2^{127}*(1+1.0-2^{-23})$より大きい(入力項目が実数型)、$2^{1023}*(1+1.0-2^{-52})$より大きい(入力項目が倍精度実数型)、または$2^{16383}*(1+1.0-2^{-112})$より大きい(入力項目が4倍精度実数型)場合、符号を変えないで、$2^{127}*(1+1.0-2^{-23})$(入力項目が実数型)、$2^{1023}*(1+1.0-2^{-52})$(入力項目が倍精度実数型)、または$2^{16383}*(1+1.0-2^{-112})$(入力項目が4倍精度実数型)を設定します	FCONVを修正します	FCONV	なし
182～183	文字型の編集を行います	修正は不可能	UNIT	なし
184	定数を無視して、文を終了します	修正は不可能	UNIT	なし
185	反復数または定数を無視して、文を終了します	修正は不可能	UNIT	なし
186～188	項目名を無視して、文を終了します	修正は不可能	UNIT	なし
189	文を無視します	修正は不可能	UNIT	なし
190	要素数を超えている定数を無視して処理を続行します	修正は不可能	UNIT	なし

番号	標準修正	利用者の修正処理	利用者修正プログラムに渡すデータ	
1073	文を無視します	修正は不可能	UNIT	なし
1111～1113	文を無視します	修正は不可能	UNIT	なし
1114	指定子に4バイトの整数型の最大値を設定して、処理を続行します	修正は不可能	UNIT	なし
1115	指定子を無視して処理を続行します	修正は不可能	UNIT	なし
1117～1127	文を無視します	修正は不可能	UNIT	なし
1146	文を無視します	修正は不可能	UNIT	なし
1147	文を無視します	修正は不可能	UNIT	なし
1148	文を無視します	修正は不可能	UNIT	なし
1149	ASYNCHRONOUS指定子にNOが指定されたとみなします	修正は不可能	UNIT	なし
1150～1151	文を無視します	修正は不可能	UNIT	なし
1152～1153	プログラムを終了します	修正は不可能	UNIT	なし
1161	プログラムを終了します	修正は不可能	UNIT	なし
1162	入出力並びを複数のFortran記録として入出力します	修正は不可能	UNIT	なし
1163	変数に4バイトの整数型の最大値を設定して処理を続行します	修正は不可能	なし	なし
1181	入力並びを無視して、処理を続行します	修正は不可能	UNIT	なし
1182～1184	文を無視します	修正は不可能	UNIT	なし
1201	正しくない文字を0とみなします	修正した文字をIDATAに代入します	IDATA	なし
1202	入力項目の値は変更せずに、処理を続行します	修正は不可能	UNIT	なし
1203	要素の型にしたがって編集処理を行い、処理を続行します	修正は不可能	UNIT	なし
1204	文を無視します	修正は不可能	UNIT	なし

UNIT: 外部ファイルの場合は装置番号(基本整数型)、内部ファイルの場合は -1を渡します。

IDATA: 入力データ上の誤っている文字。(1バイトの文字型)

FCODE: 配列で与える書式仕様中の誤っている書式コード。(1バイトの文字型)

REC: 記録変数。(4バイトの整数型)

I4CNV: 書式変換された結果の整数を設定する領域。(4バイトの整数型)

I8CNV: 書式変換された結果の整数を設定する領域。(8バイトの整数型)

FCONV: 書式変換された結果の実数を設定する領域。(実数型)

8.1.5 組込み関数エラーの処理

実行時に組込み関数で検出するエラーの原因、およびシステムの標準修正処理について、“表8.3 組込み関数に関するエラー処理”に示します。

番号202～255、259～282、1355～1370、1396～1414、1416～1417、1425～1454、1456～1461、1463～1474は、-Ncheck_intrfuncオプションが指定されている場合にのみ検出、および、エラー処理が行われます。-Ncheck_intrfuncオプションが指定されていない場合には、これらのエラーを検出せずに浮動小数点例外が発生します。浮動小数点例外は、-NRtrapオプションおよび環境変数FLIB_EXCEPT=uの指定により検出が可能です。

-Ncheck_intrfuncオプションおよび-NRtrapオプションについては“2.2 翻訳時オプション”を、環境変数FLIB_EXCEPT=uについては“3.8 実行時の環境変数”をそれぞれ参照してください。

利用者修正プログラムに渡される引数は次のとおりです。

- 復帰コード
- エラー識別番号
- “表8.3 組込み関数に関するエラー処理”に示されているデータ

表8.3 組込み関数に関するエラー処理

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
202	SIN(X) COS(X)	|X|≧8.23e+05	NaN	X	なし
203	DSIN(DX) DCOS(DX)	|DX|≧3.53d+15	NaN	DX	なし
204	QSIN(QX) QCOS(QX)	|QX|≧2**62*π	NaN	QX	なし
205	CSIN(CX) CCOS(CX)	|X1|≧8.23e+05	(NaN,NaN)	Xl	なし
206	CSIN(CX)	X2≧89.415	(SIGN(Inf,SIN(X1)), SIGN(Inf,COS(X1)))	X2	なし
		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) CDCOS(CDX)	|DX1|≧3.53d+15	(NaN,NaN)	DX1	なし
208	CDSIN(CDX)	DX2≧710.475	(DSIGN(Inf,DSIN(DX1)), DSIGN(Inf,DCOS(DX1)))	DX2	なし
		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) CQCOS(CQX)	|QX1|≧2**62*π	(NaN,NaN)	QX1	なし
210	CQSIN(CQX)	QX2≧11357.125	(QSIGN(Inf,QSIN(QX1)), QSIGN(Inf,QCOS(QX1)))	QX2	なし

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
		QX2≦-11357.125	(QSIGN(Inf,QSIN(QX1)), -QSIGN(Inf,QCOS(QX1)))		
	CQCOS(CQX)	QX2≧11357.125	(QSIGN(Inf,QCOS(QX1)), -QSIGN(Inf,QSIN(QX1)))		
		QX2≦-11357.125	(QSIGN(Inf,QCOS(QX1)), QSIGN(Inf,QSIN(QX1)))		
211	TAN(X) COTAN(X)	|X|≧8.23e+05	NaN	X	なし
212	TAN(X)	|X| が特異点(±π/2,±3π/2,...)に接近しています	Inf	X	なし
	COTAN(X)	|X| が特異点(0,±π,±2π,...)に接近しています			
213	DTAN(DX) DCOTAN(DX)	|DX|≧3.53d+15	NaN	DX	なし
214	DTAN(DX)	|DX| が特異点(±π/2,±3π/2,...) に接近しています	Inf	DX	なし
	DCOTAN(DX)	|DX|が特異点(0,±π,±2π,...)に接近しています			
215	QTAN(QX) QCOTAN(QX)	|QX|≧2**62*π	NaN	QX	なし
216	QTAN(QX)	|QX| が特異点(±π/2,±3π/2,...) に接近しています	Inf	QX	なし
	QCOTAN(QX)	|QX| が特異点(0,±π,±2π,...)に接近しています			
217	ASIN(X) ACOS(X)	|X|>1.0	NaN	X	なし
218	DASIN(DX) DACOS(DX)	|DX|>1.0	NaN	DX	なし
219	QASIN(QX) QACOS(QX)	|QX|>1.0	NaN	QX	なし
220	ATAN(X1,X2) ATAN2(X1,X2)	X1=0.0かつ X2=0.0	NaN	X1	X2
221	DATAN2(DX1,DX2)	DX1=0.0かつ DX2=0.0	NaN	DX1	DX2
222	QATAN2(QX1,QX2)	QX1=0.0かつ QX2=0.0	NaN	QX1	QX2
223	SINH(X)	X≧89.415	Inf	X	なし

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
		X≦-89.415	-Inf		
	COSH(X)	|X|≧89.415	Inf		
224	DSINH(DX)	DX≧710.475	Inf	DX	なし
		DX≦-710.475	-Inf		
	DCOSH(DX)	|DX|≧710.475	Inf		
225	QSINH(QX)	QX≧11357.125	Inf	QX	なし
		QX≦-11357.125	-Inf		
	QCOSH(QX)	|QX|≧11357.125	Inf		
226	SQRT(X)	X<0.0	NaN	X	なし
227	DSQRT(DX)	DX<0.0	NaN	DX	なし
228	QSQRT(QX)	QX<0.0	NaN	QX	なし
229	EXP(X)	X≧88.722	Inf	X	なし
230	DEXP(DX)	DX≧709.782	Inf	DX	なし
231	QEXP(QX)	QX≧11356.5	Inf	QX	なし
232	CEXP(CX)	X1≧88.722	(SIGN(Inf,COS(X1)), SIGN(Inf,SIN(X1)))	X1	なし
233	CEXP(CX)	|X2|≧8.23e+05	(NaN,NaN)	X2	なし
234	CDEXP(CDX)	DX1≧709.782	(DSIGN(Inf,DCOS(DX1)), DSIGN(Inf,DSIN(DX1)))	DX1	なし
235	CDEXP(CDX)	|DX2|≧3.53d+15	(NaN,NaN)	DX2	なし
236	CQEXP(CQX)	QX1≧11356.5	(QSIGN(Inf,QCOS(QX1)), QSIGN(Inf,QSIN(QX1)))	QX1	なし
237	CQEXP(CQX)	|QX2|≧2**62*π	(NaN,NaN)	QX2	なし
238	EXP2(X)	X≧128.0	Inf	X	なし
	2.0**X				
239	DEXP2(DX)	DX≧1024.0	Inf	DX	なし
	2.0**DX				
240	QEXP2(QX)	QX≧16384.0	Inf	QX	なし
	2.0**QX				
241	EXP10(X)	X≧38.531	Inf	X	なし
	10.0**X				
242	DEXP10(DX)	DX≧308.254	Inf	DX	なし
	10.0**DX				
243	QEXP10(QX)	QX≧4932.0625	Inf	QX	なし
	10.0**QX				
244	ALOG(X)	X=0.0	-Inf	X	なし
		X<0.0	NaN		
	ALOG10(X)	X=0.0	-Inf		

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
		X<0.0	NaN		
	ALOG2(X)	X=0.0	-Inf		
		X<0.0	NaN		
245	DLOG(DX)	DX=0.0	-Inf	DX	なし
		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	なし
		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	なし
248	CDLOG(CDX)	CDX=(0.0,0.0)	(Inf,NaN)	CDX	なし
249	CQLOG(CQX)	CQX=(0.0,0.0)	(Inf,NaN)	CQX	なし
256	LGT(C1,C2)	C1 またはC2 の文字列の中にASCIIコードに対応しないEBCDIC コードを指定しました	エラーとなった引数のASCII照合順序を26(16進数で1A)とみなします	C3	なし
	LGE(C1,C2)				
	LLT(C1,C2)				
	LLE(C1,C2)				
257	IBSET(I,POS)	POS の値が範囲を超えています	I	なし	
	IBCLR(I,POS)				
	BTEST(I,POS)		.FALSE.		
259	IX1**IX2	IX1=0 かつIX2<0	0	IX1	IX2
260	JX1**JX2	JX1=0 かつJX2<0	0	JX1	JX2
261	X**IX	X=0.0 かつIX<0	Inf	X	IX
262	X1**X2	X1=0.0 かつX2<0.0	Inf	X1	X2
263	X1**X2	X1<0.0 かつX2≠0.0	NaN	X1	X2
264	DX**IX	DX=0.0 かつIX<0	Inf	DX	IX
265	DX**JX	DX=0.0 かつJX<0	Inf	DX	JX
266	DX1**DX2	DX1=0.0 かつ DX2<0.0	Inf	DX1	DX2
267	DX1**DX2	DX1<0.0 かつ DX2≠0.0	NaN	DX1	DX2
268	QX**IX	QX=0.0 かつIX<0	Inf	QX	IX
269	QX**JX	QX=0.0 かつJX<0	Inf	QX	JX

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
270	QX1**QX2	QX1=0.0 かつ QX2<0.0	Inf	QX1	QX2
271	QX1**QX2	QX1<0.0 かつ QX2≠0.0	NaN	QX1	QX2
272	QX1**QX2	|QX1**QX2|≧2**16384	Inf	QX1	QX2
273	CX**IX	CX=(0.0,0.0) かつ IX<0	(NaN,NaN)	CX	IX
274	CDX**IX	CDX=(0.0,0.0) かつ IX<0	(NaN,NaN)	CDX	IX
275	CDX**JX	CDX=(0.0,0.0) かつ JX<0	(NaN,NaN)	CDX	JX
276	CQX**IX	CQX=(0.0,0.0) かつ IX<0	(NaN,NaN)	CQX	IX
277	CQX**JX	CQX=(0.0,0.0) かつ 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	QX1=0.0 のとき NaN。 QX1≠0.0 のとき 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	JX1=0 のとき 0。 JX1≠0 のとき 2**63-1。	QX1	QX2
1355	TANQ(X)	|X| が特異点(±1,±3,...)に接近しています	Inf	X	なし
	COTANQ(X)	|X| が特異点 (0,±2,±4,...) に接近しています			
1356	DTANQ(DX)	|DX| が特異点 (±1,±3,...) に接近しています	Inf	DX	なし
	DCOTANQ(DX)	|DX| が特異点 (0,±2,±4,...) に接近しています			
1357	ASINQ(X)	|X|>1.0	NaN	X	なし
	ACOSQ(X)				
1358	DASINQ(DX)	|DX|>1.0	NaN	DX	なし
	DACOSQ(DX)				

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
1359	ATAN2Q(X1, X2)	X1=0.0 かつ X2=0.0	NaN	X1	X2
1360	DATAN2Q(DX1,DX2)	DX1=0.0 かつ DX2=0.0	NaN	DX1	DX2
1361	SIND(X) COSD(X)	|X|≧4.72e+07	NaN	X	なし
1362	DSIND(DX) DCOSD(DX)	|DX|≧2.03d+17	NaN	DX	なし
1363	TAND(X) COTAND(X)	|X|≧4.72e+07	NaN	X	なし
1364	DTAND(DX) DCOTAND(DX)	|DX|≧2.03d+17	NaN	DX	なし
1365	TAND(X)	|X| が特異点 (±90,±270,...)に接近しています	Inf	X	なし
	COTAND(X)	|X| が特異点 (0,±180,±360,...)に接近しています			
1366	DTAND(DX)	|DX| が特異点 (±90,±270,...)に接近しています	Inf	DX	なし
	DCOTAND(DX)	|DX| が特異点 (0,±180,±360,...)に接近しています			
1367	ASIND(X) ACOSD(X)	|X|>1.0	NaN	X	なし
1368	DASIND(DX) DACOSD(DX)	|DX|>1.0	NaN	DX	なし
1369	ATAN2D(X1,X2)	X1=0.0 かつ X2=0.0	NaN	X1	X2
1370	DATAN2D(DX1,DX2)	DX1=0.0 かつ DX2=0.0	NaN	DX1	DX2
1371	ISHFTC(I,SHIFT[,SIZE])	SIZE≦0 または SIZE>BIT_SIZE(I)	SIZE にBIT_SIZE(I) を設定して処理を続行します	なし	
		|SHIFT|>SIZE	SHIFT にSIZE の値を設定して処理を続行します		
1372	IBITS(I,POS,LEN)	LEN<0 または LEN>BIT_SIZE(I)-POS	0	なし	
		POS<0 または POS≧BIT_SIZE(I)-LEN			
1373	ISHFT(I,SHIFT)	|SHIFT|>BIT_SIZE(I)	SHIFT にBIT_SIZE(I) を設定して処理を続行します	なし	

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
1375	MATMUL(MATRIX_A, MATRIX_B)	MATRIX_A の最終次元とMATRIX_Bの1 次元の寸法が異なります	プログラムを終了します	なし	
1376	SPREAD(SOURCE, DIM,NCOPIES)	DIM≦0 または DIM>(SOURCE の次元数)+1	プログラムを終了します	なし	
1377	ALL(MASK[,DIM])	DIM≦0 または DIM>(MASK の次元数)	プログラムを終了します	なし	
	ANY(MASK[,DIM])				
	PARITY(MASK[,DIM])				
	COUNT(MASK [,DIM,KIND])				
	SUM(ARRAY, DIM[,MASK])	DIM≦0 または DIM>(ARRAY の次元数)			
	PRODUCT(ARRAY, DIM[,MASK])				
	MAXVAL(ARRAY, DIM[,MASK])				
	MINVAL(ARRAY, DIM[,MASK])				
	UBOUND(ARRAY [,DIM])				
	LBOUND(ARRAY [,DIM,KIND])				
	SIZE(ARRAY[,DIM])				
	MAXLOC(ARRAY, DIM[,MASK,KIND,BACK])				
	MINLOC(ARRAY,DIM [,MASK,KIND,BACK])				
	FINDLOC(ARRAY,VA LUE,DIM[,MASK,KIN D,BACK])				
	IALL(ARRAY,DIM[,M ASK])				
	IANY(ARRAY,DIM[,M ASK])				
	IPARITY(ARRAY,DI M[,MASK])				
	NORM2(X[,DIM])	DIM≦0 または DIM>(X の次元数)			
1378	MAXLOC(ARRAY, DIM[,MASK,KIND,BA CK]) または	ARRAY とMASK の形状が異なります	プログラムを終了します	なし	

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
	MAXLOC(ARRAY [,MASK,KIND,BACK])				
	MINLOC(ARRAY, DIM[,MASK,KIND,BACK]) または MINLOC(ARRAY [,MASK,KIND,BACK])				
	SUM(ARRAY, DIM[,MASK]) または SUM(ARRAY [,MASK])				
	PRODUCT(ARRAY, DIM[,MASK]) または PRODUCT(ARRAY [,MASK])				
	MAXVAL(ARRAY, DIM[,MASK]) または MAXVAL(ARRAY [,MASK])				
	MINVAL(ARRAY, DIM[,MASK]) または MINVAL(ARRAY [,MASK])				
	PACK(ARRAY, MASK[,VECTOR])				
	FINDLOC(ARRAY,VALUE,DIM[,MASK,KIND,BACK]) または FINDLOC(ARRAY,VALUE[,MASK,KIND,BACK])				
	IALL(ARRAY,DIM[,MASK]) または IALL(ARRAY[,MASK])				
	IANY(ARRAY,DIM[,MASK])				

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
	または IANY(ARRAY[,MASK]) IPARITY(ARRAY,DIM[,MASK]) または IPARITY(ARRAY[,MASK])				
1381	PACK(ARRAY, MASK[,VECTOR])	VECTOR の配列要素数がMASK の真の配列要素数より小さいです	プログラムを終了します	なし	
1382	UNPACK(VECTOR, MASK[,FIELD])	VECTOR の配列要素数がMASK の真の配列要素数より小さいです	プログラムを終了します	なし	
1383	UNPACK(VECTOR, MASK[,FIELD])	MASK とFIELD の形状が異なります	プログラムを終了します	なし	
1384	CSHIFT(ARRAY, SHIFT[,DIM]) EOSHIFT(ARRAY, SHIFT[,BOUNDARY][,DIM])	DIM≦0 または DIM>(ARRAY の次元数)	プログラムを終了します	なし	
1385	CSHIFT(ARRAY, SHIFT[,DIM]) EOSHIFT(ARRAY, SHIFT[,BOUNDARY][,DIM])	SHIFT の形状が正しくありません	プログラムを終了します	なし	
1386	EOSHIFT(ARRAY, SHIFT[,BOUNDARY][,DIM])	BOUNDARY の形状が正しくありません	プログラムを終了します	なし	
1387	RESHAPE(SOURCE, SHAPE[,PAD][,ORDER])	SOURCE の大きさがSHAPE の各要素の値の積より小さいです	プログラムを終了します	なし	
1390	RESHAPE(SOURCE, SHAPE[,PAD][,ORDER])	SHAPE とORDERの形状が異なっています	プログラムを終了します	なし	
1391	RESHAPE(SOURCE, SHAPE[,PAD][,ORDER])	ORDER の各要素の値は1 からSHAPEの寸法までの組合せでなければなりません	プログラムを終了します	なし	
1392	REPEAT(STRING, NCOPIES)	NCOPIES<0	プログラムを終了します	なし	
1393	NEAREST(X,S)	S=0.0	X	なし	
1394	SIZE(ARRAY[,DIM]) UBOUND(ARRAY[,DIM])	ARRAY が大きさ引継ぎ配列のときDIM<1 またはDIM≧(ARRAY の次元数)	プログラムを終了します	なし	

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
1395	RESHAPE(SOURCE, SHAPE[,PAD] [,ORDER])	SOURCE とPAD の型パラメタが異なっています	プログラムを終了します	なし	
	PACK(ARRAY,MASK [,VECTOR])	ARRAY とVECTORの型パラメタが異なっています			
	UNPACK(VECTOR, MASK[,FIELD])	VECTOR とFIELDの型パラメタが異なっています			
	MERGE(TSOURCE, FSOURCE,MASK)	TSOURCE とFSOURCE の型パラメタが異なっています			
	EOSHIFT(ARRAY, SHIFT[,BOUNDARY] [,DIM])	ARRAY とBOUNDARY の型パラメタが異なっています			
1396	MODULO(A,P)	P=0	Aが実数型のときInf。 Aが整数型のとき HUGE(A)。	なし	
1397	QX1+QX2	|QX1+QX2|≧2**16384 (オーバフロー)	Inf	QX1	QX2
	QX1-QX2	|QX1-QX2|≧2**16384 (オーバフロー)			
	QX1*QX2	|QX1*QX2|≧2**16384 (オーバフロー)			
1398	QX1*QX2	|QX1*QX2|<2**(-16382) (アンダフロー)	0.0	QX1	QX2
1399	QTANQ(QX)	|QX| が特異点 (±1,±3,...)に接近しています	Inf	QX	なし
	QCOTANQ(QX)	|QX| が特異点 (0,±2,±4,...) に接近しています			
1400	QASINQ(QX)	|QX|>1.0	NaN	QX	なし
	QACOSQ(QX)				
1401	QATAN2Q(QX1,QX2)	QX1=0.0 かつ QX2=0.0	NaN	QX1	QX2
1402	QSIND(QX)	|QX|≧2**62*180	NaN	QX	なし
	QCOSD(QX)				
1403	QTAND(QX)	|QX|≧2**63*90	NaN	QX	なし
	QCOTAND(QX)				
1404	QTAND(QX)	|QX| が特異点 (±90,±270,...) に接近しています	Inf	QX	なし
	QCOTAND(QX)	|QX| が特異点 (0,±180,±360,...) に接近しています			
1405	QASIND(QX)	|QX|>1.0	NaN	QX	なし
	QACOSD(QX)				

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
1406	QATAN2D(QX1,QX2)	QX1=0.0 かつ QX2=0.0	NaN	QX1	QX2
1407	CSINQ(CX) CCOSQ(CX)	|X1|≧5.24e+05	(NaN,NaN)	X1	なし
1408	CSINQ(CX)	X2≧56.92	(SIGN(Inf,SINQ(X1)), SIGN(Inf,COSQ(X1)))	X2	なし
		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) CDCOSQ(CDX)	|DX1|≧2.25d+15	(NaN,NaN)	DX1	なし
1410	CDSINQ(CDX)	DX2≧452.30	(DSIGN(Inf,DSINQ(DX1)), DSIGN(Inf,DCOSQ(DX1)))	DX2	なし
		DX2≦-452.30	(DSIGN(Inf,DSINQ(DX1)), - DSIGN(Inf,DCOSQ(DX1)))		
	CDCOSQ(CDX)	DX2≧452.30	(DSIGN(Inf,DCOSQ(DX1)) , -DSIGN(Inf,DSINQ(DX1)))		
		DX2≦-452.30	(DSIGN(Inf,DCOSQ(DX1)) , DSIGN(Inf,DSINQ(DX1)))		
1411	CQSINQ(CQX) CQCOSQ(CQX)	|QX1|≧2**63	(NaN,NaN)	QX1	なし
1412	CQSINQ(CQX)	QX2≧7230.125	(QSIGN(Inf,QSINQ(QX1)), QSIGN(Inf,QCOSQ(QX1)))	QX2	なし
		QX2≦-7230.125	(QSIGN(Inf,QSINQ(QX1)), - QSIGN(Inf,QCOSQ(QX1)))		
	CQCOSQ(CQX)	QX2≧7230.125	(QSIGN(Inf,QCOSQ(QX1)) , -QSIGN(Inf,QSINQ(QX1)))		
		QX2≦-7230.125	(QSIGN(Inf,QCOSQ(QX1)) , QSIGN(Inf,QSINQ(QX1)))		
1413	X**JX	X=0.0 かつJX<0	Inf	X	JX
1414	CX**JX	CX=(0.0,0.0) かつ JX<0	(NaN,NaN)	CX	JX
1415	SELECTED_REAL_KIND(P,R,RADIX)	P,R,およびRADIX が省略されています	プログラムを終了します	なし	
1416	X1**X2	|X1**X2|≧2**128	Inf	X1	X2

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
1417	DX1**DX2	|DX1**DX2|≧2**1024	Inf	DX1	DX2
1418	PACK(ARRAY, MASK[,VECTOR])	MASKがスカラかつ値が真のとき、VECTOR の要素数はARRAY の要素数以上でなければなりません	プログラムを終了します	なし	
1419	EOSHIFT(ARRAY, SHIFT[,BOUNDARY] [,DIM])	ARRAY が派生型の場合、BOUNDARYを省略してはなりません	プログラムを終了します	なし	
1420	ICHAR(C [,KIND])	関数結果が、種別型パラメタKINDで表現可能な値の範囲を超えています	KIND値で指定された種別で結果を返却します	なし	
	INDEX(STRING, SUBSTRING [,BACK, KIND])				
	LBOUND(ARRAY, [, DIM, KIND])				
	LEN(STRING[, KIND])				
	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)	サポートされていない機能を使用しました	0または.FALSE.	なし	
	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)				

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
1422	IEEE_IS_NAN(X) IEEE_VALUE(X,CLASS)	サポートされていない機能を使用しました	0または.FALSE.	なし	
1425	ACOSH(X)	X<1	NaN	X	なし
1426	ACOSH(DX)	DX<1	NaN	DX	なし
1427	ACOSH(QX)	QX<1	NaN	QX	なし
1428	ATANH(X)	|X|>1	NaN	X	なし
		X=1	Inf		
		X=-1	-Inf		
1429	ATANH(DX)	|DX|>1	NaN	DX	なし
		DX=1	Inf		
		DX=-1	-Inf		
1430	ATANH(QX)	|QX|>1	NaN	QX	なし
		QX=1	Inf		
		QX=-1	-Inf		
1431	TAN(CX)	|X1|≧8.23e+05	(NaN,NaN)	X1	なし
1432	TAN(CX)	CXが特異点((±π/2,0),(±3π/2,0), …) に接近しています	REAL(CX)≧0.0のとき(Inf,0.0)。 REAL(CX)<0.0のとき(-Inf,0.0)。	CX	なし
1433	TAN(CDX)	|DX1|≧3.53d+15	(NaN,NaN)	DX1	なし
1434	TAN(CDX)	CDXが特異点((±π/2,0),(±3π/2,0), …) に接近しています	DREAL(CDX)≧0.0のとき(Inf,0.0)。 DREAL(CDX)<0.0のとき(-Inf,0.0)。	CDX	なし
1435	TAN(CQX)	|QX1|≧2.0**62*π	(NaN,NaN)	QX1	なし
1436	TAN(CQX)	CQXが特異点((±π/2,0),(±3π/2,0), …) に接近しています	QREAL(CQX)≧0.0のとき(Inf,0.0)。 QREAL(CQX)<0.0のとき(-Inf,0.0)。	CQX	なし
1437	ATAN(CX)	CX=(0.0,±1.0)	(0.0,±Inf)	CX	なし
1438	ATAN(CDX)	CDX=(0.0,±1.0)	(0.0,±Inf)	CDX	なし
1439	ATAN(CQX)	CQX=(0.0,±1.0)	(0.0,±Inf)	CQX	なし
1440	SINH(CX) COSH(CX)	|X2|≧8.23e+05	(NaN,NaN)	X2	なし
1441	SINH(CX)	X1≧89.4150	(SIGN(Inf,COS(X2)),SIGN(Inf,SIN(X2)))	X1	なし
		X1≦-89.4150	(-SIGN(Inf,COS(X2)),SIGN(Inf,SIN(X2)))		

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
	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	なし
	COSH(CDX)				
1443	SINH(CDX)	DX1≧710.475	(DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))	DX1	なし
		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*π	(NaN,NaN)	QX2	なし
	COSH(CQX)				
1445	SINH(CQX)	QX1≧11357.125	(QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))	QX1	なし
		QX1≦-11357.125	(-QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))		
	COSH(CQX)	QX1≧11357.125	(QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))		
		QX1≦-11357.125	(QSIGN(Inf,QCOS(QX2)), -QSIGN(Inf,QSIN(QX2)))		
1446	TANH(CX)	|X2|≧8.23e+05	(NaN,NaN)	X2	なし
1447	TANH(CX)	CXが特異点((0,±π/2), (0,±3π/2), …) に接近しています	(NaN,±Inf)	CX	なし
1448	TANH(CDX)	|DX2|≧3.53d+15	(NaN,NaN)	DX2	なし
1449	TANH(CDX)	CDXが特異点((0,±π/2),(0,±3π/2), …) に接近しています	(NaN,±Inf)	CDX	なし
1450	TANH(CQX)	|QX2|≧2.0**62*π	(NaN,NaN)	QX2	なし
1451	TANH(CQX)	CQXが特異点((0,±π/2),(0,±3π/2), …) に接近しています	(NaN,±Inf)	CQX	なし
1452	ATANH(CX)	CX=(±1.0,0.0)	(±Inf,0.0)	CX	なし
1453	ATANH(CDX)	CDX=(±1.0,0.0)	(±Inf,0.0)	CDX	なし
1454	ATANH(CQX)	CQX=(±1.0,0.0)	(±Inf,0.0)	CQX	なし
1455	DSHIFTL(I, J, SHIFT)	SHIFT<0 または SHIFT>BIT_SIZE(I)	SHIFT<0 のときはSHIFT に0を、SHIFT>BIT_SIZE(I)のときはSHIFT にBIT_SIZE(I)を設定して処理を続行します	なし	
	DSHIFTR(I, J, SHIFT)				
	SHIFTA(I, SHIFT)				

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
	SHIFTL (I, SHIFT)				
	SHIFTR (I, SHIFT)				
	MASKL(I [, KIND])	I<0 または I>BIT_SIZE(I)	I<0 のときはIに0を、 I>BIT_SIZE(I)のときはIに BIT_SIZE(I) を設定して処理を続行します		
	MASKR(I [, KIND])				
1456	BESSEL_J0(X)	|X|≧8.23e+05	NaN	X	なし
	BESSEL_J1(X)				
	BESSEL_JN(N, X)				
	BESSEL_JN(N1,N2,X)				
1457	BESSEL_J0(DX)	|DX|≧3.53d+15	NaN	DX	なし
	BESSEL_J1(DX)				
	BESSEL_JN(N, DX)				
	BESSEL_JN(N1,N2,DX)				
1458	BESSEL_J0(QX)	|QX|≧2**62*π	NaN	QX	なし
	BESSEL_J1(QX)				
	BESSEL_JN(N, QX)				
	BESSEL_JN(N1,N2,QX)				
1459	BESSEL_Y0(X)	X≧8.23e+05または X<0.0	NaN	X	なし
		X=0.0	-Inf		
	BESSEL_Y1(X)	X≧8.23e+05または X<0.0	NaN		
		X=0.0	-Inf		
	BESSEL_YN(N, X)	X≧8.23e+05または X<0.0	NaN		
		X=0.0	-Inf		
	BESSEL_YN(N1,N2,X)	X≧8.23e+05または X<0.0	NaN		
		X=0.0	-Inf		
1460	BESSEL_Y0(DX)	DX≧3.53d+15または DX<0.0	NaN	DX	なし
		DX=0.0	-Inf		
	BESSEL_Y1(DX)	DX≧3.53d+15または DX<0.0	NaN		
		DX=0.0	-Inf		
	BESSEL_YN(N, DX)	DX≧3.53d+15または	NaN		

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
		DX<0.0			
		DX=0.0	-Inf		
	BESSEL_YN(N1,N2,DX)	DX≧3.53d+15または DX<0.0	NaN		
		DX=0.0	-Inf		
1461	BESSEL_Y0(QX)	QX≧2**62*πまたは QX<0.0	NaN	QX	なし
		QX=0.0	-Inf		
	BESSEL_Y1(QX)	QX≧2**62*πまたは QX<0.0	NaN		
		QX=0.0	-Inf		
	BESSEL_YN(N, QX)	QX≧2**62*πまたは QX<0.0	NaN		
		QX=0.0	-Inf		
	BESSEL_YN(N1,N2,QX)	QX≧2**62*πまたは QX<0.0	NaN		
		QX=0.0	-Inf		
1462	BESSEL_JN(N, X)	N<0	NaN	なし	
	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またはN2<0	大きさ0の配列		
	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	なし
1464	GAMMA(DX)	DX≧171.6243	Inf	DX	なし
	DGAMMA(DX)				
1465	GAMMA(QX)	QX≧1.755q+03	Inf	QX	なし
	QGAMMA(QX)				
1466	LOG_GAMMA(X)	X≧0.403711e+37	Inf	X	なし

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ	
				data1	data2
	LGAMMA(X)				
	ALGAMA(X)				
1467	LOG_GAMMA(DX)	DX≧2.55634d+305	Inf	DX	なし
	LGAMMA(DX)				
	DLGAMA(DX)				
1468	LOG_GAMMA(QX)	QX≧1.048q+4928	Inf	QX	なし
	LGAMMA(QX)				
	QLGAMA(QX)				
1469	GAMMA(X)	Xが負の整数値です	NaN	X	なし
		X=±0.0	±Inf		
	LOG_GAMMA(X)	Xが0.0または負の整数値です	Inf		
	LGAMMA(X)				
	ALGAMA(X)				
1470	GAMMA(DX)	DXが負の整数値です	NaN	DX	なし
		DX=±0.0	±Inf		
	DGAMMA(DX)	DXが負の整数値です	NaN		
		DX=±0.0	±Inf		
	LOG_GAMMA(DX)	DXが0.0または負の整数値です	Inf		
	LGAMMA(DX)				
	DLGAMA(DX)				
1471	GAMMA(QX)	QXが負の整数値です	NaN	QX	なし
		QX=±0.0	±Inf		
	QGAMMA(QX)	QXが負の整数値です	NaN		
		QX=±0.0	±Inf		
	LOG_GAMMA(QX)	QXが0.0または負の整数値です	Inf		
	LGAMMA(QX)				
	QLGAMA(QX)				
1472	ERFC_SCALED(X)	X≦-9.3824	Inf	X	なし
1473	ERFC_SCALED(DX)	DX≦-26.6287	Inf	DX	なし
1474	ERFC_SCALED(QX)	QX≦-106.56	Inf	QX	なし

C1、C2、C3: 1バイトの文字型データです。C3には26(16進で1A)の値が入っています。

IX、IX1、IX2: 4バイトの整数型のデータです。

JX、JX1、JX2: 8バイトの整数型のデータです。

X、X1、X2: 実数型のデータです。

DX、DX1、DX2: 倍精度実数型のデータです。

QX、QX1、QX2: 4倍精度実数型のデータです。

CX: 複素数型のデータであり、CXは(X1,X2)を意味します。

CDX: 倍精度複素数型のデータであり、CDXは(DX1,DX2)を意味します。

CQX: 4倍精度複素数型のデータであり、CQXは(QX1,QX2)を意味します。

π : 定義であり、π ≒3.141592…です。

備考. 利用者修正プログラムに渡される引数の番地は、組込み関数またはサービスサブルーチンを利用するときに渡した引数の番地とは異なります(値は同じです)。

8.1.6 組込みサブルーチンエラーの処理

組込みサブルーチンで検出するエラーの原因、システムの標準修正処理について、“表8.4 組込みサブルーチンに関するエラー処理”に示します。利用者修正プログラムに渡される引数は次のとおりです。

- 復帰コード
- エラー識別番号
- “表8.4 組込みサブルーチンに関するエラー処理”に示されているデータ

表8.4 組込みサブルーチンに関するエラー処理

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ
1301	MVBITS(FROM,FROMPOS,LEN,TO,TOPOS)	LEN<0 または LEN>BIT_SIZE(FROM)	TOに0を設定します	なし
		FROMPOS<0 または FROMPOS>= BIT_SIZE(FROM)-LEN	TOに0を設定します	
		TOPOS<0 または TOPOS>= BIT_SIZE(FROM)-LEN	TOに0を設定します	
1302	DATE_AND_TIME([DATE,TIME,ZONE,VALES])	文字列DATEが、8文字に満たない場合	DATEの長さだけ情報を設定します	なし
		文字列TIMEが、10文字に満たない場合	TIMEの長さだけ情報を設定します	
		文字列 ZONEが、5文字に満たない場合	ZONEの長さだけ情報を設定します	
1303	DATE_AND_TIME([DATE,TIME,ZONE,VALES])	配列VALUESの大きさが、7に満たない場合	VALUESの大きさだけ情報を設定します	なし
1304	RANDOM_SEED([SIZE,PUT,GET])	PUTの配列の大きさがSIZEよりも小さい場合	PUTの大きさだけシード値を設定します	なし
		GETの配列の大きさがSIZEよりも小さい場合	GETの大きさだけシード値を設定します	
1305	RANDOM_SEED([SIZE,PUT,GET])	SIZE,PUT,GETが2個以上、同時に指定された場合	プログラムを終了します	なし
1306	MOVE_ALLOC([FROM,TO])	文字型の引数FROMとTOの長さが異なります。	プログラムを終了します	なし
1307	EXECUTE_COMMAND_LINE(COMMAND[,WAIT,EXITSTAT,CMDSTAT,CMDMSG])	システムコールでエラーが発生しました	プログラムを終了します	なし

番号	引用の形式	エラー発生の原因	標準修正処理	利用者修正プログラムに渡すデータ
1422	IEEE_GET_HALTING_MODE(FLAG,HALTING)	サポートされていない機能を使用しました	何もしません	なし
	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)	サポートされていない機能を使用しました	何もしません	なし

8.1.7 例外ハンドリングエラーの処理

本処理系における例外ハンドリングの処理は、浮動小数点演算における例外、不当なアクセス例外などです。

浮動小数点演算における例外(jwe1017i-u)は、シグナル(注)に対応するトラップを有効にして、ハンドリングします。翻訳時オプション-NRnotrapが有効な場合、浮動小数点演算における例外は検出されません。翻訳時オプション-NRnotrapについては、“2.2 翻訳時オプション”を参照してください。

注) Armアーキテクチャである富士通製CPU A64FXでサポートしているシグナルだけが対象です。そのため、サポートしていないシグナルは、翻訳時オプション-NRtrapを有効にしても検出できません。

実行時オプション-i(“3.3 実行時オプション”参照)を指定したときは、例外ハンドリングエラーに対する処理は行われません。なお、例外処理については、“8.2.2 異常終了プログラムのデバッグ”を参照してください。

例外ハンドリングの修正処理を、“表8.5 割込み発生時の修正処理”に示します。

表8.5 割込み発生時の修正処理

番号	割込みの原因	標準修正処理	利用者が修正できる処理	利用者修正プログラムに渡すデータ
17	実行時オプション -tの指定により、プログラムが終了しました	プログラムを終了します	修正は不可能	なし
18	実行時オプション-aの指定により、プログラムが異常終了しました	プログラムを終了します	修正は不可能	なし
19	不当なアクセスなどによりプログラムが異常終了しました	プログラムを終了します	修正は不可能	なし
20	異常終了処理中に再び、異常終了が発生しました	プログラムを終了します	修正は不可能	なし
1017	浮動小数点演算で結果の値が |65504|(半精度実数型または半精度複素数型)、|3.40282347E+38|(単精度実数型または単精度複素数型)、|1.797693134862316D+308|(倍精度実数型または倍精度複素数型)、|1.189731495357231765085759326628007 0Q+4932|(4倍精度実数型または4倍精度複素数型)より大きい値です	プログラムを終了します	修正は不可能	なし
	浮動小数点演算で結果の値が |6.10351562E-05|(半精度実数型または半精度複素数型)、|1.17549435E-38|(単精度実数型または単精度複素数型)、|		

番号	割込みの原因	標準修正処理	利用者が修正できる処理	利用者修正プログラムに渡すデータ
	2.225073858507201D-308|(倍精度実数型または倍精度複素数型)、|3.362103143112093506262 6778173217526Q+4932|(4倍精度実数型または4倍精度複素数型)より小さい値です			
	浮動小数点数の除算で除数が0です	プログラムを終了します	修正は不可能	なし
	浮動小数点演算で無効な演算を実行しました	プログラムを終了します	修正は不可能	なし

備考1. 標準修正処理で結果レジスタに値が設定されるときは、そのレジスタの符号は変更されません。また、最大値は以下の値です。

- 半精度実数型の場合、65504

- 単精度実数型の場合、3.40282347E+38

- 倍精度実数型の場合、1.797693134862316D+308

- 4倍精度実数型の場合、1.189731495357231765085759326628007 0Q+4932

備考2. 4倍精度実数型の演算では、上記の割込みエラーを捕捉せずに、SIGTRAPが発生することがあります。

8.1.8 そのほかのエラーの処理

本処理系で検出するエラーで入出力、組込み手続およびプログラム割込み以外のエラーの処理を、“表8.6 そのほかのエラー処理”に示します。利用者修正プログラムに渡される引数は次のとおりです。

- 復帰コード
- エラー識別番号
- “表8.6 そのほかのエラー処理”に示されているデータ

表8.6 そのほかのエラー処理

番号	標準修正処理	利用者が修正できる処理	利用者修正プログラムに渡すデータ
301	n番目の引数が省略されたものとみなして処理を続行します	修正は不可能	なし
304	CALL PRNSET(IX) IX<0のとき IX=0 IX>15のとき IX=15 として処理を続行します	修正は不可能	IX
306	プログラムを終了します	修正は不可能	なし
308	プログラムを終了します	修正は不可能	なし
311～ 318	引数の検査を継続します	修正は不可能	なし
320	処理を続行します	修正は不可能	なし
322～ 324	処理を続行します	修正は不可能	なし
329	プログラムを終了します	修正は不可能	なし
330	処理を続行します	修正は不可能	なし
1001	プログラムを終了します	修正は不可能	なし

番号	標準修正処理	利用者が修正できる処理	利用者修正プログラムに渡すデータ
1003～1008	プログラムを終了します	修正は不可能	なし
1031	プログラムを終了します	修正は不可能	なし
1032	並列数をCPU数とみなして処理を続行します	修正は不可能	なし
1033 (注)	プログラムを終了します	修正は不可能	なし
1034	処理を続行します	修正は不可能	なし
1035	プログラムを終了します	修正は不可能	なし
1036	処理を続行します	修正は不可能	なし
1038～1041	プログラムを終了します	修正は不可能	なし
1042	並列数をnumとして処理を続行します	修正は不可能	なし
1043	処理を続行します	修正は不可能	なし
1044～1046	プログラムを終了します	修正は不可能	なし
1047	処理を続行します	修正は不可能	なし
1048	プログラムを終了します	修正は不可能	なし
1049	処理を続行します	修正は不可能	なし
1050	処理を続行します	修正は不可能	なし
1051	処理を続行します	修正は不可能	なし
1052	プログラムを終了します	修正は不可能	なし
1071～1072	処理を続行します	修正は不可能	なし
1141～1143	処理を続行します	修正は不可能	なし
1231	処理を続行します	修正は不可能	なし
1232	プログラムを終了します	修正は不可能	なし
1501	プログラムを終了します	修正は不可能	なし
1551	処理を続行します	修正は不可能	なし
1561	処理を続行します	修正は不可能	なし
1563～1564	プログラムを終了します	修正は不可能	なし
1565～1566	処理を続行します	修正は不可能	なし
1569	処理を続行します	修正は不可能	なし
1570～1571	引数の検査を継続します	修正は不可能	なし
1572～1575	プログラムを終了します	修正は不可能	なし
1576～1577	処理を続行します	修正は不可能	なし
1578	プログラムを終了します	修正は不可能	なし

番号	標準修正処理	利用者が修正できる処理	利用者修正プログラムに渡すデータ
1579	処理を続行します	修正は不可能	なし
1582	処理を続行します	修正は不可能	なし
1606	プログラムを終了します	修正は不可能	なし
1652～1655	処理を続行します	修正は不可能	なし

IX: 基本整数型

注) 診断メッセージ(jwe1033i-i)は、並列実行中にプログラムが終了したことを通知します。この診断メッセージは、例えば、STOP/ERROR STOP文、EXITサービスサブルーチンを実行したり、エラーの発生回数が打切り回数に達して、並列実行中にプログラムが終了した場合に、逐次実行時と結果が異なる可能性があることを示しています。

8.2 デバッグのための機能

ここでは、Fortranプログラムのデバッグを行うための検査機能、および、実行時異常終了したときに出力されるデバッグ情報について説明します。

8.2.1 デバッグを行うための検査機能

デバッグを行うための検査機能には、プログラムの翻訳時、および、プログラムの実行時の検査機能があります。

翻訳時オプション-Ncheck_globalは、プログラムの翻訳時に次の検査を行います。

- 外部手続の定義と引用との間の手続特性の検査
- 外部手続の定義と引用仕様本体との間の手続特性の検査
- プログラム単位にまたがる共通ブロックの大きさの検査

-Haオプションおよび-Nquickdbg=argchkオプションは、プログラムの翻訳時に次の検査を行います。

- 明示的引用仕様のある手続における配列の大きさ検査

-Hsオプションおよび-Nquickdbg=subchkオプションは、プログラムの翻訳時に次の検査を行います。

- 引用した添字式および部分列範囲が、宣言した添字式および部分列範囲と矛盾していないかの検査

また、翻訳時オプション-Hおよび翻訳時オプション-Nquickdbgは、プログラムの実行時に以下の検査を行います。

表8.7 実行時の検査項目一覧

<table>
<tr><th></th><th>翻訳時オプション-H</th><th>翻訳時オプション-Nquickdbg</th></tr>
<tr><td colspan="3">引数の妥当性の検査 (ARGCHK機能)</td></tr>
<tr><td>引数の個数の検査</td><td rowspan="8">-Ha</td><td rowspan="2">-Nquickdbg=argchk</td></tr>
<tr><td>引数の型検査</td></tr>
<tr><td>引数の属性検査</td><td>－</td></tr>
<tr><td>関数の型検査</td><td>-Nquickdbg=argchk</td></tr>
<tr><td>引数の大きさの検査</td><td>－</td></tr>
<tr><td>明示的引用仕様の検査</td><td>-Nquickdbg=argchk</td></tr>
<tr><td>形状引継ぎ配列の次元数の検査</td><td>－</td></tr>
<tr><td colspan="3">添字式および部分列範囲の検査 (SUBCHK機能)</td></tr>
<tr><td>配列の引用に対する検査</td><td>-Hs</td><td>-Nquickdbg =subchk</td></tr>
</table>

	翻訳時オプション-H	翻訳時オプション-Nquickdbg
部分配列の引用に対する添字のオーバフロー検査		―
未定義データの引用の検査 (UNDEF機能)		
未定義データの検査	-Hu	-Nquickdbg =undefまたは -Nquickdbg =undefnan
割付け配列の検査		―
仮引数の実在検査		―
形状適合の検査 (SHAPECHK機能)		
形状適合の検査	-He	―
拡張検査 (EXTCHK機能)		
モジュール、サブモジュール、共通ブロックの未定義データ検査	-Hx	―
未結合ポインタの検査		―
刻み幅、増分値の検査		―
DO変数の更新検査		―
重複仮引数および拡張未定義検査 (OVERLAP機能)		
重複仮引数検査	-Ho	―
INTENT(OUT)属性をもつ大きさ引継ぎ配列の未定義検査		―
SAVE属性未定義検査		―
同時OPEN およびI/O の再帰呼び出し検査 (I/O OPEN機能)		
同時OPEN検査	-Hf	―
I/Oの再帰呼び出し検査		―

―:検査対象ではありません。

各検査項目についての詳細は、以降で説明します。

- 引数の妥当性の検査(“8.2.1.1 引数の妥当性の検査(ARGCHK機能)”を参照)
- 添字式および部分列範囲の検査(“8.2.1.2 添字式および部分列範囲の検査(SUBCHK機能)”を参照)
- 未定義データの引用の検査(“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照)
- 形状適合の検査(“8.2.1.4 形状適合の検査(SHAPECHK機能)”を参照)
- 拡張検査(“8.2.1.5 拡張検査(EXTCHK機能)”を参照)
- 重複仮引数および拡張未定義検査(“8.2.1.6 重複仮引数および拡張未定義検査(OVERLAP機能)”を参照)
- 同時OPEN およびI/O の再帰呼び出し検査(“8.2.1.7 同時OPENおよびI/Oの再帰呼び出し検査(I/O OPEN機能)”を参照)

翻訳時オプション-Hおよび翻訳時オプション-Nquickdbgの実行時検査には以下の特徴があります。

表8.8 実行時検査機能の特徴

	翻訳時オプション-H	翻訳時オプション-Nquickdbg
実行性能に与える影響	大	小
検査項目	多	少
スレッド並列 (注)	検査が無効になります	検査は有効です

注) OpenMP、自動並列実行時に検査が有効か無効であるか

翻訳時オプション-Nquickdbgによる実行時検査は、翻訳時オプション-Hによる実行時検査と比較して、検査項目が限定されますが、検査が実行性能に与える影響が軽減されます。また、OpenMP、自動並列実行プログラムに対する検査が有効となります。

翻訳時オプション-Nquickdbgによる実行時検査には、エラー検出時の診断メッセージを表示するオプションとして、-Nquickdbg=inf_detailオプションまたは-Nquickdbg=inf_simpleオプションがあります。

-Nquickdbg=inf_detailオプションが有効な場合、エラー検出時の診断メッセージとして、エラーの発生原因を識別するためのメッセージとエラー発生時の行番号、エラー発生の原因となった変数名、手続名、要素位置などの情報が表示されます。

-Nquickdbg=inf_simpleオプションが有効な場合、エラー検出時の診断メッセージとして、エラーが発生したことを通知するメッセージとエラー発生時の行番号が表示されます。ただし、-Nnolineオプションを指定した場合、行番号の値は保証されません。

-Nquickdbg=inf_simpleオプションが有効な場合は、-Nquickdbg=inf_detailオプションが有効な場合と比較して、診断メッセージに出力される情報が限定されますが、検査が実行性能に与える影響が軽減されます。

翻訳時オプション-H、翻訳時オプション-Nquickdbg および翻訳時オプション-Ncheck_globalについては、“2.2 翻訳時オプション”を参照してください。

8.2.1.1 引数の妥当性の検査(ARGCHK機能)

-Haオプションまたは -Nquickdbg=argchkオプションを指定すると、副プログラム(サブルーチンまたは関数)を引用したときに、副プログラムの仮引数と引用元の実引数について、検査されます。ただし、以下の場合については、検査されません。

- 実引数がPOINTER属性をもつ実体名の場合(実引数が派生型の変数であり、その成分がPOINTER属性をもつ場合も含まれます)
- 実引数が部分配列または配列式の場合
- 仮手続名による副プログラムを引用した場合
- 実引数が末端成分に割付け属性をもつ派生型の変数の場合
- 手続ポインタまたは型束縛手続を引用した場合
- 仮引数が、型引継ぎまたは次元引継ぎの場合

検査される項目について、以下に説明します。

8.2.1.1.1 引数の個数の検査

-Haオプションまたは-Nquickdbg=argchkオプションを指定すると、仮引数の個数に対して、実引数の個数が一致するかどうかが検査されます。この実引数の個数には、選択戻り指定子も含まれます。

引数の個数が一致しない場合、診断メッセージ(jwe0313i-w)が出力されます。

8.2.1.1.2 引数の型の検査

-Haオプションまたは-Nquickdbg=argchkオプションを指定すると、仮引数の型とそれに対応する実引数の型が一致しているかどうか検査されます。引数の型が一致しない場合、診断メッセージ(jwe0315i-w)が出力されます。

ただし、以下の場合は引数の型の検査は行われません。

- 仮引数が手続名または*のとき
- 実引数が外部手続名または選択戻り指定子のとき(仮引数と実引数のそれぞれが型をもつ外部手続名の場合は検査されます)

また、-Haオプション指定時には、次の検査も有効となります。

実引数および仮引数が派生型の変数の場合、派生型成分の型が一致しているかどうか検査されます。一致していない場合、診断メッセージ(jwe1561i-w)が出力されます。

引数の型が文字型の場合は、仮引数の長さが対応する実引数の長さを超えていないかどうか検査されます。長さが超えている場合、診断メッセージ(jwe0317i-w)が出力されます。

8.2.1.1.3 引数の属性の検査

-Haオプションを指定すると、仮引数とそれに対応する実引数の属性が一致しているかどうか検査されます。引数の属性が一致しない場合、診断メッセージ(jwe0314i-w)が出力されます。実引数がINTENT(IN)属性をもっているとき、対応する仮引数がINTENT(IN)属性をもっ

ていない場合、診断メッセージ(jwe0314i-w)が出力されます。実引数が式の結果または定数のとき、副プログラム内で仮引数に値を設定している文を記述していると、診断メッセージ(jwe0318i-w)が出力されます。

以下に、仮引数と実引数の対応を示します。

表8.9 仮引数と実引数の対応

仮引数	実引数
スカラ変数	スカラ変数、部分列、配列要素、定数、スカラ式の結果
文字型以外の配列	次の文字型以外の実引数: 全体配列、部分配列、配列要素、配列要素の部分列、配列式の結果
文字型の配列	次の文字型の実引数: スカラ変数、部分列、定数、スカラ式の結果、全体配列、部分配列、配列要素、配列要素の部分列、配列式の結果
仮手続	外部手続、組込み関数、モジュール手続
*	文番号

8.2.1.1.4 関数の型の検査

-Haオプションまたは-Nquickdbg=argchkオプションを指定すると、関数副プログラムで定義されている関数の型と引用元の外部手続の型が一致しているかどうか検査されます。関数の型が一致しない場合、診断メッセージ(jwe0311i-w)が出力されます。

また、関数副プログラムをサブルーチン副プログラムとして引用した場合、またはサブルーチン副プログラムを関数副プログラムとして引用した場合、診断メッセージ(jwe0330i-w)が出力されます。

-Haオプション指定時には、関数の型が文字型の場合に、長さが一致しているかどうか検査されます。関数の長さが一致しない場合、診断メッセージ(jwe0312iw)が出力されます。

8.2.1.1.5 引数の大きさの検査

-Haオプションを指定すると、仮引数が配列の場合にその大きさが対応する実引数の大きさを超えていないかどうかが検査されます。ただし、最終次元の上限が*または上下限が定数1の仮配列の場合、その大きさは対応する実引数の大きさに等しいものとみなされ、検査されません。

仮引数が文字型の形状引継ぎ配列であり、仮配列と実引数配列の長さ型パラメタが異なっている場合、診断メッセージ(jwe1571i-w)が出力されます。

仮配列の大きさが実引数の大きさを超えている場合、診断メッセージ(jwe0316i-w、jwe1577i-w)が出力されます。仮引数の配列に対応する実引数の大きさは、以下に示す大きさとなります。

表8.10 配列の大きさ

仮引数	実引数
配列	配列全体の大きさ
配列要素	指定された配列要素から配列の最後までの大きさ
配列要素の部分列	指定された配列要素の部分列の先頭から配列の最後までの大きさ
配列要素、配列要素の部分列以外の文字型スカラ式	文字型スカラ式の長さ

8.2.1.1.6 明示的引用仕様の検査

-Haオプションまたは-Nquickdbg=argchkオプションを指定すると、明示的引用仕様が必要な外部手続の引用において、明示的引用仕様が存在しない場合に診断メッセージ(jwe1569i-w)が出力されます。

8.2.1.1.7 形状引継ぎ配列の次元数の検査

-Haオプションを指定すると、仮引数が形状引継ぎ配列の場合、仮配列と実引数配列の次元数が一致しているかどうか検査されます。次元数が異なる場合、診断メッセージ(jwe1570i-w)が出力されます。

8.2.1.2 添字式および部分列範囲の検査(SUBCHK機能)

-Hsオプションまたは-Nquickdbg=subchkオプションを指定すると、引用した添字式および部分列範囲が宣言した添字式および部分列範囲と矛盾していないかという検査が行われます。ただし、以下の場合は、検査されません。

- POINTER属性をもつ実体名を引用した場合(派生型の成分がPOINTER属性をもつ場合も含まれます)
- 形状引継ぎ配列を引用した場合
- ベクトル添字で指定された部分配列を引用した場合
- 配列選別代入のブロック中にある配列を引用した場合
- 名前付き定数を引用した場合
- 部分列の親列がスカラ定数の場合
- 宣言式でデータを引用した場合
- 末端成分に割付け属性をもつ派生型の変数の場合
- 入出力DO形反復の添字範囲(-Hsオプション指定時は検査されます)
- 多相的変数を引用した場合
- 虚実部特定子の変数を引用した場合
- 長さ型パラメタをもつ派生型の変数を引用した場合

検査は、翻訳時と実行時に行われます。翻訳時に検出された場合は、w レベルの診断メッセージが出力されます。以下に実行時の検査項目について説明します。

8.2.1.2.1 配列の引用に対する検査

-Hsオプションまたは-Nquickdbg=subchkオプションを指定すると、部分列、配列要素、部分配列を引用したとき、その添字式または部分列範囲が宣言した範囲内にあるかどうか検査されます。宣言した範囲内にない場合、診断メッセージ(-Hs、-Nquickdbg=inf_detail:jwe0320i-w、-Nquickdbg=inf_simple:jwe1606i-u)が出力されます。

最終次元の上下限が定数1の仮配列の場合、大きさ引継ぎ配列として扱われます。-Nquickdbg=subchkオプションを指定し、実行時にエラーが検出された場合、大きさ引継ぎ配列の最終次元の上限は、8バイト整数型の最大値が表示されます。

また-Hsオプション指定時には、次の検査も有効となります。

形状明示配列の上限が下限より小さい値の場合、診断メッセージ(jwe1566i-w)が出力されます。

-Haオプションを同時に指定している場合は、最終次元の上限が*または上下限が定数1の仮配列に対して、配列要素や部分列が、対応する実引数の範囲内にあるかどうか検査されます。実引数の範囲内にない場合、診断メッセージ(jwe0322i-w)が出力されます。

8.2.1.2.2 部分配列の引用に対する添字のオーバフロー検査

-Hsオプションによる実行時検査では、次の検査が有効となります。

部分配列添字が添字三つ組である部分配列を引用した場合、第一番目の添字値と幅を足したとき、その値がオーバフローするどうか検査されます。オーバフローする場合、診断メッセージ(jwe1565i-w)が出力されます。

8.2.1.3 未定義データの引用の検査(UNDEF機能)

-Huオプション、-Nquickdbg=undefオプションまたは-Nquickdbg=undefnanオプションを指定すると、引用したデータが定義されているかどうか検査されます。

ただし、以下の場合については、検査されません。

- POINTER属性をもつ実体名を引用した場合(派生型の成分がPOINTER属性をもつ場合も含まれます。-Nquickdbg=undefオプションまたは-Nquickdbg=undefnanオプション指定時は検査されます)
- ALLOCATABLE属性をもつ派生型の成分
- 形状引継ぎ配列を引用した場合(-Nquickdbg=undefオプションまたは-Nquickdbg=undefnanオプション指定時は検査されます)
- ベクトル添字で指定された部分配列を引用した場合

- 配列選別代入のブロック中にあるデータを引用した場合
- 末端成分に割付け属性をもつ派生型の変数を引用した場合
- 長さ型パラメタをもつ派生型の変数を引用した場合
- 多相的変数を引用した場合
- 虚実部特定子の変数を引用した場合
- 構造体成分で末端成分の割付け変数を引用した場合(-Nquickdbg=undefオプションまたは-Nquickdbg=undefnanオプション指定時は検査されます)
- 構造体成分を引用した場合(-Huオプション、-Nquickdbg=undefオプション指定時は検査されます)
- 記憶列を共有する実体が異なる型をもつ場合 (-Huオプション、-Nquickdbg=undefオプション指定時は検査されます)
- 入出力DO 形反復の出力並び(-Huオプション指定時は検査されます)

また、-Huオプション指定時に-Hsを同時に指定した場合、SUBCHK機能の検査で、診断メッセージjwe0320i-w、jwe0322i-w、jwe1565i-w、jwe1566i-wが検出されているときは、UNDEF機能は無効となります。

以下に、未定義データの検査について説明します。

8.2.1.3.1 未定義データの検査

-Huオプションまたは-Nquickdbg=undefオプションを指定すると、モジュールまたはサブモジュールの宣言部、または共通ブロックに含まれない変数を引用した場合に、そのデータに値が定義されているかどうか検査されます。配列変数を引用したとき、配列要素ごとに検査が行われます。また、-Huオプションの指定時に派生型の変数を引用した場合、派生型の変数の成分ごとに検査が行われます。値を定義していない場合、診断メッセージ(-Hu、-Nquickdbg=inf_detail:jwe0323i-w、-Nquickdbg=inf_simple:jwe1606i-u)が出力されます。

-Huオプションを指定すると、関数副プログラムの終了時に、関数結果に値が定義されているかどうか、検査が行われます。

関数結果に値が定義されていない場合、診断メッセージ(-Hu、-Nquickdbg=inf_detail:jwe0323i-w、-Nquickdbg=inf_simple:jwe1606i-u)が出力されます。その際の行番号は関数副プログラムのEND文になります。

-Nquickdbg=undefオプションを指定すると、OpenMP仕様のPRIVATE指示節に現れたPOINTER属性をもつ変数を引用した場合、検査が行われます。そのPOINTER属性をもつ変数が結合していない場合、jwe1572i-sが出力されます。

データ値が以下の検査値と同じ値をもつ場合には、診断メッセージ(-Hu、-Nquickdbg=inf_detail:jwe0323i-w、-Nquickdbg=inf_simple:jwe1606i-u)が出力されますが、Fortranプログラムは正しいので、修正する必要はありません。以下の検査値を変更する場合は、環境変数FLIB_UNDEF_VALUEを指定してください。環境変数FLIB_UNDEF_VALUEの詳細は“3.8 実行時の環境変数”を参照してください。

1 バイトの整数型	:	-117
2 バイトの整数型	:	-29813
4 バイトの整数型	:	-1953789045
8 バイトの整数型	:	-8391460049216894069
半精度実数型	:	-2.301931e-04
単精度実数型	:	-5.37508134e-32
倍精度実数型	:	-4.696323204354320d-253
4 倍精度実数型	:	-9.0818487627532284154072898964213742q-4043
半精度複素数型	:	(-2.301931e-04,-2.301931e-04)
単精度複素数型	:	(-5.37508134e-32,-5.37508134e-32)
倍精度複素数型	:	(-4.696323204354320d-253,-4.696323204354320d-253)
4 倍精度複素数型	:	(-9.0818487627532284154072898964213742q-4043,
	:	-9.0818487627532284154072898964213742q-4043)
文字型	:	Z'8B'

8.2.1.3.2 無効演算例外による未定義データの検査

-Nquickdbg=undefnanオプションを指定すると、モジュールまたはサブモジュールの宣言部、または共通ブロックに含まれない変数を引用した場合に、そのデータに値が定義されているかどうか検査されます。また、配列変数を引用した場合、配列要素ごとに検査が行われます。値を定義していない場合、診断メッセージが出力されます。

出力される診断メッセージは、変数および関数結果の型により異なります。

整数型、文字型の検査では、診断メッセージ(-Nquickdbg=inf_detail: jwe0323i-w、-Nquickdbg=inf_simple: jwe1606i-u)が出力されます。

データ値が以下の検査値と同じ値をもつ場合には、診断メッセージ(-Nquickdbg=inf_detail: jwe0323i-w、-Nquickdbg=inf_simple: jwe1606i-u)が出力されますが、Fortran プログラムは正しいので修正する必要はありません。以下の検査値を変更する場合は、環境変数FLIB_UNDEF_VALUEを指定してください。環境変数FLIB_UNDEF_VALUEの詳細は“3.8 実行時の環境変数”を参照してください。

1 バイトの整数型	:	-117
2 バイトの整数型	:	-29813
4 バイトの整数型	:	-1953789045
8 バイトの整数型	:	-8391460049216894069
文字型	:	Z'8B'

実数型および複素数型の検査では、実数型および複素数型の変数の初期値がSNaNの値となります。これらの変数に値が定義されない場合、変数の引用時に無効演算例外(jwe0292i-u)が検出されます。

ただし、以下の場合には無効演算例外が検出されないことがあります。

- 実数型および複素数型の変数が、無効演算例外を発生させる命令のオペランドに現れない場合(どの命令が無効演算例外を発生させるかはCPUアーキテクチャに依存します。例えば、参照命令で例外を発生させないアーキテクチャでは、変数の参照では未定義変数が検出されません)
- 無効演算例外をマスクした場合

また、未定義変数の引用時以外にも、演算結果としてNaNが生じた場合にも無効演算例外が検出されます。

8.2.1.3.3 割付け変数の検査

-Huオプションを指定すると、割付け変数を引用した場合に割付け変数が割り付けられているかどうか検査されます。割り付けられていない場合、診断メッセージ(jwe0324i-w)が出力されます。

8.2.1.3.4 仮引数の実在検査

-Huオプションを指定すると、OPTIONAL属性をもつ仮引数を参照した場合にその仮引数が実在しているかどうか検査されます。実在していない場合、診断メッセージ(jwe1575i-s)が出力されます。

8.2.1.4 形状適合の検査(SHAPECHK機能)

-Heオプションを指定すると、配列式または配列式を含む代入文において形状適合であるかどうか検査されます。形状適合していない場合、診断メッセージ(jwe0329i-s)が出力されます。

8.2.1.5 拡張検査(EXTCHK機能)

-Hxオプションを指定すると、モジュール、サブモジュールの宣言部、および共通ブロックに属する変数を引用した場合に、その変数に値が定義されているかどうか、ポインタを引用した場合ポインタが結合されているかどうか、およびDO構文の増分値、FORALL構文の刻み幅、配列構成DO制御の増分値および添字三つ組の刻み幅が0以外の値かどうか検査されます。

モジュール、サブモジュールの宣言部、および共通ブロックに属する変数がデータ値として、特定の値をもつ場合、診断メッセージ(jwe0323i-w)が出力されますが、Fortranプログラムとしては正しいので、修正する必要はありません。この値については、“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照してください。

指示先と結合していないポインタを引用した場合、診断メッセージ(jwe1572i-s)が出力されます。ただし、派生型ポインタおよび成分ポインタは、この検査機能の対象になりません。

DO構文の増分値、FORALL構文の刻み幅、配列構成DO制御の増分値および添字三つ組の刻み幅が 0の場合、診断メッセージ(jwe1573i-s)が出力されます。

DO構文の範囲内の手続引用においてDO変数の値が更新されているかどうか検査されます。更新されている場合、診断メッセージ(jwe1574i-s)が出力されます。

-Hxオプションを指定した場合、副プログラム内のRETURN文またはEND文を実行すると、その有効域で局所的な変数は、SAVE属性をもつ変数を除いて、不定になります。

BIND属性を持つ変数は、その変数に値が定義されているかどうか検査されません。

-Hxオプションを指定すると、-Huオプションも同時に指定されたと解釈されます。-Huオプションの機能については、“8.2.1.3 未定義データの引用の検査(UNDEF機能)”を参照してください。

-Hxオプションを指定する場合は、以下の注意が必要です。

- 共通ブロックに属する変数に値を設定しているプログラム単位は、初期値設定プログラム単位を含め、すべて-Hxオプションを指定して翻訳しなければなりません。すべてのプログラム単位に-Hxオプションを指定していない場合、正しい結果が得られない場合があります。

例: 誤った-Hxオプションの使用

ファイルa.f90

```
BLOCK DATA INIT
  COMMON /CMN/ J
  DATA J/200/
END BLOCK DATA
```

ファイルb.f90

```
PROGRAM MAIN
  COMMON /CMN/ J
  PRINT *,J
END PROGRAM
```

翻訳コマンド

```
$ frtpx a.f90 -c
$ frtpx b.f90 a.o -Hx
```

a.f90を翻訳する際にも-Hxオプションを指定しなければなりません。

- モジュールを-Hxオプションを指定して翻訳した場合、そのモジュールを引用しているすべてのプログラム単位を、-Hxオプションを指定して翻訳しなければなりません。すべてのプログラム単位に-Hxオプションを指定していない場合、正しい結果が得られない場合があります。
- サブモジュールを-Hxオプションを指定して翻訳した場合、その先祖モジュールを引用しているすべてのプログラム単位および、そのサブモジュールの子孫のすべてのプログラム単位を、-Hxオプションを指定して翻訳しなければなりません。すべてのプログラム単位に-Hxオプションを指定していない場合、正しい結果が得られない場合があります。

例: 誤った-Hxオプションの使用

ファイルa.f90

```
MODULE MOD
  INTEGER :: J
END MODULE
```

ファイルb.f90

```
PROGRAM MAIN
  USE MOD
  J = 200
  CALL SUB
END PROGRAM
```

ファイルc.f90

```
SUBROUTINE SUB
  USE MOD
```

```
  PRINT *,J
END SUBROUTINE
```

翻訳コマンド

```
$ frtpx a.f90 -c -Hx
$ frtpx b.f90 -c
$ frtpx c.f90 a.o b.o -Hx
```

b.f90を翻訳する際にも-Hxオプションを指定しなければなりません。

8.2.1.6 重複仮引数および拡張未定義検査(OVERLAP機能)

-Hoオプションを指定すると、次の検査が行われます。

- 重複仮引数検査
- INTENT(OUT)属性をもつ、大きさ引継ぎ配列未定義引用検査
- SAVE属性をもつ変数の未定義引用検査

それぞれの検査項目について、説明します。

8.2.1.6.1 重複仮引数検査

同じ手続の2つの異なる仮引数の実体が重なっている場合、手続の実行中に重なっている部分の値を確定または不定にしてはいけません。-Hoオプションを指定すると、重なっている部分の値を確定または不定にしないか実行時に検査されます。検査を行い、重なっている部分の値を確定または不定にしている場合は、jwe1576i-wのメッセージが出力されます。

ただし、以下の場合については、検査されないことがあります。

- 配列選別代入文で仮引数の値が更新される場合
- FORALL代入文で仮引数の値が更新される場合
- DO変数である仮引数の値が更新される場合
- ベクトル添字で指定された部分配列によって仮引数の値が更新される場合
- 仮引数と結合する実引数が、連続する要素列ではない配列である場合
- 手続引用により、仮引数の値が更新される場合
- TARGET属性またはPOINTER属性をもつ仮引数と、TARGET属性またはPOINTER属性のいずれももたない仮引数の実体に重なりがあり、その仮引数の値が更新される場合
- 形状引継ぎ配列である仮引数と、ほかの仮引数の実体に重なりがあり、その仮引数の値が更新される場合

8.2.1.6.2 INTENT(OUT)属性をもつ大きさ引継ぎ配列の未定義検査

-Hoオプションを指定すると、INTENT(OUT)属性をもつ大きさ引継ぎ配列が引用された場合、値が定義されているか実行時に検査されます。検査を行い、INTENT(OUT)属性をもつ大きさ引継ぎ配列が未定義参照されている場合はjwe0323i-wのメッセージが出力されます。

ただし、-Hsオプションを同時に指定し、SUBCHK機能の検査で、診断メッセージjwe0320i-w、jwe0322i-w、jwe1565i-w、またはjwe1566i-wが検出されている場合、エラーが検出された変数の大きさ引継ぎ配列INTENT(OUT)未定義検査は無効となります。

また、以下の場合は、検査されません。

- POINTER属性をもつ変数を引用した場合
- ベクトル添字で指定された部分配列を引用した場合
- 配列選別代入のブロック中にある配列を引用した場合
- 末端成分に割付け属性をもつ派生型の変数を引用した場合
- 構造体成分で末端成分の割付け変数を引用した場合

8.2.1.6.3 SAVE属性未定義検査について

-Hoオプションを指定すると、SAVE属性をもつ変数が引用された場合、値が定義されているかどうか実行時に検査されます。検査を行い、SAVE属性をもつ変数が未定義参照されている場合はjwe0323i-wのメッセージが出力されます。

ただし、-Hsオプションを同時に指定し、SUBCHK機能の検査で、診断メッセージjwe0320i-w、jwe0322i-w、jwe1565i-w、またはjwe1566i-wが検出されている場合、エラーが検出された変数のSAVE属性未定義検査機能は無効となります。

また、以下の場合については、検査されません。

- 派生型の成分にPOINTER属性をもつ変数を引用した場合
- ベクトル添字で指定された部分配列を引用した場合
- 配列選別代入のブロック中にある配列を引用した場合
- 末端成分に割付け属性をもつ派生型の変数を引用した場合
- 構造体成分で末端成分の割付け変数を引用した場合

8.2.1.7 同時OPENおよびI/Oの再帰呼び出し検査(I/O OPEN機能)

-Hfオプションを指定すると、次の検査が行われます。

- 同時OPEN検査
- I/Oの再帰呼び出し検査

それぞれの検査項目について説明します。

8.2.1.7.1 同時OPEN検査機能

入出力文において、1つのファイルを同時に2つ以上の装置に接続してはいけません。-Hfオプションを指定すると、1つのファイルを同時に2つ以上の装置に接続されているか実行時に検査されます。検査を行い、1つのファイルが同時に2つ以上の装置に接続されている場合、jwe1231i-wのメッセージを出力します。

また、以下の場合については、検査されません。

- 検査の対象となっているファイルがローカルなファイルシステム以外で利用されているかつ、そのファイルは実名以外のファイル名で指定されている場合
- 検査の対象となっているファイルがaliasコマンドで別名が定義され、実名とは異なるファイル指定されている場合
- 検査の対象となっているファイルのパスに“./”が含まれている場合

8.2.1.7.2 I/Oの再帰呼び出し検査

入出力文を再帰実行してはいけません。-Hfオプションを指定すると、入出力文が再帰実行されているか実行時に検査されます。検査を行い、入出力文が再帰実行されている場合、jwe1232i-sのメッセージを出力します。

8.2.2 異常終了プログラムのデバッグ

プログラムの実行中に異常終了事象が発生した場合、シグナルが捕捉され異常終了の原因追及の手助けとなる情報が出力されます。

8.2.2.1 異常終了の原因

捕捉するシグナルとそれに対応するシグナルコードを以下に示します。

浮動小数点演算における例外(jwe1017i-u)は、シグナル(注)に対応するトラップを有効にして、ハンドリングします。翻訳時オプション-NRnotrapが有効な場合、浮動小数点演算における例外は検出されません。翻訳時オプション-NRnotrapについては、“2.2 翻訳時オプション”を参照してください。

> 注) Armアーキテクチャである富士通製CPU A64FXでサポートしているシグナルだけが対象です。そのため、サポートしていないシグナルは、翻訳時オプション-NRtrapを有効にしても検出できません。

実行時オプション-iを指定した場合には、以下のすべての例外が検出されません。

表8.11 捕捉シグナルと対応するシグナルコード

シグナル番号	シグナル番号の意味	シグナルコード		シグナルコードの意味
SIGILL(04) *	不当な命令の実行	1	ILL_ILLOPC	不正な命令コード
		2	ILL_ILLOPN	不正なオペランド
		3	ILL_ILLADR	不正なアドレッシングモード
		4	ILL_ILLTRP	不正なトラップ
		5	ILL_PRVOPC	不正な特権命令コード
		6	ILL_PRVREG	不正な特権レジスタ
		7	ILL_COPROC	コプロセッサエラー
		8	ILL_BADSTK	内部スタックエラー
SIGFPE(08)	浮動小数点例外 (注1)	3	FPE_FLTDIV	浮動小数点ゼロ除算
		4	FPE_FLTOVF	浮動小数点オーバフロー
		5	FPE_FLTUND	浮動小数点アンダフロー (注2)
		7	FPE_FLTINV	無効な浮動小数点演算
		―	―	浮動小数点例外
SIGBUS(10) *	記憶保護例外	1	BUS_ADRALN	無効なアドレス境界付けが無効
		2	BUS_ADRERR	実在しない物理アドレス
		3	BUS_OBJERR	オブジェクト固有のハードウェアエラー
SIGSEGV(11) *	セグメンテーション例外	1	SEGV_MAPERR	アドレスがオブジェクトにマップされていないアドレス
		2	SEGV_ACCERR	マップされたオブジェクトへの許可が無効
SIGXCPU(30) *	CPU占有時間による打切り	―	―	―

注1) Armアーキテクチャである富士通製CPU A64FXでは、FPE_INTDIV(除数が0の場合の整数除算例外)は検出されません。詳細は“A.2.3 除数が0の場合の整数除算例外について”を参照してください。

注2) 翻訳時オプション-NRtrapおよび環境変数FLIB_EXCEPT=u(浮動小数点アンダフロー割込み検出指示)を指定した場合に捕捉されるシグナルです。

備考. *印のシグナルは、実行時オプション-i(本処理系でシグナルを捕捉しない指示)を指定した場合にcoreファイルが生成されます。

8.2.2.2 異常終了時の出力情報

捕捉されたシグナルに応じて、以下の情報が標準エラー出力ファイルに出力されます。標準エラー出力ファイルへ出力できない場合には、端末装置に出力されます。

8.2.2.2.1 一般的な異常終了時の出力情報

SIGILL、SIGBUS、またはSIGSEGVのどれかを捕捉した場合に出力されます。これらの情報に続いてトレースバックマップが出力されます。

```
jwe0019i-u The program was terminated abnormally with signal number SSSSSSS.
           signal identifier = NNNNNNNNNN, (Detailed information.)
```

SSSSSSS: SIGILL、SIGBUS、またはSIGSEGVを示します。

NNNNNNNNNN: 異常終了の要因となったシグナルコードを示します。

(Detailed information.): 上記シグナルコード*NNNNNNNNNN*に対応する詳細情報を示します。

SIGBUSまたはSIGSEGVの原因がスタックオーバフローである可能性が高い場合、以下の情報を付加します。

```
The cause of this exception may be stack-overflow
```

8.2.2.2.2 SIGXCPUの出力情報

SIGXCPUが捕捉された場合、以下の情報が出力されます。

```
jwe0017i-u The program was terminated with signal number SIGXCPU.
```

8.2.2.2.3 実行時オプション-aを指定した場合の出力情報

実行時オプション-aを指定し、SIGIOTを捕捉した場合に出力されます。プログラム実行終了時にプログラムが使用していた一時ファイルは削除されずに強制的に異常終了され、coreファイルが生成されます。

```
jwe0018i-u The program was terminated abnormally due to runtime option -a with signal number SIGIOT.
```

8.2.2.2.4 異常終了処理中に再び異常終了が発生した場合の出力情報

異常終了処理中に再び異常終了が発生した場合、以下の情報が出力されます。

```
jwe0020i-u An error was detected during an abnormal termination process.
```

8.2.3 フック機能

フック機能は、ユーザ定義サブルーチンをFortranプログラムの特定箇所から、または、ユーザ定義関数を一定時間間隔で呼び出す機能です。ユーザ定義サブルーチンにおいてトレース情報を出力することで、またはユーザ定義サブルーチンおよびユーザ定義関数において特定の変数の値を出力することで、プログラムの動作を確認することができます。

ここでは、フック機能について説明します。

8.2.3.1 ユーザ定義サブルーチンの形式

ユーザ定義サブルーチン名は固定(USER_DEFINED_PROC)となります。

ユーザ定義サブルーチンは、以下の形式で定義してください。

【Fortran】

形式

```
SUBROUTINE USER_DEFINED_PROC(FLAG, NAME, LINE, THREAD)
INTEGER(KIND=4),INTENT(IN):: FLAG, LINE, THREAD
CHARACTER(LEN=*),INTENT(IN):: NAME
```

引数

FLAG: ユーザ定義サブルーチンの呼び出し元を示します。

FLAG	:	ユーザ定義サブルーチンの呼び出し元を示します。
= 0	:	プログラムの入口
= 1	:	プログラムの出口
= 2	:	手続の入口
= 3	:	手続の出口
= 4	:	パラレルリージョン(OpenMP/自動並列化)の入口
= 5	:	パラレルリージョン(OpenMP/自動並列化)の出口
= 6	:	一定時間間隔
= 7～99	:	システムリザーブ
= 100～	:	利用者側で利用可能
NAME	:	呼び出し元の手続名を示します。
		NAMEは、FLAGが2、3、4、5、または100以上の場合のみ参照可能です。
LINE	:	呼び出し元の行番号を示します。

LINEは、FLAGが2、3、4、5、または100以上の場合のみ参照可能です。

THREAD　：　OpenMP、自動並列化でのスレッド並列において、ユーザ定義サブルーチンを呼び出したスレッドの識別番号を示します。

THREADは、FLAGが2、3、4、5、または100以上の場合のみ参照可能です。

8.2.3.2 フック機能の注意事項

フック機能の使用時には、以下の注意事項があります。

- 手続名“USER_DEFINED_PROC”をほかの用途に使用してはなりません。
- ユーザ定義サブルーチンが定義されていない場合、動作は保証されません。
- ユーザ定義サブルーチンの引数に値を設定してはなりません。
- -Nnolineオプションを指定した場合、引数LINEの値は保証されません。
- スレッド並列化されたプログラムでは、複数のスレッドからユーザ定義サブルーチンが呼び出されることがあります。
- -CcI4I8オプションを指定してユーザ定義サブルーチンを翻訳した場合、動作は保証されません。
- -AUオプションを指定してユーザ定義サブルーチンを翻訳した場合、動作は保証されません。
- ENTRY文を含む手続では、引数NAMEに設定される手続名はENTRY文の入口名となる場合があります。
- 手続の出口は、END文の行番号になります。
- ユーザ定義サブルーチンの手続の特性は変更できません。
- ユーザ定義関数の関数入口/出口は、プログラム内の任意箇所からの呼出しを除き、ユーザ定義関数の呼出元になりません。
- ユーザ定義サブルーチン内で、直接的および間接的にSTOP文、ERROR STOP文、EXITサービスサブルーチン、およびSETRCDサービスサブルーチンを実行した場合、動作は保証されません。
- ユーザ定義サブルーチン内で使用可能なOpenMP仕様は、以下のOpenMPサービスサブルーチンだけです。
  - 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 特定箇所からのユーザ定義サブルーチン呼び出し

-Nhook_funcオプションをオブジェクトプログラムおよび実行可能プログラムの作成時に指定した場合、以下の箇所からユーザ定義サブルーチンが呼び出されます。

・ プログラムの入口/出口

・ 手続の入口/出口

また、-Kopenmpまたは-Kparallelオプションが有効な場合、上記の箇所に加えて以下の箇所からもユーザ定義サブルーチンが呼び出されます。

・ パラレルリージョン(OpenMP/自動並列化)の入口/出口

OpenMP、自動並列化の入口/出口では、引数NAMEに次の内部手続名が渡されます。

・ OpenMP “_OMP_識別番号_”

・ 自動並列化 “_PRL_識別番号_”

FortranプログラムとCプログラムの結合プログラムにおいて、-Nhook_funcオプションが指定された場合の動作を以下に示します。

・ FortranプログラムとCプログラムにおいて、それぞれのプログラム中にユーザ定義サブルーチンまたはユーザ定義関数が定義されている場合、Fortranプログラムからは Fortranプログラム中で定義されたユーザ定義サブルーチンが、Cプログラムからは Cプログラム中で定義されたユーザ定義関数が呼び出されます。

・ FortranプログラムとCプログラムにおいて、どちらか一方のプログラム中にのみユーザ定義サブルーチンまたはユーザ定義関数が定義されている場合、定義されたユーザ定義サブルーチンまたはユーザ定義関数はFortranプログラムとCプログラムの両方から呼び出されます。

特定箇所でのフック機能の使用例を以下に示します。

ユーザ定義サブルーチンのプログラム例

ファイルhook.f95

```
WRITE(*,*) 'HELLO'
CALL SUB
END

SUBROUTINE SUB
WRITE(*,*) 'SUB'
END SUBROUTINE

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 SUBROUTINE
```

実行可能プログラムの作成

```
$ frtpx -Nhook_func hook.f95
```

実行結果

```
$ ./a.out
 PROGRAM START
 HELLO
 PROC START: SUB LINE:  5
 SUB
 PROC END: SUB LINE:  7
 PROGRAM END
$
```

8.2.3.3.1 特定箇所からの呼び出しにおける注意事項

-Nhook_funcオプションの指定時には、以下の注意事項があります。

- 計算量の少ない手続が繰り返し呼び出されるような場合、手続の入口/出口からのユーザ定義サブルーチン呼び出しは、実行性能に影響を与える可能性があります。
- 計算量の少ないパラレルリージョンが繰り返し実行されるような場合、パラレルリージョンの入口/出口からのユーザ定義サブルーチン呼び出しは、実行性能に影響を与える可能性があります。
- 入出力並びから呼び出される手続からは、ユーザ定義サブルーチンは呼び出されません。

8.2.3.4 一定時間間隔でのユーザ定義関数呼び出し

-Nhook_timeオプションを実行可能プログラムの作成時に指定した場合、一定ユーザ時間の経過ごとにユーザ定義関数が呼び出されます。

ユーザ定義関数の呼び出し間隔は、環境変数FLIB_HOOK_TIMEにより指定可能です。環境変数FLIB_HOOK_TIMEを指定しない場合は、1分ごとにユーザ定義関数が呼び出されます。環境変数FLIB_HOOK_TIMEについては、“3.8 実行時の環境変数”を参照してください。

ユーザ定義関数については、“C言語使用手引書”または“C++言語使用手引書”を参照してください。

8.2.3.4.1 一定時間間隔での呼び出しにおける注意事項

-Nhook_timeオプションの指定時には、以下の注意事項があります。

- 環境変数FLIB_HOOK_TIMEの指定は、プログラムの実行途中に変更してはなりません。
- 環境変数FLIB_HOOK_TIMEに0～2147483647以外の値を指定した場合、動作は保証されません。
- ユーザ定義関数の呼び出し間隔を短くした場合、ユーザ定義関数呼び出しが、実行性能に影響する可能性があります。
- C/C++プログラムにおいて、ユーザ定義関数を記述しなければなりません。

また、一定時間間隔での呼び出しでは、非同期シグナルからユーザ定義関数が呼び出されるため、次の制限事項があります。

- ユーザ定義関数では、非同期シグナルセーフ関数を除く関数の使用はできません。
- -Nhook_timeオプションを指定した場合、SIGVTALRMに対応するシグナル処理ルーチンを定義してはなりません。
- -Nhook_timeオプションを指定した場合、ALARM(3)サービス関数を使用してはなりません。
- -Nhook_timeオプションを指定して作成された実行ファイルに対して、プロファイラを使用してはなりません。

8.2.3.5 プログラム内の任意箇所からの呼び出し

引数(FLAG:100～、NAME、LINE、THREAD)を設定して、プログラム中の任意箇所からユーザ定義サブルーチンを呼び出すことが可能です。

第9章 最適化機能

この章では、最適化機能を有効に活用するための機能や、注意事項について説明します。

9.1 最適化の概要

最適化の目的は、プログラムを可能な限り高速に実行できるようなオブジェクトプログラム(命令とデータ)を生成することです。

最適化機能を使用することにより、プログラムの実行時間を短縮することができます。

ただし一部機能によっては、コンパイラが出力するオブジェクトプログラム(命令とデータ)の大きさが大幅に増加します。そのため、実行時に必要とする記憶領域の大きさが増加する場合があります。

このため、最適化機能を使用するかどうかは、翻訳時オプションにより選択できるようになっています。

最適化オプションについては"2.2 翻訳時オプション"を参照してください。

9.1.1 標準的な最適化

標準的な最適化は最適化レベル1から3で行われます。

-Koptmsg=2オプションを有効にすると、各最適化が行われたかどうかを翻訳時の診断メッセージで知ることができます。

9.1.1.1 共通式の除去

等しい演算結果をもたらす2つの式(共通式)が存在する場合、あとの式では演算せずに前の式の演算結果を利用します。

以下に、共通式の除去の例を示します。

例:共通式の除去

[最適化後(ソースイメージ)]

```
...
A=X*Y+C
...
B=X*Y+D
```

→

```
T=X*Y
A=T+C
...
B=T+D
```

[説明]

代入式の右辺の一部であるX*Yが共通式です。2度目の計算を除去して最初の計算結果Tを使用するように変更します。Tはコンパイラが生成した変数です。

9.1.1.2 不変式の移動

ループ内で値が変化しない式(不変式)を、ループの外側に移動します。

以下に不変式の移動の例を示します

例:不変式の移動

[最適化後(ソースイメージ)]

```
DO I=1,N
  Y=A(J)*2
  X=X+Y*Z
ENDDO
```

→

```
Y=A(J)*2
T=Y*Z
DO I=1,N
  X=X+T

ENDDO
```

[説明]

文全体のY=A(J)*2および式の一部のY*Zをループ外に移動します。Tはコンパイラが生成した変数です。

この最適化により移動するのは、通常はループ内で必ず実行される文の一部または全体です。ただし、-Kpreexオプションを指定した場合は、判定文などにより、ループ内で選択的に実行される文に含まれる不変式も移動します。この最適化を、通常の不変式の移動とは区別して、不変式の先行評価と呼びます。不変式の先行評価を実施することにより、より実行時間を短縮することができます。しかし、移動による副作用が生じる場合があります

不変式の先行評価による副作用については、“9.1.2.2 不変式の先行評価”を参照してください。

9.1.1.3 演算子の強さの縮小

演算子の強さとは、演算に要する実行時間の相対的な大小関係をいいます。強い演算子ほど実行時間を多く必要とします。一般に、加算と減算の強さは同じです。乗算は加減算よりも強く、除算は乗算よりも強くなります。また、整数型と浮動小数点型の間の型変換は、加減算よりも強く乗算より弱くなります。演算子の強さの縮小とは、“乗算は加減算に”というように、演算子の強さをより弱い方向に変更することにより、実行時間を短縮する最適化のことです。

以下に強さの縮小の例を示します。

例:乗算から加算への縮小

[最適化後(ソースイメージ)]

```
DO I=1,10
  ...
  J=K+I*100
  ...
  ...
ENDDO
```

→

```
T=100
DO I=1,10
  ...
  J=K+T
  ...
  T=T+100
ENDDO
```

[説明]

演算I*100に対して最適化を行い、ループ内の演算I*100を加算T=T+100に変換します。

Tはコンパイラが生成した変数です。

9.1.1.4 ループアンローリング

ループアンローリングとは、ループ内のすべての実行文をループ内へn重に展開し、ループの繰返し数をn分の1にする最適化です。ただし、翻訳時オプション-Ksimd[=level]が有効でループ内がSIMD化された場合、実行文はSIMD長*n重に展開され、ループの繰返し数はSIMD長*n分の1に縮小されます。

ループアンローリングは、ループ外からの飛込みもループ外への飛出しもないループに対して動作します。

展開数nは、ループの繰返し数、ループ内の実行文の数と種類および使われているデータの型などから、コンパイラが最適な値を決定します。また、最適化制御行を使って、ソースプログラム上に展開数を指定することもできます。最適化制御行については、“9.10 最適化制御行(OCL)の利用”を参照してください。

繰返し数が縮小され、かつ多重に展開された実行文がループ内で一体になって最適化されるので、高速なオブジェクトプログラムが得られます。ただし、ループ内の文が多重に展開されるので、オブジェクトプログラムの大きさが増加します。

ループアンローリングはSIMD化前後で動作します。

- SIMD化前
  外側ループでSIMD化やソフトウェアパイプライニングを適用するために、内側ループの繰返し数が小さい場合に内側ループをフルアンローリングします。これをSIMD化前のフルアンローリングと呼びます。
- SIMD化後
  共通式などの最適化の促進のために、内側ループをアンローリングまたはフルアンローリングします。

ループアンローリングは、最適化レベル2(-O2)以上の場合に行われます。

翻訳時オプション-Kunroll[=n]または最適化指示子UNROLLが有効なループでは、ループアンローリングはSIMD化前後で動作します。

翻訳時オプション-Knounrollまたは最適化指示子NOUNROLLが有効なループでは、ループアンローリングは抑止されます。

SIMD化前のフルアンローリングは、最適化指示子FULLUNROLL_PRE_SIMDまたはNOFULLUNROLL_PRE_SIMDを使って動作を制御できます。-Kunroll[=*n*]オプション、-Knounrollオプション、最適化指示子UNROLL、または最適化指示子NOUNROLLが有効なルー

プに、最適化指示子FULLUNROLL_PRE_SIMDまたはNOFULLUNROLL_PRE_SIMDが指定された場合、SIMD化前のフルアンローリングでは最適化指示子FULLUNROLL_PRE_SIMDまたはNOFULLUNROLL_PRE_SIMDが優先されます。

翻訳時オプション-Koptmsg=2を指定して、最適化されたDOループとその展開数を示すメッセージを出力することができます。

配列代入および配列式を含む文(配列記述)は、DOループに展開され、ループアンローリングの対象になります。ただし、-Koptmsg=2によるメッセージは出力されません。

9.1.1.5 ループブロッキング

ループブロッキングとは、ループを多重化して、配列へのアクセスを細分化させる最適化です。これによってアクセスの局所性を高められるため、キャッシュの効率的な利用が期待できます。

ループブロッキングは、ループ外からの飛込みもループ外への飛出しもないループに対して行われます。

ループブロッキングは、最適化レベル2以上の場合に行われます。-Kloop_noblockingオプションを指定して、抑止することができます。-Koptmsg=2オプションを指定すると、最適化されたループを示すメッセージを出力することができます。

例:ループブロッキング

```
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
```

→

[最適化後(ソースイメージ)]

```
DO KK=1,NK,MK+1
  DO JJ=1,NJ,MJ+1
    DO II=1,NI,MI+1
      DO K=KK,MIN(NK,KK+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 ソフトウェアパイプライニング

ソフトウェアパイプライニングとは、命令レベルの並列化が可能なマシンにおいてループ内の命令ができるだけ並列実行されるように、命令を並べ替える最適化です。

ソフトウェアパイプライニングは、翻訳時オプション-Knoswpを指定して、抑止することができます。

ソフトウェアパイプライニングは、ループの任意の回転における実行文とそれ以降の回転における実行文を重ね合わせた命令スケジューリングを行い、ループの形状を変更します。そのため、ループには十分な繰返し数が必要になります。必要な繰返し数は、ソフトウェアパイプライニングを適用するループ内の命令列によって決まるため、他の最適化の影響を受けます。また、元のループの繰返し数が小さい場合、ループ形状を変更するために必要な繰返し数を小さくするような命令スケジューリングを行うため、ソフトウェアパイプライニングによる効果が小さくなる場合があります。

繰返し数が変数のループに対してソフトウェアパイプライニングが適用された場合、ループの繰返し数が必要な繰返し数に満たない場合を考慮し、下記の例に示すように、ソフトウェアパイプライニングを適用したループまたは元のループを選択する分岐構造を生成します。どちらのループを選択するかは、実行時に判断されます。また、オブジェクトプログラムの大きさが増加します。

例:

```
SUBROUTINE SUB(N)
INTEGER(KIND=4) :: N
(元のループ：N回転)
RETURN
END
```

[最適化後(ソースイメージ)]

```
SUBROUTINE SUB(N)
INTEGER(KIND=4) :: N
IF ( Nは必要な繰返し数以上か？) THEN
```

```
  （ソフトウェアパイプライニングを適用したループ）
ELSE
  （元のループ：N回転）
ENDIF
RETURN
END
```

翻訳時オプション-Koptmsg=2を指定して、ループに対するソフトウェアパイプライニングの適用情報を出力することができます。ソフトウェアパイプライニングが適用された場合、翻訳時メッセージjwd8204o-iおよびjwd8205o-iが同時に出力されます。ソフトウェアパイプライニングが適用されなかった場合、その理由を表示する翻訳時メッセージjwd8662o-i～jwd8674o-iのいずれかが出力されます。翻訳時オプション-Knoswpを指定している場合、およびソフトウェアパイプライニングを適用する前にフルアンローリングなどの最適化によりループがなくなった場合は、ソフトウェアパイプライニングに関するメッセージは出力されません。

翻訳時メッセージjwd8205o-iは、ソフトウェアパイプライニングを適用したループが実行時に選択されるために必要な、ソースプログラム上のループの繰返し数を表示します。また、ソフトウェアパイプライニングは最内ループを対象とするため、いくつかの最適化が適用された後では、-Koptmsg=2で出力される最適化の適用状況を加味して判断する必要があります。具体的には、ループ一重化、ループ交換、スレッド並列化、インライン展開が動作した場合には、ソフトウェアパイプライニングが対象とする最内ループの繰返し数(翻訳時メッセージjwd8205o-iで表示される繰返し数と比較すべき繰返し数)は以下のように変更されるため注意が必要です。

同時に出力される 最適化メッセージ	メッセージ番号	ソフトウェアパイプライニングが対象とする 最内ループの繰返し数
ループ一重化	jwd8330o-i	一重化された複数のループの繰返し数の積
ループ交換	jwd8211o-i	ループ交換された結果、最内となったループの繰返し数
スレッド並列化	jwd5001p-iなど	繰返し数を並列化スレッド数で割った数
インライン展開	jwd8101o-i	インライン展開された結果、最内となったループの繰返し数

翻訳時メッセージjwd8205o-iで表示される繰返し数は、ループのSIMD化により同時に実行される繰返し数を加味した値です。ただし、翻訳時オプション-Ksimd_reg_size=agnosticが有効な場合、SVEのベクトルレジスタサイズが翻訳時には不明であるため、SVEのベクトルレジスタサイズを128ビットとみなした値になります。

9.1.1.7 SIMD化

ここでは、SIMD化について説明します。

9.1.1.7.1 一般的なSIMD化

SIMD化とは、複数の同種の演算を同時実行するSIMD命令への変換を行う最適化です。

SIMD化は、翻訳オプション-Knosimdを指定して、抑止することができます。翻訳時オプション-Koptmsg=2を指定して、最適化されたループを示すメッセージを出力することができます。

9.1.1.7.2 IF構文を含むループのSIMD化

-Ksimd={ 2 | auto }を指定して、IF構文を含むループをSIMD化することができます。特に、IF構文の条件の真率が高い場合には性能向上が期待できます。性能向上がIF構文の条件の真率が高い場合に限定される理由は、CPUアーキテクチャで用意されている条件付き命令が、条件付きストア命令と条件付き転送命令のみであり、それ以外の命令は投機実行(IF構文内の命令を条件が偽の時も実行すること)されるからです。

-Ksimd=2では、命令を投機実行するため、実行時の例外発生など実行結果に副作用を生じる場合があります。

9.1.1.7.3 リストベクトル変換

IF構文の真率が低くかつIF構文内の命令数が多いループに対しては、リストベクトル変換を適用することで性能向上できる可能性があります。リストベクトル変換とは、IF構文の条件を満たすループ制御変数の値を配列に登録するループと、その登録した配列のサイズ分だけ回転して元のIF構文内の命令を実行するループへの変換です(下記例を参照)。変換した後半のループはリストアクセスのループとなりますが、SIMD化およびソフトウェアパイプライニングの対象となります。

以下にリストベクトル変換の変換例を示します。リストベクトル変換は最適化制御行で動作します。詳細は、“9.10.4 最適化指示子”の“! OCL SIMD_LISTV[({ALL | THEN | ELSE})]”の説明を参照してください。

例:

```
DO I=1, N
!OCL SIMD_LISTV
  IF (M(I)) THEN
    A(I)=B(I)+C(I)
  END IF
END DO
```

[最適化後(ソースイメージ)]

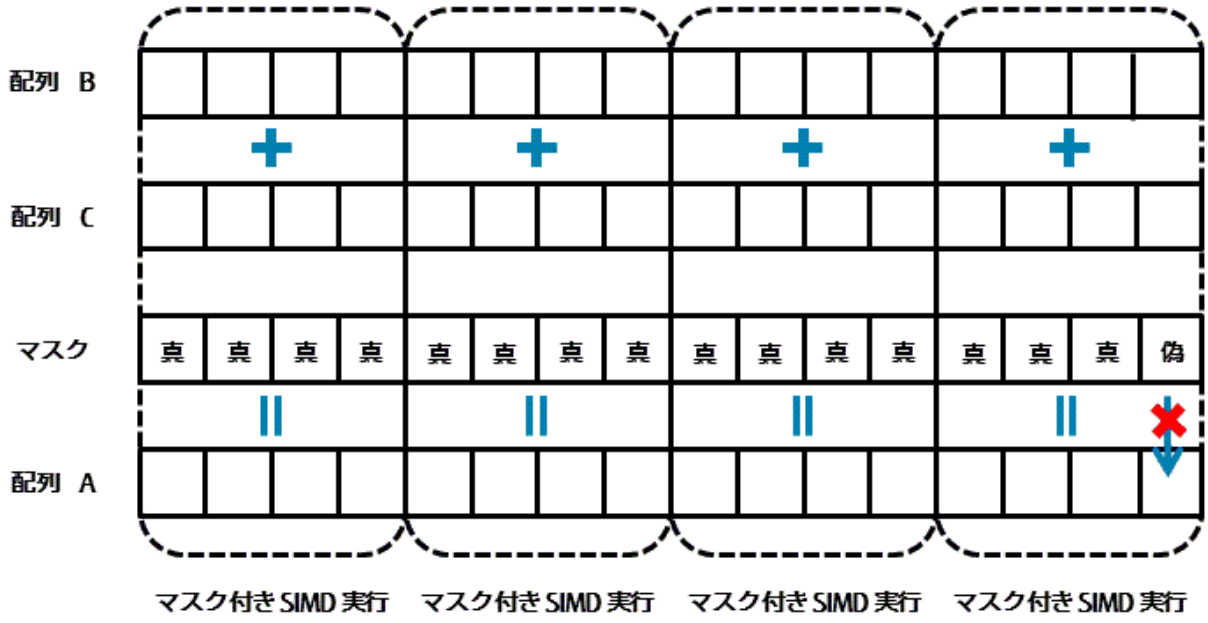
```
J=1
DO I=1,N
  IF (M(I)) THEN
    IDX(J)=I
    J=J+1
  END IF
END DO
!OCL NORECURRENCE
DO K=1,J-1
  I=IDX(K)
  A(I)=B(I)+C(I)
END DO
```

9.1.1.7.4 SIMD長の倍数冗長実行によるSIMD化機能

本機能は、ループの繰返し数がSIMD長で割り切れない場合でも、マスク付きSIMD命令を利用することで、その端数部分を含めてSIMD実行するループを生成します。

本機能の動作イメージを下図に示します。全回転がマスク付きでSIMD実行されます。

図9.1 本機能有効時の動作(4SIMDイメージ)



なお、繰返し数をSIMD長で割った余りが1や2となることが多いループでは、最適化の効果が得られない可能性があります。

本機能を抑止したい場合は、最適化制御行に最適化指示子SIMD_NOREDUNDANT_VLを指定してください。

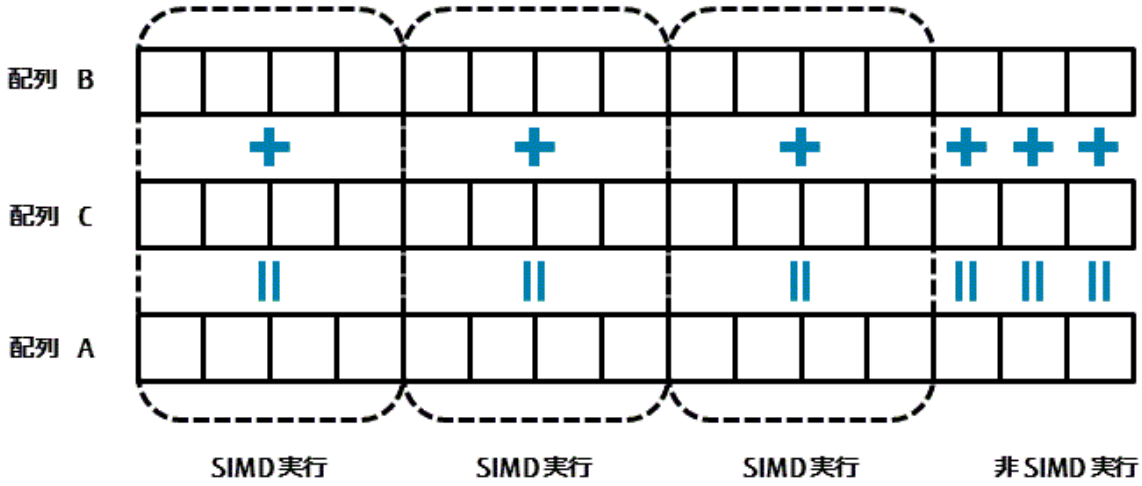
例: 本機能を抑止する最適化制御行(OCL)の指定

```
!OCL SIMD_NOREDUNDANT_VL
  DO I=1,M
```

```
    A(I)=B(I)+C(I)
END DO
```

このプログラム例の動作イメージを下図に示します。SIMD長で割り切れない端数部分I=13～15は非SIMD実行されます。

図9.2 本機能抑止時の動作(M=15の4SIMDイメージ)



また、-Kextract_stride_storeオプションを指定した場合にも本機能が抑止されます。

9.1.1.7.5 SIMD化可能な組込み関数

以下にSIMD化可能な組込み関数を示します。

数値関数

int, int4, jfix, int1, int2, real, dble, 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

数学関数

sqrt, cbrt, 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

ビット関数

not, iand, and, ior, or, ieor, xor, ishft, ishftc, lshift, shiftl, rshift, shifta, lrshft, shiftr, isha, ishc, ishl, ibset, ibclr, btest, izext, izext2, jzext, jzext2, jzext4, ble, blt, bge, bgt, poppar

関数の型や引数の型などによりSIMD化できない場合があります。

SIMD化可能な組込み関数については、“2.2 翻訳時オプション”の-Kilfuncオプションおよび-Kmfuncオプションの説明も参照してください。

9.1.1.8 ループアンスイッチング

ループアンスイッチングとは、ループ内に不変な条件で分岐するIF構文が存在する場合に、そのIF構文をループの外に出し、IF構文の条件が成立した場合のループと成立しない場合のループを作成する最適化です。ループ内にあるIF構文がループの外に追い出されることで、命令スケジューリングなどの最適化が促進されます。

ループアンスイッチングは、最適化レベル3、または翻訳時オプション-Kparallelが有効な場合に行われます。最適化レベル2以下を指定し、かつ翻訳時オプション-Kparallelを無効にすると、抑止することができます。-Koptmsg=2オプションを指定すると、最適化されたループを示すメッセージを出力することができます。

例:

[最適化後(ソースイメージ)]

```
SUBROUTINE SUB(A,B,C,N)
REAL*8 A(N)
DO I=1,N
  IF (B .EQ. C) THEN
    A(I) = 0.0D0
  ELSE
    A(I) = 1.0D0
  ENDIF
ENDDO
END SUBROUTINE
```

→

```
SUBROUTINE SUB(A,B,C,N)
REAL*8 A(N)
IF (B .EQ. C) THEN
  DO I=1,N
    A(I) = 0.0D0
  ENDDO
ELSE
  DO I=1,N
    A(I) = 1.0D0
  ENDDO
ENDIF
END SUBROUTINE
```

9.1.2 拡張された最適化

拡張最適化機能のオプションが指定されると、標準的な最適化に加えて、拡張された最適化が行われます。

ここでは、拡張された最適化の代表的な機能について説明します。

拡張最適化の機能項目によっては、コンパイラが出力するオブジェクトプログラム(命令とデータ)の大きさが大幅に増加したり、実行結果に副作用が生じたりする場合があります。

9.1.2.1 インライン展開

インライン展開とは、利用者定義の手続を引用箇所に展開する最適化です。この機能は-x*inl_arg*オプションで制御することが出来ます。-x*inl_arg*オプションの詳細は、“2.2.3 オプションの説明”を参照してください。

利用者定義の手続をインライン展開した場合、プログラム単位当たりのオブジェクトプログラムの大きさが増加する量は、下記の式で示すことができます。

(命令列の大きさ＋データ部の大きさ)×展開の回数

命令列の大きさは、実行文の種類と数から決まります。

データ部の大きさは、共通ブロック、仮引数、割付け変数、親子結合された要素を除く変数、配列および定数の大きさの合計です。

ある利用者定義の手続が同じプログラム単位内で何度もインライン展開された場合、データ部は1回分の大きさだけ増加します。

ある利用者定義の外部手続またはモジュール手続について、翻訳された原始プログラム内のすべての引用(呼出し)がインライン展開されたとしても、利用者定義の外部手続またはモジュール手続は削除されず、そのオブジェクトプログラムは、常に出力されます。これは、他のプログラム単位から呼び出される可能性があるためです。

しかし、利用者定義の内部手続については、翻訳された原始プログラム内のすべての引用(呼出し)がインライン展開された場合、利用者定義の内部手続は削除され、そのオブジェクトプログラムが出力されません。

翻訳時オプション-Koptmsg=2を指定することにより、インライン展開された場所と手続名を示すメッセージを出力することができます。

9.1.2.2 不変式の先行評価

翻訳時オプション-Kpreexを指定した場合、最適化(不変式の先行評価)によりプログラムの論理上実行されないはずの命令が実行され、エラーになる(実行が中断される)場合があります。

ただし、計算結果の精度に影響を与えることはありません。

実行場所の移動による影響は、組込み関数の引用、配列要素の引用および除算で発生する可能性があります。

以下に、実行場所の移動による影響の例を示します。

例1: 組込み関数の引用

[最適化後(ソースイメージ)]

```
DIMENSION A(100)
...

DO I=1,100
```

→

```
DIMENSION A(100)
...
T=ALOG(X)
DO I=1,100
```

```
   IF(X.GT.0.0) THEN
     A(I)=ALOG(X)
   END IF
ENDDO
...
```

```
   IF(X.GT.0.0)THEN
     A(I)=T
   END IF
ENDDO
...
```

[説明]

左側のプログラムでは、alogの引数が0.0より大きい場合に関数を呼び出すようにしているため副作用は生じません。右側のプログラムでは、最適化(不変式の先行評価)により、alog(x)がループ外に移動することで必ず実行されます。そのため、変数xの値が正でない場合、組込み関数alogで引数が不当である旨のエラーメッセージが出力されます。

例2: 配列要素の引用

```
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
...
```

→

[最適化後(ソースイメージ)]

```
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
...
```

[説明]

左側のプログラムでは、変数jがn以下のときだけ配列要素b(j)を引用しているので、配列bの宣言範囲を超えることはありません。右側のプログラムでは、最適化(不変式の先行評価)により、配列要素b(j)がループ外に移動されて、jの値に関係なく引用されます。その結果、配列の宣言範囲を超えて引用されるようになります。プログラムの範囲外を引用すると、記憶保護例外(読出し)が発生した旨のエラーメッセージが出力されて、プログラムの実行が中断します。

例3: 除算

```
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
...
```

→

[最適化後(ソースイメージ)]

```
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
...
```

[説明]

左側のプログラムでは、変数fが0.0でないときだけ除算を行っているので、除算例外のエラーが発生することはありません。右側のプログラムでは、最適化(不変式の先行評価)により、除算b/fがループ外に移動されて、fの値に関係なく実行されます。その結果、もし変数fの値が0.0であれば、除算例外が発生した旨のエラーメッセージが出力されて、プログラムの実行が中断します。

-NRtrapオプションを同時に指定した場合、本来は発生しない浮動小数点演算の割込み事象を検出する可能性があります。

9.1.2.3 演算評価方法の変更

翻訳時オプション-Kevalを指定した場合、演算評価方法の変更の最適化が行われ、計算結果に誤差を生じる場合があります。また、指数けたあふれなどの演算例外も発生する場合があります。除算の乗算化は、浮動小数点型の変数を使用する除算に対して行われます。

以下に、除算の乗算化による影響の例を示します。

例: 除算の乗算化

```
DIMENSION A(100),B(100)
...

DO I=1,100
   A(I)=B(I)/C
ENDDO
...
```

→

[最適化後(ソースイメージ)]

```
DIMENSION A(100),B(100)
...
T=1.0/C
DO I=1,100
   A(I)=B(I)*T
ENDDO
...
```

[説明]

変数cはループ内で一定です。最適化(除算の乗算化)により、変数cの逆数をループの外で計算し、ループ内の除算は逆数による乗算に変更されます。すなわち、1つの除算が除算と乗算の2つの演算によって計算されるようになります。t=1.0/cの計算結果が、変数cの型の精度内に収まらない場合は、下位のけたが切り捨てられ、最適化を行わない場合に対して精度差が生じます。また、変数cの値によっては、ループの外で逆数を計算するとき、指数けたあふれの演算例外が発生する場合があります。

9.1.2.4 ループストライピング

ループストライピングとは、ループ内の実行文を一定数(この数をストライプ長と言います)だけ展開する最適化です。ループストライピングを適用することにより、ループの繰返しによるオーバヘッドが少なくなる、または命令スケジューリングが促進されるなどの効果が期待できます。

例:

```
DO I=1,N
  A(I) = B(I) + C(I)
ENDDO
```

ストライプ長4(-Kstriping=4)で展開することが指定された場合、以下の例のように実行文がループ内で展開されます。

```
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
```

ループストライピングは、ループ内の文が多重に展開されるので、ループアンローリング同様、オブジェクトプログラムの大きさが増加します。また、多くの翻訳時間を必要とする場合があります。実行性能に関しても、使用するレジスタが増加するため実行性能が低下する場合があるので注意が必要です。

ループストライピングは、翻訳オプション -Knostriping を指定することにより、抑止することができます。

また、翻訳時オプション -Koptmsg=2 を指定することにより、最適化されたループを示すメッセージが出力されます。

9.1.2.5 zfill

zfillとは、データをメモリからロードすることなく、キャッシュ上に書き込み用の領域を確保する命令(DC ZVA)を使い、データを高速にストアする最適化です。zfillは、ループ内でストアされる配列データに対して適用されます。

ただし、同一ループ内に参照がある配列、非連続アクセスされる配列、またはIF構文配下でストアされる配列に対しては最適化されません。また、zfillが適用された場合、2次キャッシュへのプリフェッチ命令は出力されません。

zfillの最適化は対象となるストア命令の指すアドレスを起点として、256バイトを1ブロックとする単位で*N*ブロック分先のデータを最適化の対象とします。*N*はオプションまたは最適化制御行で指定する1から100までの整数値です。*N*の指定を省略した場合、コンパイラが自動的に値を決定します。

本最適化は確保した領域が必ずストアされるようなループ変形を行うため、以下の最適化が適用できなくなります。

- ループアンローリング
- ループストライピング

また、以下の場合にも、実行性能が低下することがあります。

- メモリバンド幅のボトルネック影響を受けていないプログラム
- 繰返し数が小さいループ
- -Kzfill=*N*オプションでブロック数を指定している、かつループによって書き込まれるメモリ領域のサイズが*N*ブロックよりも小さい場合

-Kzfillオプションの指定によって実行性能の低下が起きる場合は、-Kzfillオプションを指定しないでください。zfillの制御はループ単位で行うことが望ましいため、オプション指定でプログラム全体を最適化の対象にせず、最適化指示子ZFILLで指定することを推奨します。

9.1.2.6 ループバージョニング

翻訳時オプション-Kloop_versioningを指定した場合、実行時に配列のデータ依存を判断します。コンパイラは、配列のデータ依存がないものとして最適化を適用したループと配列のデータ依存があるものとして最適化を適用しないループを生成します。実行時に配列のデータ依存を判断し、どちらかのループを選択します。

本機能により、SIMD化、ソフトウェアパイプライニング、自動並列化などの最適化が促進されます。

ループバージョニングは、ループを複数生成するため、オブジェクトプログラムの大きさおよび翻訳時間が増加する場合があります。また、生成したループを選択するための判定処理がオーバヘッドとなり、実行性能が低下する場合があります。

本機能は、最内ループのみに適用されます。

翻訳時オプション-Koptmsg=2を指定することにより、最適化された箇所を示すメッセージが出力されます。

以下に、実行時に配列のデータ依存が判断できる例を示します。

例:

```
DO I=1,N
  A(I) = A(I+M) + B(I)
ENDDO
```

[最適化後(ソースイメージ)]

```
IF ((1 .GT. N+M) .OR. (N .LT. 1+M)) THEN
!OCL NORECURRENCE
  DO I=1,N                  ↑
    A(I) = A(I+M) + B(I)    | 最適化を適用
  ENDDO                     ↓
ELSE
  DO I=1,N
    A(I) = A(I+M) + B(I)
  ENDDO
ENDIF
```

翻訳時は、変数Nおよび変数Mの値が不明なため、配列Aのデータ依存も不明になります。翻訳時オプション-Kloop_versioningが有効な場合、実行時に変数Nおよび変数Mの値を用いて、最適化を適用したループまたは最適化を適用しないループのどちらかを選択します。

9.1.2.7 CLONE最適化

ループに対して変数の値による条件分岐を生成し、フルアンローリングなどのほかの最適化を促進する最適化です。CLONE最適化は最適化制御行で動作します。誤った最適化制御行の使い方をした場合、結果が保証されません。詳細は"9.10.4 最適化指示子"および"9.18 最適化指示子の誤った指定"を参照してください。

例:

```
!OCL CLONE(N==10)
DO I=1,N
  A(I) = I
ENDDO
```

[最適化後(ソースイメージ)]

```
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 リンク時最適化

ここでは、リンク時最適化について説明します。

9.1.2.8.1 リンク時最適化の概要

リンク時最適化は、複数のソースファイルからなるプログラムの場合、リンク時にファイル間をまたいで最適化を実施します。リンク時最適化は、-Kltoオプションで制御することができます。リンク時最適化を行う場合は、プログラムの翻訳時およびリンク時に-Kltoオプションを指定する必要があります。なお、リンク時にfrtpxコマンドではなくldコマンドを利用する場合は、リンク時最適化は適用されません。

例:

- 翻訳とリンクを同時に行う場合

```
$ frtpx -Kfast -Klto sample1.f90 sample2.f90 sample3.f90
```

- 翻訳とリンクを別々に行う場合

```
$ 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
```

リンク時最適化は、-Kltoオプションを指定して翻訳したオブジェクトファイルを最適化対象とします。

また、ファイルサフィックスが.oのファイルだけを最適化対象とします。そのため、.soや.aなどのライブラリは最適化対象となりません。

-Kltoオプションを指定して翻訳したオブジェクトファイルには、最適化を実施するために必要な情報が追加されているため、ファイルサイズは大きくなります。ただし、リンク時最適化に必要な情報は、実行可能プログラムには含まれることはありません。

-Kltoオプションを指定すると、コンパイラは翻訳時とリンク時の両方で、一時的にテンポラリディレクトリへリンク時最適化用の中間ファイルを出力します。このファイルは、翻訳、リンクのそれぞれが終了した際に削除されます。テンポラリディレクトリの省略値は/tmpです。テンポラリディレクトリは、環境変数TMPDIRで変更できます。

Fortranのリンク時最適化は、言語間結合では適用できません。

9.1.2.8.2 リンク時最適化の注意事項

リンク時最適化には、以下の注意事項があります。

- リンク時最適化は、オブジェクトファイルに追加された情報を元にリンク前に再度最適化を実施します。そのため、実行可能プログラム生成までに要するメモリや時間が大幅に増加する場合があります。
- リンク時最適化が適用された場合、以下に示す機能に影響を与えることがあります。

  ー 翻訳時の情報取得機能

  以下に示す翻訳時情報が、実際に適用された最適化と異なる場合があります。

  - 翻訳時オプション-Nlst=pまたは-Nlst=tで出力される翻訳情報(最適化情報)
  - 翻訳時オプション -Koptmsgで出力される最適化メッセージ

  Fortranプログラムの翻訳時情報については、“4.1 翻訳時の出力情報”を参照してください。

  ー プログラム検査

  以下に示すFortranプログラムの実行時の検査が、実施されない場合があります。

  - 引数および関数結果に関する手続引用の妥当性の検査(-Nquickdbg=argchk、-Ha)
  - 添字式および部分列範囲の検査(-Nquickdbg=subchk、-Hs)
  - 未定義データの引用の検査(-Nquickdbg=undef、-Nquickdbg=undefnan、-Hu)

  Fortranプログラムのデバッグ機能については、“8.2.1 デバッグを行うための検査機能”を参照してください。

  ー プロファイラ

  プロファイラが出力する以下の情報が保証されない場合があります。

  - コールグラフ情報

    表示される手続名に、リンク時最適化が内部的に生成した手続名が表示される場合があります。

  - ソースコード情報

    各行のコストが正しく表示されない場合があります。

  プロファイラについては、“プロファイラ使用手引書”を参照してください。

  ー 実行時情報出力機能

  実行時情報出力機能が出力する情報において、行番号がソースプログラムの行番号と一致しない場合があります。

  実行時情報については、“付録H 実行時情報出力機能”を参照してください。

  ー トレースバックマップ

  エラーが発生した文の行番号が正しくない場合があります。

  トレースバックマップについては、“4.2.2 トレースバックマップ”を参照してください。

9.1.2.9 アンロールアンドジャム

アンロールアンドジャムとは、多重ループの外側のループをアンローリングによりn重に展開し、さらに展開された内側のループを融合する最適化です。アンロールアンドジャムを適用することで、共通式の除去が促進され、実行性能が向上する場合があります。ただし、データストリーム数の増加やデータのアクセス順序の変化は、キャッシュの利用効率を低下させ、実行性能が低下する場合があります。なお、本最適化は最内ループには適用されません。

また、本最適化による実行性能への影響はループ単位で異なります。このため、本最適化は、-Kunroll_and_jam[=*N*]オプションでプログラム全体へ適用せず、最適化指示子UNROLL_AND_JAMや最適化指示子UNROLL_AND_JAM_FORCEでループ単位に適用することを推奨します。詳細は、“9.10.4 最適化指示子”のUNROLL_AND_JAMまたはUNROLL_AND_JAM_FORCEを参照してください。

翻訳時オプション-Koptmsg=2を指定することにより、最適化されたループとその展開数を示すメッセージを出力することができます。

例:

```
!OCL UNROLL_AND_JAM_FORCE(2)
DO J=1, 128
  DO I=1, 128
```

→

[最適化後(ソースイメージ)]

```
DO J=1, 128, 2
  DO I=1, 128
    A(I, J) = B(I, J) + B(I, J+1)
```

```
    A(I, J) = B(I, J) + B(I, J+1)
    ...
  END DO
END DO
```

```
    A(I, J+1) = B(I, J+1) + B(I, J+2)
    ...
  END DO
END DO
```

9.1.2.10 tree-height-reduction最適化

tree-height-reduction最適化とは、ループ中の演算に対して演算木をなるべく低くなるように演算の並び替えを行い、命令の並列性を高める最適化です。演算木とは、演算子の優先順位に従って演算式を木構造で表現したものです。葉の部分には、値を置きます。それ以外の節には、演算子を置きます。優先順位が高い演算子ほど、上位の階層に配置します。

-Keval_concurrentオプションが有効な場合に、本最適化が浮動小数点演算に対して適用されます。この場合、実行結果に副作用(計算誤差)を生じることがあります。最適化の副作用については、“9.20 浮動小数点演算に対する最適化の副作用”を参照してください。

以下に、最適化の例を示します。

例:

```
DO I=1,N
  X(I) = A(I) * B(I) + C(I) * D(I) + E(I) * F(I) + G(I) * H(I)
ENDDO
```

図9.3 tree-height-reduction最適化前の演算木

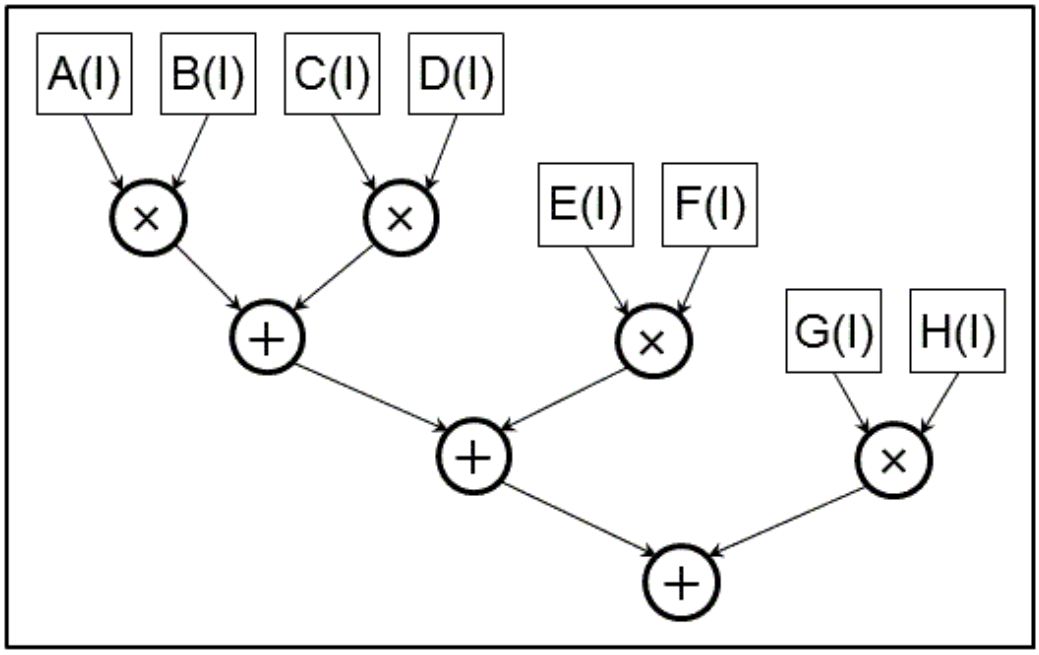
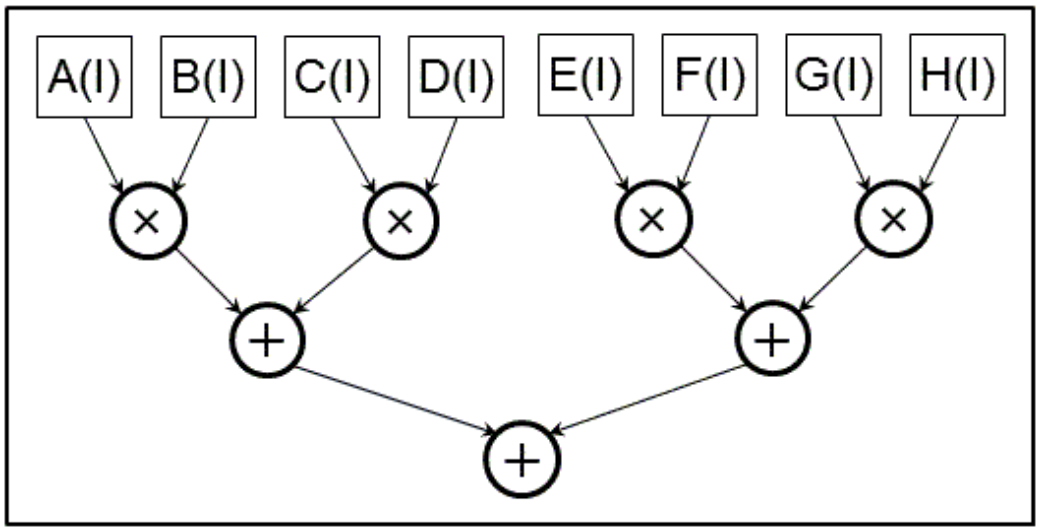


図9.4 tree-height-reduction最適化後の演算木



9.1.2.11 ループ分割

ループ分割は、以下を目的として、ループを複数の小さなループに分割する機能です。

- ソフトウェアパイプライニングの促進
- キャッシュメモリ利用効率の改善
- レジスタ不足の解消

ループ分割は、ループ外からの飛込みもループ外への飛出しもないループに対して行われます。

自動ループ分割は、-Kloop_fissionオプションおよび-Koclオプションが有効な場合に、最適化指示子LOOP_FISSION_TARGETが指定されたループに対して行われます。-Kloop_nofissionオプションを指定することにより、ループ分割を抑止することができます。-Koptmsg=2オプションを指定すると、最適化されたループを示すメッセージを出力することができます。

例:

```
!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
```

[最適化後(ソースイメージ)]

```
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 ストリップマイニング

ストリップマイニングとは、ループを小さい繰返し数で断片化する最適化です。自動ループ分割後のループにこの最適化を適用することにより、ループ分割したループ間でアクセスされるデータに対して、キャッシュメモリの利用効率の向上が期待できます。

ストリップマイニングは、-Kloop_fission_stripminingオプションを指定した場合に行われます。-Koptmsg=2オプションを指定することにより、最適化されたループを示すメッセージを出力することができます。

以下に、自動ループ分割およびストリップマイニング(-Kloop_fission_stripmining=256オプション指定時)の例を示します。

例:

[ソースプログラム]

```
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
```

[自動ループ分割後(ソースイメージ)]

```
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
```

このソースプログラムでは、ループ分割したループ間でデータを受け渡す必要があるため、ループ分割に伴い、コンパイラが一時的な配列TEMPARRAY_Qを生成します。ストリップマイニングを行わない場合、この配列の要素数はループの繰返し数と同じ値になります。

[自動ループ分割およびストリップマイニング後(ソースイメージ)]

```
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
```

分割したループの外側にループを生成し、元のループの繰返し数をストリップ長の256で断片化します。また、一時的な配列TEMPARRAY_Qの要素数は256となります。

9.2 誤りがあるプログラムに対する注意事項

以下に示すような誤ったプログラムに対して最適化を行った場合、最適化のレベルにより実行結果が異なることがあります。実行結果が異なる現象が発生した場合は、以下の点を確認してプログラムを修正してください。

- 値が定義されていない変数を引用している
- 許されない形式の値が格納されている変数を引用している
- 配列宣言を超える添字で配列要素を引用している
- Fortranの文法に違反した使い方をしている

9.3 環境の変化等の最適化機能への影響

最適化機能は、原始プログラムから変換されたコンパイラ内部情報を元に動作するため、原始プログラムと翻訳時オプションが同一でもコンパイラのバージョン・レベルの違い、修正の適用またはシステム環境の変更により結果が以前と異なる場合があります。

その結果、以下のような現象が現れる場合があります。

- 最適化機能や自動並列化機能に関する翻訳時メッセージまたは翻訳情報出力ファイル中の情報が、以前の翻訳結果と異なる。
- 演算結果に誤差程度の違いが生じる。

なお、最適化の結果の違いに関しては、上記の翻訳時メッセージや翻訳情報出力ファイルの内容を比較することにより把握できます。

9.4 実行順序の変更による影響

共通式の除去や不変式の移動などの最適化により、演算命令の実行順序が変更されてプログラム割込みの発生位置や回数が変わる場合がありますので、デバッグの際は注意を要します。

なお、定数同士の演算で演算割込みが発生する場合は、翻訳時に定数演算を行わないため、それらの演算割込みは実行時に発生することになります。

9.5 割当て形GO TO文の分岐先

割当て形GO TO文の文番号並びは、記述するのであれば正確でなければなりません。

すなわち、割当て形GO TO文では、その文の文番号並びに記述された文番号以外の文番号へ分岐してはなりません。

コンパイラは、文番号並びに記述された文番号のどれかに分岐するものとして最適化するので、もしそれ以外の文番号に分岐した場合は、意図した結果が得られない場合があります。

文番号並びが省略されている場合は、コンパイラ(最適化)は分岐する可能性のあるすべての文番号をプログラムを解析して求めます。したがって、分岐する文番号が明らかな場合は、明記したほうがより最適化されます。

9.6 仮引数での影響

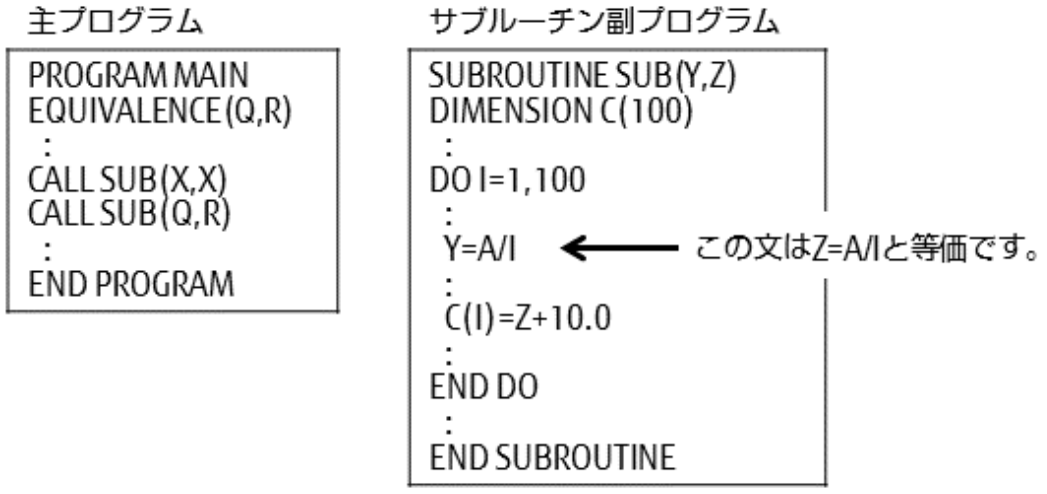
仮引数に関して、文法に違反した使い方をすると、利用者が意図した結果が得られないことがあるので、注意する必要があります。

同じ実引数との結合

同じ手続のそれぞれの仮引数は、別の変数とみなして最適化されます。したがって、異なる仮引数に対して同じ実引数が対応するような呼出しがある場合、意図した結果が得られない場合があります。実引数が異なる変数でも、それらが領域を共有している場合は同様です。文法では、このような仮引数に対して、その副プログラム内で直接および間接に値を再定義することを禁止しています。

以下に、同じ実引数との結合による影響の例を示します。

例: 同じ実引数との結合による影響



[説明]

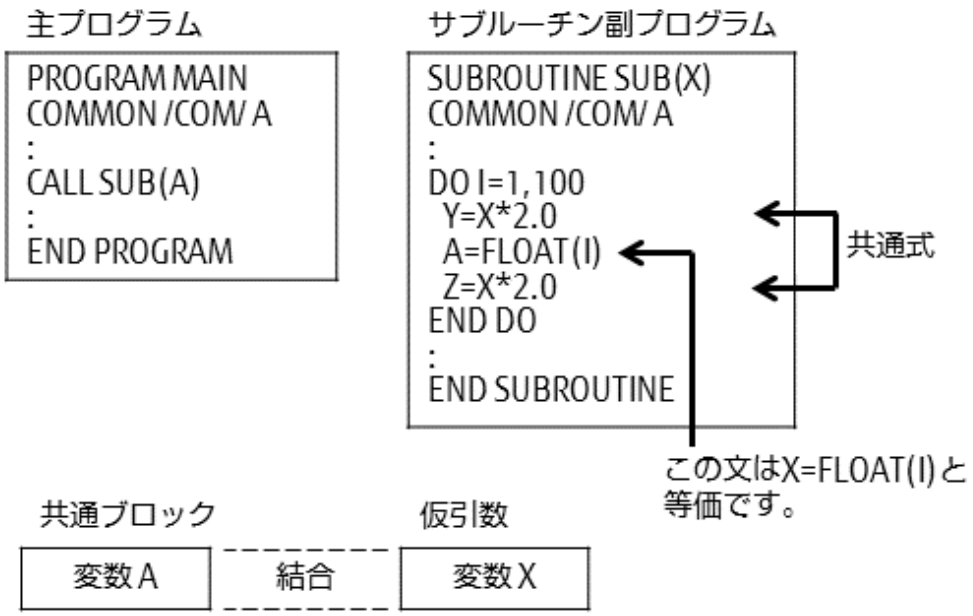
仮引数Yと仮引数Zは別の変数として最適化されます。このため、DOループ内では変数Zの値は不変であるとみなされ、最適化(不変式の移動)によりZ+10.0はループの外に移動されます。その結果、Z+10.0の演算結果にY=A/Iの演算結果は反映されません。また、変数ZがDOループ内で値が変わる場合(Z+10.0が移動されない場合)でも、もし変数Yと変数ZがDOループ内でレジスタ上に割り当てられると、Z+10.0の演算にY=A/Iの演算結果は反映されません。これは、変数Yと変数Zに異なるレジスタが割り当てられるためです。

共通ブロック内のデータとの結合

仮引数と共通ブロック内の変数は結合していない(利用者が文法に違反した記述をしていない)とみなして最適化されます。したがって、仮引数が共通ブロック内の変数と結合している場合、意図した結果が得られない場合があります。

以下に、共通ブロック内のデータとの結合による影響の例を示します。

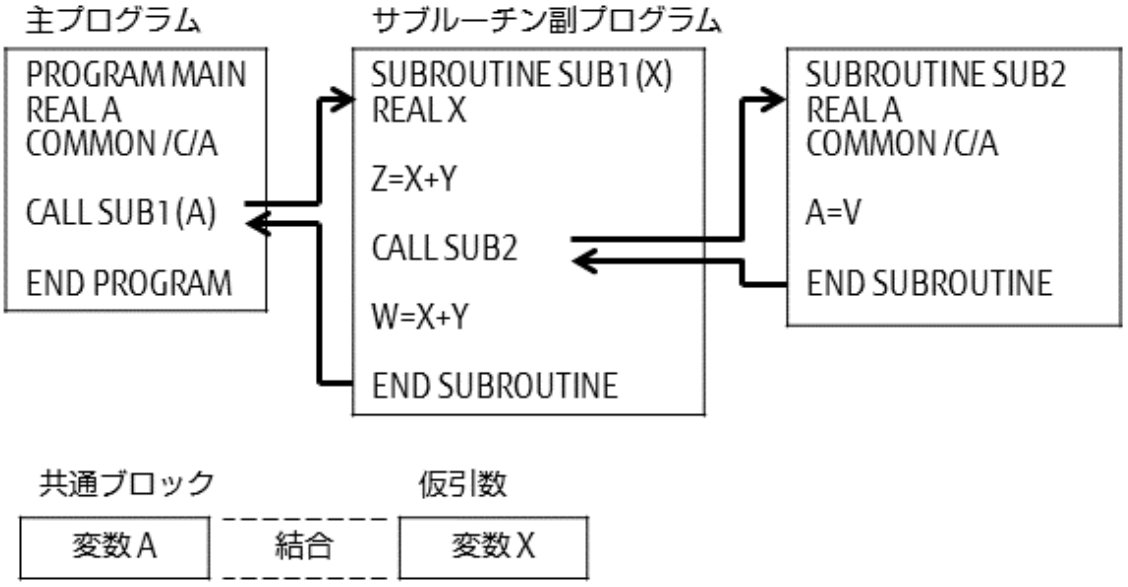
例1: 共通ブロック内のデータとの結合による影響



[説明]

仮引数Xと共通ブロック内の変数Aは別の変数として最適化されます。このため、DOループ内のX*2.0は共通式であるとみなされ、最適化(共通式の除去)によりZ=X*2.0はZ=Yに変更されます。その結果、ZにはA=FLOAT(I)で代入した値が反映されません。文法では、同じ実体に対して、仮引数と共通ブロックで同時に結合して値を定義することを禁止しています。したがって、同じ実体を仮引数で結合したり共通ブロックで結合したりすると、意図した結果が得られない場合があります。このような使い方は、文法に違反しているので、仮引数による結合または共通ブロックによる結合のどちらかに統一すべきです。

例2: 共通ブロック内のデータとの結合による影響



[説明]

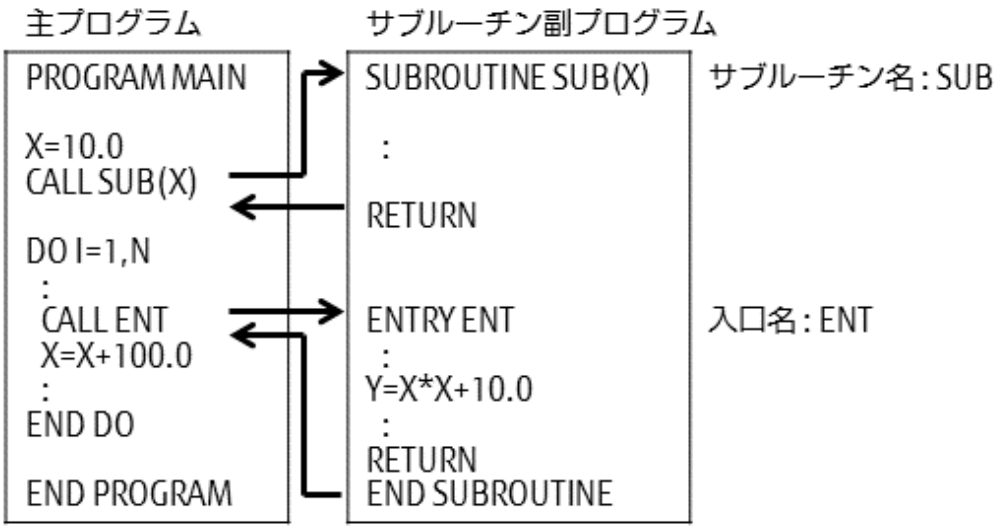
SUB1内にあるSUB2の呼出しによって共通ブロック実体Aの値が変更されます。同時に共通ブロック実体Aと結合している仮引数Xも変更されます。しかし、コンパイラは、このことを認識できないので、SUB2の呼出し前後のX+Yは、最適化(共通式の除去)の対象となります。最適化の結果、SUB2の呼出しのあとのW=X+YはW=Zに変更され、SUB2の呼出しによって変更された変数Xの値がWの結果に反映されなくなります。

ENTRY文を利用した結合

ENTRY文を含む副プログラムをある入口名、関数名またはサブルーチン名で呼出し、実引数を仮引数と結合したとします。そして、その実引数(の番地)を副プログラムに渡したと考えて、あとでその変数を実引数にせずに、別の入口名、関数名またはサブルーチン名でその副プログラムを呼び出して、副プログラム内でその変数を引用すると問題が生じる場合があります。

以下に、入口名を利用した結合による影響の例を示します。

例: 入口名を利用した結合による影響



[説明]

このような場合、文法上は、CALL ENT(X)、ENTRY ENT(X)としなければなりません。

9.7 インライン展開の制約事項

実引数と仮引数の対応関係などいくつかの制約事項があるため、インライン展開の候補になった利用者定義の手続が、必ずしもインライン展開されるとは限りません。

以下に、インライン展開の制約事項を示します。

9.7.1 呼び出される利用者定義の手続に関する制約事項

実行文の数が許容限度を超える場合

利用者定義の手続の実行文の数が許容限度を超える場合、その利用者定義の手続はインライン展開されません。
許容限度の値は、翻訳時オプション-xのパラメタで変更することができます。許容限度の標準値は、コンパイラが自動的に決定します。

配列の大きさが許容限度を超える場合

利用者定義の手続の共通ブロック、仮引数および割付け変数を除く配列(すなわち、利用者定義の手続内に局所的、かつ、領域が静的に割り付けられる配列)の大きさの合計が、許容限度を超える場合、その利用者定義の手続はインライン展開されません。
許容限度の値は、翻訳時オプション-xのパラメタで変更することができます。許容限度の標準値は無限大です。

インライン展開の制約となる文を含んでいる場合

以下に示す文を含む手続はインライン展開されません。

- ALLOCATE文
- DEALLOCATE文
- ASSIGN文
- 割当て形GO TO文
- NAMELIST文
- 変数群入出力文
- RETURN文
  整数式を指定したRETURN文がある場合、インライン展開されません。
- NULLIFY文

インライン展開の制約となる手続の場合

以下の手続はインライン展開されません。

- VOLATILE属性をもつ手続
- 関数結果が配列、多相的、または多相的成分をもつ派生型である手続
- 関数結果がVOLATILE属性をもつ手続
- 親有効域のモジュールにPRIVATE属性をもつ変数が現れるモジュール手続
- ENTRY文で定義されたモジュール手続
- 手続言語束縛指定子をもつ手続
- ASYNCHRONOUS属性をもつ手続
- 関数結果が無指定型パラメタをもつ手続
- 関数結果が文字型で、FUNCTION文またはENTRY文のいずれかに手続言語束縛指定子をもつ手続
- SAVE属性の変数をもつ手続
  共通ブロックの全体または共通ブロックの要素を除く変数(すなわち、利用者定義の手続に局所的な変数)を有効にするSAVE文がある場合、その利用者定義の手続はインライン展開されません。
- 再起的な割付け可能成分をもつ手続

- 分離モジュール手続
- 組込み関数NULLの引用を除くポインタ初期値をもつ手続
- 内部手続の定義をもつ手続
- -xaccept=module_allocatable オプションを指定していない場合で、親有効域のモジュールが割付け変数をもつモジュール手続
- -xaccept=module_allocatable オプションを指定していない場合で、割付け変数を参照結合している手続

初期値をもつ変数または配列の値が変更される場合

- DATA文または型宣言文で初期値を指定した変数の値が、直接または間接に変更される可能性がある場合、その利用者定義の手続はインライン展開されません。
- 親有効域のモジュールで、DATA文または型宣言文で初期値を指定した変数の値が、直接または間接に変更される可能性がある場合、そのモジュール内のモジュール手続はインライン展開されません。

ENTRY文を含んでいる場合

以下のENTRY文を含む副プログラムはインライン展開されません。

- 関数名またはサブルーチン名と入口名で指定されている仮引数が異なる場合

-Knoalias=sオプションを指定した場合

次のいずれかの条件の手続は、インライン展開されません。

1. 呼び出す側と呼び出される側の手続が異なるプログラム単位に現れる。かつ、
   次のいずれかが、呼び出される側の手続に現れる。
   - ポインタ属性をもつ派生型変数がある。
   - 派生型変数の成分にポインタ属性がある。
2. 呼び出す側と呼び出される側が異なるプログラム単位である。かつ、
   その呼び出される側の手続は、モジュールプログラム単位内にある。かつ、
   次のいずれかが、その呼び出される側のモジュールの宣言部に現れる。
   - ポインタ属性をもつ派生型変数がある。
   - 派生型変数の成分にポインタ属性がある。
3. 呼び出す側と呼び出される側が同じプログラム単位である。かつ、
   その呼び出される側の手続の仮引数は、派生型変数の成分にポインタ属性をもつ変数がある。

9.7.2 呼び出す側と呼び出される側の関係に関する制約事項

実引数と仮引数の対応に誤りがある場合

- 実引数と仮引数の個数が異なる
- 実引数と仮引数の型や属性が異なる
- 仮引数または実引数のどちらか一方だけに手続がある

実引数と仮引数の対応が以下に示す場合

- 仮引数が仮手続名であり、それに対応する実引数が組込み関数または、手続ポインタである
- 仮引数が*である
- 実引数が文番号である
- 実引数に配列式がある
- 実引数がPOINTER属性をもつ配列名である
- 実引数が%VALまたは組込み関数VALである
- 実引数が割付け成分をもつ型、かつその実引数は、括弧で囲まれている
- 実引数が割付け成分をもつ型の構造体構成子である

- 仮引数がOPTIONAL属性をもつ
- 仮引数がPOINTER属性をもつ
- 仮引数がALLOCATABLE属性をもつ
- 仮引数がVOLATILE属性をもつ
- 仮引数または実引数が形状引継ぎ配列である
- 仮引数がVALUE属性をもつ
- 仮引数がASYNCHRONOUS属性をもつ
- 仮引数または実引数が多相的である
- 仮引数または実引数の派生型成分が多相的である
- 仮引数が長さ型パラメタをもつ派生型である
- 仮引数が型引継ぎまたは次元引継ぎである

利用者定義の外部手続の引用方法に誤りがある場合

- 外部関数副プログラムを外部サブルーチン副プログラムとして引用している
- 外部サブルーチン副プログラムを外部関数副プログラムとして引用している
- 外部関数副プログラムで定義している型と異なる型で引用している

呼び出す利用者定義の外部手続およびモジュール手続が以下に示す場合

- 仮称指定で局所名に指定された手続

9.7.3 利用者定義の外部手続の名前に関する制約事項

同じ名前の利用者定義の外部手続がある場合

プログラムのエラーとなりインライン展開されません。

- 同じ名前の外部サブルーチン副プログラムがある
- 同じ名前の外部関数副プログラムがある
- 同じ名前の初期値設定プログラム単位がある
- 名前がない初期値設定プログラム単位が2つ以上ある

翻訳時オプション-Koptmsg=2を指定することにより、実際にインライン展開されたかどうかを示すメッセージを出力することができます。
インライン展開されなかった場合は、その理由を示すメッセージも出力されます。

9.7.4 オプションによる制約事項

-Hオプションを指定した場合

インライン展開されません。

9.7.5 要素別処理手続に関する制約事項

インライン展開の制約となる文を含んでいる場合

以下の文を含む要素別処理手続はインライン展開されません。

- DATA文
- FORMAT文
- COMMON文
- EQUIVALENCE文

インライン展開の制約となる要素別処理手続の場合

以下の要素別処理手続はインライン展開されません。

- サブルーチンである
- 関数結果が文字型である
- 要素別処理手続を呼び出す手続が、異なるプログラム単位にある
- プログラム単位に最適化制御行がある
- “*”を除く入出力のFMT指定子をもつ

9.8 CRAY仕様のポインタ変数での影響

ポインタ変数はLOC組込み関数で、そのポインタ変数に対応するアドレス保持変数にアドレスがとられたデータのみ共用関係にあると判断しています。ただし、共通ブロック実体、構造体成分または結合実体のアドレスがとられた場合、それぞれ同じ共通ブロック、構造体または結合対応に属するほかのデータとも共用関係があると判断します。

また、アドレス保持変数にLOC組込み関数以外を使用してアドレスを設定した場合は利用者が意図した結果が得られないことがあります。

さらに、アドレス保持変数が仮引数で渡ってきたときは、対応するポインタ変数が以下に示す変数とは共用関係にないと判断しているため、利用者は十分な注意が必要です。

- ほかの仮引数。ほかの仮引数がアドレス保持変数ならば、対応するポインタ変数。
- 共通ブロックに属する変数。

9.9 翻訳時オプション-Kcommonpad[=N]の影響

分割コンパイルする場合は、共通ブロックを含むファイルに対して翻訳時オプション-Kcommonpadを指定した場合、同じ名前の共通ブロック引用を含む他のファイルに対しても、翻訳時オプション-Kcommonpadを指定しなければなりません。(例1を参照)

複数のファイルに対して翻訳時オプション-Kcommonpad=*N*を指定して翻訳する場合、*N*の値は同じでなければなりません。

また、同じ共通ブロック名に対し、その要素を変えて使用している場合、翻訳時オプション-Kcommonpadを指定するとプログラムが正しく実行されないことがあります。(例2を参照)

例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
   :
```

a.f、b.f共に共通ブロックcmnを含んでいるので、翻訳時オプション-Kcommonpadを指定する場合、a.fおよびb.fの両方に指定しなければなりません。

例2:

```
SUBROUTINE SUB_A
COMMON/CMN/A,B
REAL(4),DIMENSION(10) :: A = 1.0,B = 2.0
   :
SUBROUTINE SUB_B
COMMON /CMN/X
COMPLEX(4),DIMENSION(10) :: X
```

```
PRINT *,X
   :
```

サブルーチンsub_a、sub_bにおいて、共通ブロックcmnの要素が異なるため、翻訳時オプション-Kcommonpadを指定した場合の動作は保証されません。

9.10 最適化制御行(OCL)の利用

最適化にとって有効な情報をソースプログラム上に記述することにより、最適化の効果をより高めることができます。

9.10.1 最適化制御行の記法

最適化制御行は、プログラム形式により次のいずれかです。

- 自由形式

  “!OCL”で始まる行です。“!OCL”の後には1つ以上の空白が必要です。“!OCL”の前に空白を記述することができます。

- 固定形式

  行の第1けたから第5けたが“!OCL ”(第5けたは、空白)である行です。

“!OCL”で始まる行は、通常は注釈行として扱われますが、翻訳時オプション-Koclを指定することにより、最適化制御行とみなされ、オペランドが有効になります。

以下に記法を示します。

```
!OCL□i[,i]...
```

i: 最適化指示子

□ : 1つ以上の空白

9.10.2 最適化制御行の記述位置

最適化制御行は、最適化指示子の種類によって記述できる位置が決まっています。

最適化制御行は、最適化指示子FIXEDの最適化制御行を除き(注)、プログラム単位、DOループ単位、文単位、または配列代入文単位に指定します。プログラム単位、DOループ単位、文単位、配列代入文単位は、以下のように定義されます。

- プログラム単位
  各プログラム単位の先頭。
- DOループ単位
  DO文の直前の位置。ただし、最適化制御行とDO文との間に空白、注釈行またはDOループ単位に指定可能な他の最適化制御行を含んでもかまいません。
- 文単位
  文の直前の位置。
- 配列代入文単位
  配列代入文の直前の位置。

注) 最適化指示子FIXEDの最適化制御行は、プログラム単位の宣言部に指定します。

9.10.3 ワイルドカードの指定

以下の最適化指示子のオペランドには、変数名または手続名としてワイルドカード指定が可能です。

```
NORECURRENCE
NOVREC
```

ワイルドカードの指定方法については“12.2.6.5 ワイルドカード指定”を参照してください。

9.10.4 最適化指示子

下表に、最適化制御行に記述できる最適化指示子を示します。

表9.1 最適化制御行に記述できる最適化指示子

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
ARRAY_DECLARATION_OPT	最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にします。	○	○	×	○
NOARRAY_DECLARATION_OPT	最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にしません。	○	○	×	○
ARRAY_FUSION[,*opt*] END_ARRAY_FUSION	範囲内にある配列代入文は、融合が促進されます。また、指定された最適化指示子*opt*がすべての配列代入文に対して有効となります。	×	×	×	○
ARRAY_MERGE (*array1*,*array2*[,*array3*]...) または ARRAY_MERGE (*base_array*:*array1*[,*array2*]...)	配列のマージを指示します。または、次元数の少ない配列を次元数の多い配列へマージすることを指示します。 *base_array*、*array1*、*array2*、*array3*、... は配列名です。	○	×	×	×
ARRAY_SUBSCRIPT (*array1*[,*array2*]...)	配列の次元移動を行うことを指示します。 *array1*、*array2*、... は配列名です。	○	×	×	×
ASSUME ({SHORTLOOP | NOSHORTLOOP | MEMORY_BANDWIDTH | NOMEMORY_BANDWIDTH | TIME_SAVING_COMPILATION | NOTIME_SAVING_COMPILATION})	プログラムの特徴に合わせて、最適化を調整することを指示します。	○	×	×	×
CLONE(*var*==*n1*[,*n2*]...)	ループ内で変数*var*が不変とみなして、引数で指示をした条件分岐を生成し、条件節内にループを複製する最適化を行うことを指示します。 条件式は第1引数の変数*var*と、第2引数以降で指定した値*n1*[,*n2*]...の等式とします。 *var*は整数型の変数です。 *n1*[,*n2*]...は-92233720368547758088から9223372036854775807までの10進数または名前付き定数です。	×	○	×	○
EVAL	式の評価順序を変更する最適化を有効にします。	○	○	×	○
NOEVAL	式の評価順序を変更する最適化を無効にします。	○	○	×	○
EVAL_CONCURRENT	tree-height-reduction最適化において、命令の並列性を優先することを指示します。	○	○	×	○
EVAL_NOCONCURRENT	tree-height-reduction最適化において、命令の並列性を抑え、FMA命令の利用を優先することを指示します。	○	○	×	○

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
EXTRACT_STRIDE_STORE	SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開することを指示します。	○	○	×	○
NOEXTRACT_STRIDE_STORE	SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開しないことを指示します。	○	○	×	○
FISSION_POINT[(*n*)]	ループ内の指定された位置でループを分割することを指示します。最内ループから数えて*n*重ネストされた多重ループを対象に分割します。 *n*は1～6の10進数です。 ループ分割の際にループ間でデータを受け渡す必要がある場合は、コンパイラが自動的に一時的な領域を生成します。そのため実行結果には影響ありません。	×	×	○	×
FIXED *array1* (*shape-spec-list*) [, *array2* (*shape-spec-list*)]...	ポインタ配列または割付け配列が、以下のいずれかであることを指示します。 ・ 未結合または未割付けである。 ・ 配列の上下限は、*shape-spec-list*で結合している、または割り付いている。 プログラム単位の宣言部に指定します。	×	×	×	×
FP_CONTRACT	Floating-Point Multiply-Add/Subtract 演算命令を使用します。	○	○	×	○
NOFP_CONTRACT	Floating-Point Multiply-Add/Subtract 演算命令を使用しません。	○	○	×	○
FP_RELAXED	浮動小数点除算、SQRT 関数について、高速な演算を行うことを指示します。	○	○	×	○
NOFP_RELAXED	浮動小数点除算、SQRT 関数について、通常の演算を行うことを指示します。	○	○	×	○
FULLUNROLL_PRE_SIMD[(*n*)]	SIMD化前のフルアンローリングを促進することを指示します。*n*は、対象ループの回転数の上限を表す2～100の整数値です。	×	○	×	○
NOFULLUNROLL_PRE_SIMD	SIMD化前のフルアンローリングを抑止することを指示します。	×	○	×	○
ILFUNC	組込み関数および演算をインライン展開することを指示します。	○	○	×	○
NOILFUNC	組込み関数および演算をインライン展開しないことを指示します。	○	○	×	○
ITERATIONS(max=*n1*) ITERATIONS(avg=*n2*) ITERATIONS(min=*n3*)	ループの繰返し数の最大値を*n1*、平均値を*n2*、最小値を*n3*とみなして最適化を行うことを指示します。*n1*、*n2*、および*n3*は、0～2147483647です。max、avg、およびminは個別またはコンマを区切りとして順不同の複数指定が可能です。	○	○	×	×

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
LOOP_BLOCKING(*n*)	ブロッキング機能を有効にします。*n* はブロックサイズを表す2～10000の10進数です。	○	○	×	○
LOOP_NOBLOCKING	ブロッキング機能を無効にします。	○	○	×	○
LOOP_FISSION_STRIPMINING[({ *n* | *level* })]	ループ分割時にストリップマイニングの最適化を行うことを指示します。*n*の範囲は、2～100000000の範囲です。*level*は、'L1'または'L2'です。	○	○	×	○
LOOP_NOFISSION_STRIPMINING	ストリップマイニングの最適化を抑止します。	○	○	×	○
LOOP_FISSION_TARGET[({ CL | LS })]	コンパイラによる自動ループ分割を行うことを指示します。	×	○	○	×
LOOP_FISSION_THRESHOLD(*n*)	ループ分割における分割後のループの粒度を決める閾値を指示します。*n*の範囲は、1～100です。	○	○	×	○
LOOP_INTERCHANGE(*var1*,*var2*[,*var3*]...)	指定された順序(*var1*、*var2*、…)で多重DOループの入れ換えを実施します。 *var1*、*var2*、*var3*、... はDO 変数名です。	×	○	×	×
LOOP_NOINTERCHANGE	多重DO ループの入れ換えを実施しません。	×	○	×	×
LOOP_NOFUSION	ループを融合する機能を無効にします。	○	○	×	×
LOOP_PART_SIMD	ループを分割して部分的にSIMD化する機能を有効にします。	○	○	×	○
LOOP_NOPART_SIMD	ループを分割して部分的にSIMD化する機能を無効にします。	○	○	×	○
LOOP_PERFECT_NEST	不完全多重ループを分割して完全多重ループにすることを指示します。	○	○	×	○
LOOP_NOPERFECT_NEST	不完全多重ループを完全多重ループにしないことを指示します。	○	○	×	○
LOOP_VERSIONING	ループバージョニングの最適化を行うことを指示します。	○	○	×	×
LOOP_NOVERSIONING	ループバージョニングの最適化を行わないことを指示します。	○	○	×	×
MFUNC[(*level*)]	マルチ演算関数化を行うことを指示します。 *level* は機能レベルを表す1～3の10進数です。	○	○	×	○
NOMFUNC	マルチ演算関数化を行わないことを指示します。	○	○	×	○
NOALIAS	ポインタ変数が他の変数と記憶域を共有しないことを指示します。	○	○	×	○
NOARRAYPAD(*array1*)	配列要素へのパディングを行わないことを指示します。 *array1* は配列名です。	○	×	×	×
NORECURRENCE[(*array1*[,*array2*]...)]	回転を跨いで定義引用されない配列を指示します。	○	○	×	○

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
	array1、*array2*、...は配列名です。				
NOVREC[(*array1*[,*array2*]...)]	回帰演算がないループ(SIMD命令が利用可能なループ)であることを指示します。 *array1*、*array2*、... は配列名です。	○	○	×	○
PREEX	不変式の先行評価の最適化を有効にします。	○	○	×	○
NOPREEX	不変式の先行評価の最適化を無効にします。	○	○	×	○
PREFETCH	コンパイラの自動prefetch機能を有効にします。	○	○	×	×
NOPREFETCH	コンパイラの自動prefetch機能を無効にします。	○	○	×	×
PREFETCH_CACHE_LEVEL(*c-level*)	キャッシュレベル*c-level*にデータをプリフェッチすることを指示します。 *c-level* は1、2 またはall です。	○	○	×	○
PREFETCH_CONDITIONAL	IF構文やCASE構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成することを指示します。	○	○	×	○
PREFETCH_NOCONDITIONAL	IF構文やCASE構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成しないことを指示します。	○	○	×	○
PREFETCH_INDIRECT	ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を生成することを指示します。	○	○	×	○
PREFETCH_NOINDIRECT	ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を生成しないことを指示します。	○	○	×	○
PREFETCH_INFER	プリフェッチの距離が不明な場合でも連続アクセスのプリフェッチを出力することを指示します。	○	○	×	○
PREFETCH_NOINFER	プリフェッチの距離が不明な場合は、連続アクセスのプリフェッチを出力しないことを指示します。	○	○	×	○
PREFETCH_ITERATION(*n*)	prefetch 命令を生成する際、ループの*n*回転後に参照または定義されるデータを対象とすることを指示します。 本機能では1次キャッシュにのみプリフェッチするprefetch 命令を対象とします。 *n* は1 ～ 10000 の10 進数です。 SIMD化が適用されるループの場合、*n* にはSIMD化された後のループの繰返し数を指定してください。	○	○	×	○
PREFETCH_ITERATION_L2(*n*)	prefetch 命令を生成する際、ループの*n*回転後に参照または定義されるデータを対象とすることを指示します。	○	○	×	○

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
	本機能では2次キャッシュにのみプリフェッチするprefetch 命令を対象とします。 *n* は1 ～ 10000 の10 進数です。 SIMD化が適用されるループの場合、*n* にはSIMD化された後のループの繰返し数を指定してください。				
PREFETCH_LINE(*n*)	prefetch命令を生成する際、*n*キャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。 本機能では1次キャッシュにのみプリフェッチするprefetch 命令を対象とします。 *n*は1 ～ 100 の10 進数です。	○	○	×	○
PREFETCH_LINE_L2(*n*)	prefetch命令を生成する際、*n*キャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します 本機能では2次キャッシュにのみプリフェッチするprefetch 命令を対象とします。 *n*は1 ～ 100 の10 進数です。	○	○	×	○
PREFETCH_READ (*name*[,level={ 1 | 2 }] [,strong={ 0 | 1 }])	参照されているデータに対してprefetch 命令を生成することを指示します。 *name* は配列要素または部分配列、level はプリフェッチを行うキャッシュレベル、strongはstrong prefetchとするかどうかです。	×	×	○	×
PREFETCH_WRITE (*name*[,level={ 1 | 2 }] [,strong={ 0 | 1 }])	定義されているデータに対してprefetch 命令を生成することを指示します。 *name* は配列要素または部分配列、level はプリフェッチを行うキャッシュレベル、strongはstrong prefetchとするかどうかです。	×	×	○	×
PREFETCH_SEQUENTIAL [({AUTO | SOFT})]	連続的にアクセスされる配列データに対してprefetch命令を生成することを指示します。	○	○	×	○
PREFETCH_NOSEQUENTIAL	連続的にアクセスされる配列データに対してprefetch命令を生成しないことを指示します。	○	○	×	○
PREFETCH_STRIDE [({SOFT | HARD_AUTO | HARD_ALWAYS})]	ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、プリフェッチを実施することを指示します。	○	○	×	○
PREFETCH_NOSTRIDE	ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対してprefetch命令を生成しないことを指示します。	○	○	×	○
PREFETCH_STRONG	1次キャッシュに対して生成されるprefetch命令をstrong prefetchとすることを指示します。	○	○	×	○

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
PREFETCH_NOSTRONG	1次キャッシュに対して生成されるprefetch命令をstrong prefetchとしないことを指示します。	○	○	×	○
PREFETCH_STRONG_L2	2次キャッシュに対して生成されるprefetch命令をstrong prefetchとすることを指示します	○	○	×	○
PREFETCH_NOSTRONG_L2	2次キャッシュに対して生成されるprefetch命令をstrong prefetchとしないことを指示します。	○	○	×	○
PRELOAD	ロード命令を投機実行する最適化を行うことを指示します。	○	○	×	○
NOPRELOAD	ロード命令を投機実行する最適化を行わないことを指示します。	○	○	×	○
SCACHE_ISOLATE_WAY(L2=*n1*[,L1=*n2*]) END_SCACHE_ISOLATE_WAY	1次キャッシュと2次キャッシュのセクタ1の最大ウェイ数を指示します。	○	×	○	×
SCACHE_ISOLATE_ASSIGN(*array1*[,*array2*]...) END_SCACHE_ISOLATE_ASSIGN	キャッシュのセクタ1に載せる配列を指示します。	○	×	○	×
SIMD[({ALIGNED | UNALIGNED})]	SIMD 化を有効にします。	○	○	×	○
NOSIMD	SIMD 化を無効にします。	○	○	×	○
SIMD_LISTV [({ALL | THEN | ELSE})]	IF構文内の実行文をリストベクトル変換することを指示します。	×	×	○	×
SIMD_NOREDUNDANT_VL	SIMD長の倍数冗長実行によるSIMD化機能を無効にします。	○	○	×	○
SIMD_USE_MULTIPLE_STRUCTURES	SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用します。	○	○	×	○
SIMD_NOUSE_MULTIPLE_STRUCTURES	SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用しません。	○	○	×	○
STRIPING[(*n*)]	ループストライピング機能を有効にします。 *n*は展開数(多重度)を表す2～100の整数値です。	○	○	×	○
NOSTRIPING	ループストライピング機能を無効にします。	○	○	×	○
SWP	ソフトウェアパイプライニング機能を有効にします。	○	○	×	○
SWP_FREG_RATE(*n*) SWP_IREG_RATE(*n*) SWP_PREG_RATE(*n*)	ソフトウェアパイプライニング機能における、レジスタ数の条件を変更することを指示します。 *n*はソフトウェアパイプライニングで使用可能なレジスタ数の割合(百分率)を表す1～1000までの整数値です。	○	○	×	○
SWP_POLICY({AUTO|SMALL|LARGE})	ソフトウェアパイプライニングで使用する命令スケジューリングアルゴリズムの選択基準を指示します。	○	○	×	○
SWP_WEAK	ソフトウェアパイプライニング機能を調整し、実行文の重なりを小さくします。	○	○	×	○

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
NOSWP	ソフトウェアパイプライニング機能を無効にします。	○	○	×	○
UNROLL[(*n*)]	DO ループをアンローリングさせます。 *n*は展開数(多重度)を表す2～100の整数値です。	×	○	×	○
UNROLL('full')	フルアンローリング機能を有効にします。	×	○	×	○
NOUNROLL	アンローリング機能を無効にします。	×	○	×	○
UNROLL_AND_JAM[(*n*)]	最適化の効果があると判断したループに対してアンロールアンドジャムを有効にします。 *n*は展開数(多重度)を表す2～100の整数値です。	○	○	×	×
UNROLL_AND_JAM_FORCE[(*n*)]	アンロールアンドジャムを有効にします。 *n*は展開数(多重度)を表す2～100の整数値です。	×	○	×	×
NOUNROLL_AND_JAM	アンロールアンドジャムを無効にします。	○	○	×	×
UNSWITCHING	指定されたIF構文をループアンスイッチングすることを指示します。	×	×	○	×
ZFILL[(*n*)]	ZFILLの最適化を有効にします。 *n*はDC ZVA命令の書き込みブロック数を表す1～100の10進数です。	×	○	×	○
NOZFILL	ZFILLの最適化を無効にします。	×	○	×	○

○: 最適化指示子を最適化制御行に指定できます。
×: 最適化指示子を最適化制御行に指定できません。

1. !OCL ARRAY_DECLARATION_OPT

   最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にします。

   以下に、指定の例を示します。

   例:

```
REAL*8 A(8)
!OCL ARRAY_DECLARATION_OPT
  DO I=1,N
    A(I)= 0
  ENDDO
```

   添字の動作範囲をSIMD長以下と仮定して最適化を行います。

2. !OCL NOARRAY_DECLARATION_OPT

   最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にしません。

   以下に、指定の例を示します。

例:

```
REAL*8 A(8)
!OCL NOARRAY_DECLARATION_OPT
  DO I=1,N
    A(I)= 0
  ENDDO
```

添字の動作範囲を不明として最適化を行います。

3. !OCL ARRAY_FUSION[,*opt*]
   !OCL END_ARRAY_FUSION

   複数の配列代入文の融合を促進させる場合、範囲の開始と終了のそれぞれに!OCL ARRAY_FUSIONと!OCL END ARRAY_FUSIONを記述することにより、その範囲内の融合が促進されます。ただし、この最適化制御行を記述しても、範囲内の配列代入文すべてが融合されるとは限りません。また、*opt*には最適化指示子を指定でき、範囲内のすべての配列代入文に対して有効となります。

   以下に、指定の例を示します。

   例:

```
  REAL, DIMENSION(10) :: A, B, C
     ...
!OCL ARRAY_FUSION          ↑
  A=B+ ...                 |    有効範囲
  C=C+ ...                 |
!OCL END_ARRAY_FUSION      ↓
```

4. !OCL ARRAY_MERGE(*array1*,*array2*[,*array3*]...)

   配列のマージを指示します。対象とする配列変数は、形状明示配列です。

   制約事項については、“9.15.2 局所的な配列変数マージの対象変数の制約事項”と同じ制約事項があります。

   本機能は、宣言式内および宣言部分の定数式には適用されません。

   以下に、指定の例を示します。

   例1:

```
!OCL ARRAY_MERGE(A,B,C)
  PROGRAM MAIN
  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)
```

このようなプログラムを-Koclオプションを指定して翻訳した場合、コンパイラは、以下のようなプログラムとみなして翻訳します。

```
  INTEGER,DIMENSION(3,101,102,103) :: TMP
! Aを1次元目の1要素にマージ:TMP(1,101,102,103)
! Bを1次元目の2要素にマージ:TMP(2,101,102,103)
! Cを1次元目の3要素にマージ:TMP(3,101,102,103)
  CALL SUB(TMP(1,:,:,:)) ! 配列名を形状をもつ配列要素に変更
  …
  PRINT *,TMP(2,101,102,103)
  TMP(1,:,:,:) = TMP(3,:,:,:)
  TMP(1,101,102,103) = TMP(3,1,2,3)
```

注)TMPはコンパイラで生成した変数です。

例2:

```
!OCL ARRAY_MERGE(A,B)
  PROGRAM MAIN
```

```
  INTEGER,DIMENSION(100,101,102) :: A,B
  ...
  CALL SUB(A)
  END PROGRAM
  SUBROUTINE SUB(C)
  INTEGER,DIMENSION(100) :: C
  ...
  END SUBROUTINE
```

本機能を使用した場合、MAIN側の手続SUBの引数は、Aが1次元目に要素を追加した生成変数に変更するため、SUB側の引数C(100)との結合が正しく得られないことがあります。

5. !OCL ARRAY_MERGE(*base_array*:*array1*[,*array2*]...)

ベースの配列変数に配列変数をマージします。マージすることにより、組込み関数の引数キーワードにDIMおよびMASKが指定可能な組込み関数には影響があります。

ベースの変数の最終次元の要素にマージされます。対象とする配列変数は、上下限が定数式な形状明示配列(局所的な配列および共通ブロック実体)です。

制約事項については、"9.15.2 局所的な配列変数マージの対象変数の制約事項"の制約項目の共通ブロック実体を除く項目が制約事項です。本機能は、宣言式内および宣言部分の定数式には適用されません。

以下に、指定の例を示します。

例1:

```
!OCL ARRAY_MERGE(A:B,C)
  INTEGER,DIMENSION(10,11,12,2) :: A
  INTEGER,DIMENSION(10,11,12) :: B,C
  ...
  A = 1
  B = 1
  C(1,2,3) = 1
```

以下のように配列がマージされます。

```
INTEGER,DIMENSION(10,11,12,4) :: A
A = 1
A(:,:,:,3) = 1
A(1,2,3,4) = 1
```

例2:

```
!OCL ARRAY_MERGE(A:B,C)
  INTEGER,DIMENSION(1,2,3,4) :: A,X
  INTEGER,DIMENSION(1,2,3) :: B,C
  ...
  X = A  ! 翻訳時にエラーとなります。
  A = RESHAPE((/(i,i=1,24)/),(/1,2,3,4/))
! 代入の右辺と左辺の形状が異なり、実行結果は保証されません。
  PRINT *,UBOUND(A,4)
! 6が出力される。
```

6. !OCL ARRAY_SUBSCRIPT(*array1*[,*array2*]...)

配列の次元位置の移動を指示します。対象とする配列変数は、割付け配列および上下限が定数式な形状明示配列(局所的な配列、共通ブロック実体および仮引数)です。

制約事項については、"9.14.2 配列宣言の次元移動の対象変数の制約事項"の制約項目の結合実体を除く項目が制約事項です。

本機能は、宣言式内および宣言部分の定数式には適用されません。

以下に、指定の例を示します。

例1:

```
!OCL ARRAY_SUBSCRIPT(A)
  PROGRAM MAIN
  INTEGER,DIMENSION(101,102,103,5) :: A
  CALL SUB(A)
  …
  PRINT *,A(1,2,3,4)
```

このようなプログラムを-Koclオプションを指定して翻訳した場合、コンパイラは、以下のようなプログラムとみなして翻訳します。

```
PROGRAM MAIN
INTEGER,DIMENSION(5,101,102,103) :: A
CALL SUB(A)
…
PRINT *,A(4,1,2,3)
```

例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
```

本機能を使用した場合、MAIN側の手続SUBの引数は、A(3,100,101,102)の配列とSUB側の引数B(100)と結合しますが、使用すると配列の次元数が異なるため、結果が正しく得られないことがあります。

例3:

```
!OCL ARRAY_SUBSCRIPT(A)
  PROGRAM MAIN
  INTEGER,DIMENSION(1,2,3,4) :: A,B
  …
      B = A                        ! 翻訳時にエラーとなります。
      A = RESHAPE((/(I,I=1,24)/),(/1,2,3,4/))
                                   ! 代入の右辺と左辺の形状が異なり、実行結果は保証されません。
      PRINT *,UBOUND(A,4)
                                   ! 3が出力される。
```

7. !OCL ASSUME({SHORTLOOP | NOSHORTLOOP | MEMORY_BANDWIDTH | NOMEMORY_BANDWIDTH | TIME_SAVING_COMPILATION | NOTIME_SAVING_COMPILATION})

プログラムの特徴に合わせて、最適化の調整を行うかどうかを指示します。本最適化制御行は、複数同時に指定することもできます。

!OCL ASSUME(SHORTLOOP)

プログラム中の最内ループにおいて、翻訳時にループの繰返し数が不明な場合に、繰返し数が小さいとみなして最適化を行います。自動並列化、ループアンローリング、ソフトウェアパイプライニングなどの最適化が調整または抑止される可能性があります。

!OCL ASSUME(NOSHORTLOOP)

プログラム中の最内ループにおいて、翻訳時にループの繰返し数が不明な場合に、繰返し数が大きいとみなして最適化を行います。

!OCL ASSUME(MEMORY_BANDWIDTH)

プログラム中の最内ループにおいて、メモリバンド幅がボトルネックであるとみなして最適化を行います。zfillの最適化の促進およびソフトウェアパイプライニングなどの最適化が調整または抑止される可能性があります。

!OCL ASSUME(NOMEMORY_BANDWIDTH)

プログラム中の最内ループにおいて、メモリバンド幅がボトルネックにならないとみなして最適化を行います。

!OCL ASSUME(TIME_SAVING_COMPILATION)

プログラムの翻訳時間が短くなるように最適化を調整します。

!OCL ASSUME(NOTIME_SAVING_COMPILATION)

プログラムの翻訳時間の短縮より、実行可能プログラムの高速化を優先します。

以下に、指定の例を示します。

例:

```
!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
```

対象となるループにおいて、指示された特徴に合わせて、最適化の調整を行います。

8. !OCL CLONE(*var*==*n1*[,*n2*]...)

ループ内で変数*var*の値が不変とみなして、指定された変数*var*と値*n1*[,*n2*]...の等式を条件式とする分岐を生成し、ループを複製することを指示します。分岐は指定した値の順に生成されます。本機能により、フルアンローリングなどのほかの最適化を促進します。対象となるループ内で変数*var*が更新される場合、実行結果が保証されません。詳細については、“9.18 最適化指示子の誤った指定”を参照してください。

本最適化指示子はループを複製するため、オブジェクトプログラムの大きさおよび翻訳時間が増加する場合があります。本最適化指示子は-O3オプションが有効な場合に意味があります。

*var*は整数型の変数です。種別型パラメタは1、2、4、または8です。

整数型であっても、以下の場合は指定できません。

- 構造体成分
- 配列、配列要素、または部分配列
- pointer変数
- allocatable変数
- threadprivate対象変数

また、以下の属性が付く場合も指定できません。

- ALLOCATABLE
- CHANGEENTRY
- CONTIGUOUS
- DIMENSION
- EXTERNAL
- INTRINSIC
- PARAMETER
- POINTER

n1[,*n2*]...は-9223372036854775808から9223372036854775807までの10進数または名前付き定数です。値に関わらず、数字列の後に下線“_”を用いた値の種別型パラメタは指定しないでください。

値の指定方法として、コンマ“,”による列挙、およびコロン“:”による値の範囲指定ができます。コンマ“,”とコロン“:”を組み合わせて指定することが可能です。下記の2つの指定は同等です。

```
!OCL CLONE(var==1,3:5,7)
```

```
!OCL CLONE(var==1,3,4,5,7)
```

また、1つのCLONE指示子で指定できる値の個数の上限は20個です。21個以上の値を指定した場合、個数の上限を超えた値の指定は無効とされます。

下記の指定は、30個の値を指定しているため、個数の制約を受けます。

```
!OCL CLONE(var==11:40)
```

個数の上限を超えた値の指定は無効とされるため、上記のCLONE指示子は、下記のCLONE指示子と同等に扱います。

```
!OCL CLONE(var==11:30)
```

以下に、指定の例を示します。

例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
```

コンマで指示した順番に分岐を生成し、ループが複製されます。

例2:

```
SUBROUTINE SUB(A,N,M)
...
!OCL CLONE(N==10)
!OCL CLONE(M==20)
  DO I=1,N
      A(I) = M
  ENDDO
...
END SUBROUTINE
```

→

```
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
```

同一ループに対して複数のOCLを指定した場合、指定した順番にループが複製されます。

例3:

```
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
```

→

```
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
```

多重ループにOCLを指定した場合、多重に分岐を生成し、ループが複製されます。

9. !OCL EVAL

演算評価方法を変更する最適化を行うことを指示します。

以下に、指定の例を示します。

例:

```
!OCL EVAL
  DO I=1,N
    A(I)=A(I)+B(I)+C(I)+D(I)
  ENDDO
```

→

```
DO I=1,N
  A(I)=(A(I)+B(I))+(C(I)+ D(I))
ENDDO
```

対象となるループ中で演算評価方法を変更する最適化を行います。

10. !OCL NOEVAL

演算評価方法を変更する最適化を抑止することを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOEVAL
  DO I=1,N
    A(I) = A(I) + B(I) + C(I) + D(I)
  ENDDO
```

対象となるループ中で演算評価方法を変更する最適化を行いません。

11. !OCL EVAL_CONCURRENT

tree-height-reduction最適化において、命令の並列性を優先することを指示します。

tree-height-reduction最適化については、“9.1.2.10 tree-height-reduction最適化”を参照してください。

以下に、指定の例を示します。

例:

```
!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

tree-height-reduction最適化において、命令の並列性を抑え、FMA命令の利用を優先することを指示します。

tree-height-reduction最適化については、“9.1.2.10 tree-height-reduction最適化”を参照してください。

以下に、指定の例を示します。

例:

```
!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

SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開します。

以下に、指定の例を示します。

例:

```
!OCL EXTRACT_STRIDE_STORE
DO I=1,N,2
  A(I) = B(I) + C(I)
ENDDO
```

SIMD化可能な配列Aに対し、スカラ命令を利用する最適化を行います。

14. !OCL NOEXTRACT_STRIDE_STORE

SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開しません。

以下に、指定の例を示します。

例:

```
!OCL NOEXTRACT_STRIDE_STORE
DO I=1,N,2
  A(I) = B(I) + C(I)
ENDDO
```

SIMD化可能な配列Aに対し、スカラ命令を利用せずSIMD化命令を利用します。

15. !OCL FISSION_POINT[(*n*)]

ループ分割の最適化を行うことを指示します。*n*には最内ループから数えたネスト数を指定します。

指定されたネスト数の多重ループを対象に分割します。

*n*は1～6の10進数です。*n*の指定を省略した場合、最内(1次元)ループのみを分割します。

FISSION_POINT[(*n*)]指示子は、最内ループに記述した場合のみ有効となります。

以下に、指定の例を示します。

例:

```
 DO I=1,N
   DO J=1,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
```

→

```
DO I=1,N
  DO J=1,N
    A(I,J) = A(I-1,J-1)
  ENDDO
  DO J=1,N
    A(I,J) = A(I,J) + A(I-1,J)
  ENDDO
ENDDO
```

上記の例では、Jのループで分割されます。

16. !OCL FIXED *array1* (*shape-spec-list*)[, *array2* (*shape-spec-list*)]...

ポインタ配列または割付け配列が、以下のいずれかであることを指示します。

ー 未結合または未割付けである。

ー 配列の上下限は、*shape-spec-list*で結合している、または割り付いている。

*array*は、ポインタ配列または割付け配列の名前でなければなりません。

*shape-spec*は、次の形式です。

[*割付け下限* :]*割付け上限*

*割付け下限*および*割付け上限*には、定数式または * が指定できます。

*割付け下限*および*割付け上限*は、*array*が結合状態または割付け状態であるならば、再結合または再割付けはできません。

*割付け下限*または*割付け上限*が * である場合、*array*の上限または下限は、定数で求められないことを指定します。

*割付け下限*を省略した場合は、下限は1になります。

プログラム単位の宣言部に指定します。

最適化制御行は、割付け配列またはポインタ配列の宣言よりも後に指定しなければなりません。

*割付け下限*または*割付け上限*に名前付き定数を指定する場合、最適化制御行は名前付き定数の宣言よりも後で指定しなければなりません。

FIXED最適化指示子の最適化制御行に、ほかの最適化指示子を合わせて指定することはできません。

ポインタ配列を指定した場合には、指定したポインタ配列は、連続領域でなければなりません。

親子結合するポインタ配列または割付け配列は指定できません。

例:

```
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
```

ポインタpの最終次元の上限が定数で求められないことから、FIXED最適化指示子の上限に星印(*)を指定しています。

17. !OCL FP_CONTRACT

Floating-Point Multiply-Add/Subtract 演算命令を使用した最適化を行うことを指示します。なお、Floating-Point Multiply-Add/Subtract 演算命令を使用した場合、丸め誤差程度の違いが生じることがあります。

以下に、指定の例を示します。

例:

```
!OCL FP_CONTRACT
  DO I=1,N
    A(I) = A(I) + B(I) * C(I)
  ENDDO
```

18. !OCL NOFP_CONTRACT

Floating-Point Multiply-Add/Subtract 演算命令を使用した最適化を行わないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOFP_CONTRACT
  DO I=1,N
    A(I) = A(I) + B(I) * C(I)
  ENDDO
```

19. !OCL FP_RELAXED

単精度、倍精度の浮動小数点除算、SQRT関数について、逆数近似命令を使用した高速な演算を行うことを指示します。なお、逆数近似命令を使用した場合、除算命令およびSQRT命令を利用する場合に比べて、丸め誤差程度の違いが生じることがあります。

また、-NRtrapオプションが有効、かつ-Knosimdオプションまたは-KNOSVEオプションのいずれかが有効な場合、SQRT関数に対する逆数近似命令への変換が抑止されます。したがって、-NRnotrapオプションが有効な場合と比べ、実行性能が低下する場合があります。

以下に、指定の例を示します。

例:

```
!OCL FP_RELAXED
  DO I=1, N
    A(I) = SQRT(B(I) / C(I))
  ENDDO
```

20. !OCL NOFP_RELAXED

単精度、倍精度の浮動小数点除算、SQRT関数について、逆数近似命令を使用しない除算命令およびSQRT命令を使用する演算を行うことを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOFP_RELAXED
  DO I=1, N
    A(I) = SQRT(B(I) / C(I))
  ENDDO
```

21. !OCL FULLUNROLL_PRE_SIMD[(*n*)]

SIMD化前のフルアンローリングを促進することを指示します。*n*は、対象ループの回転数の上限を表す2～100の整数値です。*n*の値が省略された場合、コンパイラが自動的に最適な値を決定します。

なお、この最適化制御行は指定した直後のDOループまたは配列記述のみが対象となります。

繰返し数が不明な場合は、最適化を行いません。

以下に、指定の例を示します。

例:

```
  DO I=1,N
!OCL FULLUNROLL_PRE_SIMD
    DO J=1,16
      A(J,I) = B(I,J) + C(I,J)
```

```
    ENDDO
  ENDDO
```

内側ループに対してSIMD化前のフルアンローリングを適用します。

22. !OCL NOFULLUNROLL_PRE_SIMD

SIMD化前のフルアンローリングを抑止することを指示します。

なお、この最適化制御行は指定した直後のDOループまたは配列記述のみが対象となります。

以下に、指定の例を示します。

例:

```
  DO I=1,N
!OCL NOFULLUNROLL_PRE_SIMD
    DO J=1,2
      A(J,I) = B(I,J) + C(I,J)
    ENDDO
  ENDDO
```

内側ループに対してSIMD化前のフルアンローリングを適用しません。

23. !OCL ILFUNC

組込み関数および演算をインライン展開することを指示します。対象を無条件にインライン展開することで性能低下が生じる可能性があります。

なお、本最適化によってインライン展開が行われると、実行結果に副作用を生じることがあります。また、-NRtrapオプションが無効な場合でも、プログラムの論理上は発生しない浮動小数点例外を発生させることがあります。

以下に、指定の例を示します。

例:

```
!OCL ILFUNC
  DO I=1,N
    A(I) = SIN(B(I))
  ENDDO
```

24. !OCL NOILFUNC

組込み関数および演算をインライン展開しないことを指示します。

以下に、指定の例を示します。

例:

```
!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*)

指定された値をループの繰返し数とみなして最適化を行うことを指示します。

n1、*n2*、*n3*の範囲は、0～2147483647です。max=*n1*、avg=*n2*、min=*n3*は、コンマを区切りとして順不同で複数指定が可能です。ただし、複数指定した値の大小関係は、*n1*≧*n2*≧*n3*とする必要があります。

max=*n1*を指定した場合は、ループの繰返し数の最大値を*n1*とみなして最適化します。

avg=*n2*を指定した場合は、ループの繰返し数の平均値を*n2*とみなして最適化します。

min=*n3*を指定した場合は、ループの繰返し数の最小値を*n3*とみなして最適化します。

本最適化指示子は、翻訳時にループの繰返し数が不明な場合に有効です。

指定した平均値*n2*は、コンパイラが最適化の参考情報として利用しますが、実際のループの繰返し数の平均値と異なる場合でも実行結果は保証されます。

注意

以下の場合は、実行結果は保証されません。

－ 実際のループの繰返し数が、指定した最大値*n1*より大きくなる場合

－ 実際のループの繰返し数が、指定した最小値*n3*より小さくなる場合

詳細は、“9.18.4 ITERATIONS指示子の例”を参照してください。

本最適化指示子は、プログラム単位、DOループ単位に指定できますが、複数ループがあるプログラムに対して、一律同じ指定が有効なケースばかりではないため、DOループ単位の指定を推奨します。

以下に、指定の例を示します。

例1: 繰返し数の平均値を指定

```
!OCL ITERATIONS(avg=32)
  DO I=1, M
    A(I) = B(I) + C(I)
  ENDDO
```

例2: 繰返し数の最小値および平均値を指定

```
!OCL ITERATIONS(min=1,avg=8)
  DO I=1, M
    A(I) = B(I) + C(I)
  ENDDO
```

例3: 繰返し数の最大値、平均値、および最小値を指定

```
!OCL ITERATIONS(max=128,avg=16,min=16)
  DO I=1, M
    A(I) = B(I) + C(I)
  ENDDO
```

26. !OCL LOOP_BLOCKING(*n*)

ブロッキングの最適化を行うことを指示します。*n*にはブロッキング時のブロックサイズを指定します。

以下に、指定の例を示します。

例:

```
!OCL LOOP_BLOCKING(80)
  DO J=1,N
    DO I=1,N
      A(I,J) = A(I,J) + B(J,I)
    ENDDO
  ENDDO
```

→

```
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
```

ブロックサイズ80でブロッキングの最適化を行います。

27. !OCL LOOP_NOBLOCKING

ブロッキングの最適化を抑止することを指示します。

以下に、指定の例を示します。

例:

```
!OCL LOOP_NOBLOCKING
  DO J=1,N
    DO I=1,N
      A(I,J) = A(I,J) + B(J,I)
    ENDDO
  ENDDO
```

28. !OCL LOOP_FISSION_STRIPMINING[({ *n* | *c-level* })]

自動ループ分割時にストリップマイニングの最適化を行うことを指示します。ストリップの長さを指定する方法には、直接長さ(*n*)を指示する方法と、各キャッシュ階層(*c-level*)に適したサイズとなるように指示する方法があります。*n*には、ストリップの長さを2から100000000までの整数値で指定できます。*c-level*には、'L1'または'L2'を指定できます。'L1'が指定された場合、キャッシュメモリの利用効率を考慮し、ストリップの長さを1次キャッシュのサイズに合わせます。'L2'が指定された場合、2次キャッシュのサイズに合わせます。*n*および*c-level*が省略された場合、コンパイラが自動的に値を決定します。本最適化指示子は、最適化制御行に最適化指示子LOOP_FISSION_TARGETが指定されている、かつ-Kloop_fissionオプションおよび-O2オプション以上が有効な場合に意味があります。

ストリップマイニングについては、“9.1.2.11.1 ストリップマイニング”を参照してください。

以下に、指定の例を示します。

例:

```
  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
```

分割したループの外側にループを生成し、ストリップ長256でストリップマイニングします。同時に、ループ間の途中結果を格納する一時的な配列をコンパイラが生成します。この配列TEMPARRAY_Qの要素数は、ストリップ長と同じ256となります。

29. !OCL LOOP_NOFISSION_STRIPMINING

ストリップマイニングの最適化を抑止することを指示します。

以下に、指定の例を示します。

例:

```
  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})]

指定されたループに対して、自動ループ分割の最適化を行うことを指示します。CLまたはLSはループ分割のアルゴリズムを指定するための指示子です。本最適化指示子は、-Kloop_fissionオプションが有効な場合に意味があります。CLおよびLSが省略された場合、CLを指定したものとみなします。

ループ分割については、“9.1.2.11 ループ分割”を参照してください。

!OCL LOOP_FISSION_TARGET(CL)

指定されたループに対して、クラスタリングアルゴリズムでループ分割を行うことを指示します。本機能では、ループ分割に伴う一時的なデータ転送のための作業配列の削減を優先したループ分割を行います。本機能により翻訳時間が増加します。

以下に、指定の例を示します。

例:

```
!OCL LOOP_FISSION_TARGET(CL)
  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
```

ループ分割に伴う一時的なデータ転送のための作業配列の削減を優先して、ループを分割します。

!OCL LOOP_FISSION_TARGET(LS)

指定されたループに対して、局所探索アルゴリズムでのループ分割を行うことを指示します。本機能では、ソフトウェアパイプライニングの促進を優先したループ分割を行います。本機能により、クラスタリングアルゴリズムに比べてさらに翻訳時間が増加します。

以下に、指定の例を示します。

例:

```
!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)
    ...
  ENDD O
```

ソフトウェアパイプライニングの促進を優先して、ループを分割します。

31. !OCL LOOP_FISSION_THRESHOLD(*n*)

自動ループ分割における分割後のループの粒度を決める閾値を指示します。*n*の範囲は、1～100です。本最適化指示子は、最適化制御行に最適化指示子LOOP_FISSION_TARGETが指定されている、かつ-Kloop_fissionオプションおよび-O2オプション以上が有効な場合に意味があります。

以下に、指定の例を示します。例1に比べて、LOOP_FISSION_THRESHOLD指示子に小さい値を指定した例2の方が細かい粒度でループが分割され、分割ループ数が増えます。

例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
```

例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*]...)

多重DOループの入れ換えを指示します。これにより、指定されたDOループの並びにすることができます。ただし、入れ換えたときに結果が異なるなど、入れ換えが不可能な場合には、行いません。

以下に、指定の例を示します。

例:

```
!OCL LOOP_INTERCHANGE(I,J)
  DO I=1,M
    DO J=1,N
      A(I,J) = A(I,J) + B(J,I)
    ENDDO
  ENDDO
```

→

```
DO J=1,N
  DO I=1,M
    A(I,J) = A(I,J) + B(J,I)
  ENDDO
ENDDO
```

LOOP_INTERCHANGEの指定のない場合は、DO変数Iで並列化されますが、LOOP_INTERCHANGEの指定によりループを入れ換え、DO変数Jで並列化されます。変数Nが変数Mより十分に大きい場合、並列化の効果が大きくなります。

33. !OCL LOOP_NOINTERCHANGE

多重DOループの入れ換えを行わないことを指示します。

以下に、指定の例を示します。

例:

```
!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

指定されたループと隣接したループの融合を抑止することを指示します。

以下に、指定の例を示します。

例:

```
!OCL LOOP_NOFUSION
  DO I=1,N
    文
  ENDDO
  DO J=1,N
    文
```

```
ENDDO
DO K=1,N
  文
ENDDO
```

IとJのループは融合されません。JとKのループは融合されます。

35. !OCL LOOP_PART_SIMD

ループを分割して部分的にSIMD化することを指示します。

以下に指定の例を示します。

例:

```
!OCL LOOP_PART_SIMD
  DO I=2,N
    A(I) = A(I)-B(I)+LOG(C(I))  ! SIMD化可能
    D(I) = D(I-1)+A(I)          ! SIMD化不可
  ENDDO
```

→

```
!OCL LOOP_PART_SIMD
  DO I=2,N
    A(I) = A(I)-B(I)+LOG(C(I))  ! SIMD実行
  ENDDO
  DO II=2,N
    D(II) = D(II-1)+A(II)       ! 非SIMD実
行
  ENDDO
```

分割されたIのループのみSIMD化されます。

36. !OCL LOOP_NOPART_SIMD

部分的にSIMD化しないことを指示します。

以下に指定の例を示します。

例:

```
!OCL LOOP_NOPART_SIMD
  DO I=1,N
    A(I) = A(I)-B(I)+LOG(C(I))
    D(I) = D(I-1)+A(I)
  ENDDO
```

部分的にSIMD化されません。

37. !OCL LOOP_PERFECT_NEST

不完全多重ループを分割して完全多重ループにすることを指示します。

以下に指定の例を示します。

例:

```
!OCL LOOP_PERFECT_NEST
  DO J=1,N                  ! 不完全多重ループ
    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                  ! 完全多重ループ
    DO I=1,N
      C(J,I) = D(J,I)+A(J)
    ENDDO
  ENDDO
```

J,Iの不完全多重ループを分割して、完全多重ループにします。

38. !OCL LOOP_NOPERFECT_NEST

不完全多重ループを完全多重ループにしないことを指示します。

以下に指定の例を示します。

例:

```
!OCL LOOP_NOPERFECT_NEST
  DO J=1,N                    ! 不完全多重ループ
    A(J) = B(J)+1
    DO I=1,N
      C(J,I) = D(J,I)+A(J)
    ENDDO
  ENDDO
```

J,Iの不完全多重ループを完全多重ループにしません。

39. !OCL LOOP_VERSIONING

ループバージョニングの最適化を行うことを指示します。

LOOP_VERSIONING指示子は、最内ループのみ有効となります。

以下に、指定の例を示します。

例:

```
  DO I=1,N
!OCL LOOP_VERSIONING
    DO J=1,N
      A(J) = A(J+M) + B(J,I)
    ENDDO
  ENDDO
```

翻訳時は、変数Nおよび変数Mの値が不明なため、配列Aのデータ依存も不明になります。LOOP_VERSIONING指示子が有効な場合、実行時に変数Nおよび変数Mの値を用いて、最適化を適用したループまたは最適化を適用しないループのどちらかを選択します。

40. !OCL LOOP_NOVERSIONING

ループバージョニングの最適化を行わないことを指示します。

LOOP_NOVERSIONING指示子は、最内ループのみ有効となります。

以下に、指定の例を示します。

例:

```
  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*)]

DOループ内の組込み関数および配列代入文の組込み関数および演算をマルチ演算関数に変換することを指示します。*level*には、1、2または3を指定することができます。*level*の詳細については、翻訳時オプション-Kmfuncを参照してください。

以下に、指定の例を示します。

例: ループ単位で記述した場合

```
  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))
  END DO
  D = LOG(C)+LOG(G)
  :
```

LOG(B(I))、LOG(F(I))がマルチ演算関数に変換されます。

42. !OCL NOMFUNC

DOループ内の組込み関数および配列代入文の組込み関数および演算をマルチ演算関数に変換しないことを指示します。

以下に、指定の例を示します。

例:

```
  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))
  END DO
  D = LOG(C)+LOG(G)
  :
```

LOG(B(I))、LOG(F(I))がマルチ演算関数に変換されません。

43. !OCL NOALIAS

NOALIAS指示子は、異なるポインタ変数が同一の記憶領域を指さないことを本処理系に指示します。

ポインタ変数が記憶領域のどこを占めるかは実行時に決まりますが、NOALIAS指示子を指定することにより、異なるポインタ変数が同一の記憶領域を指さないことを翻訳時に判断することができます。これにより、ポインタ変数に関する最適化が促進されます。ただし、ポインタ変数の結合状態がループ中で変更される場合、本最適化指示子を指定しても最適化が促進されないことがあります。

以下に、指定の例を示します。

例:

```
  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
```

[注意事項]

NOALIAS指示子が指定されたループ中に同一の記憶領域を指すポインタ変数が複数含まれる場合、実行結果は保証されません。

44. !OCL NOARRAYPAD(*array1*)

指定された配列要素へのパディング処理を行わないことを指示します。翻訳時オプション-Karraypad_const[=*N*]または-Karraypad_expr=*N*が有効な場合に意味があります。

以下に、指定の例を示します。

例:

```
!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
```

配列Aに対して、パディング処理は行われません。

45. !OCL NORECURRENCE[(*array1*[,*array2*]...)]

NORECURRENCE指示子は、DOループ内の演算対象となる配列の要素が、回転を跨いで定義引用されないことを本処理系に指示します。

これにより、配列の定義引用順序が不明で最適化できなかったDOループを、最適化の対象にします。

以下の最適化などが該当します。

－ ループスライス(自動並列化)

－ SIMD化

－ ソフトウェアパイプライニング

ここで、*array*は回転を跨いで定義引用されない配列名です。*array*にはワイルドカード指定も可能です。また、配列名を省略すると、対象範囲内のすべての配列に有効となります。ワイルドカード指定については、“12.2.6.5 ワイルドカード指定”を参照してください。

“NORECURRENCE指定のないDOループの例”の場合は、配列Aの添字式が別の配列要素L(I)のため、本処理系は、配列Aがループスライスしても問題がないか判断できません。したがって、このDOループはループスライスされません。

例: NORECURRENCE指定のないDOループ

```
REAL, DIMENSION(10000)  ::  A, B
INTEGER, DIMENSION(10000) :: L
DO I=1,10000
   A(L(I)) = A(L(I)) + B(I)
ENDDO
END
```

もし配列Aがループスライスしても問題がないと分かっているのであれば、“NORECURRENCE指示子の使用例”のようにNORECURRENCEを使用することにより、このDOループはループスライスされます。

例: NORECURRENCE指示子の使用例

```
      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
```

[注意事項]

NORECURRENCE指示子が繰返し数に依存する配列に対して誤って指定された場合、実行結果は保証されません。

46. !OCL NOVREC[(*array1*[,*array2*]...)]

NOVREC指示子は、ループ中に回帰演算となる配列がないことを指示します。ループ中の配列に対しSIMD命令を利用しますが、演算の種類やループ構造によりSIMD命令が利用されない場合もあります。

また、*array*には、ポインタ配列を指定することはできません。

以下に指定の例を示します。

例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
```

ループ中の配列がすべて回帰演算とならないことを指示します。配列AおよびBの演算についてSIMD命令が利用されます。

例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
```

データ依存が不明な配列Aに対して、回帰演算でないことを指示します。配列AおよびBの演算についてSIMD命令が利用されます。

NOVREC指示子の誤った指定については、“9.18.1 NOVREC指示子の例”を参照してください。

47. !OCL PREEX

不変式の先行評価の最適化を行うことを指示します。

以下に、指定の例を示します。

例:

```
!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
```

不変式の先行評価の最適化を行います。

48. !OCL NOPREEX

不変式の先行評価の最適化を抑止することを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOPREEX
  DO I=1,N
    IF (M(I)) THEN
      A(I) = A(I) / B(K)
    ENDIF
  ENDDO
```

不変式の先行評価の最適化を行いません。

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*)

コンパイラの自動プリフェッチ機能が動作することを指示します。自動プリフェッチ機能とは、コンパイラが自動的にprefetch命令を挿入するのに最適な位置を判断し、prefetch命令を生成する機能です。

50. !OCL PREFETCH_CONDITIONAL

IF構文やCASE構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成することを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_CONDITIONAL
  DO I=1,N
    IF(X(I) == 1.0) THEN
      A(I) = B(I)
    ENDIF
  ENDDO
```

IF構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成します。

51. !OCL PREFETCH_NOCONDITIONAL

IF構文やCASE構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成しないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_NOCONDITIONAL
  DO I=1,N
    IF(X(I) == 1.0) THEN
      A(I) = B(I)
    ENDIF
  ENDDO
```

IF構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成しません。

52. !OCL PREFETCH_INDIRECT

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を生成することを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_INDIRECT
  DO I=1,N
    A(INDX(I)) = B(INDX(I))
  ENDDO
```

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を生成します。

53. !OCL PREFETCH_NOINDIRECT

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を生成しないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_NOINDIRECT
  DO I=1,N
    A(INDX(I)) = B(INDX(I))
  ENDDO
```

ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を生成しません。

54. !OCL PREFETCH_LINE(*n*)

1次キャッシュに対してprefetch命令を生成する際、*n*キャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。*n*は1～100までの10進数です。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_LINE(20)
  DO I=1,N
```

```
    A(I) = B(I)
  ENDDO
```

ループ内で生成される1次キャッシュのprefetch命令は、20キャッシュライン先のデータをプリフェッチの対象とします。

55. !OCL PREFETCH_LINE_L2(*n*)

2次キャッシュに対してprefetch命令を生成する際、*n*キャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。*n*は1～100までの10進数です。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_LINE_L2(20)
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

ループ内で生成される2次キャッシュのprefetch命令は、20キャッシュライン先のデータをプリフェッチの対象とします。

56. !OCL PREFETCH_READ(*name*[,level={ 1 | 2 }][,strong={ 0 | 1 }])

参照されているデータに対してprefetch 命令を生成することを指示します。

*name*にはプログラム中で参照されているデータ(配列要素または部分配列)を指定します。添字には、式も指定可能です。*name*に部分配列を指定する場合は、刻み幅のない添字三つ組をもつ部分配列を指定してください。level={ 1 | 2 }を指定することにより、どのキャッシュレベルにデータをプリフェッチするかを指示します。デフォルトは、level=1です。

strong={ 0 | 1 }を指定することにより、strong prefetchとするかどうかを指示します。strong=0が指定された場合はstrong prefetchとせず、strong=1が指定された場合はstrong prefetchとします。デフォルトは、strong=1です。

なお、参照かつ定義されているデータに関しては、PREFETCH_WRITEを使用してください。

57. !OCL PREFETCH_WRITE(*name*[,level={ 1 | 2 }][,strong={ 0 | 1 }])

定義されているデータに対してprefetch 命令を生成することを指示します。

*name*にはプログラム中で定義されているデータ(配列要素または部分配列)を指定します。添字には、式も指定可能です。*name*に部分配列を指定する場合は、刻み幅のない添字三つ組をもつ部分配列を指定してください。level={ 1 | 2 }を指定することにより、どのキャッシュレベルにデータをプリフェッチするかを指示します。デフォルトは、level=1です。

strong={ 0 | 1 }を指定することにより、strong prefetchとするかどうかを指示します。strong=0が指定された場合はstrong prefetchとせず、strong=1が指定された場合はstrong prefetchとします。デフォルトは、strong=1です。

以下に、指定の例を示します。

例1: A(INDX(I+16))に対してprefetch命令を生成する場合

```
  DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)))
    T = T + A(INDX(I))
  ENDDO
```

例2: A(INDX(I+16))に対して2次キャッシュのprefetch命令を生成する場合

```
  DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)),level=2)
    T = T + A(INDX(I))
  ENDDO
```

例3: A(INDX(I+16))に対してstrong prefetchとせずにprefetch命令を生成する場合

```
  DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)),strong=0)
    T = T + A(INDX(I))
  ENDDO
```

例4: INDX(I+32)およびA(INDX(I+16))に対してprefetch命令を生成する場合

```
  DO I = 1,N
!OCL PREFETCH_READ(INDX(I+32))
!OCL PREFETCH_READ(A(INDX(I+16)))
    T = T + A(INDX(I))
  ENDDO
```

また、キャッシュ使用効率の関係上、コンパイラの自動プリフェッチ機能と排他制御したい場合は、例5のようにOCLを組み合わせることにより実現可能です。

例5: OCLを組み合わせて指定

```
!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
```

*name*には、例6のように部分配列も指定できます。部分配列を引数とするPREFETCH_READは、配列要素を引数とするPREFETCH_READを呼び出すループに展開されます。なお、プリフェッチする配列要素が連続である場合、回転毎にPREFETCH_READはされず、キャッシュラインを意識してプリフェッチが出力されます。

例6は、以下のような条件を満たしている場合に性能向上を期待できる利用方法の例です。

- OpenMPのスレッド並列指定はPREFETCH_READと後続のDO変数Iのループを含む親ループに指定している（PREFETCH_READするデータがDO変数Iで参照するデータと同じスレッドで動作することを期待）
- PREFETCH_READしたデータを参照するDO変数Iのループの命令数は多いが回転数は少ない
- PREFETCH_READするデータはDO変数Iのループの次回転で使うデータである

例6: 部分配列指定

```
!$OMP PARALLEL DO
DO J=1,N
    :
!OCL PREFETCH_READ(A(1:IDX(J),J+1),level=2,strong=1)
    :
  DO I=1,IDX(J) ! 命令数が多いループ
    :
    ... = A(I,J) + ...
    :
  ENDDO
    :
ENDDO
!$OMP END PARALLEL DO
```

また、部分配列指定の利用は、以下の仕様に注意してください。

- 自動並列（-Kparallelまたは-Kparallel_strong）との併用はできません。OpenMP（-Kopenmp）で利用してください。
- !$OMP PARALLEL DOなどのスレッド並列指定とそのターゲットの間に、部分配列を引数とするPREFETCH_READ、PREFETCH_WRITEを記述した場合、警告が出力され、無効となります。警告の対象となるOpenMPの記述は以下です。

  DO, PARALLEL DO, DO SIMD, PARALLEL DO SIMD, WORKSHARE, PARALLEL WORKSHARE, TASKLOOP, TASKLOOP SIMD

- 部分配列を引数とするPREFETCH_READ、PREFETCH_WRITEはループに展開されるため、例えば、完全ネストループの途中に挿入することで、一重化などの最適化が動作しなくなるなど、性能に影響する最適化が止まる可能性があります。

58. !OCL PREFETCH_SEQUENTIAL[({AUTO | SOFT})]

連続的にアクセスされる配列データに対してprefetch命令を生成することを指示します。なお、AUTOおよびSOFTを省略した場合はAUTOを指定したものとみなします。

!OCL PREFETCH_SEQUENTIAL(AUTO)

ループ内で使用される連続的にアクセスされる配列データに対して、ハードウェアプリフェッチを利用するか、prefetch命令を生成するかどうかをコンパイラが自動的に選択することを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_SEQUENTIAL(AUTO)
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループにおいてハードウェアプリフェッチを利用しprefetch命令を生成しません。

!OCL PREFETCH_SEQUENTIAL(SOFT)

ループ内で使用される連続的にアクセスされる配列データに対して、ハードウェアプリフェッチを利用せずに、prefetch命令を生成することを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_SEQUENTIAL(SOFT)
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループにおいてハードウェアプリフェッチを利用せずにprefetch命令を生成します。

59. !OCL PREFETCH_NOSEQUENTIAL

連続的にアクセスされる配列データに対してprefetch命令を生成しないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_NOSEQUENTIAL
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループにおいてprefetch命令を生成しません。

60. !OCL PREFETCH_STRIDE[({SOFT | HARD_AUTO | HARD_ALWAYS})]

ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、プリフェッチを実施することを指示します。なお、引数を省略した場合はSOFTを指定したものとみなします。

!OCL PREFETCH_STRIDE(SOFT)

ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、prefetch命令を生成して、プリフェッチを実施することを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_STRIDE(SOFT)
  DO I=1,N,K
    A(I) = B(I)
  ENDDO
```

対象となるループにおいて、prefetch命令を生成します。

!OCL PREFETCH_STRIDE(HARD_AUTO)

ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、prefetch命令を生成せず、ハードウェアストライドプリフェッチャーを利用して、プリフェッチを実施すること指示します。本最適化指示子を指定した場合、キャッシュ上にないデータのみプリフェッチするようストライドプリフェッチャーを設定します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_STRIDE(HARD_AUTO)
  DO I=1,N,K
    A(I) = B(I)
  ENDDO
```

対象となるループにおいてprefetch命令を生成せず、ハードウェアストライドプリフェッチャーを利用します。キャッシュ上にないデータのみプリフェッチを行うよう、ストライドプリフェッチャーを設定します。

!OCL PREFETCH_STRIDE(HARD_ALWAYS)

ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、prefetch命令を生成せず、ハードウェアストライドプリフェッチャーを利用して、プリフェッチを実施すること指示します。本最適化指示子を指定した場合、PREFETCH_STRIDE(HARD_AUTO)指示子とは異なり、常にプリフェッチを行うようストライドプリフェッチャーを設定します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_STRIDE(HARD_ALWAYS)
  DO I=1,N,K
    A(I) = B(I)
  ENDDO
```

対象となるループにおいてprefetch命令を生成せず、ハードウェアストライドプリフェッチャーを利用します。常にプリフェッチを行うよう、ストライドプリフェッチャーを設定します。

!OCL PREFETCH_NOSTRIDE

ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、prefetch命令を生成しません。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_NOSTRIDE
  DO I=1,N,K
    A(I) = B(I)
  ENDDO
```

対象となるループにおいてprefetch命令を生成しません。

61. !OCL PREFETCH_STRONG

1次キャッシュに対して生成されるprefetch命令をstrong prefetchとすることを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_STRONG
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループで生成される1次キャッシュのprefetch命令はstrong prefetchとなります。

62. !OCL PREFETCH_NOSTRONG

1次キャッシュに対して生成されるprefetch命令をstrong prefetchとしないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_NOSTRONG
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループで生成される1次キャッシュのprefetch命令はstrong prefetchとなりません。

63. !OCL PREFETCH_STRONG_L2

2次キャッシュに対して生成されるprefetch命令をstrong prefetchとすることを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_STRONG_L2
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループで生成される2次キャッシュのprefetch命令はstrong prefetchとなります。

64. !OCL PREFETCH_NOSTRONG_L2

2次キャッシュに対して生成されるprefetch命令をstrong prefetchとしないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL PREFETCH_NOSTRONG_L2
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

対象となるループで生成される2次キャッシュのprefetch命令はstrong prefetchとなりません。

65. !OCL PRELOAD

ロード命令を投機実行する最適化を行うことを指示します。

以下に、指定の例を示します。

例:

```
!OCL PRELOAD
  DO I=1,N
    IF (M(I) > 0) THEN
      A(I) = B(I) + C(I)
    ENDIF
  ENDDO
```

ロード命令を投機実行する最適化を行います。

66. !OCL NOPRELOAD

ロード命令を投機実行する最適化を行わないことを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOPRELOAD
  DO I=1,N
```

```
    IF (M(I) > 0) THEN
      A(I) = B(I) + C(I)
    ENDIF
  ENDDO
```

ロード命令を投機実行する最適化を行いません。

67. !OCL SCACHE_ISOLATE_WAY(L2=*n1*[,L1=*n2*])
!OCL END_SCACHE_ISOLATE_WAY

“9.17.2.1 最適化制御行によるソフトウェア制御”を参照してください。

68. !OCL SCACHE_ISOLATE_ASSIGN(*array1*[,*array2*...])
!OCL END_SCACHE_ISOLATE_ASSIGN

“9.17.2.1 最適化制御行によるソフトウェア制御”を参照してください。

69. !OCL SIMD[({ALIGNED | UNALIGNED})]

SIMD化することを指示します。ただし、演算の種類やループ構造によりSIMD化しない場合もあります。

以下に、指定の例を示します。

例:

```
  REAL(KIND=8),DIMENSION(10) :: A,B
    …
!OCL SIMD
  DO I=1,10
    A(I) = A(I) + B(I)
  ENDDO
```

SIMD化を行います。

!OCL SIMD({ALIGNED|UNALIGNED})

!OCL SIMDと等価です。

ALIGNEDおよびUNALIGNEDパラメタの利用は、本製品では非推奨ですが、旧製品とのソースコード互換を保つためにサポートします。

70. !OCL NOSIMD

SIMD化を行わないことを指示します。

以下に、指定の例を示します。

例:

```
  REAL(KIND=8),DIMENSION(10) :: A,B
    …
!OCL NOSIMD
  DO I=1,10
    A(I) = A(I) + B(I)
  ENDDO
```

SIMD化を行いません。

71. !OCL SIMD_LISTV[({ALL | THEN | ELSE})]

IF構文内の実行文をリストベクトル変換することを指示します。ALL、THENまたはELSEはリストベクトル変換の適用範囲を制御するための指示子です。本最適化指示子は、-O2オプション以上が有効、かつ-Ksimd=2または最適化指示子SIMDにより-Ksimd=2相当の機能が動作する場合に意味があります。ALL、THEN、およびELSEを省略した場合、ALLを指定したものとみなします。

リストベクトル変換については、“9.1.1.7.3 リストベクトル変換”を参照してください。

!OCL SIMD_LISTV(THEN)

IF THEN文の直後のブロック内の文をリストベクトル変換することを指示します。

以下に、指定の例を示します。

例:

```
!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
```

IF THEN文の直後のブロック内の文をリストベクトル変換します。

!OCL SIMD_LISTV(ELSE)

ELSE文の直後のブロック内の文をリストベクトル変換することを指示します。

以下に、指定の例を示します。

例:

```
!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
```

ELSE文の直後のブロック内の文をリストベクトル変換します。

!OCL SIMD_LISTV(ALL)

IF THEN文およびELSE文の両方のブロック内の文をリストベクトル変換することを指示します。

以下に、指定の例を示します。

例:

```
!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
```

IF THEN文およびELSE文の両方のブロック内の文をリストベクトル変換します。

72. !OCL SIMD_NOREDUNDANT_VL

SIMD長の倍数冗長実行によるSIMD化機能を無効にします。SIMD長の倍数冗長実行によるSIMD化機能については“9.1.1.7.4 SIMD長の倍数冗長実行によるSIMD化機能”の説明を参照してください。

以下に、指定の例を示します。本最適化指示子は、特定のループだけその機能を抑止することを指示するために利用します。

例:

```
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

SIMD化の際、SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用することを指示します。

本最適化指示子は、-Ksimd[={1|2|auto}]オプションまたはsimd指示子が有効、かつ-KSVEオプションが有効である場合に意味があります。

以下に、指定の例を示します。

例:

```
  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
```

対象となるループにおいて、SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用します。

74. !OCL SIMD_NOUSE_MULTIPLE_STRUCTURES

SIMD化の際、SVEのLoad Multiple Structures命令および、Store Multiple Structures命令を利用しないことを指示します。

以下に、指定の例を示します。

例:

```
  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
```

対象となるループにおいて、SVEのLoad Multiple Structures命令およびStore Multiple Structures命令を利用しません。

75. !OCL STRIPING[(*n*)]

ストライピングの最適化を行うことを指示します。ストライプ長(展開数)を*n*で指定することができます。*n*は2から100までの整数値を指定することができます。*n*を省略した場合は2を指定したものとみなします。ソースプログラムでループの繰返し数が既知の場合、*n*に繰返し数を超える値を指定しても、既知の繰返し数が優先されます。

以下に、指定の例を示します。

例1:

```
!OCL STRIPING
  DO I=1,N
    文
  ENDDO
```

文をコンパイラが決定したストライピング長でループストライピングすることを指定します。

例2:

```
!OCL STRIPING(4)
  DO I=1,N
    文
  ENDDO
```

文をストライピング長4でループストライピングすることを指定します。

例3:

```
!OCL STRIPING(8)
  DO I=1,4
    文
  ENDDO
```

ループの繰返し数(4)を超えるストライプ長(8)を指定しているので、ストライプ長はコンパイラが自動的に決定した展開数が有効となります。

例4:

```
!OCL STRIPING(4)
  SUBROUTINE SUB
  ...
  ...
  DO I=1,N
    文
  ENDDO
  DO I=1,N
    文
  ENDDO
```

プログラム単位の先頭に指定した場合、プログラム内のすべてのDO ループが対象となります。

76. !OCL NOSTRIPING

ループストライピングの最適化を抑止することを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOSTRIPING
  DO I=1,N
    文
  ENDDO
```

ループストライピングの最適化を行いません。

77. !OCL SWP

ソフトウェアパイプライニングの最適化を行うことを指示します。翻訳時に-O2オプション以上が有効な場合に意味があります。

以下に、指定の例を示します。

例:

```
!OCL SWP
  DO I=1,N
    文
  ENDDO
```

ソフトウェアパイプライニングの最適化を行います。

78. !OCL SWP_FREG_RATE(*n*)
!OCL SWP_IREG_RATE(*n*)
!OCL SWP_PREG_RATE(*n*)

以下のレジスタについて、ソフトウェアパイプライニングで使用可能な割合(百分率)を指示します。

－ 浮動小数点レジスタおよびSVEのベクトルレジスタ

－ 整数レジスタ

－ SVEのプレディケートレジスタ

SWP_FREG_RATE指示子を指定した場合は、浮動小数点レジスタおよびSVEのベクトルレジスタについて指示します。

SWP_IREG_RATE指示子を指定した場合は、整数レジスタについて指示します。

SWP_PREG_RATE指示子を指定した場合は、プレディケートレジスタについて指示します。

*n*には、ソフトウェアパイプライニングで使用可能なレジスタ数の割合(百分率)を指定します。*n*は1から1000までの整数値を指定することができます。

翻訳時に-O2オプション以上が有効な場合に意味があります。

以下に、指定の例を示します。

例:

```
!OCL SWP_FREG_RATE(120)
!OCL SWP_IREG_RATE(150)
!OCL SWP_PREG_RATE(80)
  DO I=1,N
    文
  ENDDO
```

レジスタ数に関する条件を調整したソフトウェアパイプライニングの最適化を行います。

79. !OCL SWP_POLICY({AUTO|SMALL|LARGE})

ソフトウェアパイプライニングで使用する命令スケジューリングアルゴリズムの選択基準を指示します。

SWP_POLICY(AUTO)

ループ毎に命令スケジューリングアルゴリズムを自動で選択します。

SWP_POLICY(SMALL)

小さなループ(例えば、必要レジスタ数が少ないループ)に適した命令スケジューリングアルゴリズムを使用します。

SWP_POLICY(LARGE)

大きなループ(例えば、必要レジスタ数が多いループ)に適した命令スケジューリングアルゴリズムを使用します。

80. !OCL SWP_WEAK

実行文の重なりを小さくしたソフトウェアパイプライニングの最適化を行うことを指示します。翻訳時に-O2オプション以上が有効な場合に意味があります。

以下に、指定の例を示します。

例:

```
!OCL SWP_WEAK
  DO I=1,N
    文
  ENDDO
```

実行文の重なりを小さくしたソフトウェアパイプライニングの最適化を行います。

81. !OCL NOSWP

ソフトウェアパイプライニングの最適化を抑止することを指示します。

以下に、指定の例を示します。

例:

```
!OCL NOSWP
  DO I=1,N
```

```
    文
  ENDDO
```

ソフトウェアパイプライニングの最適化を行いません。

82. !OCL UNROLL[(*n*)]

DO変数をもつDO構文の直前に記述し、ループアンローリングの展開数(多重度)を*n*で指定します。*n*は2から100までの整数値を指定することができます。*n*の値が省略された場合、コンパイラが自動的に最適な値を決定します。

また、ループの繰返し数が既知の場合は、*n*に繰返し数を超える値が指定されても、既知の繰返し数が優先されます。

なお、この最適化制御行は指定した直後のDOループのみが対象となります。

この最適化制御行が指定されていないループの展開数(多重度)は、ループの繰返し数、ループ内の実行文の数と種類および使用されているデータの型などから、コンパイラが最適と思われる値を決めます。

以下に、指定の例を示します。

例1:

```
SUBROUTINE SUB
...
!OCL UNROLL(8)
  DO I=1,N
    文
  ENDDO
...
END SUBROUTINE
```

文を8重に展開することを指定しています。

例2:

```
SUBROUTINE SUB
...
!OCL UNROLL(80)
  DO I=1,50
    文
  ENDDO
...
END SUBROUTINE
```

ループの繰返し数(50)を超える展開数(80)を指定しているので、展開数は50とみなされます。

83. !OCL UNROLL('full')

ループの展開数(多重度)に'full'を指定した場合、ソースプログラムの繰返し数だけ実行文を展開します。繰返し数が不明な場合は、最適化を行いません。

なお、この最適化制御行は指定した直後のDOループのみが対象となります。

以下に、指定の例を示します。

例1:

```
SUBROUTINE SUB
...
!OCL UNROLL('full')
  DO I=1,10
    文
  ENDDO
...
END SUBROUTINE
```

文をループの繰返し数と同じ10重に展開することを指定しています。

例2:

```
SUBROUTINE SUB
...
!OCL UNROLL('full')
  DO K=1,2
!OCL UNROLL('full')
    DO J=1,2
!OCL UNROLL('full')
      DO I=1,2
        文
      ENDDO
    ENDDO
  ENDDO
...
END SUBROUTINE
```

文を2*2*2重に展開することを指定しています。

84. !OCL NOUNROLL

ループアンローリングの最適化を抑止することを指示します。

なお、この最適化制御行は指定した直後のDOループのみが対象となります。

以下に、指定の例を示します。

例:

```
SUBROUTINE SUB
...
!OCL NOUNROLL
  DO I=1,N
    文
  ENDDO
...
END SUBROUTINE
```

ループアンローリングの最適化を行いません。

85. !OCL UNROLL_AND_JAM[(*n*)]

指定されたループに対して、アンロールアンドジャムを適用することを指示します。ただし、以下の場合には最適化を行いません。

ー 最適化の効果が期待できないと判断した場合

ー 回転を跨いだデータ依存があると判断した場合

*n*には、アンローリングの展開数(多重度)を2から100までの整数値で指定します。*n*を省略した場合、コンパイラが自動的に値を決定します。

なお、最内ループはアンロールアンドジャムの対象になりません。

本最適化指示子は-O2オプション以上が有効な場合に意味があります。

以下に、指定の例を示します。

例1:

```
!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
```

→

```
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
```

DO変数Jのループを対象とし、アンロールアンドジャムを適用します。

例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
```

指定したループが最内ループであるため、アンロールアンドジャムを適用しません。

例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
```

配列Aに回転を跨いだデータ依存があるため、アンロールアンドジャムを適用しません。

86. !OCL UNROLL_AND_JAM_FORCE[(*n*)]

回転を跨いだデータ依存はないとみなしてアンロールアンドジャムを適用することを指示します。

本最適化指示子を誤って指定した場合、実行結果は保証されません。詳細は、“9.18.3 UNROLL_AND_JAM_FORCE指示子の例”を参照して下さい。

*n*には、アンローリングの展開数(多重度)の上限を2から100までの整数値で指定します。*n*を省略した場合、コンパイラが自動的に値を決定します。

なお、本最適化指示子を指定した直後のループだけがアンロールアンドジャムの対象になります。また、最内ループはアンロールアンドジャム対象になりません。

本最適化指示子は-O2オプション以上が有効な場合に意味があります。

以下に、指定の例を示します。

例1:

```
!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
```

→

```
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
```

DO変数Jのループを対象とし、アンロールアンドジャムを適用します。

例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
```

指定したループが最内ループであるため、アンロールアンドジャムを適用しません。

87. !OCL NOUNROLL_AND_JAM

指定されたループに対して、アンロールアンドジャムを無効にすることを指示します。

以下に、指定の例を示します。

例:

```
!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
```

アンロールアンドジャムを適用しません。

88. !OCL UNSWITCHING

指定されたIF構文をループアンスイッチングすることを指示します。本最適化制御行はループ内で不変なIF構文の直前に指定してください。上記以外の箇所に記述した場合は指定が無効になります。

以下に指定の例を示します。

例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
```

IF構文をループアンスイッチングします。

なお、例2のように本最適化制御行が指定されたIF構文が含まれるループ内で、本最適化制御行が指定されていないIF構文はループアンスイッチングの対象となりません。

例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
```

最適化制御行が指定されたIF構文のみをループアンスイッチングの対象とします。

なお、ループアンスイッチングの対象となるループに多くの実行文が含まれる場合、翻訳メモリや翻訳時間が大幅に増加する場合があります。

89. !OCL ZFILL[(*n*)]

ループ内で書き込みのみを行う配列データについて、データをメモリからロードすることなく、キャッシュ上に書き込み用の領域を確保する命令を使い、書き込み動作を高速にする最適化を行うことを指示します。zfill指示子に続けて指定する1から100までの整数値*n*

によって、256バイトを1ブロックとする単位で*n*ブロック分先のデータをzfill最適化の対象とします。*n*の値が省略された場合、コンパイラが自動的に値を決定します。

zfillについては、“9.1.2.5 zfill”を参照してください。

以下に、指定の例を示します。

例1:

```
SUBROUTINE SUB(A,N)
...
!OCL ZFILL
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE
```

何ブロック先のデータをzfillの最適化の対象にするかはコンパイラが自動で判断します。

例2:

```
SUBROUTINE SUB(A,N)
...
!OCL ZFILL(1)
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE
```

1ブロック先のデータをzfillの最適化の対象にします。

90. !OCL NOZFILL

zfillの最適化を実施しないことを指示します。

以下に、指定の例を示します。

例:

```
SUBROUTINE SUB(A,N)
...
!OCL NOZFILL
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE
```

データはzfillの最適化の対象となりません。

9.11 スタック割付けの影響

翻訳時オプション-Kautoobjstack、-Kauto、-Kthreadsafe、または-Ktemparraystackが有効な場合、スタック割付けに次のような影響があります。各オプションの詳細は、“2.2 翻訳時オプション”を参照してください。

スタック領域の上限値がプログラムで使用するメモリ量より少なく設定している場合、領域が確保できなくなり、異常終了することがあります。その場合、スタック領域の上限値を大きくするか、または上記の翻訳時オプションを無効にする必要があります。

上限値を参照および変更する場合、cshではlimitコマンド、shではulimitコマンドを使用します。また、ジョブ運用ソフトウェア環境によりバッチリクエストとして実行する場合、対応するバッチキューのスタック領域の上限値(Per-process stack size limit)を大きくする必要があります。

スレッド並列プログラムの場合は、OMP_STACKSIZEまたはTHREAD_STACK_SIZEによって、必要なスレッドスタックのサイズを指定してください。詳細は、“J.2.2 実行の方法(自動並列化)”または“J.3.2 実行の方法(OpenMP仕様による並列化)”を参照してください。

OpenMPプログラムに翻訳時オプション-Knoautoを指定する場合は注意が必要です。翻訳時オプション-Knoautoの詳細は、“2.2.3 オプションの説明”を参照してください。

以下に各オプション指定による影響の例を示します。

例1: 翻訳時オプション-Kautoobjstackおよび-Kautoの影響を受ける場合

```
$ 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
  セグメント例外(コアダンプしました)
```

スタック領域が不足したため異常終了しました。

以下のどちらかで、異常終了を回避することができます。

- スタック領域の上限値を大きくして実行する。
- 翻訳時オプション-Knoautoと-Knoautoobjstackのどちらか、または両方を指定して翻訳する。

```
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ ./a.out
  1.000000000000000
```

スタック領域を大きくしたことにより正常終了するようになりました。

例2: 翻訳時オプション-Ktemparraystackおよび-Kautoの影響を受ける場合

```
$ 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
  セグメント例外 (コアダンプしました)
```

スタック領域が不足したため異常終了しました。

以下のどちらかで、異常終了を回避することができます。

- スタック領域の上限値を大きくして実行する。
- 翻訳時オプション-Knoautoと-Knotemparraystackのどちらか、または両方を指定して翻訳する。

```
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ ./a.out
  1.000000000000000
```

スタック領域を大きくしたことにより正常終了するようになりました。

9.12 配列の形状を変更する機能の影響

翻訳時オプション-Karraypad_const[=*N*]または-Karraypad_expr=*N*で変更された配列の上限の増加は、UBOUND関数とSIZE関数の結果にも反映されます。また、SUBCHK機能においても、増加後の上限がソースプログラムに記述されていたかのように動作します。

翻訳時オプション-Karraypad_const[=*N*]および-Karraypad_expr=*N*を使用する場合、実行可能プログラムを構成するすべてのプログラム単位に対して同一のオプションを指定して翻訳しなければなりません。

本機能は、個々の配列ごとに単独で適用され、EQUIVALENCE文やCOMMON文による記憶列結合や引数結合を考慮しません。そのため、本機能を使用することにより、正しくない結果が得られる場合もあるため注意が必要です。

以下に例を示します。

例1:

```
PROGRAM MAIN
REAL,DIMENSION(1024*1024)::A
CALL SUB(A,1024,1024)
  :
SUBROUTINE SUB(X,M,N)
REAL,DIMENSION(M,N)::X
  :
```

このようなプログラムを-Karraypad_expr=1オプションを指定して翻訳した場合、コンパイラは、以下のようなプログラムとみなして翻訳します。

```
PROGRAM MAIN
REAL,DIMENSION(1024*1024+1)::A
CALL SUB(A,1024,1024)
  :
SUBROUTINE SUB(X,M,N)
REAL,DIMENSION(M+1,N)::X
  :
```

本機能を使用しない場合、MAIN側のA(1025)は、SUB側のX(1,2)と結合しますが、使用するとA(1025)は、X(1025,1)と結合することになるので異なる結果が得られます。

例2:

```
INTEGER A(3,3),B(3,3)
A(:,1)=B(1,:)
A(:,2)=RESHAPE((/1,2,3/),(/3/))
```

このようなプログラムを-Karraypad_const=1を指定して翻訳した場合、それぞれの代入文の右辺と左辺の形状が異なってしまうため、これらの実行結果は保証されません。

9.13 プリフェッチを実施したことによる実行性能への影響について

翻訳時オプション-Kprefetch_sequential、-Kprefetch_stride、-Kprefetch_indirect、または-Kprefetch_conditionalが有効な場合のプリフェッチでは、ループのキャッシュ効率、分岐の有無または添字の複雑さによって、実行性能が低下することがあります。

9.14 配列宣言の次元移動

ここでは、配列宣言の次元移動について説明します。

9.14.1 配列宣言の次元移動の影響

翻訳時オプション-Karray_subscriptで配列の次元位置を移動します。移動することにより、組込み関数の引数キーワードにDIMおよびMASKが指定可能な組込み関数および形状をもつ組込み関数に影響があります。

翻訳時オプション-Karray_subscriptを使用する場合は、実行可能プログラムを構成するすべてのプログラム単位に対して同一のオプションを指定して翻訳しなければなりません。対象とする配列変数は、割付け配列および上下限が定数式な形状明示配列(局所的な配列、共通ブロック実体および仮引数)です。本機能は、宣言式内および宣言部分の定数式には適用されません。個々の配列ごとに単独で適用され、引数結合を考慮しません。そのため、本機能を使用することにより、正しくない結果が得られる場合もあるため注意が必要です。

以下に例を示します。

例1:

```
INTEGER,DIMENSION(101,102,103,5)::A
CALL SUB(A)
   :
PRINT *,A(1,2,3,4)
```

このようなプログラムを-Karray_subscriptオプションを指定して翻訳した場合、コンパイラは、以下のようなプログラムとみなして翻訳します。

```
INTEGER,DIMENSION(5,101,102,103)::A
CALL SUB(A)
   :
PRINT *,A(4,1,2,3)
```

例2:

```
PROGRAM MAIN
INTEGER,DIMENSION(100,101,102,3) :: A
   :
CALL SUB(A)
END PROGRAM
SUBROUTINE SUB(B)
INTEGER,DIMENSION(100) :: B
   :
END SUBROUTINE
```

本機能を使用した場合、MAIN側の手続SUBの引数は、A(3,100,101,102)の配列とSUB側の引数B(100)と結合しますが、使用すると配列の次元数が異なるため、結果が正しく得られないことがあります。

例3:

```
INTEGER,DIMENSION(1,2,3,4) :: A
   :
A=RESHAPE((/(i,i=1,24)/),(/1,2,3,4/))
      ! 代入の右辺と左辺の形状が異なり、実行結果は保証されません。
PRINT *,UBOUND(A,4)
      ! 3が出力される。
```

9.14.2 配列宣言の次元移動の対象変数の制約事項

以下のいずれかの条件に該当する配列変数は、配列宣言の次元移動の対象になりません。

- 派生型、文字型、または多相的
- 名前付き定数
- SAVE属性(モジュール宣言部の暗黙のSAVE属性は除く)またはTARGET属性をもつ
- ポインタ変数
- 初期値をもつ
- 結合実体
- 変数群要素
- 自動割付け配列
- モジュール内の宣言部にPUBLIC属性で宣言している

- アドレス保持変数
- 関数結果
- BLOCK内で宣言している

9.14.3 サブモジュールの祖先モジュールの制限

翻訳時オプション-Karray_subscriptを有効にして翻訳した、または最適化制御行により配列の次元移動を指示したモジュールは、サブモジュールの祖先モジュールに指定できません。

9.15 配列変数マージ

ここでは、配列宣言のマージについて説明します。

9.15.1 配列変数マージの影響

翻訳時オプション-Karray_mergeで複数の配列を1つの配列にマージします。

対象とする配列変数は、上下限が定数式な形状明示配列および自動割付け配列(注)です。本機能は、宣言式内および宣言部分の定数式には適用されません。コンパイラが自動的に1次元目の要素を追加し、要素をマージします。マージの順番は不定です。配列名の引用をマージした配列に置き換えるため翻訳エラー、もしくは意図した結果にならないことがあるため注意が必要です。個々の配列ごとに単独で適用され、宣言の下限、引数結合を考慮しません。そのため、本機能を使用することにより、正しくない結果が得られる場合があります。

注)配列宣言のすべての次元の宣言式が次のいずれかの場合は、制約事項ではありません。

- 宣言式は、1つの変数だけである、または、
- 宣言式は、定数式である。

以下に例を示します。

例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)
```

このようなプログラムを-Karray_mergeオプションを指定して翻訳した場合、コンパイラは、以下のようなプログラムとみなして翻訳します。

```
INTEGER,DIMENSION(3,101,102,103)::TMP
       ! Aを1次元目の1要素にマージ:TMP(1,101,102,103)
       ! Bを1次元目の2要素にマージ:TMP(2,101,102,103)
       ! Cを1次元目の3要素にマージ:TMP(3,101,102,103)
CALL SUB(TMP(1,:,:,:)) ! 配列名 Aをマージした配列名 TMPに変更
   :
PRINT *,TMP(2,101,102,103)
TMP(1,:,:,:) = TMP(3,:,:,:)
TMP(1,101,102,103)=TMP(3,1,2,3)
```

注)TMPは、コンパイラで生成した変数です。

例2:

```
PROGRAM MAIN
INTEGER,DIMENSION(100,101,102) :: A,B
   :
CALL SUB(A(1,1,1))
END PROGRAM
SUBROUTINE SUB(C)
INTEGER,DIMENSION(100) :: C
```

```
   :
END SUBROUTINE
```

本機能を使用した場合、MAIN側の手続SUBの引数は、Aが1次元目に要素を追加した生成変数に変更するため、SUB側の引数C(100)との結合が正しく得られないことがあります。

翻訳時オプション-Karray_merge_commonで名前付き共通ブロック実体の配列をマージします。

対象とする配列変数は、名前付き共通ブロック中にある上下限が定数式な形状明示配列です。この場合に共通ブロック名を作成します。_0_+共通ブロック名を作成します。

同じ名前付き共通ブロック中にある共通ブロック実体は、すべて同じ形状の配列変数でなければなりません。本機能は、宣言式内および宣言部分の定数式には適用されません。コンパイラが自動的に1次元目の要素を追加し、要素をマージします。マージの順番は不定です。配列名の引用をマージした配列に置き換えるため翻訳時エラー、または意図した結果にならないことがあるため注意が必要です。個々の配列ごとに単独で適用され、引数結合を考慮しません。そのため、本機能を使用することにより、正しくない結果が得られる場合があります。

以下に例を示します。

例1:

```
INTEGER,DIMENSION(101,102,103)::A,B,C
COMMON /COM/ A,B,C
CALL SUB(A)
   :
PRINT *,B(101,102,103)
```

このようなプログラムを-Karray_merge_commonオプションを指定して翻訳した場合、コンパイラは、以下のようなプログラムとみなして翻訳します。

```
INTEGER,DIMENSION(3,101,102,103)::TMP
COMMON /_0_COM/ TMP
CALL SUB(TMP(1,:,:,:)) ! 配列名Aをマージした配列名TMPに変更
   :
PRINT *,TMP(2,101,102,103)
```

注)TMPは、コンパイラで生成した変数です。

9.15.2 局所的な配列変数マージの対象変数の制約事項

以下のいずれかの条件に該当する配列変数は、局所的な配列変数マージの対象になりません。

- 派生型または文字型
- 名前付き定数
- SAVE属性またはTARGET属性をもつ
- ポインタ変数
- 初期値をもつ
- 結合実体
- 変数群要素
- モジュールまたはサブモジュール内で宣言
- アドレス保持変数
- 関数結果
- 共通ブロック実体
- 仮引数
- THREADPRIVATE指示文に指定されている

・ 自動割付け配列

配列宣言のすべての次元の宣言式が次のいずれかの場合は、制約事項ではありません。

－ 宣言式は、1つの変数だけである、または、

－ 宣言式は、定数式である。

・ BLOCK内で宣言している

9.15.3 共通ブロック実体の配列変数マージの対象変数の制約事項

以下のいずれかの条件に該当する配列変数は、共通ブロック実体の配列変数マージの対象になりません。

・ 派生型または文字型

・ TARGET属性をもつ

・ 初期値をもつ

・ 結合実体

・ 変数群要素

・ OpenMP仕様の指示文を含むプログラム単位内にある

9.16 マルチ演算関数

9.16.1 マルチ演算関数の呼出しについて

マルチ演算関数とは、1回の呼出しで複数の引数に対する同種の組込み関数計算および演算を行うことにより、実行性能を向上させた関数です。本処理系では、翻訳時オプション-Kmfuncまたは最適化制御行MFUNCを指定することにより、コンパイラがループを解析し、可能ならばマルチ演算関数に置き換えます。

複雑なループなどコンパイラが解析できない場合、ユーザプログラムからマルチ演算関数を直接CALLして利用することができます。

以下の注意が必要です。

・ サイズを指定する引数Nは、8バイトの整数型にしてください。

・ マルチ演算関数の外部手続名は、C言語の外部手続名になっています。このため、Fortranプログラムから呼び出す場合は、$pragma指定子で、外部手続名の加工方法を変更してください。

・ マルチ演算関数では高速化のために、引数チェックを省略する場合があります。このため、IEEE 754で規定されている特別な値(NaN、Infなど)が入力された場合は、異常終了することがあります。

表9.2にユーザプログラムから直接呼び出すことができるマルチ演算関数を示します。

表9.2 ユーザプログラムから直接呼び出すことができるマルチ演算関数一覧

関数／演算	引数の型	呼出しの形式	計算内容
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))

関数／演算	引数の型	呼出しの形式	計算内容
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))
べき乗演算	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

注意. マルチ演算関数をユーザプログラムから直接呼び出す場合、演算結果の返却に使う引数とその他の引数の記憶領域は異なる記憶領域としてください。記憶領域が重なると、結果が保証されないことがありますので注意してください。

以下にマルチ演算関数の呼出し例を示します。

例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
```

マルチ演算関数の呼出し

```
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)
```

例2:

IF構文の中で関数が呼ばれている場合、計算すべき要素だけを配列に格納することにより、マルチ演算関数を使用することができます。

```
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
```

マルチ演算関数の呼出し

```
INTEGER,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 翻訳時オプション-Kmfunc=3の影響

翻訳オプション-Kmfunc=3が指定された場合、IF構文を含むループのマルチ演算関数化の影響により、実行時異常終了となる場合があります。

以下に、-Kmfunc=3による影響の例を示します。

例: IF構文を含むループの例

```
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
```

[説明]

上記の例では、条件I.LE.1000を満たす場合に配列要素を実引数とする組込み関数COSを呼び出しているため、配列Bの添字の値は宣言された上下限の範囲を超えることはありません。

マルチ演算関数化により、IF構文の条件に関係なく、DO構文の繰返し数分の配列要素をマルチ演算関数化された組込み関数の引数として使用するため、配列Bは宣言範囲を超えて引用されます。そのため、記憶保護例外(読み出し)が発生した旨のエラーメッセージが出力され、プログラムの実行が中断することがあります。

対処方法としては、翻訳時オプション-Kmfunc=3を指定しないようにしてください。

9.17 セクタキャッシュのソフトウェア制御

9.17.1 セクタキャッシュの利用について

セクタキャッシュは、再利用性のあるデータが再利用性の無いデータによってキャッシュから追い出されることを防ぐことができるキャッシュ機構です。特に、スレッド並列時に複数コアが2次キャッシュをアクセスする場合、キャッシュから追い出される現象が起こりやすくなります。セクタキャッシュは、再利用性のあるデータと再利用性の無いデータをセクタ毎に分けて配置することができます。セクタキャッシュをソフトウェア制御する手段としては、環境変数または最適化制御行があります。詳細については、以降で説明しますが、環境変数では各セクタの最大ウェイ数が指定可能で、最適化指示行ではセクタ1の最大ウェイ数のみが指定可能です。

ただし、セクタキャッシュは、追い出しを防ぎたい配列のサイズを考慮せずに最大ウェイ数を指定しても性能は向上しません。キャッシュの利用効率が低下し、かえって遅くなってしまうことが考えられます。また、セクタキャッシュのソフトウェア制御をしなくても、LRU(Least Recently Used)によるキャッシュ制御は動作するため、指定しても必ず高速化できるわけではありません。セクタキャッシュで性能向上するための1つの有効な手段としては、ハードウェアモニタ情報を採取後、2次キャッシュミスが発生していることを確認した上で、再利用性のある配列サイズを把握し、その配列が載るだけのウェイ数をセクタ1のウェイ数に指定する方法があります。なお、セクタキャッシュを利用する際は、翻訳時オプション-Khpctagが有効でなければなりません。

参考

最大ウェイ数

A64FXプロセッサの1次キャッシュの最大ウェイ数は4、2次キャッシュの最大ウェイ数は16です。

なお、2次キャッシュは、アシスタントコアが常時2ウェイを使用します。

9.17.2 セクタキャッシュをソフトウェア制御する方法

本処理系には、セクタキャッシュをソフトウェア制御する手段として、以下の2つの方法があります。

- 最適化制御行によるソフトウェア制御
- 環境変数および最適化制御行によるソフトウェア制御

9.17.2.1 最適化制御行によるソフトウェア制御

!OCL SCACHE_ISOLATE_WAY(L2=*n1*[,L1=*n2*])
および
!OCL END_SCACHE_ISOLATE_WAY

SCACHE_ISOLATE_WAY指示子は、キャッシュのセクタ1の最大ウェイ数を指示します。

SCACHE_ISOLATE_WAY指示子の引数には、キャッシュのセクタ1の最大ウェイ数を指示します。L2=*n1*で2次キャッシュにおけるセクタ1の最大ウェイ数を、L1=*n2*で1次キャッシュにおけるセクタ1の最大ウェイ数をそれぞれ指示します。L1=*n2*の指定は省略可能です。その場合、2次キャッシュのセクタ1の最大ウェイ数のみを制御します。

2次キャッシュはアシスタントコアが常時2ウェイを使用します。それを踏まえて、*n1*および*n2*に指定できる範囲は以下の通りとなります。

－ 0 ≦ *n1* ≦ “2次キャッシュの最大ウェイ数 - 2”

－ 0 ≦ *n2* ≦ “1次キャッシュの最大ウェイ数”

プログラム単位に指示する場合は、プログラム単位の記述位置にSCACHE_ISOLATE_WAY指示子を記述します。指示の有効範囲はプログラム内全体となります。

プログラムの一部に対して指示する場合は、指示する範囲をSCACHE_ISOLATE_WAY指示子とEND_SCACHE_ISOLATE_WAY指示子で指定します。

なお、上記の範囲指定の指示子を入れ子に記述することはできません。

ただし、プログラム単位の指示があるプログラムの一部に範囲指定の指示を記述することは可能です。

!OCL SCACHE_ISOLATE_ASSIGN(*array1*[,*array2*...])
および
!OCL END_SCACHE_ISOLATE_ASSIGN

SCACHE_ISOLATE_ASSIGN指示子は、キャッシュのセクタ1に載せる配列を指示します。

ただし、数値型または論理型の配列を指定した場合のみ有効となります。

プログラム単位に指示する場合は、プログラム単位の記述位置にSCACHE_ISOLATE_ASSIGN指示子を記述します。指示の有効範囲はプログラム内全体となります。

プログラムの一部に対して指示する場合は、指示する範囲をSCACHE_ISOLATE_ASSIGN指示子とEND_SCACHE_ISOLATE_ASSIGN指示子で指定します。

なお、上記の範囲指定の指示を入れ子に記述することはできません。

ただし、プログラム単位の指示があるプログラムの一部に範囲指定の指示を記述することは可能です。

例

最適化制御行によるセクタキャッシュのソフトウェア制御

```
SUBROUTINE OCL_SECTOR(A,B,N,N2)
REAL(KIND=8),DIMENSION(N) :: A
REAL(KIND=8),DIMENSION(N,N2) :: B
! 10ウェイに収まるサイズの配列AをL2キャッシュで再利用
!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 環境変数および最適化制御行によるソフトウェア制御

各セクタの最大ウェイ数の初期値は、以下の環境変数でも指定することができます。環境変数を指定することにより、オブジェクトプログラムの起動時の最大ウェイ数を指定することができます。

FLIB_SCCR_CNTL

セクタキャッシュを利用することを指定する環境変数です。環境変数に設定できる値とその意味は以下のとおりです。

値	説明
TRUE	セクタキャッシュを利用します。 デフォルトです。
FALSE	セクタキャッシュを利用しません。

FLIB_L1_SCCR_CNTL

NUMAノードを1プロセスで占有していない環境でプログラムを実行する場合、2次キャッシュでセクタキャッシュは利用できません。2次キャッシュでセクタキャッシュを利用できない時に、1次キャッシュでセクタキャッシュを利用するかどうかを指定する環境変数です。FLIB_SCCR_CNTLがTRUEの時のみ有効です。

環境変数に指定できる値とその意味は以下のとおりです。

値	説明
TRUE	2次キャッシュでセクタキャッシュを利用できない場合、1次キャッシュでセクタキャッシュを利用します。 デフォルトです。
FALSE	2次キャッシュでセクタキャッシュを利用できない場合、1次キャッシュでセクタキャッシュを利用しません。 以下の実行時メッセージを出力し、セクタキャッシュの制御を無効化して、プログラムの実行を継続します。 `jwe1047i-w A sector cache couldn't be used.`

FLIB_L2_SECTOR_NWAYS_INIT

2次キャッシュのセクタの最大ウェイ数の初期値を指定する環境変数です。FLIB_SCCR_CNTLがTRUEの時のみ有効です。

1つ目のセクタ(セクタ0)の最大ウェイ数*n0*と2つ目のセクタ(セクタ1)の最大ウェイ数*n1*を以下の形式で指定します。

```
n0,n1
```

*n0*と*n1*に指定できる範囲は以下のとおりです。

- 0 <= *n0* <= “2次キャッシュの最大ウェイ数 - 2”
- 0 <= *n1* <= “2次キャッシュの最大ウェイ数 - 2”

最大ウェイ数からアシスタントコアが使用する2ウェイ分を引いた値を指定可能です。

また、競合を防ぐため

- *n0* + *n1* = “2次キャッシュの最大ウェイ数 - 2”

を満たすことを推奨します。

例

環境変数および最適化制御行によるセクタキャッシュのソフトウェア制御

以下の例の場合、オブジェクトプログラムの起動前に、環境変数FLIB_L2_SECTOR_NWAYS_INITの値として、2,10を設定することで、セクタ1の最大ウェイ数として10を採用し、配列Aをセクタ1に載せるソフトウェア制御が可能となります。

1. オブジェクトプログラムの起動前に、環境変数を設定(bashの場合)

```
FLIB_SCCR_CNTL=TRUE
export FLIB_SCCR_CNTL
FLIB_L2_SECTOR_NWAYS_INIT=2,10
export FLIB_L2_SECTOR_NWAYS_INIT
```

2. セクタ1に載せる配列を、最適化制御行により指定

```
SUBROUTINE OCL_SECTOR(A,B,N,N2)
REAL(KIND=8),DIMENSION(N) :: A
REAL(KIND=8),DIMENSION(N,N2) :: B
! 配列Aにセクタ1を割り当て
!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_WAY
END SUBROUTINE
```

9.17.2.3 例外的な指定に対する動作について

最適化指示子SCACHE_ISOLATE_WAYと環境変数FLIB_L2_SECTOR_NWAYS_INITに上限を超えた値を指定した場合、上限値を指定したものとみなされます。0より小さい値を設定した場合、その最適化制御行を無視します。なお、2バイト符号付き整数型の範囲外の値を指定した場合、動作は保証されません。

9.18 最適化指示子の誤った指定

最適化制御行を誤って指定すると、実行結果が正しく得られない場合があります。

9.18.1 NOVREC指示子の例

配列Aが回帰的データでないとわかっている場合、NOVREC指示子を有効にすれば、SIMD命令を利用します。しかし、配列Aが回帰的データの場合、誤ってNOVREC指示子を有効にすると、SIMD命令を利用しますが実行結果は保証されません。

例1:

```
!OCL NOVREC
  DO I=1,100
    A(I)=A(I+M)+ ...
  END DO
```

9.18.2 CLONE指示子の例

CLONE最適化は、生成された条件節内のループにおいて、指定した変数の値が不変であるものとした最適化を行います。そのため、対象となるループ内で変数が更新される場合、実行結果は保証されません。

例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
```

ループ内で変数が更新される場合、結果が保証されないため指定しないでください。

例2:

```
  INTEGER::N
!OCL CLONE(N==10)
  DO I=1,32
    CALL SUB(N)
  ENDDO
```

ループ内で変数が更新される可能性がある場合、結果が保証されないため指定しないでください。

9.18.3 UNROLL_AND_JAM_FORCE指示子の例

本最適化指示子は、回転を跨いだデータ依存はないとみなしてアンロールアンドジャムを適用します。本最適化指示子を誤って指定した場合、実行結果は保証されません。

例:

```
!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
```

配列Aに回転を跨いだデータ依存があるので、本最適化指示子を指定しないでください。

9.18.4 ITERATIONS指示子の例

ITERATIONS(max=*n1*)指示子およびITERATIONS(min=*n3*)指示子は、翻訳時にループの繰返し数が不明な場合に有効であり、ループの繰返し数の最大値を*n1*、最小値を*n3*とみなして最適化します。*n1*、*n3*を誤って指定した場合、実行結果は保証されません。

例:

```
!OCL ITERATIONS(max=100,min=1)
DO I=1,M  ! 実際のループの繰返し数は最大値が1000、もしくは最小値が0
    A(I) = A(I) + B(I)
END DO
```

実際のループの繰返し数が最大値100を超える、もしくは最小値が0となる場合には、実行結果は保証されません。

9.19 言語間結合における注意事項

以下に本処理系での言語間結合をする場合の注意事項を示します。

9.19.1 C言語で受け取った引数のアドレスについて

C言語で受け取った引数のアドレスを関数内で保存した場合、本処理系の仕様に違反した使い方のため、最適化により、利用者の意図した結果が得られないことがあります。(例1を参照)

このような場合、引数のアドレスを関数内で保存せず、明示的に引数で受け渡すことより、正しい結果が得られるようになります。(例2を参照)

または、例1のFortranプログラムの呼び出し側の引数を“%VAL(LOC(引数))”と記述し、引数のアドレスを渡すことを明示的にしてください。(例3を参照)

例1: 誤ったポインタの保存

```
int fun_(int *flag, int *p)
{
   static int *save;
   if (*flag == 0) {
      save = p;
      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
```

例2: ポインタを保存しない書き方

```
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
```

例3: 明示的なアドレス渡しの書き方

```
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 C言語で受け取ったポインタについて

C言語で受け取ったポインタのアドレスを関数結果またはCOMMONを通じて返した場合、本処理系の仕様に違反した使い方のため、最適化により、利用者の意図した結果が得られないことがあります。(例4を参照)

このような場合、アドレス保持変数にLOC組込み関数を使用してアドレスを設定することにより、利用者の意図した結果が得られるようになります。(例5を参照)

例4: 誤ったポインタの返却

```
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
```

例5: 正しいポインタの使い方

```
INTEGER(KIND=4) I,IP
POINTER (PTR,IP)
I = 2
PTR = LOC(I)
IP = 1
PRINT *, I
END
```

9.20 浮動小数点演算に対する最適化の副作用

最適化オプションや最適化指示子で浮動小数点演算を最適化した場合、副作用を生じる可能性があります。ここでは、その副作用(主に計算誤差)について説明します。

本処理系は、基本的にはIEEE 754に準拠したオブジェクトを作成しますが、“表9.3 浮動小数点演算に対する最適化の副作用”の計算誤差を伴う最適化により、数値演算の取り扱い方法がIEEE 754に準拠しなくなる場合があります。

各オプションの詳細は“2.2 翻訳時オプション”を、各最適化指示子の詳細は“9.10.4 最適化指示子”および“12.2.6.4 自動並列化用の最適化指示子”を参照してください。

表9.3 浮動小数点演算に対する最適化の副作用

最適化オプション	最適化指示子	副作用の内容
-Keval	EVAL	演算の評価順序を変更する最適化を行うので、計算誤差が生じることがあります。  例 `x*y + x*z → x*(y+z)` -Kevalオプションの指定に伴い、下記オプションが有効になります。それぞれの副作用も参照してください。 ・ -Kfsimpleオプション ・ -Kreductionオプション (-Kparallelオプション有効時) ・ -Ksimd_reduction_productオプション (-Ksimd[={ 1 | 2 | auto }]オプション有効時)
-Kfsimple	—	浮動小数点演算の単純化の最適化を行うので、数値演算の取り扱い方法が、下例のように、厳密にはIEEE 754に準拠しなくなります。  例 `x*0.0 → 0.0` この計算において、xがNaNである場合でも、最適化はx*0.0を0.0に置き換えます。
-Kfp_contract	FP_CONTRACT	Floating-Point Multiply-Add/Subtract演算命令を利用する最適化を行うので、丸め誤差程度の計算誤差が生じることがあります。
-Kfp_relaxed	FP_RELAXED	単精度または倍精度の浮動小数点除算またはSQRT関数に対し、逆数近似演算命令を利用する最適化を行うので、以下のような副作用が生じる可能性があります。 ・ 実行結果に丸め誤差程度の違いが生じる。 ・ -Kfzオプションの指定に関係なく、引数および戻り値に現れる非正規化数を0で置き換える。 ・ 引数および戻り値に現れる負の0を正の0で置き換える。 ・ 引数または戻り値がNaN、±Inf、正規化数の最大値に近い値、正規化数の最小値に近い値のいずれかの場合、IEEE 754に準拠しない動作となる。
-Kfz	—	flush-to-zeroモードを使用するため、アンダーフローによって非正規化数となる演算結果やソースオペランドが同符号の0.0に変わります。
-Kilfunc	—	組込み関数および演算のインライン展開で、逆数近似演算命令や三角関数補助命令などを利用したアルゴリズムを用いるため、-Kfp_relaxedと同様の副作用が生じる可能性があります。さらに、以下のような副作用が生じる可能性があります。

最適化オプション	最適化指示子	副作用の内容
		• 組込み関数の引数が“8.1.5 組込み関数エラーの処理”のエラー発生の原因に該当する場合にも、エラーメッセージは出力されずエラー処理も行なわれない。 • -Knofp_contractオプションおよび-Knofp_relaxedオプションの指定に関係なく、逆数近似演算命令およびFloating-Point Multiply-Add/Subtract命令を使用する。 また、atan2は-X03または-X08を指定した場合にもFortran95の言語仕様で計算されます。
-Kmfunc[={ 1 | 2 | 3 }]	MFUNC[(*level*)]	組込み関数および演算のマルチ演算関数への変換において、異なったアルゴリズムや逆数近似演算命令、三角関数補助命令などを利用するため、-Kfp_relaxed や-Kilfuncと同様の副作用が生じる可能性があります。 また、atan2は-X03または-X08を指定した場合にもFortran95の言語仕様で計算されます。 その他の副作用については、“9.16 マルチ演算関数”も参照してください。
-Kreduction	REDUCTION	自動並列化のリダクション最適化を行うため、演算の評価順序が変更されることで、計算誤差が生じることがあります。
-Ksimd_reduction_product	－	乗算のリダクション演算をSIMD化するため、演算の評価順序が変更されることで、計算誤差が生じることがあります。
-Kfast	－	-Kevalオプションおよび-Kilfuncオプションなどを誘導するため、計算誤差が生じることがあります。
-Kvisimpact	－	-Kfastオプションを誘導するため、計算誤差が生じることがあります。

また、本処理系は、-O1オプション以上が有効な場合に、プログラムの論理上は実行されないはずの浮動小数点演算命令を実行する最適化を行い、プログラムの論理上は発生しない浮動小数点例外を発生させることがあります。その場合では、例えば以下の現象が発生することがあります。

- 組込みモジュールIEEE_EXCEPTIONSを用いて浮動小数点状態フラグにアクセスしたときに、プログラムの論理上はセットされないフラグがセットされている。

ただし、翻訳時オプション-NRtrapを指定することで回避できる場合があります。

参考

自動並列/OpenMPによるスレッド並列時に浮動小数点演算に対する計算誤差を抑えたい場合

スレッド数の変化による計算誤差が発生しない実行可能プログラムを作成することを指示する、以下の翻訳時オプションを指定してください。ただし、実行性能が低下する可能性があります。

- -Kparallel_fp_precisionオプション (-Kparallelオプションまたは-Kopenmpオプション有効時)
- -Kopenmp_ordered_reductionオプション(-Kopenmpオプション有効時)

各オプションの詳細は、“2.2 翻訳時オプション”を参照してください。

9.21 SVEのベクトルレジスタのサイズを指定する場合の注意事項

翻訳時オプション-Ksimd_reg_size={ 128 | 256 | 512 }は、SVEのベクトルレジスタのサイズをビット単位で指定します。コンパイラは、指定された値がベクトルレジスタのサイズであるとみなして最適化を実施するため、生成された実行可能プログラムは、指定されたサイズのベクトルレジスタのSVEを実装しているCPUアーキテクチャ上においてだけ正常に動作します。

翻訳時オプション-Ksimd_reg_size={ 128 | 256 | 512 }で指定したベクトルレジスタのサイズと、実行するCPUアーキテクチャのベクトルレジスタのサイズが異なる場合、実行時異常終了する可能性があります。また、実行結果は保証されません。

なお、翻訳時オプション-Ksimd_reg_size={ 128 | 256 | 512 }で指定したベクトルレジスタのサイズが、実行するCPUアーキテクチャのベクトルレジスタのサイズより小さいことが明らかな場合、prctl(2)システムコールなどを利用して有効なベクトルレジスタのサイズを設定する必要があります。

翻訳時オプション-Ksimd_reg_size=agnosticが有効な場合、実行可能プログラムは、CPUのSVEのベクトルレジスタのサイズに関わらず実行可能です。

SVEのベクトルレジスタのサイズを変更する場合は、“A.2.2 SVEのベクトルレジスタのサイズの変更について”も参照してください。

第10章　モジュール、サブモジュール、およびモジュール引用の注意事項

モジュール名を指定したUSE文、モジュール名を祖先モジュール名として指定したSUBMODULE文、サブモジュール名を親サブモジュール名として指定したSUBMODULE文を、モジュール引用といいます。

ここでは、モジュール、サブモジュール、およびモジュール引用が存在する入力ファイルを翻訳するための注意事項、組込みモジュールの注意事項について説明します。

10.1 モジュールおよびモジュール引用の翻訳方法

モジュールおよびモジュール引用の翻訳は、以下のように行われます。

a. モジュールに対するオブジェクトは、カレントディレクトリにファイル名.oで作成されます。

b. モジュールに対する翻訳中間情報ファイルは、モジュール名.modで以下のディレクトリに作成されます。

   ー 翻訳時オプション-M*directory*で指定したディレクトリ

   ー 翻訳時オプション-M*directory*を省略した場合は、カレントディレクトリ

c. モジュールに対する翻訳中間情報ファイルは、以下の順で検索されます。

   ー 翻訳時オプション-M*directory*で指定したディレクトリ

   ー カレントディレクトリ

   ー 翻訳時オプション-I*directory*で指定したディレクトリ

リンク時にモジュールに対するオブジェクトファイルを明示的に指定する必要があります。

例1：モジュールとモジュール引用

ファイル名：a.f90

```
MODULE MOD
INTEGER::VALUE=1
END MODULE
PROGRAM MAIN
USE MOD
PRINT*,VALUE
END PROGRAM
```

翻訳コマンド

```
$ frtpx   a.f90
```

例2：異なるファイルにモジュールとモジュール引用が記述された場合1

ファイル名：a.f90

```
MODULE MOD
INTEGER :: VALUE = 1
END MODULE
```

ファイル名：b.f90

```
PROGRAM MAIN
USE MOD
PRINT *,VALUE
END PROGRAM
```

翻訳コマンド

```
$ frtpx a.f90 -c
$ frtpx b.f90 a.o
```

または

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx a.o b.o
```

例3:異なるファイルにモジュールとモジュール引用が記述された場合2

ファイル名：a.f90

```
MODULE MOD
INTEGER :: VALUE = 1
END MODULE
```

ファイル名：b.f90

```
PROGRAM MAIN
USE MOD
VALUE = VALUE + 1
CALL EXTERNAL_SUB
END PROGRAM
```

ファイル名：c.f90

```
SUBROUTINE EXTERNAL_SUB
USE MOD
PRINT *,VALUE
END SUBROUTINE
```

翻訳コマンド

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx c.f90 a.o b.o
```

または

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx c.f90 -c
$ frtpx a.o b.o c.o
```

例4:異なるファイルにモジュールとモジュール引用が記述された場合3

ファイル名：a.f90

```
MODULE MOD
INTEGER :: VALUE = 1
END MODULE
```

ファイル名：b.f90

```
PROGRAM MAIN
USE MOD
VALUE = VALUE + 1
CALL EXTERNAL_SUB
END PROGRAM
```

ファイル名：c.f90

```
SUBROUTINE EXTERNAL_SUB
USE MOD
```

```
PRINT *,VALUE
END SUBROUTINE
```

翻訳コマンド

```
$ frtpx a.f90 -c
```

a.o およびmod.mod が生成されます。

翻訳コマンド

```
$ frtpx a.f90 b.f90 c.f90
```

a.o、b.o、c.o、mod.mod およびa.out が生成されます。

例5:翻訳時オプション-Iおよび-Mの指定

ファイル名 : a.f90

```
MODULE MOD1
INTEGER :: VALUE = 1
END MODULE
```

ファイル名 : b.f90

```
MODULE MOD2
USE MOD1
INTEGER :: VALUE2 = 2
CONTAINS
SUBROUTINE MOD2_SUB
VALUE = VALUE2 + 1
END SUBROUTINE
END MODULE
```

翻訳コマンド

```
$ frtpx a.f90 -c
```

モジュールmod1のためのモジュール情報ファイルmod1.modは、カレントディレクトリに生成されます。

```
$ frtpx a.f90 -c -Mdirectory
```

モジュールmod1のためのモジュール情報ファイルmod1.modは、翻訳時オプション-Mで指定したディレクトリに生成されます。

```
$ frtpx b.f90 -c -Idirectory
```

1. モジュールmod1のためのモジュール情報ファイルmod1.modは、翻訳時オプション-Iで指定したディレクトリおよびカレントディレクトリが検索されます。

2. モジュールmod2のためのモジュール情報ファイルmod2.modは、カレントディレクトリに生成されます。

```
$ frtpx b.f90  -c  -Iinput_directory  -Moutput_directory
```

1. モジュールmod1のためのモジュール情報ファイルmod1.modは、翻訳時オプション-Mおよび-Iで指定したディレクトリおよびカレントディレクトリが検索されます。

2. モジュールmod2のためのモジュール情報ファイルmod2.modは、翻訳時オプション-Mで指定したディレクトリに生成されます。

10.2 モジュール情報の更新とモジュール引用プログラムの再翻訳

以下のどちらかの場合、モジュール引用を記述しているプログラム単位も再翻訳しなければなりません。

- モジュールの宣言部の PUBLIC属性をもつ要素を更新した場合
- PUBLIC属性をもつモジュール副プログラムの手続の特性を変更した場合

モジュールまたはサブモジュールの宣言部の要素を更新した場合、そのモジュールまたはサブモジュールの子孫サブモジュールはすべて再翻訳しなければなりません。

10.3 モジュールと制限事項

モジュール、サブモジュール、およびモジュール引用において、以下の翻訳時オプションは、同じでなければなりません。

- -AAオプション
- -AUオプション
- -Kcommonpadオプション
- -Karray_subscriptオプション

モジュールを-X9、-X03または-X08オプションで翻訳した場合、モジュール引用も-X9、-X03または-X08オプションで翻訳しなければなりません。

モジュールを-X6オプションで翻訳した場合、モジュール引用も-X6オプションで翻訳しなければなりません。

モジュールを-X7オプションで翻訳した場合、モジュール引用も-X7オプションで翻訳しなければなりません。

-Karray_subscriptオプションを有効にして翻訳した、または最適化制御行(“9.10 最適化制御行(OCL)の利用”)により配列の次元移動を指示したモジュールは、サブモジュールの祖先モジュールに指定できません。

10.4 サブモジュール

サブモジュールの翻訳は、以下のように行われます。

a. サブモジュールに対するオブジェクトは、カレントディレクトリにファイル名.oで作成されます。

b. サブモジュールに対する翻訳中間情報ファイルは、祖先モジュール名.サブモジュール名.smodで以下のディレクトリに作成されます。

   ー 翻訳時オプション-M*directory*で指定したディレクトリ

   ー 翻訳時オプション-M*directory*を省略した場合は、カレントディレクトリ

c. サブモジュールに対する翻訳中間情報ファイルは、以下の順で検索されます。

   ー 翻訳時オプション-M*directory*で指定したディレクトリ

   ー カレントディレクトリ

   ー 翻訳時オプション-I*directory*で指定したディレクトリ

結合編集時にサブモジュールに対するオブジェクトファイルを明示的に指定する必要があります。

例1:サブモジュールとモジュール引用

ファイル名：a.f90

```
module m
  interface
    module subroutine sub
    end subroutine
  end interface
end

submodule(m)smod
contains
  module subroutine sub
    print *, 'pass'
  end subroutine
end

use m
```

```
call sub
end
```

翻訳コマンド

```
$ frtpx a.f90
```

例2:モジュール、サブモジュール、その子サブモジュール、引用を異なるファイルに記述した場合

ファイル名：module.f90

```
module m
  interface
    module subroutine sub
    end subroutine
  end interface
end
```

ファイル名：parent_submodule.f90

```
submodule(m)parent_smod
  Interface
    module subroutine sub2
    end subroutine
  end interface
contains
  module subroutine sub
    call sub2
  end subroutine
end
```

ファイル名：child_submodule.f90

```
submodule(m:parent_smod)child_smod
contains
  module subroutine sub2
    print *,'pass'
  end subroutine
end
```

ファイル名：main.f90

```
program main
  use m
  call sub
end
```

翻訳コマンド

```
$ frtpx module.f90 -c
$ frtpx parent_submodule.f90 -c
$ frtpx child_submodule.f90 -c
$ frtpx main.f90 module.o parent_submodule.o child_submodule.o
```

10.5 組込みモジュール

組込みモジュールは、処理系に備わっているモジュールです。組込みモジュールには、標準組込みモジュールと非標準組込みモジュールがあります。

標準組込みモジュールには、以下があります。

- Fortran環境モジュール(ISO_FORTRAN_ENV)
- IEEE モジュール(IEEE_EXCEPTIONS、IEEE_ARITHMETICおよびIEEE_FEATURES)

- C言語プログラムとの相互利用のためのモジュール(ISO_C_BINDING)があります。

非標準組込みモジュールには、以下があります。

- サービスルーチンの手続引用仕様を定義したモジュールSERVICE_ROUTINES(“6.6.6 サービスルーチン”参照)
- OpenMPのモジュール OMP_LIBおよびOMP_LIB_KINDSがあります。
- MPIに関するモジュール MPI、MPI_で始まる名前、PMPI_で始まる名前、およびOMPI_で始まる名前があります。
- 数学ライブラリに関するモジュール FAST_DD、PLASMA、PLASMA_S、PLASMA_C、PLASMA_Z、PLASMA_D、PLASMA_ZC、およびPLASMA_DSがあります。

翻訳時オプション-AU を指定した場合、組込みモジュールの言語要素は、小文字で参照しなければなりません。

10.5.1 COMPILER_OPTIONS組込みモジュール関数

本関数結果の翻訳時オプション情報は、翻訳時オプション-Qまたは-Nlstで出力される翻訳時オプションの情報と同じです。プログラミングの際には、関数結果の文字列が1200文字以上であることを考慮してください。

10.5.2 COMPILER_VERSION組込みモジュール関数

本関数結果のバージョン情報は、翻訳時オプション-Vで出力されるFortranコンパイラの情報と同じです。プログラミングの際には、関数結果の文字列が90文字以上であることを考慮してください。

第11章　言語間結合

この章では、FortranとC/C++プログラムを結合する場合の仕様および注意事項について説明します。

注意

本章のCプログラムの記述は、C++プログラムにも適用されます。C++プログラムとの結合では、本章に加え“C++言語使用手引書”の“言語およびtrad/clang間結合における注意事項”を参照してください。

11.1 結合時の翻訳コマンドと必須オプション

Fortran、C言語、およびC++言語の翻訳コマンドには、クロスコンパイラとネイティブコンパイラの2種類があります。さらに、プログラムがMPIライブラリを利用しているかどうかにより、下表のように使い分けます。

以降の説明では、クロスコンパイラの翻訳コマンド(MPIライブラリの利用なし)を用います。MPIライブラリやネイティブコンパイラを利用する場合は、下表に従い翻訳コマンドを読み替えてください。

プログラム言語	MPIライブラリの利用	クロスコンパイラ	ネイティブコンパイラ
Fortran	なし	frtpx	frt
	あり	mpifrtpx	mpifrt
C言語	なし	fccpx	fcc
	あり	mpifccpx	mpifcc
C++言語	なし	FCCpx	FCC
	あり	mpiFCCpx	mpiFCC

Fortran、C言語 tradモード、C言語 clangモード、C++言語 tradモード、およびC++言語 clangモードのオブジェクトプログラムを組み合わせて結合することができます。tradモードとclangモードの詳細は、“C言語使用手引書”および“C++言語使用手引書”を参照してください。

オブジェクトプログラムの組合せによっては、結合時に使用できない翻訳コマンドがあります。オブジェクトプログラムの組合せと、結合時に使用する翻訳コマンドおよび必須オプションは、“表11.1 オブジェクトプログラムの組合せと結合時に使用する翻訳コマンド/必須オプション”を参照してください。翻訳コマンドとオプションの詳細は、各言語の使用手引書を参照してください。

表11.1 オブジェクトプログラムの組合せと結合時に使用する翻訳コマンド/必須オプション

結合するオブジェクトプログラム						COARRAY機能	翻訳コマンドと必須オプション
Fortran	C trad	C clang	C++ trad	C++ clang			
				STLにlibc++使用	STLにlibstdc++使用		
○	○	–	–	–	–	なし	frtpx または fccpx --linkfortran
						あり	frtpx -Ncoarray または fccpx --linkcoarray
○	–	○	–	–	–	なし	frtpx または fccpx -Nclang --linkfortran
						あり	frtpx -Ncoarray または fccpx -Nclang --linkcoarray

結合するオブジェクトプログラム						COARRAY機能	翻訳コマンドと必須オプション
Fortran	C trad	C clang	C++ trad	C++ clang			
				STLにlibc++使用	STLにlibstdc++使用		
○	―	―	○	―	―	なし	frtpx --linkstl=libfjc++ または FCCpx --linkfortran
						あり	frtpx --linkstl=libfjc++ -Ncoarray または FCCpx --linkcoarray
○	―	―	―	○	―	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray
				―	○	なし	frtpx --linkstl=libstdc++ または FCCpx -Nclang --linkfortran
						あり	frtpx --linkstl=libstdc++ -Ncoarray または FCCpx -Nclang --linkcoarray
―	○	○	―	―	―	―	fccpx -Nclang
―	○	―	○	―	―	―	FCCpx
―	○	―	―	○	―	―	FCCpx -Nclang
				―	○		FCCpx -Nclang -stdlib=libstdc++
―	―	○	○	―	―	―	FCCpx
―	―	○	―	○	―	―	FCCpx -Nclang
				―	○		FCCpx -Nclang -stdlib=libstdc++
―	―	―	○	○	―	―	FCCpx -Nclang -stdlib=libc++ (“注1”参照)
				―	○		結合できません。 (“注2”参照)
○	○	○	―	―	―	なし	frtpx または fccpx -Nclang --linkfortran
						あり	frtpx -Ncoarray または fccpx -Nclang --linkcoarray
○	○	―	○	―	―	なし	frtpx --linkstl=libfjc++ または FCCpx --linkfortran
						あり	frtpx --linkstl=libfjc++ -Ncoarray または FCCpx --linkcoarray
○	○	―	―	○	―	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++

結合するオブジェクトプログラム						COARRAY機能	翻訳コマンドと必須オプション
Fortran	C trad	C clang	C++ trad	C++ clang			
				STLにlibc++使用	STLにlibstdc++使用		
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray
				－	○	なし	frtpx --linkstl=libstdc++ または FCCpx -Nclang --linkfortran
						あり	frtpx --linkstl=libstdc++ -Ncoarray または FCCpx -Nclang --linkcoarray
○	－	○	○	－	－	なし	frtpx --linkstl=libfjc++ または FCCpx --linkfortran
						あり	frtpx --linkstl=libfjc++ -Ncoarray または FCCpx --linkcoarray
○	－	○	－	○	－	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray
				－	○	なし	frtpx --linkstl=libstdc++ または FCCpx -Nclang --linkfortran
						あり	frtpx --linkstl=libstdc++ -Ncoarray または FCCpx -Nclang --linkcoarray
○	－	－	○	○	－	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++ (“注1”参照)
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray (“注1”参照)
				－	○	なし あり	結合できません。 (“注2”参照)
－	○	○	○	－	－	－	FCCpx
－	○	○	－	○	－	－	FCCpx -Nclang -stdlib=libc++
				－	○		FCCpx -Nclang -stdlib=libstdc++
－	○	－	○	○	－	－	FCCpx -Nclang -stdlib=libc++ (“注1”参照)
				－	○		結合できません。 (“注2”参照)

結合するオブジェクトプログラム						COARRAY機能	翻訳コマンドと必須オプション
Fortran	C trad	C clang	C++ trad	C++ clang			
				STLにlibc++使用	STLにlibstdc++使用		
－	－	○	○	○	－	－	FCCpx -Nclang -stdlib=libc++ (“注1”参照)
				－	○		結合できません。 (“注2”参照)
○	○	○	○	－	－	なし	frtpx --linkstl=libfjc++ または FCCpx --linkfortran
						あり	frtpx --linkstl=libfjc++ -Ncoarray または FCCpx --linkcoarray
○	○	○	－	○	－	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray
				－	○	なし	frtpx --linkstl=libstdc++ または FCCpx -Nclang --linkfortran
						あり	frtpx --linkstl=libstdc++ -Ncoarray または FCCpx -Nclang --linkcoarray
○	○	－	○	○	－	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++ (“注1”参照)
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray (“注1”参照)
				－	○	なし	結合できません。 (“注2”参照)
						あり	
○	－	○	○	○	－	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++ (“注1”参照)
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray (“注1”参照)
				－	○	なし	結合できません。 (“注2”参照)
						あり	
－	○	○	○	○	－	－	FCCpx -Nclang -stdlib=libc++ (“注1”参照)

結合するオブジェクトプログラム						COARRAY機能	翻訳コマンドと必須オプション
Fortran	C trad	C clang	C++ trad	C++ clang			
				STLにlibc++使用	STLにlibstdc++使用		
				－	○		結合できません。 (“注2”参照)
○	○	○	○	○	－	なし	frtpx --linkstl=libc++ または FCCpx -Nclang --linkfortran -stdlib=libc++ (“注1”参照)
						あり	frtpx --linkstl=libc++ -Ncoarray または FCCpx -Nclang -stdlib=libc++ --linkcoarray (“注1”参照)
				－	○	なし	結合できません。 (“注2”参照)
						あり	

11.1.1 C++ tradモードとC++ clangモードのオブジェクト結合

C++ tradモードで作成したオブジェクトプログラムとC++ clangモードで作成したオブジェクトプログラムの結合可否は、翻訳時に使用したSTLに依存します。

C++ tradモード	C++ clangモード	翻訳コマンドと必須オプション
(libc++使用のみ)	STLにlibc++使用	FCCpx -Nclang -stdlib=libc++ (“注1”参照)
	STLにlibstdc++使用	結合できません。 (“注2”参照)

注1

以下の条件をすべて満たす場合、動作が不定になる可能性があり、動作保証できません。

1. C++ tradモードで作成したオブジェクトプログラムと、C++ clangモードで作成したオブジェクトとで、関数間のインタフェースを持つ。
2. 条件1の関数の引数または復帰値がクラス型である。

注2

C++ clangモードが使用するSTLのデフォルトはlibstdc++です。

結合させるには、オブジェクト作成時に-stdlib=libc++オプションを指定し、STLにlibc++を使用してください。

11.1.2 C++ clangモード同士のオブジェクト結合

C++ clangモードで作成したオブジェクトプログラム同士の結合可否は、翻訳時に使用したSTLに依存します。

翻訳時に使用したSTL		翻訳コマンドと必須オプション
libc++	libc++	FCCpx -Nclang -stdlib=libc++
libc++	libstdc++	結合できません。
libstdc++	libstdc++	FCCpx -Nclang -stdlib=libstdc++ (なお、-stdlib=libstdc++オプションはデフォルトなので省略可能)

11.1.3 MPIオブジェクトプログラム結合後のプロファイラ利用

結合時に以下のオプションも指定してください。

- frtpxコマンドで結合する場合： -lfjprofmpiオプション
- fccpxまたはFCCpxコマンドで結合する場合： -lfjprofmpifオプション

11.1.4 -fltoオプションを指定して作成したclangモードのオブジェクト

“-Nclang -flto”オプションを指定して作成したオブジェクトは、LLVM内部で使用される形式(通常のオブジェクトと異なる形式)になるため、clangモードでしか結合できません。

言語間結合で“-Nclang -flto”オプションを指定して作成したオブジェクトを含む場合は、結合時に“-Nclang -flto”オプションを指定してください。

11.2 言語間結合のための仕様

本処理系では、言語間結合をするために、Fortran規格仕様および富士通拡張仕様を提供しています。

言語間結合に関する次のFortran規格仕様を提供しています。

- 組込みモジュールISO_C_BINDING
- BIND文、BIND属性指定子、言語束縛指定子、手続言語束縛指定子
- VALUE文、VALUE 属性指定子
- 型引継ぎ
- 次元引継ぎ
- ISO_Fortran_binding.h

言語間結合に関する次の富士通拡張仕様を提供しています。

- 組込み関数VAL、実引数の%VAL指定
- CHANGEENTRY文
- $pragma指定子

また、翻訳時オプションとして以下の機能を提供しています。

- 外部手続名の加工方法を変更する翻訳時オプション-ml、-mldefault(“11.3 外部名の加工”参照)
- 文字型の引数の値の渡し方を変更する翻訳時オプション-Az(“11.13 翻訳時オプション-Az の注意事項”参照)

11.2.1 引数の値渡し

本処理系では、外部手続引用の実引数は、通常、アドレス渡しとなります(“11.7 引数による受渡し”参照)。この実引数を値渡しにしたい場合に、手続引用仕様でVALUE属性、または実引数で組込み関数VALまたは%VALを用います。

11.2.1.1 手続引用仕様での引数の値渡し

手続引用仕様でVALUE属性を指定することにより値渡しになります。結合できる実引数は、“11.2.2 引数の値受取り”を参照してください。以下に手続引用仕様のVALUE属性の例を示します。

例: 手続引用仕様のVALUE属性

```
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) ! 第1 引数は値渡し、第2 引数はアドレス渡しとなります
END
```

11.2.1.2 組込み関数による引数の値渡し

VAL関数は翻訳時オプション-Nobsfunが有効な場合に、組込み関数となります。

組込み関数VALおよび%VALは、外部手続引用の実引数だけに指定することができます。値渡しとする実引数は、1バイトの整数型、2バイトの整数型、4バイトの整数型、8バイトの整数型、1バイトの論理型、2バイトの論理型、4バイトの論理型、8バイトの論理型、半精度実数型、単精度実数型、または倍精度実数型のスカラ式でなければなりません。以下に組込み関数VALの例を示します。

例: 組込み関数VAL

```
EXTERNAL CSUB
CALL CSUB(VAL(2),3) ! 第1引数は値渡し、第2引数はアドレス渡しとなります
END
```

11.2.2 引数の値受取り

本処理系では、外部手続の仮引数は、通常、アドレス受取りとなります(“11.7 引数による受渡し”参照)。この仮引数を値受取りにしたい場合に、仮引数にVALUE属性を指定します。VALUE属性はVALUE文または型宣言文のVALUE属性指定子で指定することができます。

仮引数が、4倍精度複素数型、手続言語束縛指定子のない手続の文字型、大きさが16バイト以下の派生型、成分にポインタをもつ派生型、成分に割付け属性をもつ派生型、またはOPTIONAL 属性をもつ場合は、常にアドレスを受け取ります。

以下にVALUE 文の例を示します。

例: VALUE文

```
SUBROUTINE CSUB(I,K)
VALUE :: I ! 第1 引数は値受取り、第2 引数はアドレス受取りとなります
PRINT *, I,K
END SUBROUTINE
```

11.2.3 CHANGEENTRY文

CHANGEENTRY文は、翻訳時オプション-mlとの組合せで、外部手続名の加工方法を変更します。外部手続名の加工方法の変更の詳細については、“11.3 外部名の加工”を参照してください。

CHANGEENTRY 文の外部手続名は、手続言語束縛指定子をもってはなりません。

例: CHANGEENTRY文

```
CHANGEENTRY :: CSUB
CALL CSUB( )
END
```

11.2.4 $pragma指定子

$pragma指定子は、以下の形式で、注釈の記述箇所に指定できます。

```
!$pragma c (prc)
```

*prc*は、外部手続名でなければなりません。$pragma指定子は、*prc*に指定された手続がCプログラムで記述された手続であることを指定し、外部手続名の加工方法を変更します。外部手続名の加工方法の変更の詳細については、“11.3 外部名の加工”を参照してください。

$pragma 指定子の外部手続名は、手続言語束縛指定子をもってはなりません。

例: $pragma指定子

```
EXTERNAL CSUB !$pragma C(CSUB)
...
CALL CSUB ! CSUB は、C プログラムで記述された手続の呼出しとして扱われます。
END
```

11.2.5 組込みモジュールISO_C_BINDING

組込みモジュールISO_C_BINDING では、以下が提供されています。

- Cプログラムと相互利用のための名前付き定数
- Cプログラムと相互利用のための型
- Cプログラムと相互利用のための組込み手続

11.2.5.1 組込みモジュールISO_C_BINDINGで提供される名前付き定数

組込みモジュールISO_C_BINDINGで提供される名前付き定数には、以下があります。

- 種別型パラメタを表す名前付き定数
- Cプログラム文字を表す名前付き定数
- 空ポインタを表す名前付き定数

1. 組込みモジュールISO_C_BINDINGで提供されている種別型パラメタを表す名前付き定数と、その値および対応するCプログラムの型は“表11.2 Cプログラムの型と、Fortranの型および種別型パラメタの対応”を参照してください。Cプログラムとデータを相互利用するためには、Cプログラムの型と、Fortranの型および種別型パラメタを対応させる必要があります。また、相互利用するためには文字型の場合、長さは1でなければなりません。

表11.2 Cプログラムの型と、Fortranの型および種別型パラメタの対応

名前付き定数	値	対応するCプログラム	Fortranの型指定子
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
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_FLOAT16	2	_Float16	REAL
C_FLOAT	4	float	REAL
C_DOUBLE	8	double	REAL
C_LONG_DOUBLE	16	long double	REAL

名前付き定数	値	対応するCプログラム	Fortranの型指定子
C_FLOAT16_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

例: 組込みモジュールISO_C_BINDING で提供されている種別型パラメタを表す名前付き定数の利用

```
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 プログラム:

```
#include <stdio.h>
void sub(j,k,l)
int *j,*k,*l;
{
   printf ("J:%d K:%d L:%d ¥n",*j,*k,*l) ;
}
```

2. 組込みモジュールISO_C_BINDING で提供されているCプログラム文字を表す名前付き定数と、その意味は次の通りです。

名前付き定数	Cプログラム文字
C_NULL_CHAR	ヌル文字
C_ALERT	警報
C_BACKSPACE	後退
C_FORM_FEED	書式送り
C_NEW_LINE	改行
C_CARRIAGE_RETURN	復帰
C_HORIZONTAL_TAB	水平タブ
C_VERTICAL_TAB	垂直タブ

これらの名前付き定数は、長さ1の文字型です。

例: 組込みモジュールISO_C_BINDING で提供されているCプログラム文字を表す名前付き定数の利用

```
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_NULL_CHAR
CHARACTER(LEN=5)::ST
ST='ABCD'//C_NULL_CHAR ! 末尾にヌル文字の設定
...
END
```

3. 組込みモジュールISO_C_BINDING で提供されている空ポインタの名前付き定数と、その意味と型は次の通りです。

名前付き定数	意味	型
C_NULL_PTR	NULL	C_PTR
C_NULL_FUNPTR	関数への空ポインタ	C_FUNPTR

例: 組込みモジュールISO_C_BINDING で提供されている空ポインタの名前付き定数の利用

```
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        ! NULLの設定
CALL SUB(PTR)
END
```

11.2.5.2 組込みモジュールISO_C_BINDINGで提供される型

組込みモジュールISO_C_BINDINGでは、C_PTR型およびC_FUNPTR型が提供されています。C_PTR型はCプログラムのポインタ型、C_FUNPTR型はCプログラムの関数ポインタ型に対応します。C_PTRおよびC_FUNPTRは、成分非公開の派生型です。

11.2.5.3 組込みモジュールISO_C_BINDINGで提供される組込み手続

組込みモジュールISO_C_BINDINGでは、組込み関数C_LOC、C_FUNLOC、C_ASSOCIATED、C_SIZEOF、組込みサブルーチンC_F_POINTER、C_F_PROC_POINTERが提供されています。

C_LOCによって、引数に対応するCプログラムのポインタ型(C_PTR型)を取得します。

C_FUNLOCによって、引数に対応するCプログラムの関数ポインタ型(C_FUNPTR型)を取得します。

C_ASSOCIATEDによって、Cプログラムのポインタ型または関数ポインタ型を比較します。

C_F_POINTERによって、Cプログラムのポインタ型からFortranのポインタへ代入します。

C_F_PROC_POINTERによって、Cプログラムの関数ポインタ型からFortranの手続ポインタへ代入します。

C_SIZEOFによって、引数のサイズをバイト単位で取得します。

これらの組込み手続の詳細は、オンラインマニュアルintrinsic(3)を参照してください。

例: 組込みモジュールISO_C_BINDING で提供されている組込み手続の利用

```
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) ! 変数TARGETのCプログラムポインタ型の設定
CALL SUB(PTR)
END
```

C プログラム:

```
#include <stdio.h>
void sub(k)
```

```
int *k;
{
   printf ("K=%d ¥n",*k) ;
}
```

11.2.6 BIND文、BIND属性指定子、言語束縛指定子、手続言語束縛指定子

データ要素をCプログラムと相互利用する場合、言語束縛指定子を指定します。手続をCプログラムと相互利用する場合、手続言語束縛指定子を指定します。

言語束縛指定子と手続言語束縛指定子は、次の形式です。

```
BIND ( C [, NAME = スカラ文字定数式] )
```

言語束縛指定子または、手続言語束縛指定子の指定により、この言語要素の外部名が決まります。この指定による外部名を束縛ラベルといいます。

言語束縛指定子または、手続言語束縛指定子を指定することにより、束縛ラベルは次のようになります。ただし、NAME 指定子の値の先行する空白および後続の空白は意味をもちません。

- NAME 指定子がない、またはNAME 指定子の値が長さゼロの場合、束縛ラベルは言語要素の小文字の名前になります。
- NAME 指定子の値が長さゼロでない場合、この値が束縛ラベルとなります。束縛ラベル中の文字の大文字と小文字は、意味があります。

BIND文または、型宣言文の属性指定子で言語束縛指定子を指定することにより、変数または共通ブロックが、外部結合をもつCプログラム変数と相互利用することを指定します。言語束縛指定子の指定により、データ要素はBIND属性をもちます。

手続言語束縛指定子は、SUBROUTINE文、FUNCTION文、またはENTRY文に指定することができ、その手続をCプログラムと相互利用することを指定します。手続言語束縛指定子は、内部手続のSUBROUTINE文およびFUNCTION文には指定できません。

例: 言語束縛指定子と手続言語束縛指定子の指定

```
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プログラム:

```
#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文

BIND文は、BIND属性を指定します。次の形式です。

```
言語束縛指定子[ :: ] 束縛要素並び
```

束縛要素には、変数名または、/共通ブロック名/を指定できます。ここで変数名を指定する場合、そのBIND 文は、モジュールの宣言部に現れなければなりません。

言語束縛指定子でNAME指定子を指定する場合、束縛要素並びは1つでなければなりません。

例: BIND文を使用したCプログラムとの相互利用

```
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プログラム:

```
#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 型宣言文の属性指定子によるBIND属性

型宣言文の属性指定子に言語束縛指定子を指定することにより、BIND属性を指定します。

型宣言文の言語束縛指定子は、モジュールの宣言部で指定する場合にだけ書くことができます。また、型宣言文の言語束縛指定子でNAME指定子を指定する場合、型宣言文の変数名並びは1つでなければなりません。

例: 型宣言文の属性指定子による言語束縛指定子の指定

```
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プログラム:

```
#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 手続言語束縛指定子

手続言語束縛指定子は、SUBROUTINE文、FUNCTION文、またはENTRY文の仮引数並びに続けて指定します。手続言語束縛指定子を指定する場合、仮引数がない場合でも、仮引数並びを囲む括弧は省略できません。

例: 手続言語束縛指定子の指定

```
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プログラム:

```
#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 文、VALUE 属性指定子

VALUE文または、型宣言文の属性指定子でVALUE属性を指定することにより、指定された仮引数がVALUE属性をもちます。

VALUE属性の仮引数をもつ手続を呼び出す場合、その手続は明示的引用仕様をもたなければなりません。

仮引数が、VALUE属性をもつ場合、次のいずれかの場合を除き、引数は値渡しになります。

- 手続言語束縛指定子のない手続の文字型
- 成分にポインタをもつ派生型
- 成分に割付け属性をもつ派生型
- OPTIONAL属性をもつ

例: VALUE文、VALUE属性の指定

```
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プログラム:

```
void sub(int);
int  foo(int k1,int k2)
{
   sub(k1) ;
   sub(k2) ;
   return 0 ;
}
```

11.2.8 派生型とC言語の構造体型との相互利用

Fortran の派生型をC言語の構造体型と相互利用する場合、次の条件のすべてを満たす必要があります。

- Fortranの派生型は、BIND属性をもたなければなりません。
- Fortranの派生型は、連続型であってはなりません。
- Fortranの派生型は、型パラメタをもってはなりません。
- Fortranの派生型は、EXTENDS属性をもってはなりません。
- Fortranの派生型は、型束縛手続部をもってはなりません。
- Fortranの派生型の各成分は、相互利用可能な型および型パラメタでなければなりません。
- Fortranの派生型の各成分は、ポインタまたは割付けであってはなりません。
- Fortranの派生型とC言語の構造体型とが同じ個数の成分をもたなければなりません。
- Fortranの派生型の各成分は、C言語の構造体型の対応する成分の型と相互利用可能である型および型パラメタをもたなければなりません。

11.2.9 型引継ぎ

型引継ぎは、型指定子TYPE(*)で宣言します。無制限多相的の仮引数です。

仮引数が、次元引継ぎでない型引継ぎの場合、Cプログラムの仮引数はvoidへのポインタです。

例: 次元引継ぎでない型引継ぎの利用

Fortranプログラム:

```
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プログラム:

```
#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 ;
}
```

実行結果:

```
 Value : 10
 Value : 11
 Value : 12.000000
```

11.2.10 次元引継ぎ

有効実引数の次元数、形状、大きさを引継ぐ仮引数です。

仮引数が、次元引継ぎの場合、Cプログラムの仮引数は CFI_cdesc_t(“11.2.11 ISO_Fortran_binding.h”参照)へのポインタです。

例1: CONTIGUOUSな配列での次元引継ぎの利用

Fortranプログラム:

```
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プログラム:

```
#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 ;
}
```

実行結果:

```
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
```

例2: CONTIGUOUSでない配列での次元引継ぎの利用

Fortranプログラム:

```
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プログラム:
“例1: CONTIGUOUSな配列での次元引継ぎの利用”と共通

実行結果:

```
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
```

例3: スカラでの次元引継ぎの利用

Fortranプログラム:

```
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プログラム:
“例1: CONTIGUOUSな配列での次元引継ぎの利用”と共通

実行結果:

```
Element length  : 4
 Type           : int
 Attribute      : other
 Rank           : 0
 Value          : 2
```

11.2.11 ISO_Fortran_binding.h

ISO_Fortran_binding.hは、本処理系が提供するヘッダを格納するディレクトリに存在します。

Cプログラムのヘッダファイルです。次元引継ぎ(“11.2.10 次元引継ぎ”参照)との言語間結合で使用します。

このヘッダをインクルードしたC言語プログラムは、このヘッダで定義されていないCFI_で始まる名前は、使用してはなりません。

ISO_Fortran_binding.hでは、以下が提供されています。

- 構造体CFI_cdesc_t
- 構造体CFI_dim_t
- マクロおよび型定義

本処理系では、ISO_Fortran_binding.hに関するC関数は提供していません。

11.2.11.1 構造体CFI_cdesc_t

CFI_cdesc_tは、C構造体の型定義名です。以下のメンバをもちます。

1. base_addr

   実体のアドレスです。

2. elem_len

   スカラの場合、バイト単位の大きさ。配列の場合、1要素のバイト単位の大きさです。

3. version

   ISO_Fortran_binding.h のCFI_VERSIONと同じ値です。

4. rank

   配列の次元数です。スカラの場合、ゼロです。

5. type

   型のコード値です。型とそのマクロ名は、“11.2.11.3 ISO_Fortran_binding.hの型およびマクロ名”を参照してください。

6. attribute

   属性のコード値です。本処理系では、属性は“nonallocatable nonpointer”だけです。

7. dim

   dimは配列で宣言されています。その要素数は、実体の次元数と同じです。dimは、構造体CFI_dim_tです。

11.2.11.2 構造体CFI_dim_t

CFI_dim_tは、C構造体の型定義名です。配列の1つの次元に対する下限、要素数およびメモリ間隔情報があります。以下のメンバをもちます。

1. lower_bound

   各次元の下限値です。値はゼロです。

2. extend

   各次元の要素数の値です。大きさ引継ぎ配列の最終次元では-1です。

3. sm

   各次元のメモリ間隔の値です。連続した要素のアドレス間隔と等しい値です。

11.2.11.3 ISO_Fortran_binding.hの型およびマクロ名

CFI_CDESC_Tは、1つの引数をもつ関数形式マクロです。その引数は、C記述子の次元数です。その引数は整定数式で0以上かつ、30以下でなければなりません。

CFI_index_tは、下限、要素数およびメモリ間隔の型定義名です。

CFI_MAX_RANKマクロは、値30をもちます。本処理系での配列次元数の最大値です。

CFI_VERSIONマクロは、値1をもちます。本処理系のISO_Fortran_binding.hのバージョンです。

CFI_attribute_otherマクロは、ポインタ属性および割付け属性でないことを指定します。本処理系は、ISO_Fortran_binding.hのポインタ属性および割付け属性はサポートしていません。

CFI_type_tは、型コードのための型定義名です。マクロ名に対応する型は、“表11.3 ISO_Fortran_binding.hのマクロ名と対応する型”を参照してください。

表11.3 ISO_Fortran_binding.hのマクロ名と対応する型

マクロ名	Cプログラムの型
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
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 *

マクロ名	Cプログラムの型
CFI_type_struct	相互利用可能なCプログラム構造体
CFI_type_other	指定できません

11.3 外部名の加工

オブジェクトの外部名は、原始プログラムで記述した手続名を加工して出力されます。CプログラムでFortranの手続名を参照する場合は、Fortranと同じ規則で加工した名前を原始プログラムに記述する必要があります。

下表に、Fortranの手続名の加工方法を示します。

表11.4 Fortranの手続名の加工方法

手続名		加工名
主プログラム名		MAIN__
外部手続名		外部手続名_
初期値設定プログラム単位	BLOCK DATA 文に名前の指定がある場合	初期値設定プログラム単位名_
	BLOCK DATA 文に名前の指定がない場合	_BLKDT__
スタートアップルーチン名		main

備考1. 翻訳時オプション -AUが指定されていない場合、外部手続および名前の指定がある初期値設定プログラム単位の加工名はすべて小文字に統一されます。

備考2. スタートアップルーチンは、Fortranを起動するルーチンです。

翻訳時オプション-AUが指定された場合の外部名は原始プログラムで指定したつづりになります。手続言語束縛指定子をもつ手続の加工名は、束縛ラベルになります。外部手続名の加工方法は、翻訳時オプション-mldefaultの指定により変更することができます。翻訳時オプション-mldefaultの指定は、手続言語束縛指定子をもつ手続、CHANGEENTRY文に指定された手続および$pragma 指定子に指定された手続の外部手続名に対しては有効になりません。

下表に、-mldefaultオプション指定時の外部名の加工方法を示します。

表11.5 翻訳時オプション-mldefaultの指定と外部名の加工方法

翻訳時オプション -mldefaultの指定	外部名の加工方法
-mldefault=cdecl	外部手続名
-mldefault=frt	外部手続名_
指定なし	

CHANGEENTRY文に指定された手続の外部名の加工方法は、翻訳時オプション-mlの指定により異なります。

下表に、CHANGEENTRY文に指定された手続の外部名の加工方法を示します。

表11.6 CHANGEENTRY文に指定された手続の外部名の加工方法

翻訳時オプション-mlの指定	外部名の加工方法
-ml=cdecl	外部手続名
-ml=frt	外部手続名_
指定なし	-mldefaultオプションの指定に従います

下表に、$pragma指定子に指定された手続の外部名の加工方法を示します。

表11.7 $pragma指定子に指定された手続の外部名の加工方法

$pragma指定子の指定	外部名の加工方法
c(*prc*)	外部手続名

11.4 Cとの型の対応

FortranプログラムとCプログラムとの間でデータを受け渡す場合は、データ型の対応をとる必要があります。Fortranデータの内部表現については、“5.1 データの表現”を参照してください。

BIND属性を指定した場合のFortranとCの対応するデータ型は、“表11.2 Cプログラムの型と、Fortranの型および種別型パラメタの対応”を参照してください。

BIND属性を指定しない場合のFortranとCの対応するデータ型を下表に示します。

表11.8 FortranとCの対応するデータ型

型	Fortran	C
1バイトの論理型	LOGICAL(1) L1	unsigned char L1 ;
2バイトの論理型	LOGICAL(2) L2	short int L2 ;
4バイトの論理型	LOGICAL(4) L4	int L4 ;
8バイトの論理型	LOGICAL(8) L8	long long int L8 ;
1バイトの整数型	INTEGER(1) I1	signed char I1 ;
2バイトの整数型	INTEGER(2) I2	short int I2 ;
4バイトの整数型	INTEGER(4) I4	int I4 ;
8バイトの整数型	INTEGER(8) I8	long long int I8 ;
半精度実数型	REAL(2) R2	_Float16 R2;
単精度実数型	REAL(4) R4	float R4 ;
倍精度実数型	REAL(8) R8	double R8 ;
4倍精度実数型	REAL(16) R16	long double R16 ;
半精度複素数型	COMPLEX(2) C4	_Float16 _Complex C4 ;
単精度複素数型	COMPLEX(4) C8	struct { float r,i;} C8 ;
倍精度複素数型	COMPLEX(8) C16	struct { double r,i;} C16 ;
4倍精度複素数型	COMPLEX(16) C32	struct { long double r,i; } C32 ;
文字型	CHARACTER(10) S	char S [10] ;
派生型	TYPE TAG INTEGER I4 REAL(8) R8 END TYPE TAG TYPE(TAG)::D	struct tag { int i4; double r8; } d;

11.5 手続の呼出し

FortranからCの手続を呼び出す場合、またはCからFortranの手続を呼び出す場合には、手続名の加工方法を同じにする必要があります。Fortranの手続名の加工方法については、“11.3 外部名の加工”を参照してください。

以下に手続呼出しの例を示します。

例1: 手続言語束縛指定子を指定して、Fortran プログラムからC の関数を呼び出す場合

Fortran プログラム:

```
INTERFACE
  SUBROUTINE SUB() BIND(C)
  END SUBROUTINE
END INTERFACE
CALL  SUB( )
END
```

C プログラム:

```
#include <stdio.h>
void sub( )
{
   printf ("%s¥n","** sub **");
}
```

Fortran の手続呼出しSUBの外部名は、束縛ラベルsub となるため、C の関数名は、sub を使用します。

例2: C プログラムから、手続言語束縛指定子が指定されたFortran の手続を呼び出す場合

C プログラム:

```
void sub( );
int c_( )
{
   sub( );
   return 0;
}
```

Fortran プログラム:

```
SUBROUTINE SUB() BIND(C)
PRINT *,'** SUB **'
END SUBROUTINE
```

Fortranのサブルーチン SUBの外部名は、束縛ラベルsubになり相互利用できます。

例3: 手続言語束縛指定子を指定しないで、Fortranプログラムからの関数を呼び出す場合

Fortranプログラム:

```
EXTERNAL  SUB
CALL SUB( )
END
```

Cプログラム:

```
#include <stdio.h>
void sub_( )
{
   printf ("%s¥n","** sub **");
}
```

Fortranの手続呼出しSUBは、外部名の加工によりsub_となるため、Cの関数名は、sub_を使用します。

例4: Cプログラムから、手続言語束縛指定子がないFortran手続を呼び出す場合

Cプログラム:

```
void sub( );
int c_( )
{
   sub( );
   return 0;
}
```

Fortranプログラム:

```
SUBROUTINE SUB( )
CHANGEENTRY :: SUB
PRINT *,'** SUB **'
END SUBROUTINE
```

Fortranのサブルーチン SUBの外部名の加工方法をCプログラムに合わせるため、手続名をCHANGEENTRY文に指定し、翻訳時オプション-ml=cdeclを指定して翻訳します。

11.5.1 Cのプログラムに最初に制御を渡す方法

最初に制御が渡るプログラムがCで記述される場合、その関数名はMAIN__でなければなりません。また、mainは、存在してはいけません。ただし、翻訳時オプション-mlcmain=main指定時には、C主プログラムの関数名にmainを使用することができます。

例1: Cのプログラムに最初に制御を渡す場合

Cプログラム: a.c

```
void sub_( );
int MAIN__( )
{
   sub_( );
   return 0;
}
```

Fortranプログラム: b.f95

```
SUBROUTINE SUB( )
PRINT *,'SUB'
END SUBROUTINE
```

翻訳、リンクおよび実行:

```
$ fccpx -c a.c
$ frtpx a.o b.f95
$ ./a.out
  SUB
$
```

例2: Cのプログラムに最初に制御を渡す場合(関数名がmainの場合)

Cプログラム: a.c

```
void sub_( );
int main( )
{
   sub_( );
   return 0;
}
```

Fortranプログラム: b.f95

```
SUBROUTINE SUB( )
PRINT *,'SUB'
END SUBROUTINE
```

翻訳、リンク、および実行:

```
$ fccpx -c a.c
$ frtpx -mlcmain=main a.o b.f95
$ ./a.out
  SUB
$
```

11.5.2 Cの標準ライブラリを呼び出す方法

FortranプログラムからCの標準ライブラリを呼び出す場合、手続言語束縛指定子を指定する方法と指定しない方法があります。

11.5.2.1 手続言語束縛指定子を指定してC の標準ライブラリを呼び出す方法

手続言語束縛指定子を指定してC の標準ライブラリを呼び出すことができます。

例: 手続言語束縛指定子を指定して、Cの標準ライブラリを呼び出す場合

```
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
```

この例では、文字型スカラの実引数は、文字型配列の仮引数に対応しています。

翻訳および実行:

```
$ frtpx a.f95
$ ./a.out
same string
$
```

11.5.2.2 手続言語束縛指定子を指定しないでCの標準ライブラリを呼び出す方法

この場合、Fortranプログラム中で呼び出すCの標準ライブラリをCHANGEENTRY文に指定し、-ml=cdeclオプションを指定して翻訳する必要があります(“11.3 外部名の加工”参照)。また、FortranとCでは引数の受渡し方法が違うので注意が必要です。(“11.7 引数による受渡し”参照)。

例: 手続言語束縛指定子を指定しないで、C の標準ライブラリを呼び出す場合

Fortranプログラム: a.f95

```
INTEGER :: STRCMP
CHANGEENTRY :: STRCMP
CHARACTER(LEN=5) :: ST1,ST2
ST1 = 'ABCD¥0' ; ST2 = 'ABCD¥0'    ! 翻訳時オプション-Aeを指定する必要があります。
IF (STRCMP(ST1,ST2)==0) PRINT *," same string "
END
```

翻訳および実行:

```
$ frtpx a.f95 -ml=cdecl -Ae
$ ./a.out
  same string
$
```

11.6 関数値受渡し

ここでは、手続言語束縛指定子を指定しない手続におけるCプログラムとFortranプログラムとの間での、関数値によるデータの受渡しについて説明します。手続言語束縛指定子を指定した手続については、FortranとCの対応するデータ型は、“表11.2 Cプログラムの型と、Fortranの型および種別型パラメタの対応”を参照してください。

11.6.1 手続言語束縛指定子を指定しないFortranの関数値のCでの受取り

Fortranの関数値とCでのその呼出しおよび受取り方法を以下の表に示します。

Fortranが返却する関数値	C側の受取り方	CからFortranを呼び出す例
INTEGER(1)	signed char	res = sub_( );

Fortranが返却する関数値	C側の受取り方	CからFortranを呼び出す例
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)	int	
LOGICAL(8)	long long int	
COMPLEX(2)	構造体	
COMPLEX(4)		
COMPLEX(8)		
COMPLEX(16)		
CHARACTER	void	sub_(&res,len); (注)
派生型	構造体	res = sub_( );

sub_: 呼び出す Fortranの手続名。

res : 関数値の返却域。対応する型については、“11.4 Cとの型の対応”を参照してください。

len: 関数値の文字長。Fortranは8バイト整数型の値として文字長を受け取ります。

注) 手続言語束縛指定子のない手続における文字型の関数値の返却は、実引数域を使用します。この場合、結果域の番地を第1引数、結果域の長さを第2 引数でそれぞれ受渡し、実際の引数は、第3 引数以降で受け渡されます。

例1: 整数型関数値の受取り

Cプログラム:

```
#include <stdio.h>
int ifun_( );
int main( )
{
  int i;
  i = ifun_( );
  printf("%d¥n",i);
  return 0;
}
```

Fortranプログラム:

```
FUNCTION IFUN( )
INTEGER IFUN
IFUN= 100
END
```

例2: 倍精度実数型関数値の受取り

Cプログラム:

```
#include <stdio.h>
double r8fun_( );
int main( )
```

```
{
  double r8;
  r8 = r8fun_( );
  printf("%f¥n",r8);
  return 0;
}
```

Fortranプログラム:

```
FUNCTION R8FUN( )
REAL(8) R8FUN
R8FUN = 12.2D+0
END
```

例3: 文字型関数値の受取り

Cプログラム:

```
#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プログラム:

```
FUNCTION CFUN( )
CHARACTER (LEN=*) CFUN
CFUN = "1234567890"
END
```

文字型関数CFUNの関数値を受け取るには、結果値の返却域の番地を第1引数に設定して、関数値の長さを第2引数に設定します。このとき、Fortranは文字型関数CFUNの長さを8バイト整数型の値として受け取ります。

例4: 派生型関数値の受取り

Cプログラム:

```
#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プログラム:

```
FUNCTION DFUN( )
TYPE TAG
  INTEGER(4) I
  REAL(8) FR8
END TYPE
TYPE (TAG)::DFUN
DFUN%I = 10
```

```
DFUN%FR8 = 1.2D+0
END
```

11.6.2 手続言語束縛指定子を指定しないFortranでのCの関数値の受取り

Cの関数値と、手続言語束縛指定子を指定しないFortranでのその呼出しおよび受取り方法を以下の表に示します。

Cが返却する関数値	Fortran側の受取り方	FortranからCを呼び出す例
void	受け取らない	CALL SUB( )
signed char	INTEGER(1)	RES = SUB( )
short int	INTEGER(2)	
int	INTEGER(4) または LOGICAL(4)	
long long int	INTEGER(8)	
_Float16	REAL(2)	
float	REAL(4)	
double	REAL(8)	
long double	REAL(16)	
構造体	派生型	

SUB: 呼び出すCの関数名。加工名については“11.3 外部名の加工”を参照してください。

RES: 関数値の返却域。

例1: 整数型関数値の受取り

Fortranプログラム:

```
INTEGER(4) I,IFUN
I = IFUN( )
PRINT *,I
END
```

Cプログラム:

```
int ifun_( )
{
  return 100;
}
```

例2: 派生型関数値の受取り

Fortranプログラム:

```
TYPE TAG
  INTEGER(4) I
  REAL(8) FR8
END TYPE
TYPE (TAG) :: DFUN,D
D = DFUN()
PRINT *,D%I,D%FR8
END
```

Cプログラム:

```
struct tag
{ int i;
  double fr8; } ;
struct tag dfun_()
```

```
{
  struct tag d;
  d.i = 100;
  d.fr8 = 1.2;
  return(d);
}
```

11.7 引数による受渡し

手続間の引数の受渡しは、標準ではアドレス渡しとなります。Fortran から値渡しをする場合は、“11.2.1 引数の値渡し” を参照してください。Fortran で値受取りとする場合は、“11.2.2 引数の値受取り” を参照してください。

文字型の引数を受け渡す場合は、引数長を8バイト整数型の値の実引数として渡します。引数長を表す実引数は、指定されたすべての引数の後ろに、文字型の引数の個数分だけ存在します。ただし、手続言語束縛指定子をもつ手続および、VALUE 属性の引数では引数長は渡されません。

ここでは、手続言語束縛指定子を指定しない手続で、引数によってデータの受け渡しを行う方法について説明します。

11.7.1 手続言語束縛指定子FortranからCへの引数の受渡し

Fortranは、標準では実引数はアドレス渡しのため、Cで仮引数を宣言する場合にはアドレスを受け取るようにします。Fortranの実引数を値渡しにする場合は、手続引用仕様で仮引数にVALUE属性を指定するか、組込み関数VALまたは%VALを実引数に指定します。これらについては、“11.2.1 引数の値渡し”を参照してください。

例: 整数型、実数型引数の受渡し

Fortranプログラム(主プログラム):

```
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プログラム:

```
#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 Cから手続言語束縛指定子を指定しないFortranへの引数の受渡し

Fortranは、標準では仮引数はアドレス受取りのため、Cで実引数を記述する場合にはアドレスを渡すようにします。Fortranの仮引数を値受取りとする場合は、仮引数にVALUE属性を指定します。これらについては、“11.2.2 引数の値受取り”を参照してください。

例1: 整数型引数の受渡し

Cプログラム(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プログラム:

```
INTEGER FUNCTION FUN(X,Y)
INTEGER(4) X,Y
WRITE(*,*) " *** FUN *** "
FUN = X+Y
END
```

例2: 文字型引数の受渡し

Cプログラム(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プログラム:

```
CHARACTER(LEN=*) FUNCTION FUN(CH)
CHARACTER(LEN=*) CH
FUN = CH//'k'
END
```

文字型関数FUNは1つの文字型の引数をもつ手続ですが、Cからこの手続を呼び出すのに四つの引数を渡します。第1引数は関数値の返却域の番地を渡し、第2引数は関数値の長さを渡します。第3引数は引数chの番地を渡し、第4引数は引数chの長さを渡します。ここでFortranは、文字型関数FUNの長さおよび引数chの長さを8バイト整数型として受け取ります。

11.8 外部変数による受渡し

外部変数と結合しデータを受け渡すには、BIND属性を指定する方法とBIND属性を指定しない方法があります。

11.8.1 BIND属性を指定する外部変数の受渡し

共通ブロックまたはモジュール変数にBIND属性を指定した場合、その共通ブロックまたは変数は、外部変数と結合できます。外部名は、変数または共通ブロックの束縛ラベルになります。

例1: BIND属性を指定した外部変数との結合

```
MODULE MOD
    INTEGER,BIND(C)::INT_VARIABLE  ! Cプログラムの外部変数int_variableと結合します。
END
```

例2: BIND属性を指定した名前付き共通ブロックの受渡し

Fortranプログラム:

```
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プログラム:

```
#include <stdio.h>
struct tag { int i,j; } ext;
void sub( )
{
   printf("i=%d j=%d¥n",ext.i,ext.j);
}
```

共通ブロックEXT にBIND属性を指定して受け渡します。

11.8.2 BIND属性を指定しない外部変数の受渡し

BIND属性を指定しない場合、Fortran の共通ブロックに属するデータをC では構造体の外部変数で受け渡すことができます。Fortran の共通ブロック名とCの外部変数名は以下の対応でなければなりません。

共通ブロックの指定	C の構造体の外部変数名
名前付き共通ブロック	共通ブロック名_
無名共通ブロック	_BLNK__

共通ブロック名のつづりは、翻訳時オプション-AU が指定されていない場合は小文字です。翻訳時オプション-AU が指定されている場合、原始プログラムに記述したつづりとなります。

例1: BIND属性を指定しない名前付き共通ブロック

Fortranプログラム:

```
INTEGER I,J
COMMON /EXT/I,J
I = 1
J = 1
CALL SUB( )
END
```

Cプログラム:

```
#include <stdio.h>
struct tag { int i,j; } ext_;
void sub_( )
{
   printf("i=%d j=%d¥n",ext_.i,ext_.j);
}
```

共通ブロックEXTを参照するためには、名前を小文字にして、末尾にアンダースコアを付けた名前ext_を使用します。

例2: BIND属性を指定しない無名共通ブロック

Fortranプログラム:

```
COMMON // X,Y
X = 1.0
Y = 2.0
CALL SUB( )
END
```

Cプログラム:

```
#include <stdio.h>
struct tag { float x,y;} _BLNK__ ;
void sub_( )
{
   printf("x=%f y=%f¥n",_BLNK__.x, _BLNK__.y);
}
```

11.9 ファイルによる受渡し

ファイルによってデータの受渡しを行う場合、そのファイルはCLOSEされた状態でなければなりません。

FortranおよびCプログラムのCLOSEされたファイルの条件は以下の処理がなされたファイルです。

条件:

```
Fortran: CLOSE文
C: close(2)
```

11.10 配列の記憶順序の相違

FortranとCでは、2次元以上の配列における各配列要素の記憶順序が異なります。したがって、

配列を受け渡す場合は注意が必要です。

FortranとCの配列の相違を、以下の図に示します。

(Fortranの配列)

INTEGER(4) K(2,3)

Fortranの記憶順序

K(1,1)
K(2,1)
K(1,2)
K(2,2)
K(1,3)
K(2,3)

(Cの配列)

int k[2][3];

Cの記憶順序

k[0][0]
k[0][1]
k[0][2]
k[1][0]
k[1][1]
k[1][2]

11.11 Cへの文字定数の受渡し

Fortranから渡した文字定数の末尾には、¥0(ヌル文字)は設定されていません。そのため、Cプログラムの中でstrlenなどの関数を使用する際には、以下のいずれかの対応が必要です。

- 組込みモジュールISO_C_BINDINGのC_NULL_CHAR(ヌル文字)を文字列の末尾に追加する。
- Fortranの定数の末尾に¥0(ヌル文字)を追加する。ただし、翻訳時オプション-Aeを指定する必要があります。

・ 翻訳時オプション-Azを指定する。(引数に文字定数を指定した場合)

例1: BIND属性とC_NULL_CHARを使用した文字定数の受渡し

Fortranプログラム:

```
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) ! 末尾にヌル文字を追加
END
```

Cプログラム:

```
#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');
}
```

例2: BIND属性を指定せず、ヌル文字を設定した文字定数の受渡し

Fortranプログラム:

```
! CALL SUBA('12345')
CALL SUBA('12345¥0') ! 末尾に¥0 を追加
END
```

C プログラム:

```
#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');
}
```

備考. Fortranプログラムの翻訳では、翻訳時オプション-Aeの指定が必要です。

11.12 実行可能プログラムの復帰値

以下に示すように、FortranとC/C++プログラムとの言語間結合では、最初に制御を渡すプログラムにより実行可能プログラムの復帰値が決定します。

次の場合には、実行可能プログラムの復帰値にFortranの復帰コードが設定されます。

Fortranの復帰コードの詳細は、“3.6 実行コマンドの復帰値” を参照してください。

・ 最初に制御を渡すプログラムがFortranで記述されている場合

次の場合には、実行可能プログラムの復帰値にC/C++言語の復帰値が設定されます。

・ 最初に制御を渡すプログラムがC/C++言語で記述されている場合

11.13 翻訳時オプション-Az の注意事項

翻訳時オプション-Azを指定することにより、定数の末尾に¥0(ヌル文字)を追加することができますが、Fortran 77 以前の仕様で記述されたプログラムで一部動作が保証されません。

例: 翻訳時オプション-Az

```
CALL SUB('12')
END
SUBROUTINE SUB(I)
INTEGER(4) I
IF (I.NE.'12') PRINT *,'ok'
END SUBROUTINE
```

上記プログラムのCALL 文は、以下のように変更してください。

```
CALL SUB('12  ')
```

11.14 Cプログラムとの結合の制限事項

CプログラムとFortranプログラムを結合する場合には、以下のような制限事項および注意事項があります。また、最適化機能と組み合わせた注意事項は、“9.19 言語間結合における注意事項”を参照してください。

・ BIND属性を指定した場合、Fortranの言語要素とCの言語要素は、相互利用可能でなければなりません。

・ BIND属性による束縛ラベルは、他の外部加工名(“11.3 外部名の加工”参照)と一致してはなりません。

・ 関数引用の形でCプログラムを呼び出す場合、関数は、“11.6 関数値受渡し”に示される対応をとらなければなりません。

・ CプログラムからFortranプログラムを呼び出す場合、Fortranプログラムは、主プログラムであってはなりません。

・ 主プログラムがCプログラムの場合、その名前はMAIN__でなければなりません。ただし、翻訳時オプション-mlcmain=main指定時には、mainを使用することができます。

・ Cプログラムでsignal(2)、sigaction(2)を使用する場合は、実行時オプション-iによりFortranの割込み処理を抑止する必要があります。また実行時オプション-t による実行打ち切り時間の指定、ALARMサービス関数、SIGNALサービス関数を使用してはなりません。

・ 標準入出力ファイル以外のファイルは、閉じてから他言語のプログラムに渡さなければなりません。

・ 選択戻り指定子付きCALL文は、呼出し先もFortranの場合についてのみ有効です。選択戻り指定子付きCALL文でCプログラムを呼び出すと、誤動作のおそれがあるので注意してください。

・ Cプログラムから呼び出されるFortranのサブルーチン副プログラムは、仮引数並びに*を含んではなりません。このようなサブルーチン副プログラムをCプログラムから呼び出すと、誤動作のおそれがあるので注意してください。

・ Fortranの手続からCプログラムのsetjmpまたはlongjmp関数を呼び出してはなりません。

・ 外部変数と結合(“11.8 外部変数による受渡し”参照)し、Cプログラムで外部変数に初期値を設定している場合、リンク時に警告が出力され、Fortranの変数の境界は、Cプログラムの外部変数の境界値に変更されます。

・ Cプログラムでsetjmpまたはlongjmp関数を呼び出す場合、その手続は、Fortranの手続を呼び出してはなりません。

・ Fortranの手続をシグナル処理ルーチンとして呼び出す場合は、VOLATILE属性の変数だけを引用することができます。

・ Fortranの手続からCプログラムの可変個数の引数をもつ関数を呼び出してはなりません。

・ 組込みモジュールISO_C_BINDINGの言語要素C_PTR型の変数は、デバッガを用いてデバッグできません。

・ C++プログラムで例外処理(try、catch、throw)を含む場合、その手続は、Fortranの手続を呼び出してはなりません。

・ C++プログラムが含まれる場合の言語間結合の方法は、“C++言語使用手引書”の“言語およびtrad/clang間結合における注意事項”を参照してください。

11.15 C++プログラムとの結合方法

C++プログラムとFortranプログラムを結合する場合、リンク時に--linkstl=*stl_kind*オプションを指定してリンクします。C++プログラムの翻訳時に使用したSTLと同じSTLを--linkstl=*stl_kind*オプションで指定する必要があります。

Fortranプログラム: b.f95

```
SUBROUTINE SUB( )
PRINT *,'SUB'
END SUBROUTINE
```

C++プログラム: a.cc

```
#include <iostream>
extern "C" {
  void sub_( );
}
int MAIN__( )
{
  sub_( );
  std::cout << " C++ Program" << std::endl;
  return 0;
}
```

例1: STLがlibfjc++の場合

C++コンパイラのtradモードのSTLです。

翻訳、リンク、および実行:

```
$ FCCpx -c a.cc
$ frtpx a.o b.f95 --linkstl=libfjc++
$ ./a.out
 SUB
 C++ Program
$
```

例2: STLがlibc++の場合

C++コンパイラのclangモードにおいて、-stdlib=libc++オプションを指定したときに使用されるSTLです。

翻訳、リンク、および実行:

```
$ FCCpx -Nclang -stdlib=libc++ -c a.cc
$ frtpx a.o b.f95 --linkstl=libc++
$ ./a.out
 SUB
 C++ Program
$
```

例3: STLがlibstdc++の場合

libstdc++はC++コンパイラのclangモードにおいて、デフォルトで使用されるSTLです。または、-stdlib=libstdc++オプションを指定したときに使用されるSTLです。

翻訳、リンク、および実行:

```
$ FCCpx -Nclang -c a.cc
$ frtpx a.o b.f95 --linkstl=libstdc++
$ ./a.out
 SUB
 C++ Program
$
```

第12章　並列化機能

この章では、LLVM OpenMPライブラリを使用してFortranプログラムを並列処理する方法について記述しています。

富士通OpenMPライブラリを使用する場合は、“付録J 富士通OpenMPライブラリ”を参照してください。

注意

以下の条件を満たしている場合、本処理系はLLVM OpenMPライブラリを使用します。

- プログラムのリンク時に、-Nlibompオプションが有効である。

参考

並列処理用ライブラリ(LLVM OpenMPライブラリおよび富士通OpenMPライブラリ)と結合可能なオブジェクトファイルの組合せを下表に示します。

tradモードおよびclangモードについては、“C言語使用手引書”または“C++言語使用手引書”を参照してください。

	Fortran オブジェクトファイル	C/C++オブジェクトファイル	
		tradモード (-Nnoclangオプション)	clangモード (-Nclangオプション)
LLVM OpenMPライブラリ (-Nlibompオプション)	結合可	結合可	結合可
富士通OpenMPライブラリ (-Nfjomplibオプション)	結合可	結合可	**結合不可**

12.1 並列処理の概要

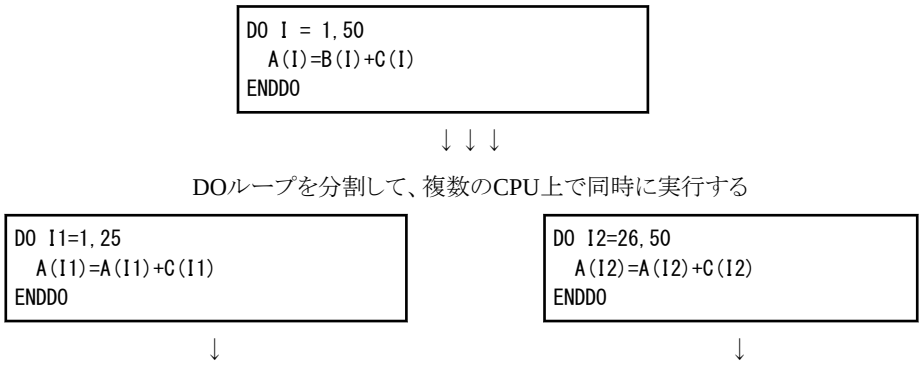
ここでは、Fortranプログラムの並列処理の概要と本処理系の並列機能の特徴について説明しています。

12.1.1 並列処理とは

並列処理という言葉は広い意味に使われていますが、ここで述べる並列処理とは、単独で動作可能なCPUを同時に複数使って1つのプログラムを実行することを意味します。ここでは、複数のプログラムを同時に実行するマルチジョブのことを、並列処理の定義に含めないこととします。

同時に複数のCPUを利用して、並列処理する様子を下図に示します。

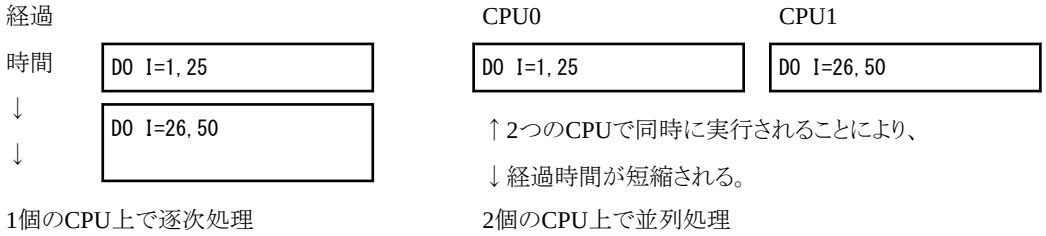
図12.1 並列処理のイメージ



12.1.2 並列処理の効果

並列処理の効果は、複数のCPUを同時に使うことによって、プログラムの実行時間が短縮されることです。例えば、“図12.1 並列処理のイメージ”に示したように、DOループを分割して、2つのDOループを並列実行できたとすると、このDOループを実行するのにかかる時間は、理想的には半分になります(“図12.2 並列処理による経過時間の短縮”)。

図12.2 並列処理による経過時間の短縮



しかし、並列処理によってプログラム実行の経過時間の短縮は可能ですが、プログラム実行に要するCPU時間を減らすことはできません。なぜなら、並列処理では複数のCPUを利用しますが、それぞれのCPUの消費するCPU時間の合計は、プログラムを逐次的に実行した場合のCPU時間と同等か、または、並列処理に必要なオーバヘッドの分だけ増加するためです。

並列処理プログラムの性能は、一般に、実行に要したCPU時間の短縮ではなく、経過時間の短縮で評価することに注意してください。逐次的に動作していたプログラムを並列処理によって実行した場合、並列処理のためのオーバヘッドによってプロセッサ全体で消費するCPU時間の合計は、逐次処理時のCPU時間よりも大きくなります。

12.1.3 並列処理で効果を得るための条件

並列処理で経過時間を短縮するためには、複数のCPUを同時に使える計算機環境が必要です。本処理系を使用して作成した並列処理プログラムは、単一のCPUしかもたないハードウェアでも実行可能ですが、この場合には経過時間の短縮は望めません。また、複数のCPUをもつハードウェア上でも、他のジョブが動いていて計算機全体としてCPU時間が不足しているような状況下では、経過時間の短縮は難しくなります。これは、並列処理プログラムの実行のために、同時に複数のCPUが割り当てられる機会が少なくなるためです。

つまり、並列処理で効果を得るためには、複数のCPUをもつハードウェア上で、かつ、システム全体としてCPU処理能力に余裕のある環境下で、並列処理プログラムを実行する必要があります。

また、並列処理のためのオーバヘッドを相対的に小さくするため、ループの繰返し数またはループ中の実行文数が多いことが必要です。

12.1.4 本処理系の並列機能の特徴

本処理系は、自動並列化機能およびOpenMP仕様による並列化機能を提供します。

自動並列化は、本処理系が自動的にプログラムを並列処理することです。自動並列化の対象となるのは、本処理系が並列化可能であると認識できるDOループと配列操作の文に限られますが、プログラムの書き換えなどのユーザ負担を軽減できます。

また、本処理系には、自動並列化を促進する最適化制御行が用意されています。最適化制御行は、プログラマがコンパイラに自動並列化の参考になる情報を通知し、効率の良いオブジェクト生成を促進するために用いられます。

自動並列化の機能を利用する場合、逐次実行可能な原始プログラムに、何ら手を加える必要はありません。したがって、プログラムがFortran規格に合致している限り、他の処理系へ原始プログラムを移植するのは容易です。

自動並列化については、“12.2 自動並列化”を参照してください。

OpenMP仕様による並列化については、“12.3 OpenMP仕様による並列化”を参照してください。

12.2 自動並列化

ここでは、本処理系で提供している自動並列化について説明します。

12.2.1 翻訳の方法

自動並列化の機能を使用するには、翻訳コマンドに以下のオプションを指定します。

```
-Kparallel [-Nlibomp]
```

12.2.1.1 自動並列化のための翻訳時オプション

自動並列化機能に関連するオプションを以下に示します。各オプションの詳細については、“2.2 翻訳時オプション”を参照してください。

```
-K{parallel|parallel_strong|visimpact} [ ,reduction,instance=N,array_private,parallel_iteration=N,
dynamic_iteration,ocl,optmsg=2,independent=pgm_nm,loop_part_parallel,parallel_fp_precision,
region_extension ] [-Nlibomp]
```

12.2.2 実行の方法

自動並列化されたプログラムを実行させるときは、環境変数OMP_NUM_THREADSで、並列に動作させるスレッドの数を指定することができます。また、環境変数OMP_STACKSIZEで、スレッド毎のスタック領域の大きさを指定することができます。さらに、環境変数OMP_WAIT_POLICYで、同期待ち処理の動作を指定することができます。

その他の実行に関する手続は、逐次実行のときと同じです。

12.2.2.1 スレッド数

並列に動作するスレッドの数は、まず以下の優先順位で決定されます。

1. 環境変数OMP_NUM_THREADSの指定値
2. システムで利用できるCPU数

自動並列では以下のOpenMP仕様の環境変数を使用することが可能です。OpenMP仕様の環境変数の詳細については、OpenMP仕様書を参照してください。

環境変数OMP_NUM_THREADS

実行中に使用するスレッドの数を指定します。

12.2.2.2 実行時の領域

環境変数OMP_STACKSIZEで、スレッド毎のスタック領域の大きさを指定することができます。

環境変数OMP_STACKSIZE

スレッド毎のスタック領域の大きさをバイト、Kバイト、Mバイト、Gバイト、Tバイト単位で指定することができます。デフォルト値は8Mバイトです。

12.2.2.3 同期待ち処理

環境変数OMP_WAIT_POLICYで、同期待ち処理の動作を指定することができます。

環境変数OMP_WAIT_POLICY

ACTIVE	同期が獲得できるまでスピン待ちを行います。
PASSIVE	スピン待ちを行わず、サスペンド待ちを行います。

デフォルトは、PASSIVEです。

経過時間を重視する場合には、ACTIVEを選択してください。CPU時間を重視する場合は、PASSIVEを選択してください。

12.2.2.4 サービス関数および組込みサブルーチン使用時の注意

- ALARMサービス関数

  動作は保証されません。逐次実行プログラムだけで使用してください。

- KILLおよびSIGNALサービス関数

  デッドロックの危険性があります。逐次実行プログラムだけで使用してください。

- FORKサービス関数

  動作は保証されません。逐次実行プログラムだけで使用してください。

- SYSTEM、SH、CHMODサービス関数、およびEXECUTE_COMMAND_LINE組込みサブルーチン

  メモリの使用量が2倍になり実行性能が低下する可能性があります。逐次実行プログラムだけでの使用をお勧めします。

12.2.2.5 スレッドのCPUバインド

環境変数GOMP_CPU_AFFINITYまたはOMP_PROC_BINDを使用して、スレッドの固定のCPUへのバインドを制御することができます。

環境変数GOMP_CPU_AFFINITYは、環境変数OMP_PROC_BINDより優先されます。

環境変数GOMP_CPU_AFFINITY

スレッドを指定されたcpuid順にCPUバインドします。

スレッド数が、指定したcpuidの数を超える場合は、先頭から繰り返して使用します。

各cpuidは、コンマ(',')または空白(' ')で区切る必要があります。

cpuidは、以下のような形式で増分値付き範囲指定ができます。

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: 範囲指定の最初のcpuid(0≦*cpuid1*＜CPU_SETSIZE)
cpuid2: 範囲指定の最後のcpuid(0≦*cpuid2*＜CPU_SETSIZE)
inc: 増分値(1≦*inc*＜CPU_SETSIZE)

なお以下の条件を満たす必要があります。

```
cpuid1≦cpuid2
```

*cpuid1*から*cpuid2*の範囲のcpuidを増分値*inc*ごとにすべてを指定した場合と等価になります。

cpuidは、上記のように指定はできますが、実際に使用できるcpuidは、実行開始時のプロセスのCPU affinityの範囲内のcpuidのみになります。

CPU_SETSIZEについては、CPU_SET(3)を参照してください。

注意

cpuidが実行開始時のプロセスのCPU affinityの範囲外である場合、エラーを出力しプログラムが終了しますので、設定値を見直してください。

例

環境変数GOMP_CPU_AFFINITYの指定例

一 例1:

```
$ export GOMP_CPU_AFFINITY="12,14,13,15"
```

12,14,13,15の順でスレッドにバインドします。
スレッド数5以上の場合は、先頭から繰り返して使用します。

— 例2:

```
$ export GOMP_CPU_AFFINITY="12-19"
```

12,13,14,15,16,17,18,19の順でスレッドにバインドします。
スレッド数9以上の場合は、先頭から繰り返して使用します。

— 例3:

```
$ export GOMP_CPU_AFFINITY="12-19:2"
```

12,14,16,18の順でスレッドにバインドします。
スレッドが5以上の場合は、先頭から繰り返して使用します。

— 例4:

```
$ export GOMP_CPU_AFFINITY="12-16:2,13,19"
```

12,14,16,13,19の順でスレッドにバインドします。
スレッドが6以上の場合は、先頭から繰り返して使用します。

環境変数OMP_PROC_BIND

スレッドアフィニティを制御します。

TRUE、FALSE、またはカンマで区切ったMASTER、CLOSE、SPREADのリストのいずれかを設定します。リストの値は、ネストレベルに対応するPARALLELリージョンで使用するスレッドアフィニティポリシーを設定します。

デフォルトは、CLOSEです。

FALSEを設定した場合、スレッドアフィニティは無効になり、PARALLEL構文のPROC_BIND指示節も無視されます。

FALSE以外を設定した場合、スレッドアフィニティは有効になり、初期スレッドはプレースリストの最初のプレースにバインドされます。

MASTERを設定した場合、すべてのスレッドはマスタースレッドと同じプレースにバインドされます。

CLOSEを設定した場合、スレッドはマスタースレッドがバインドされているプレースの次から連続したプレースにバインドされます。

SPREADを設定した場合、マスタースレッドのプレースパーティションが分割され、スレッドは各サブパーティションの1つのプレースにバインドされます。

TRUEを指定した場合、SPREADを指定した場合と同様の効果になります。

プレースについては環境変数OMP_PLACESを参照してください。

環境変数OMP_PLACES

スレッドアフィニティを制御するプレースリストを定義します。

プレースとは、スレッドが実行されるプロセッサの順序付けされていない集合です。プレースリストとは、利用可能なプレースの順序付けされたリストです。

環境変数OMP_PLACESには、抽象名または明示的なプレースリストのどちらかを、以下の形式で設定します。

```
{ abstract-name[(num-places)] | place-list }
```

abstract-name

抽象名です。抽象名には以下を指定します。デフォルトは、CORESです。

抽象名	意味
THREADS	プレースは、1ハードウェアスレッド単位になります。 本処理系ではシステムのCPU資源の最小単位になりCORESと等価になります。
CORES	プレースは、1コア単位になります。 本処理系ではシステムのCPU資源の最小単位になりTHREADSと等価になります。
SOCKETS	プレースは、1ソケット単位になります。

抽象名	意味
	本処理系ではNUMAノード単位になります。

num-places

抽象名形式のプレース数です。1以上の正の整数です。

place-list

明示的なプレースリストです。以下の形式で指定します。

```
{ place:length[:stride] | [!]place[,place-list] }  (注1)
```

place

“{”*place1*“}”(注2)

place1

{ *cpuid*:*length*[:*stride*] | [!]*cpuid*[,*place1*] }

length

インターバル指定の要素数です。1以上の正の整数です。

stride

インターバル指定の増分値です。1以上の正の整数で、デフォルト値は1です。

cpuid

システムのCPU資源の最小単位の識別番号です。

注1)排他演算子!は直後の指定を除外することを指示します。

注2)“{”および“}”は選択記号ではありません。半角中括弧を使用して指定することを意味します。

例

環境変数OMP_PLACESの指定例

- 例1:

```
$ export OMP_PLACES="CORES"
```

抽象名“CORES”のプレースリストを定義しています。

- 例2:

```
$ export OMP_PLACES="CORES(4)"
```

抽象名“CORES”の4プレースのプレースリストを定義しています。

- 例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"
```

上記3つの定義は、すべて等価です。

cpuid12～14、15～17、18～20、および21～23の同じ4つのプレースを定義しています。

12.2.3 翻訳・実行の例

例1:

```
$ frtpx -Kparallel,reduction,ocl -Nlibomp test1.f
$ ./a.out
```

例1は、リダクションの最適化と最適化制御行を有効にして、原始プログラムを翻訳します。このプログラムが実行時に使用するスレッド数は、実行時環境によって決まります。スレッド数の詳細については、“12.2.2.1 スレッド数”を参照してください。

例2:

```
$ frtpx -Kparallel -Koptmsg=2 -Nlibomp test2.f
Fortran diagnostic messages: program name(main)
  jwd5001p-i "test2.f", line 2: このDOループを並列化しました。(名前:i)
$ export OMP_NUM_THREADS=2
$ ./a.out
$ export OMP_NUM_THREADS=4
$ ./a.out
```

例2は、-Koptmsg=2オプションを指定して翻訳して、自動並列化されたことを確認します。環境変数OMP_NUM_THREADSに値 2を設定して、2個のスレッドを使用して動作させます。次に、環境変数OMP_NUM_THREADSに値4を設定して、4個のスレッドを使用して動作させます。

12.2.4 並列化プログラムのチューニング

並列化したプログラムをチューニングする手段として、基本プロファイラを利用することができます。基本プロファイラにより、プログラムのどの文の実行コストが高いかを知ることができます。プログラムを高速に実行させるには、実行コストの高い文が並列に動作するようにプログラミングします。

基本プロファイラについては、“プロファイラ使用手引書”を参照してください。

12.2.5 自動並列化機能の詳細

ここでは、本処理系のもつ自動並列化機能について説明します。

12.2.5.1 自動並列化の対象

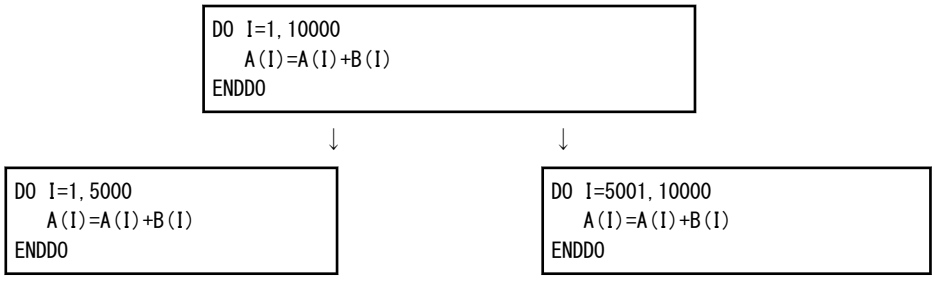
自動並列化機能は、Fortran原始プログラム中のDOループ(多重DOループも含みます)および配列操作の文(配列式、配列代入)を並列化の対象にします。

12.2.5.2 ループスライスとは

自動並列化機能は、DOループを複数に分割します。そして分割されたDOループを並列に実行することで、経過時間の短縮を図ります。この並列化をループスライスと呼びます。

下図に、ループスライスのイメージを示します。

図12.3 ループスライスのイメージ

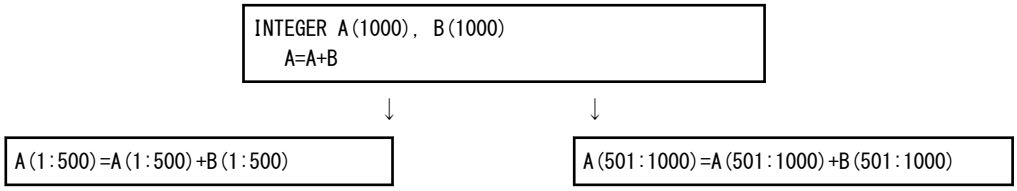


12.2.5.3 配列操作と自動並列化

自動並列化は、DOループの他に、配列操作の文(配列式、配列代入)を対象にします。

下図に、配列操作の文の自動並列化のイメージを示します。

図12.4 配列操作の文の自動並列化

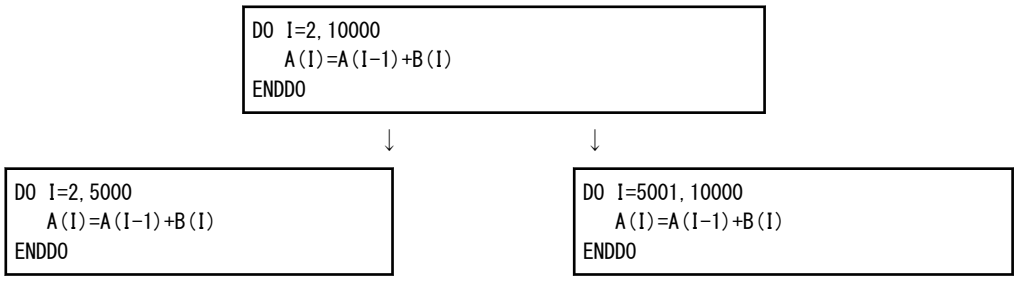


12.2.5.4 コンパイラによる自動ループスライス

本処理系は、ループスライスを行ってもデータの定義引用順序が変わらないようなDOループを自動並列化の対象として選択して、並列実行の結果が逐次実行の結果と同じになるようにします。

“図12.5 ループスライスができないDOループの例”のDOループは、自動ループスライスの対象にならない例です。このDOループでは、本来、DO変数Iが5000のときに定義された配列要素A(5000)の値を、DO変数Iが5001のときに引用しなければなりません。しかし、このDOループを単純にループスライスして並列に実行すると、A(5000)を定義する前に引用する可能性があり、実行結果を誤ることになります。

図12.5 ループスライスができないDOループの例

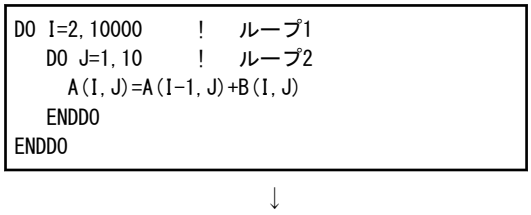


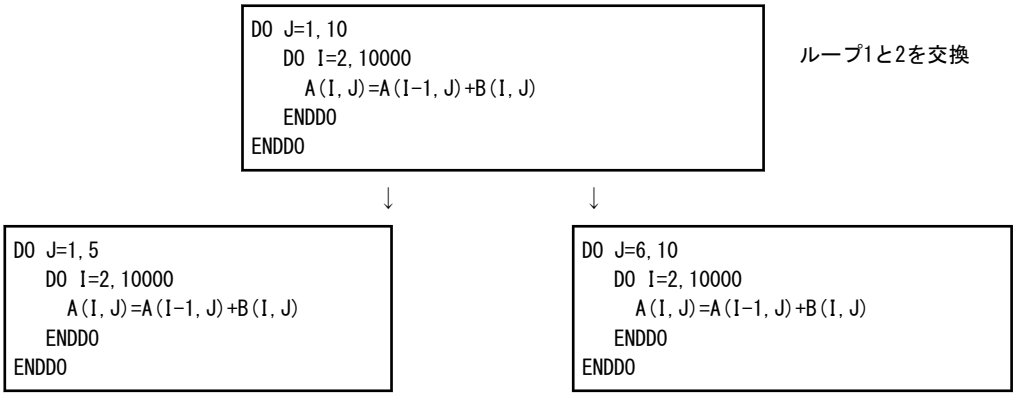
12.2.5.5 ループ交換と自動ループスライス

多重DOループに対してループスライスを実施する場合、本処理系は、ループスライスが可能なDOループを選択し、可能ならばDOループの交換を行い、できるだけ外側のDOループをループスライスします。これは、できるだけ並列処理制御の回数を少なくしてオーバヘッドを削減し、実行性能を向上させるためです。

“図12.6 多重 DOループでのループ交換”は、多重DOループにおいて、DOループが交換されてループスライスされた例を示しています。DO変数Jについて並列化されますが、Jを外側のループにすることで、並列処理制御の回数を少なくすることができます。

図12.6 多重 DOループでのループ交換





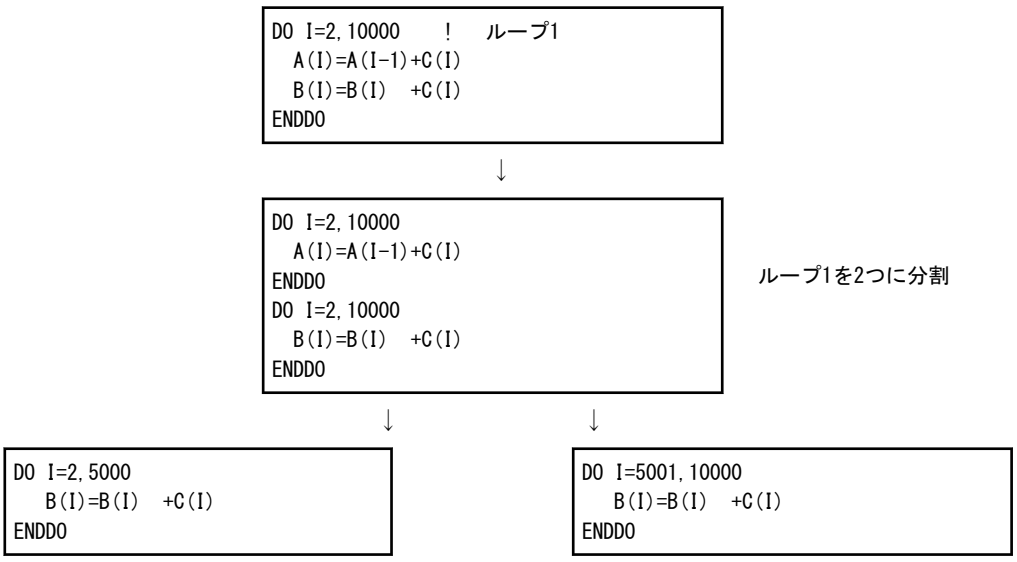
12.2.5.6 ループ分割と自動ループスライス

“図12.7 DOループの分割”のDOループでは、配列Aは、ループスライスを行うとデータの定義引用順序が変わるため、ループスライスすることができません。配列Bは、データの定義引用順序が変わらないので、ループスライスできます。このような場合、配列Aを定義している文と配列Bを定義している文を2つのDOループに分割して、配列Bを定義するDOループの方をループスライスします。

このようなループ分割による部分的な自動並列化は、-Kloop_part_parallelオプションが有効、かつ、ループスライスの効果が十分見込まれるとコンパイラが判断した場合に適用されます。

下図に、DOループが分割されてループスライスされた例を示します。

図12.7 DOループの分割

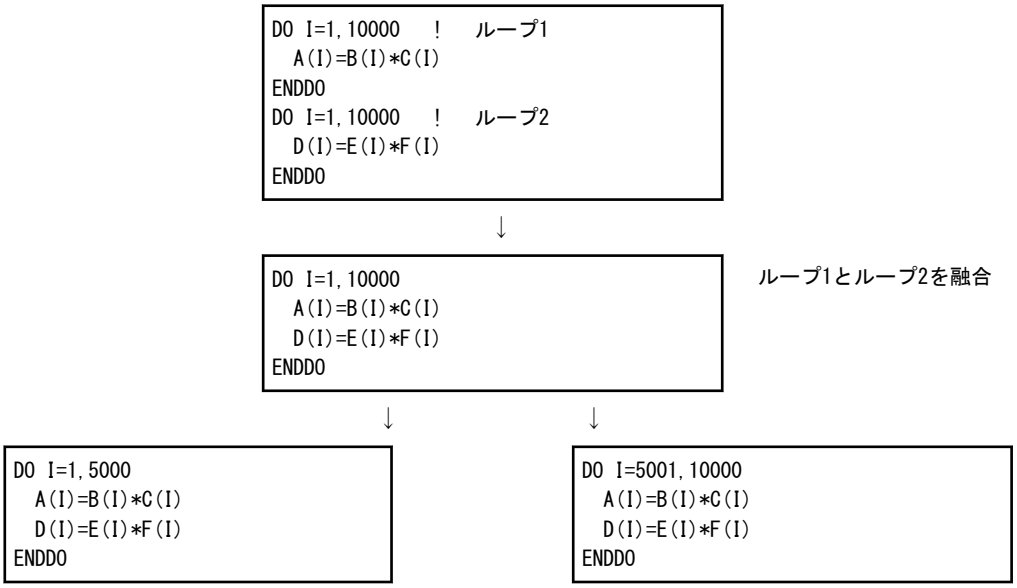


12.2.5.7 ループ融合と自動ループスライス

“図12.8 DOループの融合”では、ループの繰返し数が同じDOループが2つ連続しています。このような場合、ループを1つに融合することで、ループ制御のオーバヘッドを軽減するとともに、並列処理制御の回数を減らすことができます。

下図に、DOループを融合してループスライスされた例を示します。

図12.8 DOループの融合



12.2.5.8 リダクションによるループスライス

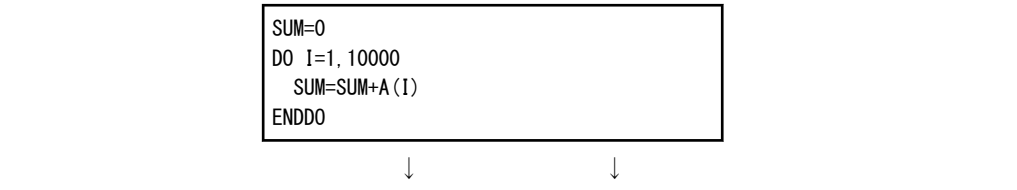
翻訳時オプションに、-Kparallelと-Kreductionを同時に指定したときに自動並列化される場合があります。この並列化によって実行結果に計算誤差を生じることがありますが、加算および乗算などの交換可能な演算の順序を変更してループスライスを行います。

リダクションの対象になるのは、DOループ内に以下の演算が含まれる場合です。

- 総和を求める演算がある場合
  例: S=S+A(I)
- 総積を求める演算がある場合
  例: P=P*A(I)
- 内積を求める演算がある場合
  例: P=P+A(I)*B(I)
- 最小値を求める演算がある場合
  例: X=MIN(X,A(I))
- 最大値を求める演算がある場合
  例: Y=MAX(Y,A(I))
- 論理和を求める演算がある場合
  例: N=N.OR.A(I)
- 論理積を求める演算がある場合
  例: M=M.AND.A(I)

下図に、リダクションによるループスライスの例を示します。

図12.9 リダクションによる自動ループスライス



```
SUM1=0
DO I=1,5000
  SUM1=SUM1+A(I)
ENDDO
```

```
SUM2=0
DO I=5001,10000
  SUM2=SUM2+A(I)
ENDDO
```

```
SUM=SUM+SUM1+SUM2
```

12.2.5.9 ループスライスされないDOループ

以下に示すDOループはループスライスの対象となりません。

- 並列実行しても実行時間が短縮されないと予想される場合
- ループスライスの対象とならない型の演算を含む場合
- 外部手続、内部手続またはモジュール手続の引用を含む場合
- DOループの形が複雑な場合
- 入出力文または組込みサブルーチンを含む場合
- データの定義引用順序が逐次実行のときと変わるおそれが有る場合

並列実行しても実行時間が短縮されないと予想される場合

DOループの繰返し数が小さいまたはDOループ内の演算数が少ない場合は、並列実行に伴うオーバヘッドのため、ループスライスを行うと、逐次処理した場合よりも性能が低下することがあります。そのため、性能が向上しないと予想されるDOループについては、本処理系はループスライスを行いません。

以下に、繰返し数が小さく、演算数が少ないDOループの例を示します。

図12.10 繰返し数が小さく、演算数が少ないDOループ

```
DO I=1,10          ! 繰返し数が小さく、かつ、演算数も少ないため
  A(I)=A(I)+B(I)   ! ループスライスの対象とはなりません。
ENDDO
```

ループスライスの対象とならない型の演算を含む場合

以下の型の演算を含むDOループはループスライスの対象となりません。内部DOループに以下の型の演算が含まれているDOループも同様です。

- 文字型
- 派生型

外部手続、内部手続またはモジュール手続の引用を含む場合

外部手続、内部手続またはモジュール手続の引用を含むDOループはループスライスの対象となりません。内部DOループに、外部手続、内部手続またはモジュール手続の引用が含まれているDOループも同様です。

ただし、最適化制御行によって、並列化を促進することができます。詳細については、“12.2.6 最適化制御行”を参照してください。

また、外部手続または内部手続がインライン展開された場合は、並列化の対象になります。

以下に、サブルーチンの引用を含むDOループの例を示します。

図12.11 サブルーチンの引用を含むDOループ

```
DO J=1,10                          …サブルーチンの引用が含まれているので、
  DO I=1,10000                       ループスライスの対象とはなりません。
    A(I,J)= A(I,J) +B(I,J)
    CALL SUB(A)
  ENDDO
ENDDO
```

DOループの形が複雑な場合

以下に示すDOループは、形が複雑なため、ループスライスの対象となりません。

- DOループの内側から外側へ飛出しがあるDOループ
- 複雑な構造のDOループ

以下に、飛出しのあるDOループの例を示します。

図12.12 飛出しのあるDOループ

```
      DO J=1,10                        …DOループ外へ飛出しがあるため、
        DO I=1,10000                     ループスライスの対象とはなりません。
          A(I,J)=A(I,J)+B(I,J)
          IF(A(I,J).LT.0)  GOTO 20
        ENDDO
      ENDDO
      ...
      ...
      ...
   20 CONTINUE
```

以下に、複雑な構造のDOループの例を示します。

図12.13 複雑な構造のDOループ

```
      DO J=1,10000                     …DOループの構造が複雑なため、
        IF(N>0) THEN                     ループスライスの対象とはなりません。
          ASSIGN 10 TO I
        ELSE IF(N<0) THEN
          ASSIGN 20 TO I
        ELSE
          ASSIGN 30 TO I
        ENDIF
        GOTO I,(10,20,30)
   10 A(J)=A(J)+0
   20 A(J)=A(J)+1
   30 A(J)=A(J)+2
      ENDDO
```

入出力文または組込みサブルーチンを含む場合

入出力文または組込みサブルーチンを含むDOループは、ループスライスの対象となりません。

組込み関数は、ループスライスの対象になるものとならないものがありますが、診断メッセージによって知ることができます。

データの定義引用順序が逐次実行のときと変わるおそれが有る場合

“図12.5 ループスライスができないDOループの例”の例で説明したように、データの定義引用順序が逐次実行のときと変わるDOループはループスライスの対象となりません。

12.2.5.10 自動並列化状況の表示

自動並列化が行われたかどうか、行われたとしたらどのように行われたかを知りたいときは、翻訳時オプションに-Kparallelと-Koptmsg=2を同時に指定してください。翻訳時の診断メッセージによって、知ることができます。

自動並列化が行われなかった理由も、同様に知ることができます。

以下に診断メッセージの例を示します。

例1: “図12.3 ループスライスのイメージ”のプログラムを翻訳したときのメッセージ

```
jwd5001p-i "test.f", line 2: このDOループを並列化しました。(名前:I)
```

例2: “図12.5 ループスライスができないDOループの例”のプログラムを翻訳したときのメッセージ

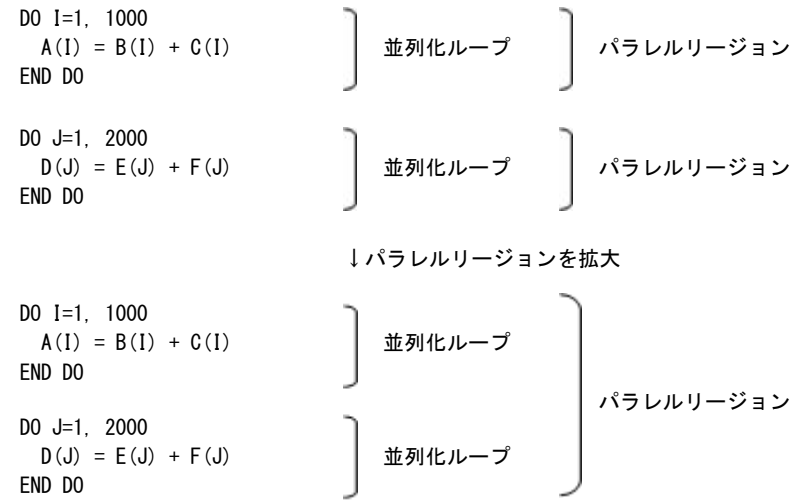
```
jwd5201p-i "test.f", line 2: データの引用の順序が逐次実行と変わるため、このDOループは並列化できません。(名前:A)
```

12.2.5.11 パラレルリージョンの拡大

パラレルリージョンとは、自動並列化のために複数のスレッドを生成してから解放するまでの範囲です。通常、このパラレルリージョンは、“図12.14 パラレルリージョンの拡大の例”のように、1つのパラレルリージョンに対して1つの並列化ループが生成されます。このとき、実行時にパラレルリージョンを生成するためのコストが、並列化オーバヘッドとなります。

この2つの並列化ループに対して、パラレルリージョンの拡張を行うことにより、1つのパラレルリージョンに対して2つの並列化ループが生成されます。その結果、パラレルリージョンを生成する回数が1回だけになり、並列化オーバヘッドコストを削減することができます。

図12.14 パラレルリージョンの拡大の例



上記以外にも、“図12.15 パラレルリージョンの拡大の例(多重ループ)”のようなループに対してもパラレルリージョンの拡大を行うことにより、並列化オーバヘッドを削減することができます。この場合、パラレルリージョンを生成する回数が1000回だったのに対して、最適化後は1回だけになります。

図12.15 パラレルリージョンの拡大の例(多重ループ)

```
DO I=1, 1000
  DO J=1, 2000
    A(J) = B(J) + C(J)
  END DO
END DO
```

並列化ループ

パラレルリージョン

↓パラレルリージョンを拡大

```
DO I=1, 1000
  DO J=1, 2000
    A(J) = B(J) + C(J)
  END DO
END DO
```

並列化ループ

パラレルリージョン

12.2.5.12 ブロック分割とサイクリック分割

自動並列化ループの分割方法には、ブロック分割とサイクリック分割があります。

ブロック分割とは、ループの繰返し数をスレッド数で分割したブロックを、各スレッドに割り当てる方式です。

サイクリック分割とは、ループの繰返し数を任意のサイズで分割したブロックを、各スレッドに順番に割り当てる方式です。自動並列化の分割方式のデフォルトは、ブロック分割です。最適化指示子PARALLEL_CYCLICが指定された場合、サイクリック分割します。

図12.16 ブロック分割とサイクリック分割

```
      SUBROUTINE SUB(A,B)
      INTEGER A(20,20),B(20,20)
      DO J=1,20
        DO I=J,20
          A(I,J) = B(I,J)
        ENDDO
      ENDDO
      END
```

"図12.16ブロック分割とサイクリック分割"に対してブロック分割またはサイクリック分割を適用した場合、ループの繰返し数は以下のように各スレッドに割り当てられます。

- ブロック分割の場合

  ループの繰返し数をスレッド数2で割ったブロックを、以下のように各スレッドに割り当てます。

  ```
  スレッド0: {1,2,3,4,5,6,7,8,9,10}
  スレッド1: {11,12,13,14,15,16,17,18,19,20}
  ```

- サイクリック分割のブロックサイズが2の場合

  ループの繰返し数を2回転(ブロックサイズ2)で分割したブロックを、以下のように各スレッドに順番に割り当てます。

  ```
  スレッド0: {1,2}, {5,6}, {9, 10}, {13,14}, {17,18}
  スレッド1: {3,4}, {7,8}, {11,12}, {15,16}, {19,20}
  ```

12.2.6 最適化制御行

本処理系には、自動並列化を促進するために、最適化制御行(OCL: Optimization Control Line)が用意されています。

自動並列化を促進する最適化制御行を有効にするには、翻訳時オプションに-Kparallelと-Koclを同時に指定してください。

12.2.6.1 最適化制御行の記法

最適化制御行は、行の第1けたから第5けたが"!OCL "である行です。!OCL に続いて、自動並列化用の最適化指示子を1個以上指定します。

以下に記法を示します。

```
!OCL i [, i] ....
```

i: 自動並列化用の最適化指示子

12.2.6.2 最適化制御行の記述位置

最適化制御行は、最適化指示子の種類によって記述できる位置が決まっています。

自動並列化のための最適化制御行は、total位置、loop位置、または配列代入位置に指定します。各位置は以下のように定義されます。

- total位置
  各プログラム単位の先頭。
- loop位置
  DO文の直前の位置。ただし、最適化制御行とDO文との間に注釈行またはloop位置に指定可能な他の最適化制御行を含んでもかまいません。
- 配列代入位置
  配列代入文の直前の位置。

例:

```
!OCL SERIAL ← total位置
  SUBROUTINE SUB(B,C,N)
  INTEGER A(N),B(N),C(N)
  DO I=1,N
    A(I) = B(I) + C(I)
  ENDDO
  PRINT *, FUN(A)
!OCL PARALLEL ← loop位置
  DO I=1,N
    A(I) = B(I) * C(I)
  ENDDO
  PRINT *, FUN(A)
!OCL SERIAL ← 配列代入位置
  A = 0
  PRINT *,FUNC(A)
  END SUBROUTINE
```

12.2.6.3 自動並列化と最適化指示子

自動並列化用の最適化指示子を指定した場合であっても、ループスライスの対象とならないDOループについては、最適化指示子の効果は無効になります。ループスライスの対象とならないDOループについては、“12.2.5.9 ループスライスされないDOループ”を参照してください。

12.2.6.4 自動並列化用の最適化指示子

最適化制御行は、指定される最適化指示子の種類によって、機能が異なります。

“表12.1 自動並列化用の最適化指示子一覧”を示します。自動並列化用の最適化指示子は、プログラマがコンパイラに自動並列化の参考になる情報を通知し、効率の良いオブジェクトを生成するために用いられます。

表12.1 自動並列化用の最適化指示子一覧

最適化指示子	意味概略	total位置	loop位置	文単位	配列代入文位置
ARRAY_PRIVATE	プライベート配列化機能を有効にします。	○	○	×	×
NOARRAY_PRIVATE	プライベート配列化機能を無効にします。	○	○	×	×
INDEPENDENT[(*e*[,*e*]...)]	手続(*e*)引用のあるDO ループのループスライスを指示します。	○	○	×	×
LOOP_PART_PARALLEL	ループを分割して部分的に自動並列化する機能を有効にします。	○	○	×	○
LOOP_NOPART_PARALLEL	ループを分割して部分的に自動並列化する機能を無効にします。	○	○	×	○
SERIAL	自動並列化機能を無効にします。	○	○	×	○
PARALLEL	自動並列化機能を有効にします。	○	○	×	○
PARALLEL_CYCLIC[(*n*)]	ブロックサイズ(*n*)でサイクリック分割することを指示します。	×	○	×	○
PARALLEL_STRONG	ループの繰返し数が小さい、または演算数が少ないために並列化されていないループを並列化させることを指示します。	○	○	×	○
REDUCTION	リダクション演算の並列化を有効にします。	○	○	×	○
NOREDUCTION	リダクション演算の並列化を無効にします。	○	○	×	○
TEMP[(*var*[,*var*]...)]	DO ループ内で一時的に使用している変数(*var*)を指示します。	○	○	×	×
TEMP_PRIVATE(*var*[,*var*]...)	変数(*var*)をスレッド内でローカルな領域に割り付けることを指示します。これにより、スレッド間のデータ依存が解消され、自動並列化が促進されます。	×	○	×	×
FIRST_PRIVATE(*var*[,*var*]...)	変数(*var*)をスレッド内でローカルな領域に割り付けることを指示します。また、変数の初期値の引用があることを指示します。これにより、スレッド間のデータ依存が解消され、自動並列化が促進されます。	×	○	×	×
LAST_PRIVATE(*var*[,*var*]...)	変数(*var*)をスレッド内でローカルな領域に割り付けることを指示します。また、ループ終了後に変数の引用があることを指示します。これにより、スレッド間のデータ依存が解消され、自動並列化が促進されます。	×	○	×	×
var1 op var2 または *var1 op cnst*	変数*var1*と変数*var2*の大小関係または変数*var1*と定数*cnst*の大小関係を指示します。	○	○	×	○

以下に各最適化指示子について説明します。

また、わかりやすさのため、本章の説明に使用する原始プログラムリストには、並列化される文に記号(p)、部分的に並列化される文に記号(m)および並列化されない文に記号(s)を表示して説明します。

ARRAY_PRIVATE

ARRAY_PRIVATE指示子は、ループ内のプライベート化可能な配列を認識し、プライベート化することにより並列化を促進させることを本処理系に指示します。

ARRAY_PRIVATE指示子は、以下の形式です。

```
!OCL ARRAY_PRIVATE
```

ARRAY_PRIVATE指示子は、loop位置およびtotal位置に記述することができます。

ARRAY_PRIVATEの効果は、記述位置によって以下の違いがあります。

- loop位置に記述された場合
  対応するDOループに対して有効となります。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループに対して有効となります。

“図12.17 ARRAY_PRIVATE指示子の使用例”の場合、ARRAY_PRIVATE指示子を指定することにより、翻訳時オプション-Karray_privateの指定がない場合でも、プライベート化可能な配列 Aを認識し、プライベート化することにより並列化を促進することができます。

図12.17 ARRAY_PRIVATE指示子の使用例

```
      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 SUBROUTINE
```

NOARRAY_PRIVATE

NOARRAY_PRIVATE指示子は、ループ内のプライベート化可能な配列に対してプライベート化しないことを本処理系に指示します。

NOARRAY_PRIVATE指示子は、以下の形式です。

```
!OCL NOARRAY_PRIVATE
```

NOARRAY_PRIVATE指示子は、loop位置およびtotal位置に記述することができます。

NOARRAY_PRIVATEの効果は、記述位置によって以下の違いがあります。

- loop位置に記述された場合
  対応するDOループに対して有効となります。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループに対して有効となります。

“図12.18 NOARRAY_PRIVATE指示子の使用例”の場合、NOARRAY_PRIVATE指示子を指定することにより、翻訳時オプション-Karray_privateの指定されている場合でもプライベート化可能な配列Aは、プライベート化されません。

図12.18 NOARRAY_PRIVATE指示子の使用例

```
      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
      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
      ENDDO
      PRINT*,A,B
      END SUBROUTINE
```

INDEPENDENT[(*e*[,*e*]...)]

INDEPENDENT指示子は、DOループ内の手続を引用しても、逐次実行のときと動作が変わらないことを本処理系に指示します。これにより、手続引用のあるDOループをループスライスの対象にします。

INDEPENDENT指示子は、以下の形式です。

```
!OCL INDEPENDENT[ (e[,e]...) ]
```

ここで、*e*はループスライスに影響しない手続名です。*e*にはワイルドカード指定も可能です。また、手続名を省略すると、対象範囲内のすべての手続に有効となります。ワイルドカード指定の方法については、“12.2.6.5 ワイルドカード指定”を参照してください。

手続*e*は、翻訳時オプション-Kthreadsafeおよび-Kparallelを指定して翻訳しなければなりません。

INDEPENDENT指示子は、loop位置およびtotal位置に記述することができます。

INDEPENDENTの効果は、記述位置によって以下の違いがあります。

- loop位置に記述された場合
  対応するDOループに対して有効となります。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループに対して有効となります。

“図12.19 INDEPENDENT指定のないDOループの例”は、手続FUNを引用しているため、本処理系は、DOループがループスライス可能であるかどうか判断できません。

図12.19 INDEPENDENT指定のないDOループの例

```
REAL FUN,A(10000)
DO I=1,10000
  J=I
  A(I)=FUN(J)
ENDDO
  ...
FUNCTION FUN(J)
INTEGER J
REAL FUN
FUN=SQRT(REAL(J**2+3*J+6))
END
```

もし手続 FUNの引用のあるDOループをループスライスしても、実行結果に影響を与えないと分かっているのであれば、“図12.20 INDEPENDENT指示子の使用例”のようにINDEPENDENTを使用することにより、このDOループはループスライスされます。

図12.20 INDEPENDENT指示子の使用例

```
       REAL FUN,A(10000)
 !OCL INDEPENDENT(FUN)
p      DO I=1,10000
p        J=I
p        A(I)=FUN(J)
p      ENDDO
       ...
       RECURSIVE FUNCTION FUN(J)
       INTEGER J
       REAL FUN
       FUN=SQRT(REAL(J**2+3*J+6))
       END
```

[注意事項]

INDEPENDENT指示子に、ループスライス不可能な手続を誤って指定した場合、本処理系は、誤ったループスライスを行うことがあります。なお、ループスライス不可能な手続の例として、以下のようなループがあります。

- 手続間で依存関係がある手続

LOOP_PART_PARALLEL

LOOP_PART_PARALLEL指示子は、ループを分割して部分的に自動並列化することを本処理系に指示するために使用します。

LOOP_PART_PARALLEL指示子は、以下の形式です。

```
!OCL LOOP_PART_PARALLEL
```

LOOP_PART_PARALLEL指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

LOOP_PART_PARALLELの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループと、その内側のDOループを対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループを対象とします。

“図12.21 LOOP_PART_PARALLEL指示子の使用例”の場合、LOOP_PART_PARALLEL指示子を指定することにより、翻訳時オプション-Kloop_part_parallelの指定がない場合でも、ループを分割して部分的に並列化できます。

図12.21 LOOP_PART_PARALLEL指示子の使用例

```
       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))  ! 自動並列化可能
s        D(I) = D(I-1)+A(I)          ! 自動並列化不可能
p      ENDDO
```

LOOP_NOPART_PARALLEL

LOOP_NOPART_PARALLEL指示子は、部分的に自動並列化しないことを本処理系に指示するために使用します。

LOOP_NOPART_PARALLEL指示子は、以下の形式です。

```
!OCL LOOP_NOPART_PARALLEL
```

LOOP_NOPART_PARALLEL指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

LOOP_NOPART_PARALLELの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループと、その内側のDOループを対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループを対象とします。

“図12.22 LOOP_NOPART_PARALLEL指示子の使用例”の場合、LOOP_NOPART_PARALLEL指示子を指定することにより、対象とするループは部分的に並列化されません。

図12.22 LOOP_NOPART_PARALLEL指示子の使用例

```
      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

SERIAL指示子は、DOループのループスライスを抑止する場合に使用します。

例えば、あるDOループは、逐次実行させた方が速いとわかっている場合に使用します。

SERIAL指示子は、次の形式です。

```
!OCL SERIAL
```

SERIAL指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

SERIALの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文に対するループスライスを抑止します。
- loop位置に記述された場合
  対応するDOループと、その内側のDOループに対するループスライスを抑止します。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループに対するループスライスを抑止します。

“図12.23 SERIAL指定のないプログラムの例 ”のプログラムで、ループをループスライスしたくない場合、“図12.24 SERIAL指示子の使用例”のようにSERIALを指定することにより、ループのループスライスを止めることができます。

図12.23 SERIAL指定のないプログラムの例

```
      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           ENDDO
p         ENDDO
          END
```

図12.24 SERIAL指示子の使用例

```
      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)
         ENDDO
      ENDDO
      END
```

PARALLEL

PARALLEL指示子は、SERIALの効果を打ち消して、一部のDOループだけをループスライスの対象とする場合に使用します。

PARALLEL指示子は、次の形式です。

```
!OCL PARALLEL
```

PARALLEL指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

PARALLELの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文をループスライスの対象とします。
- loop位置に記述された場合
  対応するDOループと、その内側のDOループをループスライスの対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループをループスライスの対象とします。

“図12.25 PARALLEL指定のないプログラムの例”のプログラムで、ループ2のDOループだけをループスライスの対象にしたい場合、“図12.26 PARALLEL指示子の使用例”のようにSERIALとPARALLELを併用することにより、ループ2のDOループだけをループスライスの対象にすることができます。

図12.25 PARALLEL指定のないプログラムの例

```
      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                    ! ループ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                    ! ループ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                    ! ループ3
p         DO I=1,100
p           A3(I,J) = A3(I,J) + B3(I,J)
p         ENDDO
p       ENDDO

        RETURN
        END
```

図12.26 PARALLEL指示子の使用例

```
 !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                    ! ループ1
          DO I=1,100
            A1(I,J) = A1(I,J) + B1(I,J)
          ENDDO
        ENDDO

 !OCL PARALLEL
p       DO J=1,N2                    ! ループ2
p         DO I=1,100
p           A2(I,J) = A2(I,J) + B2(I,J)
p         ENDDO
p       ENDDO

        DO J=1,N3                    ! ループ3
          DO I=1,100
            A3(I,J) = A3(I,J) + B3(I,J)
          ENDDO
        ENDDO

        RETURN
        END
```

PARALLEL_CYCLIC[(*n*)]

PARALLEL_CYCLIC指示子は、コンパイラが自動並列化可能と解析したDOループに対してサイクリック分割を行うことを本処理系に指示するために使用します。本最適化指示子が適用された場合、並列効果の見積もりは行わずに並列実行します。

サイクリック分割については、“12.2.5.12 ブロック分割とサイクリック分割”を参照してください。

PARALLEL_CYCLIC指示子は、次の形式です。

```
!OCL PARALLEL_CYCLIC[(n)]
```

ここで、*n*はサイクリック分割するブロックのサイズです。*n*には1～10000の値が指定できます。省略値は1です。

PARALLEL_CYCLIC指示子は、配列代入位置、およびloop位置に記述することができます。

PARALLEL_CYCLICの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループを対象とします。

“図12.27 PARALLEL_CYCLIC指示子の使用例”の場合、DOループはブロックサイズを2とするサイクリック分割が行われます。

図12.27 PARALLEL_CYCLIC指示子の使用例

```
        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

PARALLEL_STRONG指示子は、ループの繰返し数が小さい、または演算数が少ないために並列化されていないループを並列化させることを指示するために使用します。

PARALLEL_STRONG指示子は、以下の形式です。

```
!OCL PARALLEL_STRONG
```

PARALLEL_STRONG指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

PARALLEL_STRONGの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループとその内側DOループを対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループを対象とします。

“図12.28 PARALLEL_STRONG指示子の使用例”の場合、ループの繰返し数が小さいことによる性能向上が得られないため並列化が抑止されますが、PARALLEL_STRONG指示子を指定することにより、並列化することができます。

図12.28 PARALLEL_STRONG指示子の使用例

```
      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      ENDDO
```

```
      PRINT*,A
      END
```

REDUCTION

REDUCTION指示子は、リダクション演算が含まれるDOループを並列化させることを本処理系に指示するために使用します。

REDUCTION指示子は、以下の形式です。

```
!OCL REDUCTION
```

REDUCTION指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

REDUCTIONの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループとその内側DOループを対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループを対象とします。

“図12.29 REDUCTION指示子の使用例”の場合、REDUCTION指示子を指定することにより、翻訳時オプション-Kreductionの指定がない場合でも対象とするリダクション演算を含むDOループを並列化することができます。

図12.29 REDUCTION指示子の使用例

```
      REAL S,A(5000)
 !OCL REDUCTION
p     DO I=1,5000
p       S=S+A(I)
p     ENDDO
      END
```

NOREDUCTION

NOREDUCTION指示子は、リダクション演算が含まれるDOループを並列化させないことを本処理系に指示するために使用します。

NOREDUCTION指示子は、以下の形式です。

```
!OCL NOREDUCTION
```

NOREDUCTION指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

NOREDUCTIONの効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループとその内側DOループを対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループを対象とします。

“図12.30 NOREDUCTION指示子の使用例”の場合、NOREDUCTION指示子を指定することにより、対象とするリダクション演算を含むDOループの並列化を抑止することができます。

図12.30 NOREDUCTION指示子の使用例

```
        REAL S,A(5000)
 !OCL NOREDUCTION
s       DO I=1,5000
s         S=S+A(I)
s       ENDDO
        END
```

TEMP[(*var*[,*var*]...)]

TEMP指示子は、DOループ内で引用されている変数が、そのDOループの中で一時的に使用されていることを本処理系に指示するために使用します。これにより、並列化したDOループの実行性能を向上させることができます。

TEMP指示子は、以下の形式です。

```
!OCL TEMP[ (var[,var]...) ]
```

ここで、*var*はDOループ内で一時的に使用している変数名です。*var*にはワイルドカード指定も可能です。また、変数名を省略すると、対象範囲内のすべての変数名に有効となります。ワイルドカード指定については、“12.2.6.5 ワイルドカード指定”を参照してください。

TEMP指示子は、loop位置およびtotal位置に記述することができます。

TEMPの効果は、記述位置によって以下の違いがあります。

- loop位置に記述された場合
  対応するDOループ内の指定された変数が、一時的な変数であることを示します。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループ内の指定された変数が、一時的な変数であることを示します。

“図12.31 TEMP指定のないプログラムの例”の場合、変数TがDOループの中でしか使用されていなくても、Tが共通ブロック要素のため、本処理系は、TがサブルーチンSUBの中で参照されていると判断し、Tの値が正しくなるようなループスライスを行います。

図12.31 TEMP指定のないプログラムの例

```
        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         ENDDO
p       ENDDO
        ...
        ...
        ...
        CALL SUB
        END
```

この場合、DOループの終了時のTの値が、サブルーチンSUBの中で参照されないことが分かるのであれば、“図12.32 TEMP指示子の使用例”のようにTEMPでTを指定します。これにより、DOループの終了時のTの値を正しくする命令が不要になるので、実行性能が向上します。

図12.32 TEMP指示子の使用例

```
          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            ENDDO
p          ENDDO
          ...
          ...
          ...
          CALL SUB
          END
```

[注意事項]

TEMP指示子に、一時的に使用されている変数以外の変数を誤って指定した場合、本処理系は、誤ったループスライスを行うことがあります。

TEMP_PRIVATE(*var*[,*var*]...)

TEMP_PRIVATE指示子は、自動並列化の対象となっているループ内において、指定された変数を各スレッドでローカル変数として扱うことを本処理系に指示するために使用します。

なお、以下のような最適化メッセージが出力された場合、本指示子で対象データを各スレッドでローカルな領域に割り付けることにより、自動並列化の阻害要因が解消され、該当ループの自動並列化が促進されます。

```
jwd5208p-i "file", line n: 定義引用の順序が分からないため、定義引用順序が逐次実行と変わる可能性があり、このDOループは並列化できません。(名前:var)
```

TEMP_PRIVATE指示子は、以下の形式です。

```
!OCL TEMP_PRIVATE(var[,var]...)
```

*var*にはワイルドカード指定も可能です。ワイルドカード指定の方法については、“12.2.6.5 ワイルドカード指定”を参照してください。

TEMP_PRIVATE指示子は、loop位置に記述することができます。なお、この最適化制御行は指定した直後のDOループのみ対象となります。

本指示子に指定できる変数は以下のとおりです。

- 整数型、実数型または論理型のスカラ変数
- 以下の条件を満たす配列変数
  - 整数型、実数型または論理型である。かつ、
  - 配列の大きさが翻訳時に確定している。

本指示子が適用された場合、以下の最適化メッセージが出力されます。

```
jwd5013p-i "file", line n: 最適化指示子TEMP_PRIVATEを変数'var'に適用しました。
```

本指示子を使用する上での注意事項ついては、“12.2.7.4 最適化制御行の使い方の注意”を参照してください。

“図12.33 TEMP_PRIVATE指定のないDOループのプログラム例”は、外側ループでは配列Aの定義引用順序が不明のため、並列化ができません。

図12.33 TEMP_PRIVATE指定のないDOループのプログラム例

```
      INTEGER A(100), B(100), C(100,100)
      INTEGER M,N
      DO J=1,100                  ! 自動並列化の対象となるループ
        DO I=1,M                  ! Mの値が不明のため、配列Aの定義される範囲が不明
          A(I) = B(I)
        ENDDO
        DO I=1,N                  ! Nの値が不明のため、配列Aの参照される範囲が不明
          C(I,J) = A(I)
        ENDDO
      ENDDO
      PRINT*, C
      END
```

“図12.34 TEMP_PRIVATE指示子の使用例”のようにTEMP_PRIVATE指示子を指定し、配列Aの依存関係を明確にすることにより、外側ループで並列化することができます。

図12.34 TEMP_PRIVATE指示子の使用例

```
 !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*]...)

FIRST_PRIVATE指示子は、自動並列化の対象となっているループ内において、指定された変数を各スレッドでローカル変数として扱い、また、スレッドごとに変数の初期値の引用があることを本処理系に指示するために使用します。

なお、以下のような最適化メッセージが出力された場合、本指示子で対象データを各スレッドでローカルな領域に割り付けることにより、自動並列化の阻害要因が解消され、該当ループの自動並列化が促進されます。

```
jwd5208p-i "file", line n: 定義引用の順序が分からないため、定義引用順序が逐次実行と変わる可能性があり、このDOループは並列化できません。(名前:var)
```

FIRST_PRIVATE指示子は、以下の形式です。

```
!OCL FIRST_PRIVATE(var[,var]...)
```

*var*にはワイルドカード指定も可能です。ワイルドカード指定の方法については、“12.2.6.5 ワイルドカード指定”を参照してください。

FIRST_PRIVATE指示子は、loop位置に記述することができます。なお、この最適化制御行は指定した直後のDOループのみ対象となります。

本指示子に指定できる変数は以下のとおりです。

- 整数型、実数型または論理型のスカラ変数
- 以下の条件を満たす配列変数
  - 整数型、実数型または論理型である。かつ、

- 配列の大きさが翻訳時に確定している。

本指示子が適用された場合、以下の最適化メッセージが出力されます。

```
jwd5014p-i "file", line n: 最適化指示子FIRST_PRIVATEを変数'var'に適用しました。
```

本指示子を使用する上での注意事項ついては、"12.2.7.4 最適化制御行の使い方の注意"を参照してください。

"図12.35 FIRST_PRIVATE指定のないDOループのプログラム例"は、配列Aの定義引用順序が不明のため、本処理系では、外側ループでは並列化ができません。

図12.35 FIRST_PRIVATE指定のないDOループのプログラム例

```
      INTEGER A(100), B(100,100), C(100,100), D(100)
      INTEGER M,N
      DO J=1,100              ! 自動並列化の対象となるループ
        DO I=1,M              ! Mの値が不明のため、配列Aの参照される範囲が不明
          B(I,J) = A(I)
        ENDDO
        DO I=N,100            ! Nの値が不明のため、配列Aの定義される範囲が不明
          A(I) = B(I,J) + D(I)
          C(I,J) = A(I)
        ENDDO
      ENDDO
      PRINT*, C
      END
```

"図12.36 FIRST_PRIVATE指示子の使用例"のようにFIRST_PRIVATE指示子を指定し、配列Aの依存関係を明確にすることにより、外側ループで並列化することができます。

図12.36 FIRST_PRIVATE指示子の使用例

```
 !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*]...)

LAST_PRIVATE指示子は、自動並列化の対象となっているループ内において、指定された変数を各スレッドでローカル変数として扱い、また、最終スレッドで定義されたローカル変数の値を引用することを本処理系に指示するために使用します。

なお、以下のような最適化メッセージが出力された場合、本指示子で対象データを各スレッドでローカルな領域に割り付けることにより、自動並列化の阻害要因が解消され、該当ループの自動並列化が促進されます。

```
jwd5208p-i "file", line n: 定義引用の順序が分からないため、定義引用順序が逐次実行と変わる可能性があり、このDOループは並列化できません。(名前:var)
```

LAST_PRIVATE指示子は、以下の形式です。

```
!OCL LAST_PRIVATE(var[,var]...)
```

varにはワイルドカード指定も可能です。ワイルドカード指定の方法については、“12.2.6.5 ワイルドカード指定”を参照してください。

LAST_PRIVATE指示子は、loop位置に記述することができます。なお、この最適化制御行は指定した直後のDOループのみ対象となります。

本指示子に指定できる変数は以下のとおりです。

- 整数型、実数型または論理型のスカラ変数
- 以下の条件を満たす配列変数
  - 整数型、実数型または論理型である。かつ、
  - 配列の大きさが翻訳時に確定している。

本指示子が適用された場合、以下の最適化メッセージが出力されます。

```
jwd5015p-i "file", line n: 最適化指示子LAST_PRIVATEを変数'var'に適用しました。
```

本指示子を使用する上での注意事項ついては、“12.2.7.4 最適化制御行の使い方の注意”を参照してください。

“図12.37 LAST_PRIVATE指定のないDOループのプログラム例”は、配列Aの定義引用順序が不明のため、本処理系では、外側ループでは並列化ができません。

図12.37 LAST_PRIVATE指定のないDOループのプログラム例

```
      INTEGER A(100), B(100,100), C(100,100), D(100)
      INTEGER M,N
      DO J=1,100              ! 自動並列化の対象となるループ
        DO I=1,M
          B(I,J) = I
        ENDDO
        DO I=N,100            ! Nの値が不明のため、配列Aの定義される範囲が不明
          A(I) = B(I,J) + D(I)
          C(I,J) = A(I)
        ENDDO
      ENDDO
      PRINT *, A, C
      END
```

“図12.38 LAST_PRIVATE指示子の使用例”のようにLAST_PRIVATE指示子を指定し、配列Aの依存関係を明確にすることにより、外側ループで並列化することができます。

図12.38 LAST_PRIVATE指示子の使用例

```
 !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
        PRINT*, A, C
```

var1 op var2

var1 op cnst

この指示子は、変数と変数の大小関係または変数と定数の大小関係を本処理系に指示するために使用します。

ここで*var1*および*var2*は、変数名です。*cnst*は、整定数または名前付き定数です。以下の形式です。

```
!OCL var1 op var2
!OCL var1 op cnst
```

*op*に指定できる関係演算子を以下に示します。

演算子	意味
.LT. および <	小さい
.LE. および <=	小さいか等しい
.EQ. および ==	等しい
.NE. および /=	等しくない
.GT. および >	大きい
.GE. および >=	大きいか等しい

*var1 op var2*または*var1 op cnst*指示子は、配列代入位置、loop位置、およびtotal位置に記述することができます。

*var1 op var2*または*var1 op cnst*指示子の効果は、記述位置によって以下の違いがあります。

- 配列代入位置に記述された場合
  対応する配列代入文を対象とします。
- loop位置に記述された場合
  対応するDOループとその内側DOループを対象とします。
- total位置に記述された場合
  対応するプログラム単位内のすべてのDOループを対象とします。

“図12.39 変数名op整定数指示子の使用例”の場合、最適化制御行が記述されていないと配列要素の添字の値の大小関係が分からないため、配列Aの定義引用順序が逐次実行のときと変わる可能性があり並列化されません。この場合、変数名*op*整定数指示子を指定することにより、配列Aの定義引用順序が逐次実行のときと変わらないと分かるので並列化することができます。

図12.39 変数名op整定数指示子の使用例

```
      REAL,DIMENSION(10000)::A,B
 !OCL M.GT.2000
p     DO I=1,2000
p       A(I)=A(I+M)+B(I)
p     ENDDO
      END
```

12.2.6.5 ワイルドカード指定

以下の最適化指示子のオペランドには、変数名または手続名としてワイルドカード指定が可能です。

```
INDEPENDENT
TEMP
```

ワイルドカード指定は、ワイルドカードと呼ばれる文字と英数字の組合せです。ワイルドカード指定を用いると、ワイルドカード指定によって表現された文字列に合致する変数名または手続名を一括して指定するのと同じ効果があります。

ワイルドカードには、“*”と“?”の2種類があり、それぞれ以下のような文字列に合致します。

```
*: 任意の英数字から構成される、長さ1以上の文字列に合致する。
?: 任意の英数字1文字に合致する。
```

なお、1つのワイルドカード指定の中に、2つ以上のワイルドカードを含むことはできません。

```
!OCL TEMP (W*)
```

この例で、"W*"は、先頭の文字がWであり、長さが2以上の任意の変数名を表します。

例えば、WORK1、W2、WORK3という変数名は、この指定の中に含まれます。

```
!OCL INDEPENDENT (SUB?)
```

この例で、"SUB?"は、先頭の3文字がSUBであり、長さが4の任意の手続名を表します。

例えば、SUB1、SUB2、SUBXという配列名は、この指定の中に含まれます。

12.2.7 自動並列化機能を使うときの留意事項

本処理系の自動並列化機能を使用する場合の留意事項について説明します。

12.2.7.1 翻訳時オプション-Kparallel,instance=N指定時の注意

翻訳時オプション-Kparallel,instance=*N*で、並列動作のスレッド数を指定する場合、実行時のスレッド数と同じ値でなければなりません。並列動作のスレッド数については、“12.2.2.1 スレッド数”を参照してください。

-Kinstanceに誤った値を指定した場合、以下の実行時メッセージを出力し、実行を打ち切ります。

```
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 並列処理の入れ子での注意

ループスライスされたDOループ内で手続を引用していて、引用先の手続中にループスライスされるDOループを含む場合、結果的に並列実行部分が入れ子になります。このような場合、内側のDOループは、逐次に実行されます。このようなDOループを含む原始プログラムを、-Kparallel,instance=*N*を指定して翻訳してはいけません。

“図12.40 並列処理が入れ子になった場合”に、スライスされたDOループが逐次に実行される例を示します。このようなDOループを含む原始プログラムを-Kparallel,instance=*N*を指定して翻訳すると、実行結果は保証されません。

図12.40 並列処理が入れ子になった場合

ファイル: a.f

```
!OCL INDEPENDENT
      DO I=1,100      ! このループは並列実行する
        J=I
        CALL SUB(J)
      ENDDO
      END

      RECURSIVE SUBROUTINE SUB(N)
      ...
      DO I=1,10000      ! このループは逐次実行する
        A(I)=1/B(I)**N
      ...
      ENDDO
      END
```

a.fの原始プログラムを以下のように翻訳した場合、実行結果は保証できません。

```
$ frtpx -Kparallel,instance=4 -Nlibomp a.f （誤った使い方）
```

このような誤りを防ぐために、ループスライスされるDOループ内で引用している手続に、最適化制御行“!OCL SERIAL”を以下のように指定します。

```
!OCL SERIAL
      RECURSIVE SUBROUTINE SUB(N)
      ...
      DO I=1,10000
        A(I)=1/B(I)**N
      ENDDO
      ...
      END
```

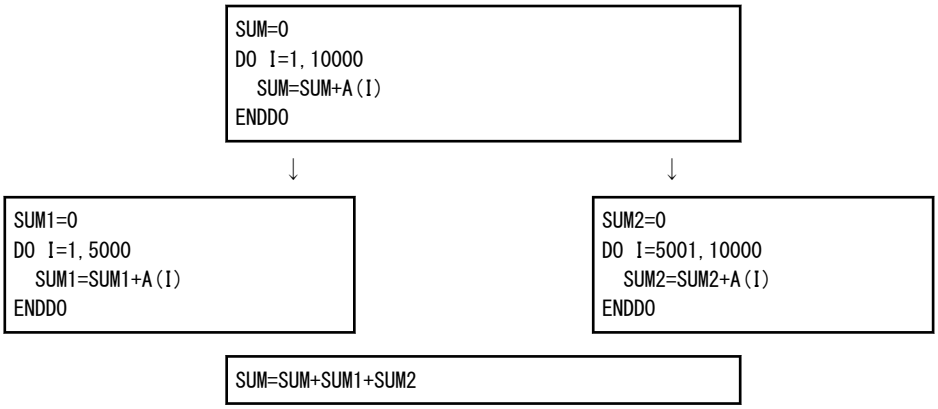
12.2.7.3 翻訳時オプション-Kparallel,reduction指定時の注意

翻訳時オプション-Kparallel,reductionを指定して翻訳したとき、並列実行の結果が、逐次実行の結果と異なることがあります。これは、並列実行のときの演算順序が、逐次実行の演算順序と異なる場合があるためです。

“図12.41リダクションによる精度差”は、“12.2.5.8リダクションによるループスライス”の説明で使用した例です。変数SUMは、逐次実行では、配列要素A(1)からA(10000)まで順次に加算されます。並列実行では、配列要素A(1)からA(5000)まで順次に加算した値を変数SUM1として、配列要素A(5001)からA(10000)まで加算した値をSUM2とします。そして、SUM1とSUM2を加算した値をSUMとします。

つまり、逐次実行と並列実行では、配列Aの要素の値を累計する順番が違うので、演算結果に精度差が生じることがあります。

図12.41 リダクションによる精度差



12.2.7.4 最適化制御行の使い方の注意

最適化指示子の誤った使い方を以下に示します。

本処理系は、最適化指示子の指示どおりに、誤ったループスライスを行う場合があります。

- 最適化指示子TEMP

  下のプログラムは、変数Tに対して、誤ってTEMPを指定しています。DOループの実行後、変数Tの値は保証されないので、変数LASTに期待した値が代入されないことがあります。

```
!OCL TEMP(T)
      DO I=1,1000
        T=A(I)+B(I)
```

```
        C(I)=T+D(I)
      ENDDO
      LAST=T
```

・ 最適化指示子INDEPENDENT

下のプログラムは、サブルーチンSUBに対して、誤ってINDEPENDENTを指定しています。サブルーチンSUBの引用によって、配列Aの定義・引用の順序が変わるため、結果が正しくない場合があります。

```
      COMMON A(1000),B(1000)
!OCL INDEPENDENT(SUB)
      DO I=2,1000
        A(I)=B(I)+1.0
        CALL SUB(I-1)
      ENDDO
      ...
      END
      RECURSIVE SUBROUTINE SUB(J)
      COMMON A(1000)
      A(J)=A(J)+1.0
      END
```

・ 最適化指示子TEMP_PRIVATE

下のプログラムは、変数Aに対して、誤ったTEMP_PRIVATEを指定しています。各スレッドで不定な値を参照することになるため、実行結果は保証されません。

```
!OCL TEMP_PRIVATE(A)
      DO I=1,1000
        B(I) = A
      ENDDO
```

・ 最適化指示子LAST_PRIVATE

下のプログラムは、配列Aに対して、誤ったLAST_PRIVATEを指定しています。配列Aが最終スレッドで定義されない場合があるため、実行結果は保証されません。

```
!OCL LAST_PRIVATE(A)
      DO J=1,1000
        DO I=J,500
          A(I) = B(J)
        ENDDO
      ENDDO
      PRINT *, A
```

12.2.7.5 並列処理中の入出力文および組込み手続

ループスライスされたDOループ内で手続を引用していて、引用先の手続中に入出力文、組込みサブルーチンまたはループスライスの対象とならない組込み関数を含む場合、動作は保証されません。並列実行に伴うオーバヘッドのため、逐次処理した場合より実行性能が低下したり、入出力文の実行結果が逐次処理の場合と変わるおそれがあります。

“図12.42 並列処理中の入出力文”に、ループスライスされたDOループ内で引用された手続で、入出力文が実行される例を示します。

図12.42 並列処理中の入出力文

```
!OCL INDEPENDENT(SUB)
      DO I=1,10000
        J=I
        CALL SUB(J)
      ENDDO
```

```
...
END
RECURSIVE SUBROUTINE SUB(N)
...
PRINT*,N
...
END
```

12.3 OpenMP仕様による並列化

ここでは、本処理系で提供しているOpenMP仕様による並列化について説明します。OpenMP仕様 については、“OpenMP Architecture Review Board”のホームページを参照してください。

本処理系は、以下の仕様をサポートしています。ただし、デバイスに関連する構文、環境変数、およびランタイムライブラリルーチンはサポートしていません。

サポートする仕様
OpenMP 4.5
OpenMP 5.0の一部 (注)

注) 以下の機能を使用できます。

- TASK構文のIN_REDUCTION指示節
- TASKGROUP構文のTASK_REDUCTION指示節
- TASKLOOP構文のREDUCTION指示節およびIN_REDUCTION指示節

12.3.1 翻訳の方法

原始プログラムの翻訳およびリンクを行うには、翻訳コマンドに以下のオプションを指定します。指定するオプションは、OpenMP仕様による並列化を行うか否かで異なります。

OpenMP仕様による並列化	指定するオプション	
	翻訳時	リンク時
OpenMP仕様による並列化およびSIMD化を行う場合	-Kopenmp	-Kopenmp -Nlibomp
OpenMP仕様によるSIMD化だけを行い、並列化は行わない場合	-Kopenmp_simd	指定するオプションはありません

12.3.1.1 OpenMPプログラムのための翻訳オプション

ここでは、OpenMPプログラムを翻訳するためのオプションについて説明します。

-Kopenmp

-Kopenmpオプションは、OpenMP指示文を有効にし、原始プログラムを翻訳します。

-Kopenmpオプションは、リンク時にも指定する必要があります。-Kopenmpオプションを指定して翻訳されたオブジェクトプログラムが含まれる場合は、リンク時に-Kopenmpオプションを指定する必要があります。

 例

```
$ frtpx a.f90 -Kopenmp -c
$ frtpx a.o -Kopenmp -Nlibomp
```

-Kopenmp_simd

-Kopenmp_simdオプションは、OpenMP仕様のSIMD構文、DECLARE SIMD構文、DECLARE REDUCTION指示文、およびSIMD指示節をもつORDERED構文だけを有効にし、原始プログラムを翻訳します。

-Nlibomp

OpenMPライブラリとして、LLVM OpenMPライブラリを使用することを指示します。

-Nlibompオプションは、リンク時に指定する必要があります。

12.3.1.2 OpenMPプログラムの最適化情報を出力するための翻訳時オプション

ここでは、OpenMPプログラムの最適化情報を出力するための翻訳時オプションについて説明します。

-Nlst=p

翻訳時オプション-Nlst=pが指定されたとき、OpenMP仕様の指示文によって指示された実行文について、最適化情報の表示は以下のようになります。

- 並列に実行される可能性がある文については、pが表示されます。冗長に実行される文についてはこの表示はされません。DO指示文またはSECTIONS指示文に直接囲まれた実行文がこれに該当します。WORKSHARE指示文内では、並列化された文だけがpと表示されます。
- 同時に1スレッドだけで実行される文については、sが表示されます。MASTER指示文、SINGLE指示文、CRITICAL指示文またはORDERED指示文に直接囲まれた実行文がこれに該当します。ATOMIC指示文の直後の実行文は、その全体が排他的に1スレッドで実行されるとき、sが表示されます。
- 1つの実行文の中で、並列に実行される可能性がある部分と、同時に1スレッドだけで実行される部分が混在しているとき、mが表示されます。WORKSHARE指示文内のATOMIC指示文の直後の実行文などが、これに該当する場合があります。

ネストした並列実行では、その実行文を囲む最も内側のPARALLELリージョンに対する並列化情報を表示します。そのPARALLELリージョンがどのように呼ばれるか(並列実行中に呼ばれるか逐次実行中に呼ばれるか)は表示に影響を与えません。

-Koptmsg=2

OpenMP仕様のSIMD化やCOLLAPSEなど実行性能に関連する診断メッセージを出力します。

12.3.2 実行の方法

OpenMPプログラムを実行させる手続は、逐次実行のときと同じです。

12.3.2.1 実行時の環境変数

環境変数OMP_WAIT_POLICY

環境変数OMP_WAIT_POLICYで、同期待ち処理の動作を指定することができます。

ACTIVE	同期が獲得できるまでスピン待ちを行います。
PASSIVE	スピン待ちを行わず、サスペンド待ちを行います。

デフォルトは、PASSIVEです。

経過時間を重視する場合には、ACTIVEを選択してください。CPU時間を重視する場合は、PASSIVEを選択してください。

環境変数OMP_PROC_BIND

スレッドアフィニティを制御します。

TRUE、FALSE、またはカンマで区切ったMASTER、CLOSE、SPREADのリストのいずれかを設定します。リストの値は、ネストレベルに対応するPARALLELリージョンで使用するスレッドアフィニティポリシーを設定します。

デフォルトは、CLOSEです。

FALSEを設定した場合、スレッドアフィニティは無効になり、PARALLEL構文のPROC_BIND指示節も無視されます。

FALSE以外を設定した場合、スレッドアフィニティは有効になり、初期スレッドはプレースリストの最初のプレースにバインドされます。

MASTERを設定した場合、すべてのスレッドはマスタースレッドと同じプレースにバインドされます。

CLOSEを設定した場合、スレッドはマスタースレッドがバインドされているプレースの次から連続したプレースにバインドされます。

SPREADを設定した場合、マスタースレッドのプレースパーティションが分割され、スレッドは各サブパーティションの1つのプレースにバインドされます。

trueを指定した場合、SPREADを指定した場合と同様の効果になります。

プレースについては環境変数OMP_PLACESを参照してください。

環境変数OMP_PLACES

スレッドアフィニティを制御するプレースリストを定義します。

プレースとは、スレッドが実行されるプロセッサの順序付けされていない集合です。プレースリストとは、利用可能なプレースの順序付けされたリストです。

環境変数OMP_PLACESには、抽象名または明示的なプレースリストのどちらかを、以下の形式で設定します。

```
{ abstract-name[(num-places)] | place-list }
```

abstract-name

抽象名です。抽象名には以下を指定します。デフォルトは、CORESです。

抽象名	意味
THREADS	プレースは、1ハードウェアスレッド単位になります。 本処理系ではシステムのCPU資源の最小単位になりCORESと等価になります。
CORES	プレースは、1コア単位になります。 本処理系ではシステムのCPU資源の最小単位になりTHREADSと等価になります。
SOCKETS	プレースは、1ソケット単位になります。 本処理系ではNUMAノード単位になります。

num-places

抽象名形式のプレース数です。1以上の正の整数です。

place-list

明示的なプレースリストです。以下の形式で指定します。

```
{ place:length[:stride] | [!]place[,place-list] }  (注1)
```

place

“{”*place1*“}”(注2)

place1

{ *cpuid*:*length*[:*stride*] | [!]*cpuid*[,*place1*] }

length

インターバル指定の要素数です。1以上の正の整数です。

stride

インターバル指定の増分値です。1以上の正の整数で、デフォルト値は1です。

cpuid

システムのCPU資源の最小単位の識別番号です。

注1)排他演算子!は直後の指定を除外することを指示します。

注2)“{”および“}”は選択記号ではありません。半角中括弧を使用して指定することを意味します。

例

環境変数OMP_PLACESの指定例

- 例1:

```
$ export OMP_PLACES="CORES"
```

抽象名“CORES”のプレースリストを定義しています。

- 例2:

```
$ export OMP_PLACES="CORES(4)"
```

抽象名“CORES”の4プレースのプレースリストを定義しています。

- 例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"
```

上記3つの定義は、すべて等価です。

cpuid12～14、15～17、18～20、および21～23の同じ4つのプレースを定義しています。

環境変数OMP_STACKSIZE

利用者は、環境変数OMP_STACKSIZEの値として、スレッド毎のスタック領域の大きさをバイト、Kバイト、Mバイト、Gバイト、Tバイト単位で指定することができます。デフォルト値は8Mバイトです。

環境変数GOMP_CPU_AFFINITY

スレッドを指定されたcpuid順にCPUバインドします。

スレッド数が、指定したcpuidの数を超える場合は、先頭から繰り返して使用します。

各cpuidは、コンマ(',')または空白(' ')で区切る必要があります。

cpuidは、以下のような形式で増分値付き範囲指定ができます。

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: 範囲指定の最初のcpuid(0≦*cpuid1*＜CPU_SETSIZE)
cpuid2: 範囲指定の最後のcpuid(0≦*cpuid2*＜CPU_SETSIZE)
inc: 増分値(1≦*inc*＜CPU_SETSIZE)

なお以下の条件を満たす必要があります。

```
cpuid1≦cpuid2
```

*cpuid1*から*cpuid2*の範囲のcpuidを増分値*inc*ごとにすべてを指定した場合と等価になります。

cpuidは、上記のように指定はできますが、実際に使用できるcpuidは、実行開始時のプロセスのCPU affinityの範囲内のcpuidのみになります。

CPU_SETSIZEについては、CPU_SET(3)を参照してください。

注意

cpuidが実行開始時のプロセスのCPU affinityの範囲外である場合、エラーを出力しプログラムが終了しますので、設定値を見直してください。

例

環境変数GOMP_CPU_AFFINITYの指定例

- 例1:

```
$ export GOMP_CPU_AFFINITY="12,14,13,15"
```

12,14,13,15の順でスレッドにバインドします。
スレッド数5以上の場合は、先頭から繰り返して使用します。

- 例2:

```
$ export GOMP_CPU_AFFINITY="12-19"
```

12,13,14,15,16,17,18,19の順でスレッドにバインドします。
スレッド数9以上の場合は、先頭から繰り返して使用します。

- 例3:

```
$ export GOMP_CPU_AFFINITY="12-19:2"
```

12,14,16,18の順でスレッドにバインドします。
スレッドが5以上の場合は、先頭から繰り返して使用します。

- 例4:

```
$ export GOMP_CPU_AFFINITY="12-16:2,13,19"
```

12,14,16,13,19の順でスレッドにバインドします。
スレッドが6以上の場合は、先頭から繰り返して使用します。

12.3.2.2 OpenMP仕様の環境変数

利用者は、OpenMP仕様で規定された以下の環境変数を使用することが可能です。詳細については、OpenMP仕様を参照してください。

環境変数OMP_SCHEDULE

スケジュールタイプがRUNTIMEのDO指示文およびPARALLEL DO指示文に対して、実行すべきスケジュールタイプおよびチャンクサイズを指定します。

環境変数OMP_NUM_THREADS

実行中に使用するスレッドの数を指定します。

環境変数OMP_DYNAMIC

動的スレッド調整機能の有効または無効を指定します。

環境変数OMP_PROC_BIND

スレッドをCPUにバインドする方法を指定します。

環境変数OMP_NESTED

PARALLELリージョンのネスト機能の有効または無効を指定します。

環境変数OMP_STACKSIZE

スレッド毎のスタック領域の大きさを指定します。

環境変数OMP_WAIT_POLICY

同期待ち処理の動作を指定します。

環境変数OMP_MAX_ACTIVE_LEVELS

ネストしている活動状態のPARALLELリージョンの最大数を指定します。

環境変数OMP_THREAD_LIMIT

OpenMPプログラムで実行するスレッド数の最大数を指定します。

環境変数OMP_PLACES

スレッドが利用できるプレースリストを指定します。

環境変数OMP_CANCELLATION

キャンセレーション機能の有効または無効を指定します。

環境変数OMP_MAX_TASK_PRIORITY

タスク優先度の最大値を指定します。

環境変数OMP_DISPLAY_ENV

OpenMPバージョンとOpenMP内部制御変数の初期値を表示するかどうかを指定します。

12.3.2.3 実行時の注意事項

本処理系のOpenMP仕様による並列化機能を使用する場合の注意事項を、以下に示します。

実行時の変数割付け

スレッド毎のスタック領域の大きさを、特定の大きさで確保したい場合には、環境変数OMP_STACKSIZEを使用してください。

環境変数OMP_STACKSIZEについては、“12.3.2.1 実行時の環境変数”を参照してください。

CPU数の上限値

CPU数の上限値は、システムで利用できるCPU数です。

スレッド数

OpenMPのスレッド数は、以下の優先順位に従って決まります。自動並列化のスレッド数は、“12.2 自動並列化”の記述に従います。

1. PARALLEL指示文のNUM_THREADS指示節の指定値
2. OMP_SET_NUM_THREADSルーチンの指定値
3. 環境変数OMP_NUM_THREADSの指定値
4. CPU数の上限値

動的スレッド調整機能が有効な場合、上記の優先順位で決まったスレッド数はCPU数の上限内に収まるように調整されます。

動的スレッド調整機能が無効な場合、上記の優先順位で決まったスレッド数となります。1CPU当たりのスレッド数が1を超える場合には、並列に実行されることを意図したスレッドが時分割で実行されることになります。このような場合、スレッド間の同期処理のオーバヘッドが大きくなり、実行性能が低下することがあります。システムによる負荷も考慮して、1CPU当たりのスレッド数が1以下になるようにスレッド数を設定することをお勧めします。

サービス関数および組込みサブルーチン使用時の注意

ALARMサービス関数

動作は保証されません。逐次実行プログラムだけで使用してください。

KILLおよびSIGNALサービス関数

デッドロックの危険性があります。逐次実行プログラムだけで使用してください。

FORKサービス関数

動作は保証されません。逐次実行プログラムだけで使用してください。

SYSTEM、SH、CHMODサービス関数およびEXECUTE_COMMAND_LINE組込みサブルーチン

メモリの使用量が2倍になり実行性能が低下する可能性があります。逐次実行プログラムだけでの使用をお勧めします。

スレッドのCPUバインド

スレッドは固定のCPUにバインドされませんが、環境変数GOMP_CPU_AFFINITYまたはOMP_PROC_BINDを使用することにより、スレッドの固定のCPUへのバインドを制御することができます。

環境変数GOMP_CPU_AFFINITYは環境変数OMP_PROC_BINDよりも優先されます。

12.3.3 実現依存の仕様

OpenMP仕様において、処理系の実現に任されている仕様については、本処理系では、以下のように実現しています。各項目の詳細については、OpenMP仕様を参照してください。

メモリモデル

4バイトを超えるか4バイト境界を跨ぐ変数に対して、

- 2つのスレッドからの書き込みが同時に起こるとき

  明示的な排他制御が行われなければ、変数の値はどちらの値にもならず不定となることがあります。

- 2つのスレッドから書き込みと読み出しが同時に起こるとき

  明示的な排他制御が行われなければ、読み出される変数の値は書き込み前の値でも書き込み後の値でもなく不定となることがあります。

内部制御変数

各内部制御変数の初期値は以下の通りです。

- nthreads-varの初期値は、CPUの上限値です。
- dyn-varの初期値は、FALSEです。
- run-sched-varの初期値は、チャンクサイズなしのSTATICです。
- def-sched-varの初期値は、チャンクサイズなしのSTATICです。
- bind-varの初期値は、FALSEです。
- stacksize-varの初期値は、8Mバイトです。
- wait-policy-varの初期値は、PASSIVEです。
- thread-limit-varの初期値は、2147483647です。
- max-active-levels-varの初期値は、2147483647です。
- place-partition-varの初期値は、CORESです。本処理系では、THREADSと等価です。

動的スレッド調整機能

本処理系のOpenMP仕様による並列化機能では、動的スレッド調整機能を実現しています。この機能の効果は、“12.3.2.3 実行時の注意事項”を参照してください。

デフォルトの状態では、動的スレッド調整機能は無効です。

ループ指示文

一重化したループの繰返し数の計算に使用する変数の型は、8バイト整数型です。

内部制御変数run-sched-varにAUTOが設定されたときのSCHEDULE(RUNTIME)指示節の効果は、SCHEDULE(STATIC)となります。

SECTIONS構文

SECTIONS構文内の構造化ブロックのスレッドへの割当ては、STATICスケジュールと同じ方法で行われます。

SINGLE構文

SINGLEリージョンは、最初にそのリージョンに到達したスレッドによって実行されます。

SIMD構文

一重化したループの繰返し数の計算に使用する変数の型は、8バイト整数型です。

SIMD長はコンパイラが自動的に決定する値になります。

ALIGNED指示節に*alignment*パラメタが指定されていないときは、*alignment*パラメタに次の値が指定されたものとみなされます。

翻訳時に-KSVEオプションが有効な場合	リストアイテムの型のアライメント
翻訳時に-KNOSVEオプションが有効な場合	16

DECLARE SIMD構文

SIMDLEN指示節が指定されていないときのSIMD長は、次のようになります。

- 翻訳時に-KSVEオプションが有効な場合

  SIMD長は、実行時に決定されます。

- 翻訳時に-KNOSVEオプションが有効な場合

  SIMD長は、ベクトル化対象の引数と戻り値の中で最も小さい型の大きさで決定されます。型の大きさにより、SIMD長は以下のようになります。

型の大きさ(byte)	SIMD長
1	16と8
2	8と4
4	4と2
8	2
16	

ALIGNED指示節に*alignment*パラメタが指定されていないときは、*alignment*パラメタに次の値が指定されたものとみなされます。

翻訳時に-KSVEオプションが有効な場合	リストアイテムの型のアライメント
翻訳時に-KNOSVEオプションが有効な場合	16

TASKLOOP構文

GRAINSIZE指示節およびNUM_TASKS指示節の指定がない場合、これらの値はループ繰り返し数を考慮して適切に設定されます。

一重化したループの繰返し数の計算に使用する変数の型は、8バイト整数型です。

CRITICAL構文

本処理系ではロックのヒントをサポートしていません。したがって、HINT指示節を使用した場合、HINT指示節は指定されてないものとみなされます。

ATOMIC構文

2つのATOMICリージョンは、更新する変数の型または型パラメタが異なっていれば、独立に(排他的でなく)実行されます。型と型パラメタが一致しているとき、以下の場合にはアドレスが異なっていても排他的に実行される場合があります。

- 論理型、複素数型、1バイト整数型、2バイト整数型、または4倍精度(16バイト)実数型の変数の更新。
- 配列要素の更新であり、添字式が字面上一致していないとき。
- 対象の式が明示的または暗黙の型変換を伴うとき。

OMP_SET_NUM_THREADSルーチン

OMP_SET_NUM_THREADSルーチンの呼び出しは、引数が0以下の値のとき、1を設定したとみなされます。この値として、システムがサポートするスレッド数を超える数値を設定してはなりません。

OMP_SET_SCHEDULEルーチン

実装依存のスケジュールタイプはありません。

OMP_SET_MAX_ACTIVE_LEVELSルーチン

OMP_SET_MAX_ACTIVE_LEVELSルーチンが明示的なPARALLELリージョンから呼ばれた場合、呼び出しは無効になります。
OMP_SET_MAX_ACTIVE_LEVELSルーチンの引数が0以上の整数ではない場合、呼び出しは無効になります。

OMP_GET_MAX_ACTIVE_LEVELSルーチン

OMP_GET_MAX_ACTIVE_LEVELSルーチンはプログラム内のどの場所からでも呼び出されることができ、内部制御変数max-active-levels-varの値を返します。

OMP_GET_PLACE_PROC_IDSルーチン

指定されたプレースの利用できるプロセッサを識別する数値を返します。プロセッサの定義はオペレーティングシステムが管理するCPUになります。

OMP_INIT_LOCK_WITH_HINTおよびOMP_INIT_NEST_LOCK_WITH_HINTルーチン

本処理系ではロックのヒントをサポートしていません。したがって本ルーチンを使用した場合、hintがないものとみなされます。

環境変数OMP_SCHEDULE

環境変数OMP_SCHEDULEに指定されたスケジュールタイプが有効なスケジュールタイプではない場合、指定は無効となり、デフォルトスケジュール(チャンクサイズなしのSTATIC)が適用されます。

環境変数OMP_SCHEDULEに指定されたスケジュールタイプがSTATIC、DYNAMIC、またはGUIDEDであり、指定されたチャンクサイズが正ではない場合、チャンクサイズは次のようになります。

STATIC	: チャンクサイズなし
DYNAMICおよびGUIDED	: 1

環境変数OMP_NUM_THREADS

環境変数OMP_NUM_THREADSのリストに0を設定すると、1を設定した場合と同じ効果をもちます。0未満を設定すると、“12.3.2.3 実行時の注意事項”のスレッド数に従ってスレッド数が決定されます。この値として、システムがサポートするスレッド数を超える数値を設定してはなりません。

環境変数OMP_PROC_BIND

環境変数OMP_PROC_BINDに使用できない値を設定すると、指定は無効となり、デフォルトの値(CLOSE)が使用されます。以下の条件をすべて満たす場合、余ったスレッドは可能な限り均等にプレースに割り当てられます。

- OMP_PROC_BINDにSPREADまたはCLOSEが指定されている
- T(スレッド数)がP(プレース数)より大きい
- T/Pが割り切れない

環境変数OMP_PLACES

環境変数OMP_PLACESに設定された値が定義された形式を満たしていない場合、指定は無効となり、デフォルトの値(CORES)が使用されます。

環境変数OMP_DYNAMIC

環境変数OMP_DYNAMICにTRUEでもFALSEでもない値を設定すると、指定は無効となり、デフォルトの値(FALSE)が使用されます。

環境変数OMP_NESTED

環境変数OMP_NESTEDにTRUEでもFALSEでもない値を設定すると、指定は無効となり、デフォルトの値(FALSE)が使用されます。

環境変数OMP_STACKSIZE

環境変数OMP_STACKSIZEに設定された値が定義された形式を満たしていない場合、指定は無効となり、デフォルトの値(1Mバイト)が使用されます。

環境変数OMP_WAIT_POLICY

ACTIVEの振舞いはスピン待ちです。PASSIVEの振舞いはサスペンド待ちです。

環境変数OMP_MAX_ACTIVE_LEVELS

環境変数OMP_MAX_ACTIVE_LEVELSに設定された値が0以上の整数ではない場合、指定は無効となり、デフォルトの値(2147483647)が使用されます。

環境変数OMP_THREAD_LIMIT

環境変数OMP_THREAD_LIMITに設定された値が正の整数ではない場合、指定は無効となり、デフォルトの値(2147483647)が使用されます。

THREADPRIVATE指示文

スレッドプライベート変数は、以下の条件が満たされる場合に限って、連続するPARALLELリージョンを跨いでその値、割付け状態、および結合状態が維持されます。この条件は、動的スレッド調整機能とネスト並列の設定には無関係です。

- 連続するPARALLELリージョンは、別の明示的なPARALLELリージョンの内側にネストされていない。
- 両方のPARALLELリージョンを実行するために使用されるスレッドの数が同じである。

これらの条件が満たされないとき、あとのPARALLELリージョンの入口でのスレッドプライベート変数の値、割付け状態、および結合状態は、原則として不定になります。ただし、割付け状態は、以下の条件を満たす場合に限り、あとのPARALLELリージョンの入口で割付けられていない状態になります。

- あとのPARALLELリージョンの直前で、割付けられていない状態である。
- それ以前に実行されたすべてのPARALLELリージョンの出口において、すべてのスレッドで割付けられていない状態である。

SHARED指示節

非組込み手続に共有変数を渡す場合、手続の引用前に共有変数の値を一時的な記憶域にコピーし、手続の引用後に実引数領域に値が戻されることがあります。これは、以下の条件を満たす場合に起こることがあります。

a. 実引数が以下のいずれかの場合
   - 共有変数
   - 共有変数の部分実体
   - 共有変数と結合した実体
   - 共有変数の部分実体と結合した実体

b. さらに、実引数が以下のいずれかの場合
   - 部分配列
   - ベクトル添字がある部分配列
   - 形状引継ぎ配列
   - ポインタ配列

c. この実引数に結合される仮引数は、形状明示配列または大きさ引継ぎ配列である。

実行時ライブラリ定義

本処理系のOpenMP仕様による並列化機能では、omp_lib.hという名前のインクルードファイルおよびomp_libという名前のモジュールを提供しています。これらには総称名による引用仕様宣言は含まれていません。

モジュールomp_libおよびomp_lib.hを引用する場合には以下の注意が必要です。

- 翻訳時オプション-AUを指定した場合、モジュール名omp_libおよび各ライブラリルーチン名は、小文字でなければなりません。
- 翻訳時オプション-CcdII8、-CcI4I8、-CcdLL8、-CcL4L8、-Ccd4d8、または-Cca4a8を指定した場合、ライブラリルーチンの引数および結果の対応する型が変換されます。実行時に対応する実行時オプション-Liや-Lbを指定する必要があります。

12.3.4 OpenMP仕様の明確化および制限事項

ここでは、OpenMP仕様についての、本処理系での解釈および制限事項について説明します。

12.3.4.1 ASSIGN文による文番号指定および割当て形GO TO文

ASSIGN文は、それが現れる構造化ブロックの範囲外の文番号を引用することはできません。また、PARALLELリージョン内の文番号を、PARALLELリージョンの範囲外のASSIGN文で引用することはできません。

割当て形GO TO文による、構造化ブロックの範囲内への飛込みまたは範囲外への飛出しはできません。

12.3.4.2 ATOMIC構文およびリダクション識別子における追加の関数

ATOMIC構文およびリダクション識別子において、以下の組込み関数を指定することが可能です。

組込み関数　　:　　AND、OR、XOR

12.3.4.3 THREADPRIVATE使用時の注意事項

THREADPRIVATE指示文を使用する場合、同じ共通ブロック名は、プログラム内のすべてのプログラム単位および手続において、THREADPRIVATEの指定に関しては、同じ指定でなければなりません。

THREADPRIVATEに指定された共通ブロックは、大きさを拡張して使用することはできません。

12.3.4.4 インライン展開

DECLARE SIMD指示文を除くOpenMP指示文を含んでいる利用者定義の手続は、インライン展開されません。

12.3.4.5 PARALLELリージョンからの内部手続呼出し

PARALLELリージョン内から呼び出される内部手続の中で引用されている親手続の変数は、PARALLELリージョン内でプライベート化されていても、SHAREDであるとみなされます。

例

```
  :
  I=1                    この"I"はSHARED
!$OMP PARALLEL PRIVATE(I)
  I=2                    ↑
  PRINT *,I              |"I"は、この範囲でPRIVATE
  CALL C_PROC()          ↓
!$OMP END PARALLEL
  CONTAINS
  SUBROUTINE C_PROC()
   :                     ↑
  PRINT *,I              |"I"は、この範囲でSHARED
   :                     ↓
  END SUBROUTINE
   :
```

12.3.4.6 多相的変数

多相的として宣言された変数は、THREADPRIVATE指示文、REDUCTION指示節、IN_REDUCTION指示節、およびTASK_REDUCTION指示節に指定することはできません。無制限多相的として宣言された変数は、次の指示節に指定することはできません。

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- COPYPRIVATE
- COPYIN

- DEPEND
- ALIGNED
- UNIFORM

12.3.4.7 OPTIONAL属性

PRIVATE、FIRSTPRIVATE、LASTPRIVATE、REDUCTION、IN_REDUCTION、TASK_REDUCTION、またはLINEAR指示節にOPTIONAL属性の変数を指定した場合、その有効範囲では、その変数はPRESENT組込み関数の実引数には指定できません。

12.3.4.8 結合名

OpenMP仕様の指示文にSELECT TYPE構文の結合名を指定することはできません。

12.3.4.9 選択子

SELECT TYPE構文で結合名をもつ選択子の変数は、次の指示節に指定することはできません。

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- COPYPRIVATE
- COPYIN
- REDUCTION
- IN_REDUCTION
- TASK_REDUCTION

12.3.4.10 長さ型パラメタをもつ派生型の変数

長さ型パラメタをもつ派生型変数は、THREADPRIVATE指示文に指定することはできません。長さ型パラメタをもつ派生型変数は、次の指示節に指定することはできません。

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- REDUCTION
- IN_REDUCTION
- TASK_REDUCTION
- COPYPRIVATE
- COPYIN
- DEPEND
- ALIGNED
- UNIFORM

12.3.4.11 型引継ぎ変数

型引継ぎ変数は、次の指示節に指定することはできません。

- PRIVATE
- FIRSTPRIVATE

- LASTPRIVATE
- REDUCTION
- IN_REDUCTION
- TASK_REDUCTION
- COPYPRIVATE
- COPYIN
- DEPEND
- ALIGNED
- UNIFORM

12.3.4.12 次元引継ぎ変数

次元引継ぎ変数は、次の指示節に指定することはできません。

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- REDUCTION
- IN_REDUCTION
- TASK_REDUCTION
- COPYPRIVATE
- COPYIN
- DEPEND
- ALIGNED
- UNIFORM

12.3.5 プログラミングの注意事項

ここでは、OpenMP仕様のプログラミングにおいての注意事項について説明します。

12.3.5.1 PARALLELリージョンおよび明示的なTASKリージョンの実現

PARALLEL構文およびTASK構文の構造化ブロックは、翻訳処理により、内部手続化されます。

PARALLEL構文から生成された内部手続は、“_OMP_*識別番号A*_”という名前をもちます。*識別番号A*はプログラム単位内でPARALLEL構文を識別する一意の数字です。

TASK構文から生成された内部手続は、“_TSK_*識別番号B*_”という名前をもちます。*識別番号B*はプログラム単位内でTASK構文を識別する一意の数字です。

12.3.5.2 THREADPRIVATE変数の実現

THREADPRIVATE変数は、スレッド・ローカル・ストレージ(Thread-Local Storage)を使用して実現しています。スレッド・ローカル・ストレージのサイズがオフセットの範囲を超えた場合、リンク時にエラーが発生します。

エラーメッセージ例:

```
relocation truncated to fit: R_AARCH64_TLSLE_ADD_TPREL_HI12 against symbol 'varname_'
```

このエラーが発生した場合、-Ktls_size={12|24|32|48}オプションで適切なオフセットのサイズを指定してください。-Ktls_sizeオプションの詳細は、“2.2.3 オプションの説明”を参照してください。

12.3.5.3 OpenMPプログラムの自動並列化

本処理系の-Kopenmpオプションと-Kparallelオプションは、同時に指定可能です。同時に指定した場合、自動並列化されるDOループは以下のように制限されます。

- OpenMPの構文内にあるDOループは、自動並列化の対象とはなりません。
- OpenMPの指示文を静的に内部に含むDOループは、自動並列化の対象とはなりません。
- OpenMPのDO指示文で並列化されるDOループがある場合、以下のDOループは、自動並列化の対象とはなりません。
  - OpenMPのDO指示文で並列化されるDOループ自身
  - OpenMPのDO指示文で並列化されるDOループの内側にあるDOループ

12.3.5.4 ランタイムライブラリルーチン

翻訳時オプション-mldefault=cdeclを指定する場合は、モジュールomp_libを引用してください。

モジュールomp_libを引用していないプログラムでは、同一名のC/C++の関数が実行されるため、パラメタを持つルーチンを実行すると、実行結果が異なる、性能が低下する、エラーを出力しプログラムが終了する、など、プログラムが正しく動作しない場合があります。

OMP_SET_NUM_THREADSルーチンを実行する例

- プログラム例

```
      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
```

- 実行結果
  - 翻訳時オプション-mldefault=cdeclを指定していない場合

```
Max Threads= 12
```

  - 翻訳時オプション-mldefault=cdeclを指定した場合

```
Max Threads= 4199088   (注)
```

    注) 正しい値が出力されません。

    正しい実行結果を得るためには、以下のようにモジュールomp_libを引用してください。

```
      program example
      use omp_lib                    ! omp_libの引用
      call omp_set_num_threads(12)
!$omp single
      print *, "Max Threads=",omp_get_max_threads()
!$omp end single
      end
```

12.3.6 OpenMPプログラムのデバッグ

本処理系が用意しているデバッグ機能については、“第8章 プログラムのデバッグ”を参照してください。

なお、OpenMPプログラムのデバッグには、以下の制限があります。

- DECLARE SIMD構文によって生成された手続は、デバッガを用いてデバッグできません。

- PARALLELリージョンの範囲内では、翻訳時オプション-Hにより提供されるデバッグのための検査機能は、無効になります。

12.4 実行時メッセージ

LLVM OpenMPライブラリを使用した場合、以下の形式で実行時メッセージを標準エラーに出力します。

```
OMP: type message
```

OMP:

LLVM OpenMPライブラリのメッセージであることを表します。

type

メッセージの種別を以下のいずれかで表します。

type	説明
Hint	OpenMPに関する助言(ヒント)です。
Info	OpenMPに関する情報です。
Warning	OpenMPに関する警告メッセージです。致命的なエラーではありません。本メッセージが出力された場合でも、実行を継続します。
Error	OpenMPに関する致命的なエラーメッセージです。本メッセージが出力された場合は、実行を中断します。
System error	システムに関する致命的なエラーメッセージです。本メッセージが出力された場合は、実行を中断します。

message

メッセージ本文です。

例

環境変数OMP_STACKSIZEにFXシステムの許容値を超える値をスタックサイズとして設定すると、以下の実行時メッセージが表示されます。

```
$ 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.
```

注意

以下の実行時オプションは、LLVM OpenMPライブラリの実行時メッセージの制御に使用できません。各実行時オプションの詳細は、“3.3 実行時オプション”を参照してください。

- -e*num*
- -l*elvl*
- -Re
- -Rp

付録A　制限事項および注意事項

A.1　制限事項

Fortranコンパイラを使用する場合、原始プログラムの大きさや複雑さには制限があります。これには、1つのFortran文に関するものと、複数個の文の組合せによるものとがあります。以下に各制限について説明します。

A.1.1　関数引用、部分配列、配列要素引用および部分列引用の入れ子の深さ

関数引用、部分配列、配列要素引用および部分列引用の組合せにおいて、その入れ子の深さは合計255重までです。

ただし、255重以下の入れ子であってもコンパイラの作業域の制限により処理できないことがあります。

例: 関数引用および配列要素引用の入れ子の深さ

```
INTEGER IA(10)
EXTERNAL FUN,IFUN
      :
X = FUN(IFUN(IA(1))) !入れ子の深さは 3です
```

A.1.2　DO構文、CASE構文、IF構文、SELECT TYPE構文、BLOCK構文、CRITICAL構文、およびASSOCIATE構文の入れ子の深さ

DO構文、CASE構文、IF構文、SELECT TYPE構文、BLOCK構文、CRITICAL構文、およびASSOCIATE構文の組合せにおいて、その入れ子の深さは、50重までです。

例: DO構文およびIF構文の入れ子の深さ

```
DO I=1,10
   A(I) = A(I) + 1    !入れ子の深さは1です
   IF (I <= 5) THEN
       CALL S             !入れ子の深さは2です
       DO J = 1,5
          B(J) = A(J)      !入れ子の深さは3です
       END DO
       CALL T             !入れ子の深さは2です
   END IF
END DO
```

A.1.3　DO形並びの入れ子の深さ

入出力文およびDATA文におけるDO形並びの入れ子の深さは、合計25重までです。

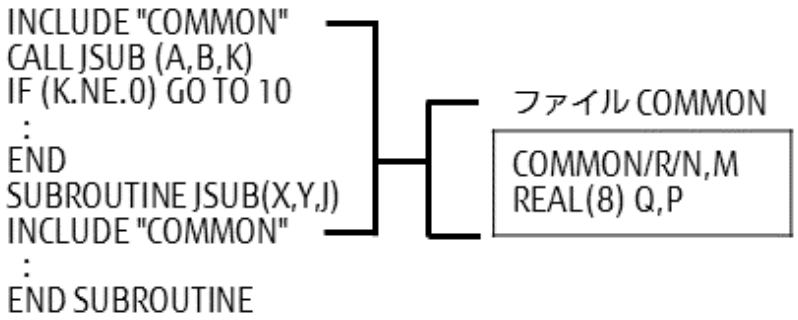
例:DO形並びの入れ子の深さ

```
PRINT *, ((A(I,J),I=1,2),J=1,3) !入れ子の深さは2です
```

A.1.4　INCLUDE行の入れ子の深さ

INCLUDE行の入れ子の深さは、合わせて16重までです。

例: INCLUDE行の入れ子の深さ



この例のINCLUDE 行の入れ子の深さは、1 です。

A.1.5 配列宣言子の制限

コンパイラは、部分配列または配列要素引用時に配列の要素位置を求める計算を減らすため、配列宣言子に対して1つの値*T*を求めます。

配列宣言子には、2つの制限があります。

すべての配列宣言子の制限

次の式で表される*T*の絶対値および計算の途中における絶対値が92233720368547758O7を超えてはいけません。

$$T = l_1 * s + \sum_{i=2}^{n} \{ l_i * (\prod_{m=2}^{i} d_{m-1} * s) \}$$

T: 配列宣言子に対して求める値
l: 各次元の寸法の下限
s: 配列要素の長さ
n: 配列の次元数
d: 各次元の大きさを示す寸法

例: すべての配列宣言子の制限

```
PARAMETER (IL=10000000,IU=10000999)
INTEGER(8) A(IL:IU,IL:IU,IL:IU,IL:IU,IL:IU)
配列Aにおいての T（配列宣言子に対して求める値）は、80080080080080000000となり、この配列宣言子は記述できません。
```

大きさゼロの配列宣言子の制限

非定数宣言式の上下限をもつ大きさゼロの配列の配列宣言子において、次の式で表される*T*の絶対値および計算の途中における絶対値が92233720368547758O7を超えてはいけません。

$$T = s * (\prod_{i=1}^{n} u_i - l_i + 1)$$

T: 配列宣言子に対して求める値
s: 配列要素の長さ
n: 配列の次元数
u: 各次元の寸法の上限
l: 各次元の寸法の下限

A.1.6 部分配列の制限

部分配列の添字三つ組の

```
添字1 : 添字2 : 刻み幅
```

において、

```
（添字2－添字1＋刻み幅）／刻み幅
```

の演算の途中でオーバフローが発生してはいけません。

A.1.7 分岐命令から分岐先命令への距離による影響

分岐命令において、分岐先アドレスの即値フィールドの大きさには制限があります。そのため、分岐命令から分岐先命令への距離が一定以上離れている時、アセンブラで以下のようなメッセージが出力され、そのプログラムを翻訳することができない場合があります。

```
/var/tmp/asmAAAa006jR.s: Assembler messages:
/var/tmp/asmAAAa006jR.s:10: Error: relocation overflow
```

この現象は、数千行以上の実行文を含むループに対して-H系オプションと-Kthreadsafeオプションを同時に指定して翻訳した場合や、巨大なループの中に配列式が多数含まれている場合に発生する可能性が高くなります。

このような現象が起きた場合、ループを分割するなどプログラムを修正する、または-O0オプションを有効にすることにより回避することが可能です。-O0オプションを有効な場合、最適化が行われないので実行時間が長くなることがありますので注意が必要です。

A.2 注意事項

A.2.1 デバッガを用いたデバッグについて

ここでは、デバッガを利用してデバッグする際の注意事項について説明します。

最適化されたプログラムのデバッグ

-O1オプション以上を有効にして翻訳したオブジェクトプログラムをデバックする場合、以下の制約があります。

- 変数のデバッグは、手続の入口でだけ可能です。デバッグできる変数は、仮引数、モジュールまたはサブモジュールで宣言された変数、および共通ブロックで宣言された変数だけです。また、次の制約があります。
  - 4倍精度実数型、単精度複素数型、倍精度複素数型、4倍精度複素数型、派生型、および多相的な仮引数はデバッグできません。
  - 長さ型パラメタが定数ではない文字型、上下限が定数ではない配列、および形状引継ぎ配列の仮引数はデバッグできません。
  - 手続内で参照されない仮引数はデバッグできません。
- 局所変数のデバッグは保証されません。
- インライン展開された手続はデバッグの対象になりません。
- 行番号の表示は保証されません。

これらの制約なしでデバッグしたい場合は、-O0オプションを指定して翻訳してください。

A.2.2 SVEのベクトルレジスタのサイズの変更について

本処理系では、プログラムの実行中にSVEのベクトルレジスタのサイズが変更されないことを仮定して、オブジェクトプログラムを生成します。

SVEを利用したプログラムの実行中に、prctl(2)システムコールなどを利用してSVEの有効なベクトルレジスタのサイズを変更した場合、その動作は保証されません。

SVEのベクトルレジスタのサイズを変更したい場合は、SVEを利用したプログラムの実行前に設定してください。

A.2.3 除数が0の場合の整数除算例外について

Armアーキテクチャである富士通製CPU A64FXでは、除数が0の場合の整数除算例外は検出されません。

翻訳時オプション-NRtrapを指定し、除数が0である場合に、整数除算例外が検出されることを前提としたプログラムでは、演算結果が異なる場合があります。

「整数除算の直前に除数の値を確認する」ことをお勧めします。

 例

整数除算の直前に除数の値を確認する例

```
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 リンクエラー(undefined reference to)について

プログラムの翻訳時において、configureなどにより以下のような操作をしていた場合、通常行われる本処理系独自オブジェクトのリンクを抑止するため、リンクエラー(undefined reference to)となる場合があります。

- 本処理系がリンカに渡している-Lオプションを、本処理系の翻訳コマンドに直接指定してリンクを行う。

 例

リンクエラーの例

```
/opt/FJSVxos/devkit/aarch64/rfs/usr/lib64/crt1.o: In function `_start':
(.text+0x18): undefined reference to `main'
```

環境変数FCOMP_LINK_FJOBJを設定することで回避できる場合があります。環境変数FCOMP_LINK_FJOBJについては、“2.3 翻訳コマンドの環境変数”を参照してください。

付録B　エンディアン変換コマンド

利用者は、エンディアン変換コマンドfcvendianおよびfcvendianpxを使用することにより、データのエンディアンを変換することができます。

B.1　fcvendianおよびfcvendianpxコマンドの形式

以下に、fcvendianおよびfcvendianpxコマンドの形式を示します。

コマンド名	オペランド
fcvendian	□ *file1* □ *file2* □ *type*
fcvendianpx	

□: 1個以上の空白が必要なことを意味します。

file1: 入力ファイル

file2: 出力ファイル

type: 変換するデータの型(2、4、8、16のいずれかの値)

オペランドを指定しない場合、fcvendianおよびfcvendianpxコマンドの使用方法が表示されます。

以下に、例を示します。

例1: 8バイトデータが格納されたデータファイルを変換する

```
$ fcvendian infile outfile 8
```

例2: fcvendianコマンドの使用方法を表示する

```
$ fcvendian
```

B.2　fcvendianおよびfcvendianpxコマンドの復帰値

以下に、fcvendianおよびfcvendianpxコマンドが設定する復帰コードを示します。

復帰コード	意味
0	正常に終了しました。
1	fcvendianまたはfcvendianpxコマンドでエラーが発生しました。

B.3　fcvendianおよびfcvendianpxコマンドの出力情報

エンディアン変換コマンドfcvendianおよびfcvendianpxが出力する情報として診断メッセージがあります。診断メッセージは、コマンド行に指定されたオペランドに誤りがある場合、オペランドの値に誤りがある場合、指定されたファイルがオープンできない場合、作業領域が獲得できない場合に標準エラー出力ファイルに出力されます。

fcvendianコマンドが出力する診断メッセージを以下に示します。

```
Usage: fcvendian infile outfile type
```

```
fcvendian:Invalid operand
```

```
fcvendian: Input file open error
```

```
fcvendian: Output file open error
```

```
fcvendian: Memory allocate error
```

fcvendianpxコマンドが出力する診断メッセージを以下に示します。

```
Usage: fcvendianpx infile outfile type
```

```
fcvendianpx:Invalid operand
```

```
fcvendianpx: Input file open error
```

```
fcvendianpx: Output file open error
```

```
fcvendianpx: Memory allocate error
```

B.4 留意事項

データの型が混在するファイルは正しくデータ変換できない場合があります。

付録C　プリプロセッサ

ここではC言語プリプロセッサおよびFortranプリプロセッサについて説明します。

ファイルサフィックスが.F、.FOR、.F90、.F95、.F03、.F08のいずれかの場合、または、-Cppオプション指定されている場合、プリプロセッサが動作します。このとき、C言語のプリプロセッサとFortranのプリプロセッサを選択するオプション-Ccppおよび-Cfppが指定できます。デフォルトは、Fortranプリプロセッサが選択されます。

プリプロセッサオプションの詳細については、“2.2 翻訳時オプション”を参照してください。

C.1　C言語プリプロセッサ

C言語の仕様に準拠した前処理を行います。

C.2　Fortranプリプロセッサ

C言語仕様よりFortran言語仕様を優先し、注釈および継続行指示などFortran構文を考慮した前処理を行います。

次の点で注意が必要です。

- Fortran構文(継続行指示、注釈、固定形式と自由形式など)が考慮されます。
- Fortran注釈構文もマクロ置換の対象になります。ただし、誤りがあってもFortran注釈構文の中ではメッセージが出力されません。
- ホレリス定数およびH形編集記述子は、文字文脈と解釈されず、マクロ置換の対象になります。
- 2進定数表現、8進定数表現、16進定数表現を表すB、O、X、Z、b、o、xおよびzは、前処理指令の影響を受けずマクロ置換されません。
- 自由形式で文字文脈を継続する場合、‘&’はその行の空白でない最後の文字でなければならず、継続行の空白でない最初の文字は‘&’でなければなりません。
- Fortran原始プログラムを前処理することにより行番号が移動することがあるため、本オプションの利用には注意が必要です。この場合は-Pオプションを指定して、生成された前処理済みの出力ファイルを参照して対応してください。
- 前処理指令により制限桁を超えた行が現れ、かつ、-Pオプション指定されるとき、生成された前処理済みの出力ファイルへ#行番号!fpp startまたは#行番号!fpp endの形式の制御行が生成されることがあります。この出力ファイルをコンパイラの再入力とする場合は、これらの制御行はコンパイラで利用されるため、変更しないでください。

次にFortranプリプロセッサの特徴を例で説明します。

C.2.1　Fortranの継続行

Fortranの継続行を考慮します。

- 固定形式では、6桁目が継続行の指示となります。前行の行末文字と継続行の7桁目が連続しているとして解析されます。

例:aaaはbbbに置換されます。

```
#define aaa bbb
!23456789
      aa
     *a=1
```

- 自由形式の行末の文字&および継続行の最初の文字&は継続行を指示する文字と解釈され、&の直前と&の直後の文字は連続しているとして解析されます。

例:aaaはbbbに置換されます。

```
#define aaa bbb
aa&
  &a=1
```

C.2.2 Fortranの注釈行

Fortranの注釈行を考慮します。

・ C言語の注釈とFortranの注釈が記述できます。C言語形式の注釈は、空白に置換されます。

例:C言語の注釈とFortranの注釈は有効です。

```
/* C言語の注釈
    C言語の注釈 */
   ! Fortranの注釈
```

C.2.3 Fortranの注釈とC言語の注釈

・ Fortranの注釈中にC言語の注釈の開始「/*」が現れた場合、C言語の注釈の開始は無効です。

例:C言語形式の注釈の開始は無効です。

```
!  /* Fortranの注釈です
注釈にはなりません */
```

・ C言語の注釈中にFortranの注釈が現れた場合、Fortranの注釈は無効です。

例:Fortranの注釈は無効となり、代入文は有効です。

```
/*  !  */ kkk=1
```

C.2.4 Fortranの注釈構文中のマクロ置換

Fortranの注釈構文中のマクロ置換は考慮されます。

・ Fortranの注釈構文中も、マクロ置換の対象となります。これは、FortranでのOpenMPの接頭辞および最適化制御行などが注釈文字で始まることを考慮した仕様です。

例:Fortran注釈構文中のaaaはbbbに置換されます。

```
#define aaa bbb
!OCL NOARRAYPAD(aaa)
       integer::aaa(100)
       aaa=10
!      aaa=20
!$OMP PARALLEL SHARED(aaa)
       :
```

・ Fortranの注釈構文中は、マクロ置換で誤りを検出してもメッセージは出力されません。

例:注釈中は、マクロ置換で誤りを検出してもメッセージは出力されません。

```
#define a(i) b(2,i)
a(5,10)=1  ! メッセージが出力されます
! a(5,10)=1 メッセージが出力されません
```

C.2.5 固定形式における第72けたを超えた文字の扱い

固定形式の場合、注釈と前処理指令を除き、第72けたを超えた文字は無視されます。ただし、翻訳時オプション-Fwideを指定することで第73けたから第255けたまで有効にすることができます。

例:固定形式において文字Cが第73けたであるため、ABCはABと解釈されDへ置換されません。第73けたから第255けたを有効とする場合は、翻訳時オプション-Fwide を指定してください。

```
#define ABC D
!       1         2         3         4         5         6         7
!2345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
                                                                      ABC
```

```
    * = 100
     PRINT *,D
     END
```

付録D　GNU互換オプション

本処理系は、GNU Fortranの翻訳時オプション(以降、GNU互換オプションと呼びます)をサポートします。

ここでは、本処理系がサポートするGNU互換オプションについて説明します。

GNU互換オプションの詳細については、GCCのホームページを参照してください。

本処理系がサポートするGNU互換オプション

--print-file-name=include

本処理系が提供するincludeディレクトリを表示することを指示します。

--print-prog-name={ as | ld | objdump | ranlib | ar }

翻訳コマンドが呼び出すプログラムを表示することを指示します。

--shared

リンカに対して、共有オブジェクトを作成するよう指示します。

--version

コンパイラのバージョンと著作権の情報を標準出力に出力します。

-cpp

前処理指令Fortran原始プログラムをC言語のプリプロセッサで前処理することを指示します。

-Xlinker *option*

*option*をリンカへのオプションとして渡すことを指示します。

-f{ align-loops[=*N*] | no-align-loops }

ループの先頭アライメントを2の累乗バイトの境界に合わせるか否かを指示します。省略時は、-fno-align-loopsが適用されます。

-ffp-contract=fast

Floating-Point Multiply-Add/Subtract演算命令を使用した最適化を行うことを指示します。

本オプションを指定した場合、実行結果に副作用(丸め誤差程度の違い)を生じることがあります。

-f{ inline-functions | no-inline-functions }

利用者定義の手続きをインライン展開するか否かを指示します。インライン展開の対象とする利用者定義の手続きは、コンパイラが自動的に決定します。省略時は、-fno-inline-functionsオプションが適用されます。

-floop-parallelize-all

自動並列化を行うことを指示します。ただし、並列化の効果が期待できないものに対しては、自動並列化を行いません。

ファイル名の並びに本オプションを指定して翻訳したオブジェクトプログラムが含まれている場合、リンク時も本オプションを指定してください。

-flto

リンク時最適化を行うことを指示します。

本オプションは、プログラムの翻訳時およびリンク時に指定する必要があります。

-f{ omit-frame-pointer | no-omit-frame-pointer }

手続呼出しにおけるフレームポインタレジスタを保証するか否かを指示します。省略時は、-fno-omit-frame-pointerオプションが適用されます。

-fopenmp

OpenMP仕様を有効にすることを指示します。

ファイル名の並びに-fopenmpオプションを指定して翻訳されたオブジェクトプログラムが含まれている場合、-fopenmpオプションを指定する必要があります。

-f{ openmp-simd | no-openmp-simd }

OpenMP仕様のSIMD構文、DECLARE SIMD構文、DECLARE REDUCTION指示文、およびSIMD指示節をもつORDERED構文だけを有効にするか否かを指示します。省略時は、-fno-openmp-simdオプションが適用されます。

-f{ optimize-sibling-calls | no-optimize-sibling-calls }

末尾呼出しの最適化を行うか否かを指示します。省略時は、-fno-optimize-sibling-callsが適用されます。

-f{ pic | PIC }

位置独立コード(PIC)を生成することを指示します。

本オプションは、翻訳時の指定だけが有効となります。

-f{ pie | PIE }

位置独立コード(PIC)を生成することを指示します。

本オプションは、翻訳時の指定だけが有効となります。

-f{ plt | no-plt }

位置独立コード(PIC)での外部シンボルへのアクセスにProcedure Linkage Table(PLT)を使用するか否かを指示します。省略時は、-fpltオプションが適用されます。

-fprofile-dir=*path*

コードカバレッジ機能使用時に必要な.gcdaファイルの格納先ディレクトリを指示します。*path*には、格納先ディレクトリ名を相対パスまたは絶対パスで指定します。

-f{ schedule-insns | no-schedule-insns }

レジスタ割付け前に、命令スケジューリングの最適化を行うか否かを指示します。省略時は、-fno-schedule-insnsが適用されます。

-f{ schedule-insns2 | no-schedule-insns2 }

レジスタ割付け後に、命令スケジューリングの最適化を行うか否かを指示します。省略時は、-fno-schedule-insns2が適用されます。

-funroll-loops

ループアンローリングの最適化を行うことを指示します

-f{ unsafe-math-optimizations | no-unsafe-math-optimizations }

flush-to-zeroモードを使用するか否かを指示します。省略時は、-fno-unsafe-math-optimizationsオプションが適用されます。

-funsafe-math-optimizationsオプションを指定した場合、実行結果に副作用を生じることがあります。

本オプションは、プログラムのリンク時に指定する必要があります。

-g{ dwarf | dwarf-4 }

DWARF Version 4形式のデバッグ情報を出力します。

本オプションは、-gオプションと等価です。

-isystem *dir*

INCLUDEファイルまたはモジュール情報の検索時に、*dir*を標準のディレクトリとして追加します。

複数の-isystemオプションでディレクトリが複数指定された場合、指定された順に検索します。

ファイルが絶対パス名で指定された場合、指定された絶対パス名だけ検索します。ディレクトリが存在しない場合、本オプションは無効となります。

-march=*archi*[+*features*]

アーテクチャを指定します。

*archi*には、armv8-a、armv8.1-a、armv8.2-a、またはarmv8.3-aを指定します。

*features*には、sveまたはnosveを指定します。*features*の省略時は、sveが適用されます。

-marchオプションの省略時は、-march=armv8.3-a+sveが適用されます。

-mcmodel={ small | large }

実行可能プログラムおよび共有オブジェクトの、コード領域と静的データ領域の最大値を指示します。省略時は、-mcmodel=smallオプションが適用されます。

-mcmodel=small

リンク後のコード領域と静的データ領域の合計が4GB以内になると仮定し、効率の良いオブジェクトプログラムを生成します。

-mcmodel=large

リンク後のコード領域が4GB以内になると仮定します。静的データ領域の大きさに制約はありません。静的データ領域が大きくリンク時にエラーが発生する場合は、本オプションを指定します。

-mcpu=*cpu*

ターゲットのプロセッサを指定します。*cpu*には、a64fx、thunderx2t99、またはgenericを指定します。省略時は、-mcpu=a64fxが適用されます。

-m{ pc-relative-literal-loads | no-pc-relative-literal-loads }

手続内のコード領域が1MB以内であるとして扱い、リテラルプールに1命令でアクセスします。省略時は、-mno-pc-relative-literal-loadsが適用されます。

-mtls-size={ 12 | 24 | 32 | 48 }

スレッド・ローカル・ストレージ(Thread-Local Storage)へのアクセスに必要なオフセットのサイズを指定します。単位はビットです。

-mtune=*cpu*

最適化のターゲットとするプロセッサを指定します。*cpu*には、a64fx、thunderx2t99、またはgenericを指定します。省略時は、-mtune=a64fxが適用されます。

-pthread

POSIXスレッドライブラリを利用するマルチスレッドオブジェクトを生成することを指示します。

-rdynamic

リンカに対して、すべてのシンボルを動的シンボルテーブルに追加するよう指示します。本オプションは、リンカに-export-dynamicオプションを渡します。

-v

翻訳コマンドが呼び出すコンパイラ、アセンブラおよびリンカの実行コマンドを表示します。

-w

すべての警告を抑止することを指示します。

付録E　データおよびメモリ領域

ここでは、本処理系におけるデータおよびメモリ領域の属性について説明します。

E.1　データのサイズおよびアライメント

基本的なデータの型、種別型パラメタ、サイズ、およびアライメントの表を以下に示します。

データの型	種別型パラメタ	サイズ(byte)	アライメント(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
派生型	-	-	成分内で最大のアライメントと8を比較し、大きい方の値

E.2　メモリ領域

メモリ領域の主な用途を以下に示します。

領域	用途
Text	命令が配置されるメモリ領域
静的データ(.data)	初期化ありの静的データが格納されるメモリ領域
静的データ(.bss)	初期化なしの静的データが格納されるメモリ領域
プロセススタック	プロセス/メインスレッド用のスタック領域
スレッドスタック	子プロセス用のスタック領域
動的メモリ確保領域	動的にメモリの確保を要求する時に割り当てられるメモリ領域 または、 スレッドヒープ領域/スレッドスタックとして割り当てられるメモリ領域
共有メモリ	プロセス間で共有するメモリ領域

E.3 データの割付け

変数を割り付けるメモリ領域は、変数の種別や初期値の有無により異なります。

例

```
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))
```

このプログラムでは、各配列を以下のように割り付けます。

配列名	説明
a	局所配列であり、初期設定がないため、プロセススタック領域に割り付けます。
b	局所配列であり、初期設定があるため、静的データ領域(.data)に割り付けます。
c	SAVE属性をもつ配列であり、初期設定がないため、静的データ領域(.bss)に割り付けます。
d	SAVE属性をもつ配列であり、初期設定があるため、静的データ領域(.data)に割り付けます。
e	大域配列であり、初期設定がないため、静的データ領域(.bss)に割り付けます。
f	大域配列であり、初期設定があるため、静的データ領域(.data)に割り付けます。
g	割付け配列のため、動的メモリ確保領域に割り付けます。

E.4 スタック領域へのデータ割付け

スタック領域へのデータ割付け順序を、以下の翻訳時オプションによって制御できます。

-N{ reordered_variable_stack | noreordered_variable_stack }

自動割付け変数をスタック領域に割り付ける順序をコンパイラに指示します。

-Nreordered_variable_stackオプションが指定された場合、以下の順序で自動割付け変数の割付け順を決めます。

1. アライメントの降順
2. アライメントが等しい場合は、データサイズの降順
3. アライメントおよびデータサイズが等しい場合は、ソースプログラム内の宣言文の記載順

アライメントの降順に自動割付け変数を割り付けると、プログラム全体のスタック領域を減らすことができます。

-Nnoreordered_variable_stackオプションが指定された場合、ソースプログラム内の宣言文の記載順に自動割付け変数を割り付けます。

デフォルトは、-Nnoreordered_variable_stackです。

-Nnolineオプション、-g0オプション、または-Koclオプションが有効な場合、割付け順序は保証されません。

例

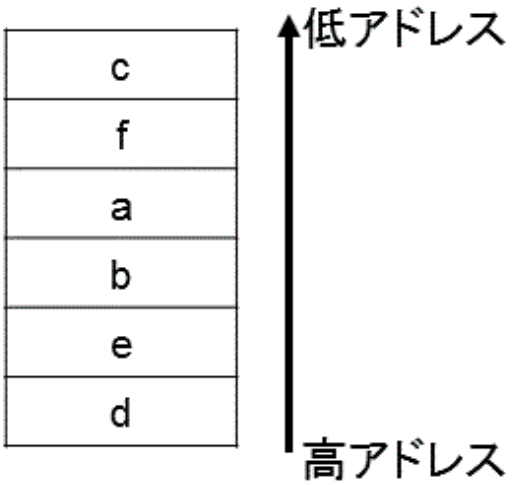
- ソースプログラム

```
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
```

- データ割付け順序を指定した場合

  データ割付け順序を指定した場合、自動割付け変数は下図のようにスタック領域に割り付けられます。プログラム全体のスタック領域を減らすことができます。

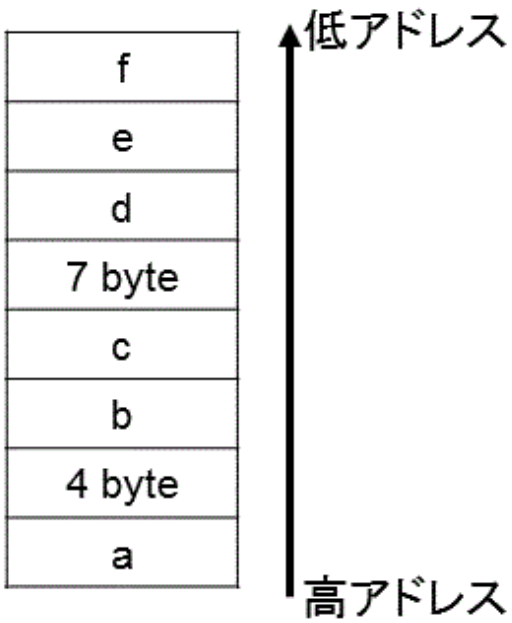
  図E.1 -Nreordered_variable_stackオプション有効時の割付け



・ データ割付け順序を指定しない場合

データ割付け順序を指定しない場合、自動割付け変数は下図のようにスタック領域に割り付けられます。

図E.2 -Nnoreordered_variable_stackオプション有効時の割付け



付録F　半精度型を使用できる組込み手続および入出力文

F.1　組込み手続

本処理系において、半精度実数型/半精度複素数型を使用できる組込み手続を下表に示します。

総称名	半精度実数型	半精度複素数型
ABS	○	○
AIMAG	×	○
ALLOCATED	○	○
ASSOCIATED	○	○
CMPLX	○	○
CSHIFT	○	○
DCMPLX	○	○
DBLE	○	○
DIGITS	○	×
DOT_PRODUCT	○	○
EOSHIFT	○	○
EPSILON	○	×
HUGE	○	×
INT	○	○
IS_CONTIGUOUS	○	○
KIND	○	○
LBOUND	○	○
LOC	○	○
MATMUL	○	○
MAX	○	×
MAXEXPONENT	○	×
MAXVAL	○	×
MIN	○	×
MINEXPONENT	○	×
MINVAL	○	×
MOVE_ALLOC	○	○
NORM2	○	×
PACK	○	○
PRECISION	○	○
PRESENT	○	○
PRODUCT	○	○
QCMPLX	○	○
QEXT	○	○
RADIX	○	×

総称名	半精度実数型	半精度複素数型
RANGE	○	○
REAL	○	○
RESHAPE	○	○
SHAPE	○	○
SIGN	○	×
SIZE	○	○
SIZE_OF	○	○
SPACING	○	×
SPREAD	○	○
STORAGE_SIZE	○	○
SUM	○	○
TINY	○	×
TRANSFER	○	○
UBOUND	○	○
UNPACK	○	○
VAL	○	○

○: 使用できます。

×: 使用できません。

F.2 入出力文

本処理系において、半精度実数型/半精度複素数型を使用できる入出力文を下表に示します。

入出力文	半精度実数型	半精度複素数型
書式なし順番探査READ文	○	○
書式なし順番探査WRITE文	○	○
書式なし直接探査READ文	○	○
書式なし直接探査WRITE文	○	○
書式なし流れ探査READ文	○	○
書式なし流れ探査WRITE文	○	○
出力並びINQUIRE文	○	○

○: 使用できます。

付録G　コードカバレッジ機能

ここでは、コードカバレッジ機能について説明します。

コードカバレッジ機能は、実行時に実行文の実行回数を計測し、プログラムのコード網羅率を調べる機能です。

コードカバレッジ機能では、オープンソースソフトウェアであるコンパイラ基盤LLVMに含まれるコードカバレッジツール(以降、llvm-covコマンドと呼びます)を使用します。llvm-covコマンドの使用方法については、manコマンドを使用して、llvm-covコマンドのオンラインマニュアルを参照してください。

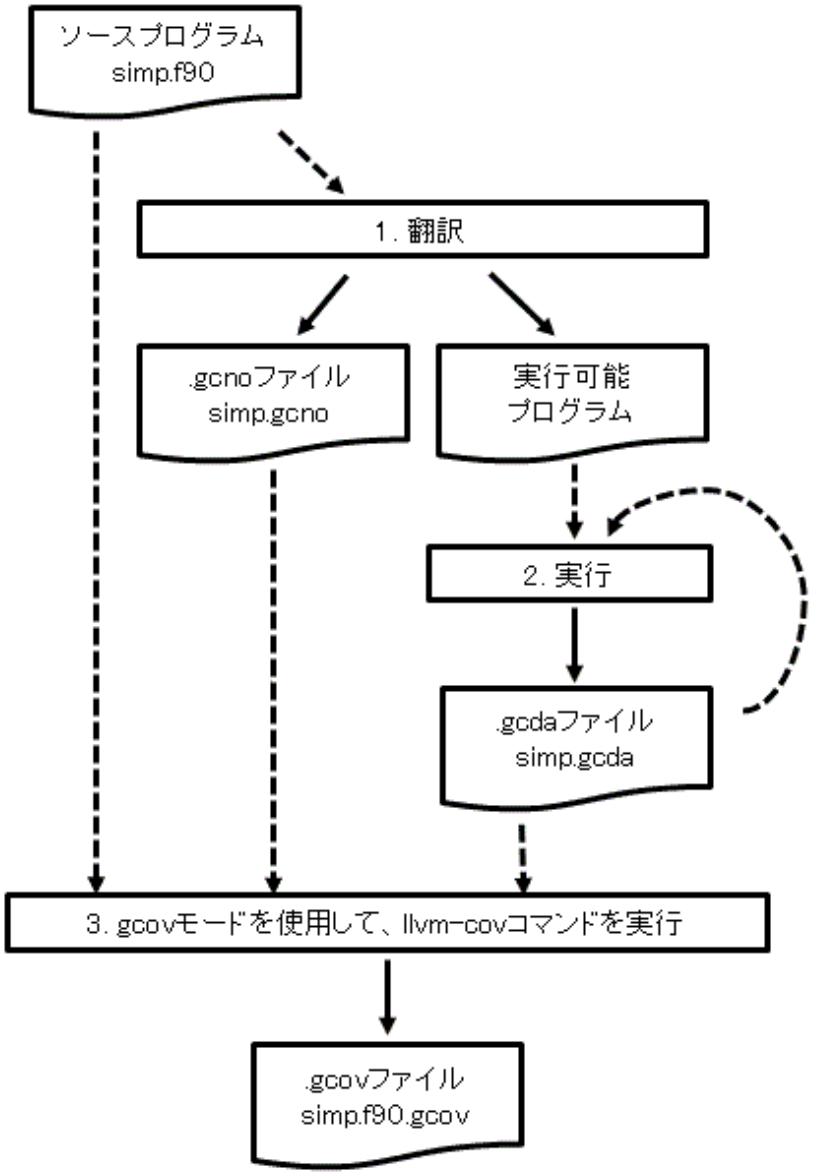
G.1 コードカバレッジ機能の使用方法

コードカバレッジ機能は、以下の手順で使用します。

1. 翻訳
2. 実行
3. gcovモードを使用して、llvm-covコマンドを実行

コードカバレッジ機能の手続を下図に示します。

図G.1 コードカバレッジ機能の手続



G.2 コードカバレッジ機能使用時に必要なファイル

コードカバレッジ機能を使用する時に必要なファイルについて説明します。

G.2.1 .gcnoファイル

.gcnoファイルは、翻訳時に得られる情報を含んだバイナリファイルです。

ファイル生成タイミング	翻訳時に生成されます。
ファイル名	ソースプログラムの拡張子を.gcnoに変更したファイル名になります。
ファイル生成場所	翻訳を行ったディレクトリに生成されます。

以下に、.gcnoファイルの生成例を示します。

例

```
$ frt -Ncoverage ../simp.f90
$ ls
a.out simp.gcno
```

-oオプションを指定してアセンブラソースファイルおよびオブジェクトファイルを生成した場合は、-oオプションが.gcnoファイルにも適用されます。

以下に、-oオプションを指定した場合の生成例を示します。

例

```
$ frt -Ncoverage ../simp.f90 -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
```

G.2.2 .gcdaファイル

.gcdaファイルは、実行時に得られる情報を含んだバイナリファイルです。

ファイル生成タイミング	実行時に生成されます。 同一名のファイルがすでに存在する場合は、そのファイルに情報が累積されます。
ファイル名	ソースプログラムの拡張子を.gcdaに変更したファイル名になります。
ファイル生成場所	翻訳を行ったディレクトリに生成されます。

以下に、.gcdaファイルの生成例を示します。

例

```
$ frt -Ncoverage ../simp.f90
$ ls
a.out simp.gcno
$ ./a.out
$ ls
a.out simp.gcda simp.gcno
```

-oオプションを指定してアセンブラソースファイルおよびオブジェクトファイルを生成した場合は、-oオプションが.gcdaファイルにも適用されます。

以下に、-oオプションを指定した場合の生成例を示します。

例

```
$ 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
```

.gcdaファイルの格納先ディレクトリは-Nprofile_dir=*dir_name*オプションで変更できます。

以下に、-Nprofile_dir=*dir_name*オプションを指定した場合の生成例を示します。

例

```
$ 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 コードカバレッジ機能の注意事項

コードカバレッジ機能使用時の注意事項について説明します。

- 実行回数を計測する命令がオブジェクトファイル内に追加されるため、実行性能が低下する場合があります。
- 実行時に生成される.gcdaファイルの格納先ディレクトリは、翻訳時に決定されます。
  - 翻訳後に実行可能プログラムを移動しても、.gcdaファイルの格納先ディレクトリは変わりません。
  - .gcdaファイルの格納先ディレクトリを変更したい場合は、翻訳時に-Nprofile_dir=*dir_name*オプションを使用して変更してください。
- -Kcmodel=largeオプションを必要とするプログラムでは、使用できません。
- 以下の場合、実行回数を正しく計測できないことがあります。
  - 1行に複数の実行文が含まれる場合
  - STOP文、ERROR STOP文、EXITサービスサブルーチン、およびSETRCDサービスサブルーチンが呼ばれた場合
  - 例外が捕捉された場合
  - ソースプログラムに#line指令が含まれる場合
  - コンパイラの最適化が適用された場合

付録H　実行時情報出力機能

ここでは、実行時情報出力機能について説明します。

実行時情報出力機能は、プログラムの高速化や性能確認の足掛かりとなる情報を、翻訳時オプションおよび実行時環境変数を指定することによって、プログラム実行時に計測して出力する機能です。

H.1　実行時情報出力機能の使用方法

ここでは、実行時情報出力機能の使用方法について説明します。

H.1.1　情報を取得できる範囲

実行時情報出力機能で情報を取得できる範囲を以下に示します。

表H.1 情報を取得することができる範囲

翻訳時オプション	実行時環境変数	情報取得可能な範囲
-Nrt_tune	FLIB_RTINFO	プログラム開始から終了まで
-Nrt_tune_func		利用者定義の手続ごと
-Nrt_tune_loop		ループごと

翻訳時オプション-Nrt_tuneおよび実行時環境変数FLIB_RTINFOを指定することにより、実行時情報出力機能が有効となり、プログラムの開始から終了までの情報を取得することができます。また、翻訳時オプション-Nrt_tune_funcまたは-Nrt_tune_loopを指定することにより、利用者定義の手続ごとの情報、ループごとの情報も取得することができます。

翻訳時オプションについては、“2.2 翻訳時オプション”を参照してください。

MPIプログラムの場合はランク0、COARRAYプログラムの場合は像1、のプロセスの情報が出力されます。

利用者定義の手続ごと

翻訳時オプション-Nrt_tune_funcを指定すると、利用者定義の手続ごとの情報が出力されます。

注意

- 手続の中から手続が呼び出された場合、呼び出された手続の情報は、呼出し元の手続の情報に含まれません。
- 同じ手続が複数箇所で呼び出された場合、合計値がその手続の情報になります。
- 出力される手続は、ユーザが定義した手続のみで、システムコールやライブラリルーチンについては情報が取得されません。

ループごと

翻訳時オプション-Nrt_tune_loopを指定すると、ループごとに以下の情報が出力されます。

- 逐次ループごとの情報
- OpenMP指示文で並列化された箇所内のループごとの情報

逐次ループのコストは、逐次ループごとの情報として出力されます。OpenMP指示文で並列化された箇所内にループが存在する場合は、OpenMP指示文で並列化された箇所内のループごとの情報として出力されます。OpenMP指示文で並列化された箇所内のループの情報は、マスタースレッドの情報のみが出力されます。

注意

- ループの中から手続が呼び出された場合、呼び出された手続の情報は、呼出し元のループの情報に含まれます。
- ループの中からループが呼び出された場合、呼び出されたループの情報は、呼出し元のループの情報に含まれます。

H.1.2 実行時環境変数

実行時情報出力機能で使用する環境変数について、以下に示します。

実行時環境変数のオペランドに指定される値は、英大文字と英小文字を区別します。オペランドが不要である環境変数については、オペランドに指定した値は無視されます。

表H.2 出力する情報を制御する環境変数

実行時環境変数		意味
変数名	オペランド	
FLIB_RTINFO	なし	実行時情報を出力します。
FLIB_RTINFO_NOFUNC	なし	翻訳時オプション-Nrt_tune_funcを指定した場合でも、利用者定義の手続ごとの情報出力を抑止します。
FLIB_RTINFO_NOLOOP	なし	翻訳時オプション-Nrt_tune_loopを指定した場合でも、ループごとの情報出力を抑止します。
FLIB_RTINFO_CSV	ファイル名	取得した情報を、CSV形式でファイルに出力します。また、オペランドには任意のファイル名が指定可能です。オペランドを省略した場合は、flib_rtinfo.csvというファイルに出力します。
	-	

H.2 実行時情報出力機能が出力する情報

ここでは、実行時情報出力機能が出力する情報について説明します。

H.2.1 出力情報

実行時情報出力機能が出力する情報を以下に示します。

表H.3 出力情報

翻訳時オプション	実行時環境変数	出力情報
-Nrt_tune	FLIB_RTINFO	・プログラムの開始から終了までの以下の情報を出力 ー 経過時間 ー ユーザCPU時間 ー システムCPU時間
-Nrt_tune_func		・-Nrt_tuneの情報に加えて、利用者定義の手続ごとに、以下の情報を出力 ー 手続のコスト ー 手続のコストの割合 ー 手続の呼出し1回当たりのコスト ー 手続の開始行番号 ー 手続の終了行番号 ー 手続の呼出し回数 ー 手続名
-Nrt_tune_loop		・-Nrt_tuneの情報に加えて、逐次ループごとおよびOpenMP指示文で並列化された箇所内のループごとに、以下の情報を出力 ー ループのコスト ー ループのコストの割合 ー ループの呼出し1回当たりのコスト

翻訳時オプション	実行時環境変数	出力情報
		― ループの開始行番号 ― ループの終了行番号 ― ループの呼出し回数 ― ループのネストレベル ― 生成手続名

H.2.2 出力形式

実行時情報は、テキスト形式またはCSV形式で出力することができます。

テキスト形式

環境変数FLIB_RTINFO_CSVを指定しない場合、情報はテキスト形式で標準出力ファイルに出力されます。

情報の出力形式の詳細については、“H.2.3 出力例”を参照してください。

CSV形式

環境変数FLIB_RTINFO_CSVを指定することにより、情報をCSV形式で出力することができます。オペランドにファイル名を指定した場合、指定したファイルに取得した情報が出力されます。ファイル名を指定しない場合、flib_rtinfo.csvというファイルに取得した情報が出力されます。すでに指定したファイルが存在する場合は、既存のファイルに情報が上書きされます。ファイルのオープンに失敗した場合は、実行時メッセージjwe1653i-wが出力され、標準出力に情報が出力されます。

CSV形式で出力する場合、データとなる行の情報は以下に示す構成に従います。出力する情報がない場合、“-”が出力されます。

表H.4 CSV形式で出力されるカラム構成

カラム位置	出力される情報
1カラム目	出力された情報の種別を示す識別子 [COST] : プログラム全体の情報 [COST_ROUTINE] : 手続ごとの情報 [COST_LOOP_SERIAL] : 逐次ループごとの情報 [COST_LOOP_PRL] : OpenMP指示文で並列化された箇所内のループごとの情報
2カラム目以降	出力する情報によって異なります。

情報の出力形式の詳細については、“H.2.3 出力例”を参照してください。

H.2.3 出力例

実行時情報出力機能の情報の出力結果の例を以下に示します。

実行時情報の出力例

```
++=============================================================================================++
||                                EXECUTION PERFORMANCE INFORMATION                            ||
++=============================================================================================++

+(1)--------+(2)--------+(3)--------+
| Elapsed   | User      | System    |
|      (sec)|      (sec)|      (sec)|
+-----------+-----------+-----------+
|     0.8820|     0.8909|     0.0030|
+-----------+-----------+-----------+

Routine (4)
+(5)--------+(6)---+(7)--------+(8)----+(9)----+(10)---+(11)-------------------------+
```

```
| 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)
+-----------+------+-----------+-------+-------+-------+(15)+------------------------------+
| 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) 経過時間

(2) ユーザCPU時間

(3) システムCPU時間

(4) 手続ごとの情報(-Nrt_tune_func指定時)

(5) コスト(単位は秒)

(6) コストが全体に占める割合

(7) 呼出し1回当たりのコスト

(8) 開始行番号

(9) 終了行番号

(10) 呼出し回数

(11) 手続名

(12) 各手続の情報

(13) 各手続の合計の情報

(14) 逐次ループごとの情報(-Nrt_tune_loop指定時)

(15) ネストレベル

(16) 各ループの情報

(17) 各ループの合計の情報

(18) OpenMP指示文で並列化された箇所内のループごとの情報(-Nrt_tune_loop指定時)

備考.ループごとの情報取得時には、手続名として内部手続名が出力されます。ネストされたループの場合には、手続名が括弧で囲まれます。

実行時情報の出力例(FLIB_RTINFO_CSV指定時)

```
=====================================
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 実行時情報出力機能の注意事項

ここでは、実行時情報出力機能の注意事項について説明します。

- -Nrt_tune_funcまたは-Nrt_tune_loopを指定し、手続ごとまたはループごとの情報を取得する場合、取得する情報が増加し、実行時間が増加する可能性があります。また、情報が取得できない場合、診断メッセージjwe1655i-iが出力されることがあります。エラー処理中に出力された場合、診断メッセージの出力される順序が正しくない場合があります。
- プログラム中にGOTO文、STOP文などがある場合、または例外が捕捉された場合、正しく情報を計測できない場合があります。
- コンパイラの最適化などにより、以下の影響を受ける可能性があります。
  - ー ループごとの情報を取得する場合、計測の対象となるループが変わる場合があります。
  - ー 行番号がソースプログラムの行番号と一致しない場合があります。
  - ー ループのネスト関係がソースプログラムのネストと一致しない場合があります。次のような最適化によって、ループ情報を正しく認識できない可能性があります。
    - 手続のインライン展開
    - ループアンローリング
    - ループブロッキング
    - ループ交換
    - ループ融合
    - ループストライピング
    - ループ分割
    - ループピーリング
    - ループアンスイッチング
    - マルチ演算関数化・SIMD化

付録I　ジョブ運用ソフトウェア連携高速化機能の利用

ジョブ運用ソフトウェアのジョブ(以降、ジョブと呼びます)では、FXシステムのCPUで実装されている高速化機能(コア間ハードウェアバリアとセクタキャッシュ)を利用することができます。

ここでは、FXシステム向けの高速化機能を使用するためのプログラムの翻訳および実行について説明します。

ジョブの作成方法およびジョブの投入方法については、ジョブ運用ソフトウェアのマニュアルを参照してください。

I.1　コア間ハードウェアバリア

コア間ハードウェアバリア(以降、ハードウェアバリアと呼びます)とは、スレッド並列処理における同期処理を高速にし、実行性能を向上させるハードウェア機能です。ここでは、LLVM OpenMPライブラリを使用して、ハードウェアバリアを利用するためのプログラムの翻訳および実行について説明します。

I.1.1　翻訳

ハードウェアバリアをスレッド間バリアとして利用するプログラムを作成するためには、-Kparallelオプションまたは-Kopenmpオプションを指定します。

I.1.2　実行

環境変数FLIB_BARRIER

OpenMPまたは自動並列を使用する場合のスレッド間バリアの種類を制御できます。指定できる値とその意味は、以下のとおりです。デフォルトは、SOFTです。

HARD

ハードウェアバリアを使用します。

使用できない場合、診断メッセージjwe1050i-wを出力し、ソフトウェアバリアを使用して実行を続けます。

ハードウェアバリアを使用する場合、以下のOpenMP機能のすべてまたは一部が使用できません。これらの機能を使用した場合、診断メッセージjwe1051i-wを出力し、機能を無効化して実行を続けます。

- スレッド数の制御

  スレッド数は、以下の優先順位に従って決まります。

  1. 環境変数OMP_NUM_THREADSの指定値

  2. CPU数の上限値(ジョブによりプロセスに割り当てられたCPU数)

  環境変数OMP_NUM_THREADSの指定値がCPU数の上限値を超える場合、スレッド数はCPU数の上限値になります。

  NUM_THREADS指示節およびOMP_SET_NUM_THREADSルーチンに指定する値は、上記で決まるスレッド数と一致していなければなりません。環境変数OMP_DYNAMICおよびOMP_SET_DYNAMICルーチンの設定に関わらず、スレッド数はCPU数の上限値を超えることはできません。

- スレッドアフィニティの制御

  ジョブによりプロセスに割り当てられたCPUを使用してスレッドアフィニティを設定します。

  環境変数OMP_PROC_BINDおよびPARALLEL構文のPROC_BIND指示節の設定に関わらず、OpenMPのスレッドアフィニティ機能は無効化されます。他の方法でスレッドアフィニティを設定した場合、動作は保障されません。

- ネスト並列

  環境変数OMP_NESTEDおよびOMP_SET_NESTEDルーチンの設定に関わらず、ネストしたPARALLELリージョンは常に逐次化されます。

- タスク

  常に即時に実行するタスクを生成します。

- キャンセレーション

  環境変数OMP_CANCELLATIONの設定に関わらず、常にCANCEL構文とキャンセレーションポイントの効果は無効化されます。

SOFT

ソフトウェアバリアを使用します。(省略値)

注意

- スレッド数が1の場合、ハードウェアバリアは使用できません。
- ハードウェアバリアは同じNUMAノード内でのみ使用できます。したがって、プロセスが複数のNUMAノードに跨る場合は、NUMAノード内はハードウェアバリアが使用されますがNUMAノード間はソフトウェアバリアが使用されるためNUMAノード内にのみと比べて性能は大きく低下します。バリア性能を優先する場合は、1プロセスが1NUMAノード内に収まる使い方を推奨します。
- 以下の場合、1つのプロセスがコア間ハードウェアバリア資源を占有できません。そのためハードウェアバリアの動作は不定になります。診断メッセージjwe1044i-uまたはjwe1045i-uを出力しプログラムを終了、または、異常終了することがあります。

  ー プロセスを富士通のMPI機能以外で生成する場合

  例:

  - Fortran FORK/SYSTEM/SHサービス関数などを使用する場合
  - C forkシステムコール/system関数などを使用する場合

例

例1:スレッド並列ジョブで、ハードウェアバリアを使用するプログラムを実行

```
$ cat job.sh
#!/bin/sh
export FLIB_BARRIER=HARD
./a.out

$ pjsub -L "node=1" job.sh
```

例2:ハイブリッド(プロセスおよびスレッド)並列ジョブ(1次元で1つのジョブを実行:プロセス数にノード数を設定)で、ハードウェアバリアを利用するプログラムを実行

```
$ 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 セクタキャッシュ

セクタキャッシュの制御方法については、“9.17 セクタキャッシュのソフトウェア制御 ”を参照してください。

ここでは、実行上の注意を説明します。

実行上の注意

1次キャッシュと2次キャッシュでセクタキャッシュを利用できる実行環境は以下の通りです。

実行環境	1次キャッシュで セクタキャッシュ利用	2次キャッシュで セクタキャッシュ利用
NUMAノードを1プロセスで占有できる	利用可	利用可
NUMAノードを1プロセスで占有できない	利用可 (注)	利用不可

注) 環境変数FLIB_L1_SCCR_CNTL=FALSEを指定した場合、利用しません。環境変数FLIB_L1_SCCR_CNTLについては、“9.17.2.2 環境変数および最適化制御行によるソフトウェア制御”を参照してください。

また以下の場合、NUMAノードを1プロセスで占有できる場合を識別できません。そのためセクタキャッシュの動作は不定になります。性能が大幅に劣化する、診断メッセージjwe1048i-uを出力しプログラムを終了する、またはプログラムが異常終了することがあります。

- プロセスを富士通のMPI機能以外で生成する場合

  例:

  ー Fortran FORK/SYSTEM/SHサービス関数などを使用する場合

  ー C forkシステムコール/system関数などを使用する場合

- スレッドを富士通の自動並列/OpenMP機能以外で生成する場合

  例:

  ー pthread関数などでスレッドを制御する場合

実行環境に関わらず無条件にセクタキャッシュ制御機能を無効化したい場合は、環境変数FLIB_SCCR_CNTLにFALSEを指定します。詳細は、“9.17.2.2 環境変数および最適化制御行によるソフトウェア制御”を参照してください。

付録J　富士通OpenMPライブラリ

ここでは、富士通OpenMPライブラリを使用してFortranプログラムを並列処理する方法について説明します。

LLVM OpenMPライブラリを使用する場合は、“第12章 並列化機能”を参照してください。

注意

以下の条件を満たしている場合、本処理系は富士通OpenMPライブラリを使用します。

- プログラムのリンク時に、-Nfjomplibオプションが有効である。

参考

並列処理用ライブラリ(LLVM OpenMPライブラリおよび富士通OpenMPライブラリ)と結合可能なオブジェクトファイルの組合せを下表に示します。

tradモードおよびclangモードについては、“C言語使用手引書”または“C++言語使用手引書”を参照してください。

	Fortran オブジェクトファイル	C/C++オブジェクトファイル	
		tradモード (-Nnoclangオプション)	clangモード (-Nclangオプション)
LLVM OpenMPライブラリ (-Nlibompオプション)	結合可	結合可	結合可
富士通OpenMPライブラリ (-Nfjomplibオプション)	結合可	結合可	**結合不可**

J.1　並列処理の概要

“12.1 並列処理の概要”を参照してください。

J.2　自動並列化

ここでは、本処理系で提供している自動並列化について説明します。

J.2.1　翻訳の方法(自動並列化)

自動並列化の機能を使用するには、翻訳コマンドに以下のオプションを指定します。

- 翻訳時

```
-Kparallel
```

- リンク時

```
-Kparallel -Nfjomplib
```

自動並列化のための翻訳時オプション

自動並列化機能に関連するオプションを以下に示します。各オプションの詳細については、“2.2 翻訳時オプション”を参照してください。

```
-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　実行の方法(自動並列化)

自動並列化されたプログラムを実行させるときは、環境変数PARALLELまたはOMP_NUM_THREADSで、並列に動作させるスレッドの数を指定することができます。また、環境変数THREAD_STACK_SIZEまたはOMP_STACKSIZEで、スレッドごとのスタック領域の大きさを指定することができます。さらに、環境変数FLIB_SPINWAITまたはOMP_WAIT_POLICYで、同期待ち処理の動作を指定することができます。

その他の実行に関する手続は、逐次実行の場合と同じです。

環境変数PARALLEL

環境変数PARALLELで、並列に動作させるスレッド数を指定することができます。指定できる値は、1～2147483647の整数値です。

スレッド数の決定方法の詳細は、“スレッド数”を参照してください。

スレッド数

並列に動作するスレッド数は、以下の優先順位に従って決まります。

1. 環境変数PARALLELの値
2. 環境変数OMP_NUM_THREADSの値
3. ジョブで利用できるCPU数
4. 1スレッド

環境変数PARALLELの値

リンク時に-Kopenmpオプションが指定され、かつ環境変数FLIB_FASTOMPにTRUEが設定されている場合、環境変数OMP_NUM_THREADSに指定値があれば、その値と環境変数PARALLELの指定値は一致していなければなりません。

一致していない場合には、スレッド数には小さい方の値が採用されます。この時、jwe1042i-wのメッセージが出力されます。

上記の優先順位で決まったスレッド数と、CPU数の上限値を比較して、小さい方を最終的なスレッド数として採用します。

ジョブで利用できるCPU数の詳細については、“J.5.1 CPU資源の管理”の“ジョブで利用できるCPU数”を参照してください。

なお、CPU数の上限値は、次のように決定されます。

ジョブの場合	ジョブで利用できるCPU数
ジョブでない場合	システムのCPU数

環境変数OMP_NUM_THREADSについては、“J.3.2 実行の方法(OpenMP仕様による並列化)”の“実行時の環境変数”を参照してください。

環境変数THREAD_STACK_SIZEおよびOMP_STACKSIZE

THREAD_STACK_SIZE

環境変数THREAD_STACK_SIZEで、スレッドごとのスタック領域の大きさをKバイト単位で指定することができます。指定できる値は、1～18446744073709551615の整数値です。

環境変数OMP_STACKSIZEが指定されている場合、大きい方の指定値がスレッドごとのスタック領域の大きさの値になります。

OMP_STACKSIZE

環境変数OMP_STACKSIZEで、スレッドごとのスタック領域の大きさをバイト、Kバイト、Mバイト、Gバイト単位で指定することができます。

環境変数THREAD_STACK_SIZEが指定されている場合、大きい方の指定値がスレッドごとのスタック領域の大きさの値になります。

スタック領域の詳細は、“実行時の領域”を参照してください。

実行時の領域

自動並列化されたDOループ内でローカルなスカラ変数(翻訳時オプション-Knoautoobjstackが有効な場合の自動割付けデータ実体は除く)は、スレッドごとのスタック領域に割り付けられます。これらの変数が多い場合には、スタック領域を十分な大きさに拡張する必要があります。

スレッドごとのスタック領域を特定の大きさで確保したい場合は、環境変数THREAD_STACK_SIZEまたはOMP_STACKSIZEで指定してください。

環境変数の指定がない場合、スレッドごとのスタック領域は、プロセスのスタック領域と同じ大きさで確保されます。
ただし、「プロセスのスタック領域の制限値」が「使用可能な実装メモリと仮想メモリの大きさの小さい方をスレッド数で割った値」より大きい場合、スレッドごとのスタック領域は下記の仮定値で確保されます。

```
仮定値(byte) = ( min(使用可能な実装メモリの大きさ, 仮想メモリ大きさ) / スレッド数 ) / 5
```

なお、この仮定値は、動作を保証するものではありません。プロセススタック領域の制限値に適切な値を設定することをお勧めします。プロセスのスタック領域の制限値は、ulimit(bash組込みコマンド)などで設定が可能です。
仮想メモリの大きさとは、ulimit(bash組込みコマンド)などで仮想記憶の最大サイズとして表示される値です。
スレッド数については、“J.2.2 実行の方法(自動並列化)”の“スレッド数”を参照してください。

注意

一部のpthreadライブラリでは、fixed stackになっているため、スレッドスタックサイズを上記のように変更できないことがあります。同一環境であっても、スタティック版はfixed stackであることが多いため、できるだけ共有版のpthreadライブラリを使用するようにしてください。

同期待ち処理

環境変数FLIB_SPINWAITまたはOMP_WAIT_POLICYで、同期待ち処理の動作を指定することができます。

FLIB_SPINWAITの値

unlimited	スピン待ちを行います。 デフォルトです。
0	サスペンド待ちを行います。
n s	*n*秒間スピン待ちし、その後サスペンド待ちに移ります。 *n*は0以上の整数です。*n*に続けて、単位の"s"を指定します。
n ms	*n*ミリ秒間スピン待ちし、その後サスペンド待ちに移ります。 *n*は0以上の整数です。*n*に続けて、単位の"ms"を指定します。

スピン待ちとは、CPU時間を消費する方法で待たせる方式です。逆に、サスペンド待ちはCPU時間を消費せずに待たせます。スピン待ちを行う方が並列処理のオーバヘッドが少ないため、経過時間を重視する場合には、unlimitedを選択してください。CPU時間を重視する場合は、0を選択してください。

OMP_WAIT_POLICYの値

ACTIVE	スピン待ちを行います。 デフォルトです。
PASSIVE	サスペンド待ちを行います。

経過時間を重視する場合には、ACTIVEを選択してください。CPU時間を重視する場合は、PASSIVEを選択してください。

環境変数FLIB_SPINWAITが指定されている場合は、環境変数FLIB_SPINWAITの設定を有効とします。

打切り時間設定時の注意

ulimit(1)コマンドなどにより打切り時間を設定した場合、正常に動作しないことがあります。正常に動作しない場合は、Fortranの例外ハンドリング処理を実行時オプション-Wl,-iにより無効にしてください。または、実行時オプション-Wl,-tにより打切り時間を設定してください。

サービス関数および組込みサブルーチン使用時の注意

“12.2.2.4 サービス関数および組込みサブルーチン使用時の注意”を参照してください。

スレッドのCPUバインド

スレッドのCPUへのバインドは、ジョブで実行するか否かで異なります。

ジョブ実行しない場合

スレッドは固定のCPUにバインドされませんが、環境変数FLIB_CPU_AFFINITYを使用することによりスレッドの固定のCPUへのバインドを制御することができます。

環境変数FLIB_CPU_AFFINITY

スレッドを指定されたcpuid順にCPUバインドします。

スレッド数が、指定したcpuidの数を超える場合は、先頭から繰り返して使用します。

各cpuidは、コンマ(',')または空白(' ')で区切る必要があります。

cpuidは、以下のような形式で増分値付き範囲指定ができます。

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: 範囲指定の最初のcpuid(0 <= *cpuid1* < CPU_SETSIZE)
cpuid2: 範囲指定の最後のcpuid(0 <= *cpuid2* < CPU_SETSIZE)
inc: 増分値(1 <= *inc* < CPU_SETSIZE)

なお以下の条件を満たす必要があります。

```
cpuid1 <= cpuid2
```

*cpuid1*から*cpuid2*の範囲のcpuidを増分値*inc*ごとにすべてを指定した場合と等価になります。

cpuidは、上記のように指定はできますが、実際に使用できるcpuidは、実行開始時のプロセスのCPU affinityの範囲内のcpuidのみになります。

CPU_SETSIZEについては、CPU_SET(3)を参照してください。

注意

cpuidが実行開始時のプロセスのCPU affinityの範囲外である場合、エラーが出力されますので、設定値を見直してください。

例

例1:

```
$ export FLIB_CPU_AFFINITY="12,14,13,15"
```

12,14,13,15の順でスレッドにバインドします。
スレッド数5以上の場合は、先頭から繰り返して使用します。

例2:

```
$ export FLIB_CPU_AFFINITY="12-19"
```

12,13,14,15,16,17,18,19の順でスレッドにバインドします。
スレッド数9以上の場合は、先頭から繰り返して使用します。

例3:

```
$ export FLIB_CPU_AFFINITY="12-19:2"
```

12,14,16,18の順でスレッドにバインドします。
スレッドが5以上の場合は、先頭から繰り返して使用します。

例4:

```
$ export FLIB_CPU_AFFINITY="12-16:2,13,19"
```

12,14,16,13,19の順でスレッドにバインドします。
スレッドが6以上の場合は、先頭から繰り返して使用します。

ジョブ実行する場合

スレッドをCPUへバインドしない場合、ジョブ実行しない場合と同様に、環境変数FLIB_CPU_AFFINITYを使用することができます。

スレッドをCPUへバインドする場合、環境変数FLIB_CPU_AFFINITYは無視されます。

スレッドのCPUへのバインドについては、"J.5.2 スレッドのCPUへのバインド"を参照してください。

J.2.3 翻訳・実行の例

例1:

```
$ frtpx -Kparallel,reduction,ocl -Nfjomplib test1.f
$ ./a.out
```

リダクションの最適化と最適化制御行を有効にして、原始プログラムを翻訳します。このプログラムが実行時に使用するスレッド数は、実行時環境によって決まります。スレッド数の詳細については、"J.2.2 実行の方法(自動並列化)"の"スレッド数"を参照してください。

例2:

```
$ frtpx -Kparallel -Koptmsg=2 -Nfjomplib test2.f
Fortran diagnostic messages: program name(main)
  jwd5001p-i "test2.f", line 2: このDOループを並列化しました。(名前:i)
$ export PARALLEL=2
$ ./a.out
$ export PARALLEL=4
$ ./a.out
```

環境変数PARALLELに値2を設定して、2個のスレッドを使用して動作させます。

次に、環境変数PARALLELに値4を設定して、4個のスレッドを使用して動作させます。

J.2.4 並列化プログラムのチューニング

"12.2.4 並列化プログラムのチューニング"を参照してください。

J.2.5 自動並列化

"12.2.5 自動並列化機能の詳細"を参照してください。

J.2.6 最適化制御行

"12.2.6 最適化制御行"を参照してください。

J.2.7 自動並列化機能を使うときの留意事項

"12.2.7 自動並列化機能を使うときの留意事項"を参照してください。

J.2.8 他のマルチスレッドプログラムとの結合

自動並列化プログラム以外の、マルチスレッドプログラムのオブジェクトプログラムとの結合について、制限事項を示します。

本処理系のOpenMPプログラム

本処理系で翻訳時オプション-Kopenmpを指定し作成したOpenMPプログラムのオブジェクトプログラムについては、本処理系で-Kparallelオプションを指定し作成したオブジェクトプログラムと結合可能です。

他のマルチスレッドプログラム

本処理系で作成した以外の、他のマルチスレッドプログラムのオブジェクトプログラムとは、結合できません。

ただし、環境変数FLIB_PTHREADを使用する場合、pthreadプログラムを結合することができます。

環境変数FLIB_PTHREAD

環境変数FLIB_PTHREADで、pthreadプログラムとの結合を制御できます。指定できる値は、以下のとおりです。省略値は、0になります。

値	説明
0 (省略値)	pthreadスレッドを制御スレッドとしてのみ使用するpthreadプログラムを結合できます。 ただし以下の制限があります。 ・ pthreadスレッドはサスペンド待機する必要があります。 ・ pthreadプログラムは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳してはなりません。(注) ・ pthreadプログラムでは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳したルーチンを使用してはなりません。(注) ・ pthreadプログラムでは、Fortranルーチンを使用してはなりません。(注)
1	pthreadスレッドで並列処理を行うpthreadプログラムを結合できます。 pthreadスレッドを制御スレッドとして使用するpthreadプログラムとの結合もできます。 ただし以下の制限があります。 ・ pthread並列処理とOpenMP/自動並列処理を順番に実行する必要があります。(注) pthread並列処理からOpenMP/自動並列処理を実行してはなりません。 また、OpenMP/自動並列処理からpthread並列処理を実行してはなりません。 ・ pthreadスレッドはサスペンド待機する必要があります。 ・ pthreadプログラムは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳してはなりません。(注) ・ pthreadプログラムでは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳したルーチンを使用してはなりません。(注) ・ pthreadプログラムでは、Fortranルーチンを使用してはなりません。(注) ・ pthreadプログラムでは、スレッド並列版の数学ライブラリを使用してはなりません。(pthreadプログラムを、-SSL2BLAMPオプションを指定して翻訳してはなりません) (注) ・ pthreadスレッドの生成前にチームスレッド数2以上のOpenMP並列処理を実行する必要があります。また、このスレッド数をあとで変更できません。 また本機能を使用する場合、以下の機能が有効になります。 ・ FLIB_SPINWAIT=0 ・ FLIB_CPUBIND=off 上記の環境変数に異なる値を指定した場合、動作は保証されません。

注) 制限機能を使用した場合、動作は保証されません。

J.3　OpenMP仕様による並列化

ここでは、本処理系で提供しているOpenMP仕様による並列化について説明します。OpenMP仕様 については、“OpenMP Architecture Review Board”のホームページを参照してください。

本処理系は、以下の仕様をサポートしています。

サポートする仕様
OpenMP 3.1
OpenMP 4.0の一部 (注)
OpenMP 4.5の一部 (注)

注) 以下の機能を使用できます。

- SIMD構文
- DECLARE SIMD構文

J.3.1　翻訳の方法(OpenMP仕様による並列化)

原始プログラムの翻訳およびリンクを行うには、翻訳コマンドに以下のオプションを指定します。指定するオプションは、OpenMP仕様による並列化を行うか否かで異なります。

OpenMP仕様による並列化	指定するオプション	
	翻訳時	リンク時
OpenMP仕様による並列化およびSIMD化を行う場合	-Kopenmp	-Kopenmp -Nfjomplib
OpenMP仕様によるSIMD化だけを行い、並列化は行わない場合	-Kopenmp_simd	指定するオプションはありません

OpenMPプログラムのための翻訳時オプション

ここでは、OpenMPプログラムを翻訳するための翻訳時オプションについて説明します。

-Kopenmp

-Kopenmpオプションは、OpenMP指示文を有効にし、原始プログラムを翻訳します。

-Kopenmpオプションは、リンク時にも指定する必要があります。-Kopenmpオプションを指定して翻訳されたオブジェクトプログラムが含まれる場合は、リンク時に-Kopenmpオプションを指定する必要があります。

 例

```
$ frtpx a.f90 -Kopenmp -c
$ frtpx a.o -Kopenmp -Nfjomplib
```

-Kopenmp_simd

-Kopenmp_simdオプションは、OpenMP仕様のSIMD構文、DECLARE SIMD構文、DECLARE REDUCTION指示文、およびSIMD指示節をもつORDERED構文だけを有効にし、原始プログラムを翻訳します。

-Nfjomplib

OpenMPライブラリとして、富士通OpenMPライブラリを使用することを指示します。

-Nfjomplibオプションは、リンク時に指定する必要があります。

OpenMPプログラムの最適化情報を出力するための翻訳時オプション

“12.3.1.2 OpenMPプログラムの最適化情報を出力するための翻訳時オプション”を参照してください。

J.3.2　実行の方法(OpenMP仕様による並列化)

OpenMPプログラムを実行させる手続は、逐次実行と同じです。

実行時の環境変数

以下の環境変数を使用できます。

環境変数	説明
FLIB_FASTOMP	高速実行時ライブラリの使用を制御することができます。 以下の値を指定できます。 TRUE(デフォルト) 高速実行時ライブラリを使用します。 FALSE 高速実行時ライブラリを使用しません。 高速実行時ライブラリは、OpenMP仕様の一部を制限することにより、実行時の高速化を実現しています。高速実行時ライブラリには、以下のような特徴があります。 ・ ネストした並列化がない、および、スレッドがCPUに1対1に対応付けられていることを前提とすることにより、高速化を図っています。 ・ ジョブ運用ソフトウェア環境では、高速なハードウェアバリアを使用することができます。 ただし、高速実行時ライブラリを使用できるプログラムには、以下の制限があります。 ・ NUM_THREADS指示節およびOMP_SET_NUM_THREADSサブルーチンで指定するスレッド数は、実行時環境によって決まるスレッド数と一致していなければなりません。 実行時環境によって決まるスレッド数については、“実行時の注意事項”(ジョブ運用ソフトウェア環境の場合には“J.5 ジョブ運用ソフトウェア連携高速化機能の利用”)を参照してください。 高速実行時ライブラリを使用するとき、OpenMP仕様の環境変数とOpenMP仕様による実行の制御は、以下のような制限を受けます。 ・ 環境変数OMP_NESTEDとOMP_SET_NESTEDサブルーチンによる設定に関わらず、ネストしたPARALLELリージョンは常に逐次化されます。 ・ 環境変数OMP_DYNAMICとOMP_SET_DYNAMICサブルーチンによる設定に関わらず、スレッド数は使用可能なCPU数を超えることはできません。 使用可能なCPU数については、“実行時の注意事項”(ジョブ運用ソフトウェア環境の場合には“J.5 ジョブ運用ソフトウェア連携高速化機能の利用”)を参照してください。
FLIB_SPINWAIT	同期待ち処理の動作を指定することができます。 環境変数OMP_WAIT_POLICYが同時に指定されている場合、環境変数FLIB_SPINWAITの設定が有効になります。 以下の値を指定できます。 unlimited(デフォルト) 同期が獲得できるまでスピン待ちを行います。 0 スピン待ちを行わず、サスペンド待ちを行います。 *n* s *n*秒間スピン待ちし、その後サスペンド待ちに移ります。 *n*は0以上の整数です。*n*に続けて、単位の"s"を指定します。

環境変数	説明
	n ms *n*ミリ秒間スピン待ちし、その後サスペンド待ちに移ります。 *n*は0以上の整数です。*n*に続けて、単位の"ms"を指定します。 スピン待ちは、スレッド間で同期を待ち合うとき、CPU時間を消費する方法で待たせる方式です。 サスペンド待ちは、CPU時間を消費せずに待たせます。 スピン待ちを行う方が並列処理のオーバヘッドが少ないため、経過時間を重視する場合には、unlimitedを選択してください。 CPU時間を重視する場合は、0を選択してください。
OMP_WAIT_POLICY	同期待ち処理の動作を指定することができます。 環境変数FLIB_SPINWAITが同時に指定されている場合、環境変数FLIB_SPINWAITの設定が有効になります。 以下の値を指定できます。 ACTIVE(デフォルト) 同期が獲得できるまでスピン待ちを行います。 PASSIVE スレッドはCPUにバインドされません。 経過時間を重視する場合には、ACTIVEを選択してください。CPU時間を重視する場合は、PASSIVEを選択してください。
OMP_PROC_BIND	スレッドのCPUへのバインドを制御することができます。 以下の値を指定できます。 TRUE スレッドは実行開始時のプロセスのCPU affinityの範囲内のcpuid順にCPUバインドされます。 FALSE スレッドはCPUにバインドされません。 環境変数FLIB_CPUBINDまたはFLIB_CPU_AFFINITYの指定は、環境変数OMP_PROC_BINDの指定より優先されます。 ただし、以下の場合には環境変数OMP_PROC_BINDの指定が有効になります。 ・ 環境変数FLIB_CPUBINDがoff、かつ、環境変数FLIB_CPU_AFFINITYの指定なし 環境変数FLIB_CPUBINDについては、“J.5.2 スレッドのCPUへのバインド”を参照してください。 環境変数FLIB_CPU_AFFINITYについては、“実行時の注意事項”を参照してください。
THREAD_STACK_SIZE	スレッドごとのスタック領域の大きさをKバイト単位で指定することができます。 環境変数OMP_STACKSIZEが指定されている場合、大きい方の指定値がスレッドごとのスタック領域の大きさの値になります。 詳細は、“実行時の注意事項”を参照してください。
OMP_STACKSIZE	スレッドごとのスタック領域の大きさをバイト、Kバイト、Mバイト、Gバイト単位で指定することができます。 環境変数THREAD_STACK_SIZEが指定されている場合、大きい方の指定値がスレッドごとのスタック領域の大きさの値になります。 詳細は、“実行時の注意事項”を参照してください。

OpenMP仕様の環境変数

以下のOpenMP仕様の環境変数を使用できます。OpenMP仕様の環境変数の詳細は、OpenMP仕様書を参照してください。

OpenMP仕様の環境変数	説明
OMP_SCHEDULE	スケジュールタイプがRUNTIMEのDO指示文およびPARALLEL DO指示文に対して、実行すべきスケジュールタイプおよびチャンクサイズを指定します。
OMP_NUM_THREADS	実行中に使用するスレッドの数を指定します。
OMP_DYNAMIC	動的スレッド調整機能の有効または無効を指定します。
OMP_PROC_BIND	スレッドをCPUにバインドするかどうかを指定します。
OMP_NESTED	PARALLELリージョンのネスト機能の有効または無効を指定します。
OMP_STACKSIZE	スレッドごとのスタック領域の大きさを指定します。
OMP_WAIT_POLICY	同期待ち処理の動作を指定します。
OMP_MAX_ACTIVE_LEVELS	ネストしている活動状態のPARALLELリージョンの最大数を指定します。
OMP_THREAD_LIMIT	OpenMPプログラムで実行するスレッド数の最大数を指定します。

実行時の注意事項

本処理系のOpenMP仕様による並列化機能を使用する場合の注意事項を、以下に示します。

実行時の変数割付け

本処理系のOpenMP仕様による並列化機能を使用したプログラムは、メインプログラム以外の手続内でローカルな変数(翻訳時オプション-Knoautoobjstackが有効な場合の自動割付けデータ実体は除く)およびプライベート変数をスタック領域に割り付けます。手続内でローカルな変数およびプライベート変数に必要な領域が巨大な場合には、スタック領域を十分な大きさに拡張する必要があります。

スレッドごとのスタック領域の大きさを特定の大きさで確保したい場合は、環境変数THREAD_STACK_SIZEまたはOMP_STACKSIZEで指定してください。

環境変数の指定がない場合、スレッドごとのスタック領域は、プロセスのスタック領域と同じ大きさで確保されます。
ただし、「プロセスのスタック領域の制限値」が「使用可能な実装メモリと仮想メモリの大きさの小さい方をスレッド数で割った値」より大きい場合、スレッドごとのスタック領域は下記の仮定値で確保されます。

```
仮定値(byte) = ( min(使用可能な実装メモリの大きさ，仮想メモリ大きさ) / スレッド数 ) / 5
```

なお、この仮定値は、動作を保証するものではありません。プロセススタック領域の制限値に適切な値を設定することをお勧めします。
プロセスのスタック領域の制限値は、ulimit(bash組込みコマンド)などで設定が可能です。
仮想メモリの大きさとは、ulimit(bash組込みコマンド)などで仮想記憶の最大サイズとして表示される値です。
スレッドごとのスタック領域の大きさを一意にするために動的にスレッド数を変更できる機能を考慮しません。したがって、この式では“FLIB_FASTOMP=TRUE”相当のスレッド数を使用します。
動的にスレッド数を変更する場合は、あらかじめOMP_NUM_THREADSに最大スレッド数を設定しておくことをお勧めします。

環境変数については、“J.3.2 実行の方法(OpenMP仕様による並列化)”の“実行時の環境変数”および“OpenMP仕様の環境変数”を参照してください。

CPU数の上限値

以下の値を、CPU数の上限値とします。

ジョブの場合	ジョブのCPU数
ジョブでない場合	システムのCPU数

スレッド数

並列実行のスレッド数は、高速実行時ライブラリを使用するときOpenMPと自動並列化で共通となり、そうでないときそれぞれ独立に決定されます。高速実行時ライブラリの使用は、環境変数FLIB_FASTOMPの設定によって制御することができます。環境変数については、“J.3.2 実行の方法(OpenMP仕様による並列化)”の“実行時の環境変数”および“OpenMP仕様の環境変数”を参照してください。

高速実行時ライブラリを使用するとき(FLIB_FASTOMPがTRUEのとき)

スレッド数は、以下の優先順位に従って決まります。

1. 環境変数OMP_NUM_THREADSの指定値
2. 環境変数PARALLELの指定値
3. ジョブで利用できるCPU数
4. 1スレッド

この優先順位で決まるスレッド数がCPU数の上限値を超えるとき、スレッド数はCPU数の上限値となります。

NUM_THREADS指示節およびOMP_SET_NUM_THREADSサブルーチンによって指定する値は、ここで決まるスレッド数と一致していなければなりません。異なる値を指定した場合には、プログラムの実行が終了されます。この時にjwe1041i-sのメッセージが出力されます。

高速実行時ライブラリを使用しないとき(FLIB_FASTOMPがFALSEのとき)

OpenMPのスレッド数は、以下の優先順位に従って決まります。自動並列化のスレッド数は、“J.2 自動並列化”の記述に従います。

1. PARALLEL指示文のNUM_THREADS指示節の指定値
2. OMP_SET_NUM_THREADSサブルーチンの指定値
3. 環境変数OMP_NUM_THREADSの指定値
4. 環境変数PARALLELの指定値
5. ジョブで利用できるCPU数
6. 1スレッド

動的スレッド調整機能が有効な場合、上記の優先順位で決まったスレッド数と、CPU数の上限値を比較して、小さい方を最終的なスレッド数として採用します。

動的スレッド調整機能が無効な場合、上記の優先順位で決まったスレッド数となり、CPU数の上限を越えることができます。1CPU当たりのスレッド数が1を超える場合には、並列に実行されることを意図したスレッドが時分割で実行されることになります。このような場合、スレッド間の同期処理のオーバヘッドが大きくなり、実行性能が低下することがあります。システムによる負荷も考慮して、1CPU当たりのスレッド数が1以下になるようにスレッド数を設定することをお勧めします。

ジョブ運用ソフトウェア環境下で実行する場合のCPU数の詳細については、“J.5.1 CPU資源の管理”を参照してください。

打切り時間設定時の注意

ulimit(1)コマンドなどにより打切り時間を設定した場合、正常に動作しないことがあります。正常に動作しない場合は、Fortranの例外ハンドリング処理を実行時オプション-Wl,-iにより無効にしてください。または、実行時オプション-Wl,-tにより打切り時間を設定してください。

サービスルーチン関数および組込みサブルーチン使用時の注意

- ALARMサービス関数

  動作は保証されません。逐次実行プログラムだけで使用してください。

- KILLおよびSIGNALサービス関数

  デッドロックの危険性があります。逐次実行プログラムだけで使用してください。

- FORKサービス関数

  動作は保証されません。逐次実行プログラムだけで使用してください。

- SYSTEM、SH、およびCHMODサービス関数

  メモリの使用量が2倍になり実行性能が低下する可能性があります。逐次実行プログラムだけでの使用をお勧めします。

スレッドのCPUバインド

スレッドのCPUへのバインドは、ジョブで実行するか否かで異なります。

ジョブ実行しない場合

スレッドは固定のCPUにバインドされませんが、環境変数FLIB_CPU_AFFINITYを使用してスレッドの固定のCPUへのバインドを制御することができます。

環境変数FLIB_CPU_AFFINITY

スレッドを指定されたcpuid順にCPUバインドします。

スレッド数が、指定したcpuidの数を超える場合は、先頭から繰り返して使用します。

各cpuidは、コンマ(',')または空白文字(' ')で区切る必要があります。

cpuidは、以下の形式で増分値付き範囲指定ができます。

```
cpuid1[-cpuid2[:inc]]
```

cpuid1:範囲指定の最初のcpuid(0 <= *cpuid1* < CPU_SETSIZE)

cpuid2:範囲指定の最後のcpuid(0 <= *cpuid2* < CPU_SETSIZE)

inc:増分値(1 <= *inc* < CPU_SETSIZE)

なお、以下の条件を満たす必要があります。

```
cpuid1 <= cpuid2
```

*cpuid1*から*cpuid2*の範囲のcpuidを増分値*inc*ごとにすべて指定した場合と等価になります。

cpuidは、上記のように指定できますが、実際に使用できるcpuidは、実行開始時のプロセスのCPU affinityの範囲内のcpuidのみになります。

CPU_SETSIZEについては、CPU_SET(3)を参照してください。

注意

cpuidが実行開始時のプロセスのCPU affinityの範囲外である場合、エラーが出力されますので、設定値を見直してください。

例

例1:

```
$ export FLIB_CPU_AFFINITY="12,14,13,15"
```

12,14,13,15の順でスレッドにバインドします。
スレッド数5以上の場合は、先頭から繰り返して使用します。

例2:

```
$ export FLIB_CPU_AFFINITY="12-19"
```

12,13,14,15,16,17,18,19の順でスレッドにバインドします。
スレッド数9以上の場合は、先頭から繰り返して使用します。

例3:

```
$ export FLIB_CPU_AFFINITY="12-19:2"
```

12,14,16,18の順でスレッドにバインドします。
スレッドが5以上の場合は、先頭から繰り返して使用します。

例4:

```
$ export FLIB_CPU_AFFINITY="12-16:2,13,19"
```

12,14,16,13,19の順でスレッドにバインドします。
スレッドが6以上の場合は、先頭から繰り返して使用します。

ジョブ実行する場合

スレッドをCPUへバインドしない場合、ジョブ実行しない場合と同様に、環境変数FLIB_CPU_AFFINITYを使用することができます。

スレッドをCPUへバインドする場合、環境変数FLIB_CPU_AFFINITYは無視されます。

スレッドのCPUへのバインドについては、“J.5.2 スレッドのCPUへのバインド”を参照してください。

複数のスレッドからの出力情報

プログラムの実行時に同一プログラム内で複数のスレッドからエラーが検出された場合、診断メッセージおよびトレースバックマップは、スレッド単位に出力されます。したがって、PARALLELリージョン内では、スレッド数分の情報が出力されます。

J.3.3 実現依存の仕様

OpenMP仕様で、処理系の実現に任されている仕様については、本処理系では以下のように実現しています。各項目の詳細は、OpenMP仕様書を参照してください。

メモリモデル

4バイトを超えるか、4バイト境界を跨ぐ変数に対して、

- 2つのスレッドからの書き込みが同時に起こるとき

  明示的な排他制御が行われなければ、変数の値はどちらの値にもならず不定となることがあります。

- 2つのスレッドから書き込みと読み出しが同時に起こるとき

  明示的な排他制御が行われなければ、読み出される変数の値は書き込み前の値でも書き込み後の値でもなく不定となることがあります。

内部制御変数

各内部制御変数の初期値は以下のとおりです。

内部制御変数	初期値
nthreads-var	1
dyn-var	TRUE
run-sched-var	チャンクサイズなしのSTATIC
def-sched-var	チャンクサイズなしのSTATIC
bind-var	FALSE
stacksize-var	“J.3.2 実行の方法(OpenMP仕様による並列化)”の“実行時の注意事項”のスレッドごとのスタック領域の大きさ
wait-policy-var	ACTIVE
thread-limit-var	2147483647
max-active-levels-var	2147483647

動的スレッド調整機能

本処理系のOpenMP仕様による並列化機能では、動的スレッド調整機能を実現しています。この機能の効果については、“J.3.2 実行の方法(OpenMP仕様による並列化)”の“実行時の注意事項”を参照してください。

デフォルトの状態では、動的スレッド調整機能は有効です。

ループ指示文

一重化したループの繰返し数の計算に使用する変数の型は、8バイト整数型です。

内部制御変数run-sched-varにAUTOが設定されたときのSCHEDULE(RUNTIME)指示節の効果は、SCHEDULE(STATIC)となります。

SECTIONS構文

SECTIONS構文内の構造化ブロックのスレッドへの割当ては、DYNAMICスケジュールと同じ方法で行われます。スレッドは、SECTIONSリージョンに到達するか、その中の1つの構造化ブロックの実行を終了したとき、その順序で次の構造化ブロックと対応付けられます。

SINGLE構文

SINGLEリージョンは、最初にそのリージョンに到達したスレッドによって実行されます。

SIMD構文

一重化したループの繰返し数の計算に使用する変数の型は、8バイト整数型です。

SIMD長はコンパイラが自動的に決定する値になります。

ALIGNED指示節に*alignment*パラメタが指定されていないときは、*alignment*パラメタに次の値が指定されたものとみなされます。

翻訳時に-KSVEオプションが有効な場合	リストアイテムの型のアライメント
翻訳時に-KNOSVEオプションが有効な場合	16

DECLARE SIMD構文

SIMDLEN指示節が指定されていないときのSIMD長は、次のようになります。

- 翻訳時に-KSVEオプションが有効な場合

  SIMD長は、実行時に決定されます。

- 翻訳時に-KNOSVEオプションが有効な場合

  SIMD長は、ベクトル化対象の引数と戻り値の中で最も小さい型の大きさで決定されます。型の大きさにより、SIMD長は以下のようになります。

型の大きさ(byte)	SIMD長
1	16と8
2	8と4
4	4と2
8	2
16	

ALIGNED指示節に*alignment*パラメタが指定されていないときは、*alignment*パラメタに次の値が指定されたものとみなされます。

翻訳時に-KSVEオプションが有効な場合	リストアイテムの型のアライメント
翻訳時に-KNOSVEオプションが有効な場合	16

ATOMIC構文

2つのATOMICリージョンは、更新する変数の型または型パラメタが異なっていれば、独立に(排他的でなく)実行されます。型と型パラメタが一致しているとき、以下の場合にはアドレスが異なっていても排他的に実行される場合があります。

- 論理型、複素数型、1バイト整数型、2バイト整数型、または4倍精度(16バイト)実数型の変数の更新
- 配列要素の更新であり、添字式が字面上一致していないとき
- 対象の式が明示的または暗黙の型変換を伴うとき

OMP_SET_NUM_THREADSルーチン

OMP_SET_NUM_THREADSルーチンの呼び出しは、引数が0以下の値のとき、何の効果もありません。この値として、システムがサポートするスレッド数を超える数値を設定してはなりません。

OMP_SET_SCHEDULEルーチン

実装依存のスケジュールタイプはありません。

OMP_SET_MAX_ACTIVE_LEVELSルーチン

OMP_SET_MAX_ACTIVE_LEVELSルーチンが明示的なPARALLELリージョンから呼ばれた場合、呼び出しは無効になります。
OMP_SET_MAX_ACTIVE_LEVELSルーチンの引数が0以上の整数ではない場合、呼び出しは無効になります。

OMP_GET_MAX_ACTIVE_LEVELSルーチン

OMP_GET_MAX_ACTIVE_LEVELSルーチンはプログラム内のどの場所からでも呼び出されることができ、内部制御変数max-active-levels-varの値を返します。

環境変数OMP_SCHEDULE

OMP_SCHEDULEに指定されたスケジュールタイプが有効なスケジュールタイプではない場合、指定は無効となり、デフォルト値(チャンクサイズなしのSTATIC)が適用されます。

OMP_SCHEDULEに指定されたスケジュールタイプがSTATIC、DYNAMIC、またはGUIDEDであり、指定されたチャンクサイズが正ではない場合、チャンクサイズは次のようになります。

スケジュールタイプ	チャンクサイズ
STATIC	チャンクサイズなし
DYNAMIC	1
GUIDED	

環境変数OMP_NUM_THREADS

OMP_NUM_THREADSのリストに0以下の値を設定すると、1を設定した場合と同じ効果をもちます。この値として、システムがサポートするスレッド数を超える数値を設定してはなりません。

環境変数OMP_PROC_BIND

OMP_PROC_BINDにTRUEでもFALSEでもない値を設定すると、指定は無効となり、デフォルト値(FALSE)が適用されます。

環境変数OMP_DYNAMIC

OMP_DYNAMICにTRUEでもFALSEでもない値を設定すると、指定は無効となり、デフォルト値(TRUE)が適用されます。

環境変数OMP_NESTED

OMP_NESTEDにTRUEでもFALSEでもない値を設定すると、指定は無効となり、デフォルト値(FALSE)が適用されます。

環境変数OMP_STACKSIZE

OMP_STACKSIZEに設定された値が定義された形式を満たしていない場合、指定は無効となり、デフォルト値が適用されます。

環境変数OMP_WAIT_POLICY

ACTIVEの振舞いはスピン待ちです。PASSIVEの振舞いはサスペンド待ちです。

環境変数OMP_MAX_ACTIVE_LEVELS

OMP_MAX_ACTIVE_LEVELSに設定された値が0以上の整数ではない場合、指定は無効となり、デフォルト値(2147483647)が適用されます。

環境変数OMP_THREAD_LIMIT

OMP_THREAD_LIMITに設定された値が正の整数ではない場合、指定は無効となり、デフォルト値(2147483647)が適用されます。

THREADPRIVATE指示文

スレッドプライベート変数は、以下の条件が満たされる場合に限って、連続するPARALLELリージョンを跨いでその値、割付け状態、および結合状態が維持されます。この条件は、動的スレッド調整機能とネスト並列の設定には無関係です。

- 連続するPARALLELリージョンは、別の明示的なPARALLELリージョンの内側にネストされていない。
- 両方のPARALLELリージョンを実行するために使用されるスレッドの数が同じである。

これらの条件が満たされないとき、あとのPARALLELリージョンの入口でのスレッドプライベート変数の値、割付け状態、および結合状態は、原則として不定になります。ただし、割付け状態は、以下の条件を満たす場合に限り、あとのPARALLELリージョンの入口で割付けられていない状態になります。

- あとのPARALLELリージョンの直前で、割付けられていない状態である。
- それ以前に実行されたすべてのPARALLELリージョンの出口において、すべてのスレッドで割付けられていない状態である。

SHARED指示節

非組込み手続に共有変数を渡す場合、手続の引用前に共有変数の値を一時的な記憶域にコピーし、手続の引用後に実引数領域に値が戻されることがあります。これは、以下の条件を満たす場合に起こることがあります。

a. 実引数が以下のいずれかの場合
   - 共有変数
   - 共有変数の部分実体
   - 共有変数と結合した実体
   - 共有変数の部分実体と結合した実体

b. さらに、実引数が以下のいずれかの場合
   - 部分配列
   - ベクトル添字がある部分配列
   - 形状引継ぎ配列
   - ポインタ配列

c. この実引数に結合される仮引数は、形状明示配列または大きさ引継ぎ配列である

実行時ライブラリ定義

本処理系のOpenMP仕様による並列化機能では、omp_lib.hという名前のインクルードファイルおよびomp_libという名前のモジュールを提供しています。これらには総称名による引用仕様宣言は含まれていません。

モジュールomp_libおよびomp_lib.hを引用する場合には以下の注意が必要です。

- 翻訳時オプション-AUを指定した場合、モジュール名omp_libおよび各ライブラリルーチン名は、小文字でなければなりません。
- 翻訳時オプション-CcdII8、-CcI4I8、-CcdLL8、-CcL4L8、-Ccd4d8または-Cca4a8を指定した場合、ライブラリルーチンの引数および結果の対応する型が変換されます。実行時に対応する実行時オプション-Lb、-Liを指定する必要があります。
- omp_lib.hで宣言されている名前付き定数が使用されない場合、診断メッセージjwd2004i-iが出力されますが、実行結果には影響しません。

J.3.4 OpenMP仕様の明確化および制限事項

“12.3.4 OpenMP仕様の明確化および制限事項”を参照してください。

J.3.5 プログラミングの注意事項

“12.3.5 プログラミングの注意事項”を参照してください。

J.3.6 他のマルチスレッドプログラムとの結合

OpenMPプログラム以外の、マルチスレッドプログラムのオブジェクトプログラムとの結合について説明します。

本処理系の自動並列化プログラム

本処理系で-Kparallelオプションを指定して作成した自動並列化プログラムのオブジェクトプログラムは、本処理系で-Kopenmpオプションを指定して作成したオブジェクトプログラムと結合可能です。

他のマルチスレッドプログラム

本処理系で作成した以外の、他のマルチスレッドプログラムのオブジェクトプログラムとは、結合できません。

ただし、環境変数FLIB_PTHREADを使用する場合、pthreadプログラムを結合することができます。

環境変数FLIB_PTHREAD

環境変数FLIB_PTHREADで、pthreadプログラムとの結合を制御できます。指定できる値は、以下のとおりです。省略値は、0です。

値	説明
0(省略値)	pthreadスレッドを制御スレッドとしてのみ使用するpthreadプログラムを結合できます。 ただし以下の制限があります。 ・ pthreadスレッドはサスペンド待機する必要があります。 ・ pthreadプログラムは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳してはなりません。(注) ・ pthreadプログラムでは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳したルーチンを使用してはなりません。(注) ・ pthreadプログラムでは、Fortranルーチンを使用してはなりません。(注)
1	pthreadスレッドで並列処理を行うpthreadプログラムを結合できます。 pthreadスレッドを制御スレッドとして使用するpthreadプログラムとの結合もできます。 ただし、以下の制限があります。 ・ pthread並列処理とOpenMP/自動並列処理を順番に実行する必要があります。(注) pthread並列処理からOpenMP/自動並列処理を実行してはなりません。 また、OpenMP/自動並列処理からpthread並列処理を実行してはなりません。 ・ pthreadスレッドはサスペンド待機する必要があります。 ・ pthreadプログラムは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳してはなりません。(注) ・ pthreadプログラムでは、-Kopenmpオプションまたは-Kparallelオプションを指定して翻訳したルーチンを使用してはなりません。(注) ・ pthreadプログラムでは、Fortranルーチンを使用してはなりません。(注) ・ pthreadプログラムでは、スレッド並列版の数学ライブラリを使用してはなりません。(pthreadプログラムを、-SSL2BLAMPオプションを指定して翻訳してはなりません)(注) ・ pthreadスレッドの生成前にチームスレッド数2以上のOpenMP並列処理を実行する必要があります。また、このスレッド数をあとで変更できません。 また、本機能を使用する場合、以下の機能が有効になります。 ・ FLIB_SPINWAIT=0 ・ FLIB_CPUBIND=off 上記の環境変数に異なる値を指定した場合、動作は保証されません。

注) 制限機能を使用した場合、動作は保証されません。

J.3.7 OpenMPプログラムのデバッグ

“12.3.6 OpenMPプログラムのデバッグ”を参照してください。

J.4 I/Oバッファ並列転送

I/Oバッファ並列転送とは、Fortranプログラムの配列領域と入出力文で使用するバッファ領域の間のデータ転送をスレッド並列により高速化する機能です。

J.4.1　翻訳の方法

自動並列化の場合

自動並列化を行う翻訳時オプション-Kparallelを指定します。翻訳時オプション-Kparallelについては“J.2.1 翻訳の方法(自動並列化)”を参照してください。

OpenMP仕様による並列化の場合

OpenMP仕様による並列化を行う翻訳時オプション-Kopenmpを指定します。翻訳時オプション-Kopenmpについては“J.3.1 翻訳の方法(OpenMP仕様による並列化)”を参照してください。

J.4.2　実行の方法

必要に応じて、以下の環境変数を使用します。

FLIB_IOBUFCPY

I/Oバッファ並列転送を行うための環境変数です。

環境変数FLIB_IOBUFCPY指定できるのはMPだけです。MP以外を指定した場合、I/Oバッファ並列転送は行われません。

指定例

```
$ export FLIB_IOBUFCPY=MP
```

FLIB_IOBUFCPY_SIZE *size*

I/Oバッファ並列転送を行うための条件を変更します。

環境変数FLIB_IOBUFCPY_SIZEに*size*を指定したとき、データの転送量および入出力バッファが*size*Kバイト以上の場合、書式なし入出力文において並列転送が実行されます。*size*の指定はKバイト単位で、1 <= *size*の値が指定できます。

指定例

```
$ export FLIB_IOBUFCPY_SIZE=16
```

自動並列化の場合

- 環境変数FLIB_IOBUFCPYにMPを指定します。
- スレッド数を環境変数PARALLELで指定します。スレッド数については、“J.2.2 実行の方法(自動並列化)”の“スレッド数”を参照してください。

OpenMP仕様による並列化の場合

- 環境変数FLIB_IOBUFCPYにMPを指定します。
- スレッド数を環境変数OMP_NUM_THREADSで指定します。スレッド数については、“J.3.2 実行の方法(OpenMP仕様による並列化)”の“スレッド数”を参照してください。

J.4.3　I/Oバッファ並列転送の条件

I/Oバッファ並列転送が実行されるには、以下の条件が必要です。

- 対象となる入出力文は、書式なし入出力文
- 対象となる入出力文は、パラレルリージョン外および自動並列化された部分外に存在する
- 環境変数FLIB_IOBUFCPY_SIZEが指定されていない場合、データの転送量および入出力バッファの値が、スレッド数Kバイト＋30Kバイト以上
- 環境変数FLIB_IOBUFCPY_SIZEにsizeが指定されている場合、データの転送量および入出力バッファの値が、sizeKバイト以上

入出力バッファのサイズは、順番探査入出力文または流れ探査入出力文の場合、実行時オプション-gまたは実行時の環境変数fuxxbfで指定してください。直接探査入出力文の場合、open文のrecl指定子の値と実行時オプション-dで指定してください。

実行時オプション-gの詳細については、“3.3 実行時オプション”を参照してください。

実行時の環境変数fuxxbfについては、“3.8 実行時の環境変数”を参照してください。

J.4.4 注意事項

I/Oバッファ並列転送が行われても、入出力バッファが小さい場合または転送されるデータサイズが小さい場合は、性能が劣化することがあります。

J.4.5 翻訳・実行の例

- プログラム(test.f)

```
CHARACTER CH((30+4)*1024)
CH='X'
OPEN(10,FORM="UNFORMATTED")
WRITE(10) CH
CLOSE(10)
END
```

- 自動並列化の場合

```
$ frtpx -Kparallel -Nfjomplib test.f
$ export FLIB_IOBUFCPY=MP
$ export PARALLEL=4
$ ./a.out -Wl,-g34
```

環境変数FLIB_IOBUFCPYにMPを指定してI/Oバッファ並列転送機能を有効にします。環境変数PARALLELに4を指定して、スレッド数を4にします。実行時オプション-g34を指定して、入出力バッファを34Kバイト(スレッド数4＋30Kバイト)にします。

- OpenMP仕様による並列化の場合

```
$ frtpx -Kparallel -Nfjomplib test.f
$ export FLIB_IOBUFCPY=MP
$ export OMP_NUM_THREADS=4
$ ./a.out -Wl,-g34
```

環境変数FLIB_IOBUFCPYにMPを指定してI/Oバッファ並列転送機能を有効にします。環境変数OMP_NUM_THREADSに4を指定して、スレッド数を4にします。実行時オプション-g34を指定して、入出力バッファを34Kバイト(スレッド数4＋30Kバイト)にします。

J.5 ジョブ運用ソフトウェア連携高速化機能の利用

ジョブ運用ソフトウェアのジョブ(以降、ジョブと呼びます)では、FXシステムのCPUで実装されている高速化機能(コア間ハードウェアバリアとセクタキャッシュ)を利用することができます。

ここでは、FXシステム向けの高速化機能を使用するためのプログラムの翻訳および実行について説明します。

J.5.1 CPU資源の管理

ジョブでは、利用できるCPU資源を厳密に管理できます。

- ジョブで利用できるCPU数

  ジョブのプロセス内で利用できるCPU数です。詳細は、“J.5.1 CPU資源の管理”の“ジョブで利用できるCPU数”を参照してください。

- CPU数の上限値

  ジョブで利用できるCPU数を上限値とします。

プログラム実行時のスレッド数の決定において、CPU数の上限値が考慮されます。自動並列化におけるスレッド数の詳細は、“J.2.2 実行の方法(自動並列化)”の“スレッド数”を参照してください。OpenMPにおけるスレッド数の詳細は、“J.3.2 実行の方法(OpenMP仕様による並列化)”の“スレッド数”を参照してください。

ジョブで利用できるCPU数

ジョブが利用できる仮想ノードあたりのCPU資源と仮想ノードあたりのプロセス数から、1プロセスで利用するCPU数の上限値を決定します。この上限値を「ジョブで利用できるCPU数」と呼びます。

ジョブで利用できるCPU数 = *cpunum_on_node* / *procnum_on_node*

cpunum_on_node ： 1仮想ノードあたりの使用できるCPU数
1～1仮想ノードの全CPU数

procnum_on_node ： 1仮想ノードあたりのプロセス数
スレッド並列ジョブの場合は、1
ハイブリッド(プロセスおよびスレッド)並列ジョブの場合は、1～1仮想ノードの全CPU数

割り切れない場合は、切り捨てられます。

MPIプログラムを実行する場合、VCOORDファイルでプロセスのCPU数を設定すると、その値をジョブで利用できるCPU数とします。MPIの詳細については、“MPI使用手引書”を参照してください。

FLIB_USE_ALLCPU

環境変数FLIB_USE_ALLCPUで、ジョブで利用できるCPU数を制御できます。

指定できる値とその意味は以下のとおりです。デフォルトは、FALSEです。

TRUE

ジョブで利用できるCPU数を*cpunum_on_node*とします。つまり、プロセス数に関係なく1仮想ノードあたりの使用できるCPUのすべてを1プロセスで使用します。したがって、1仮想ノードあたりのプロセス数が2以上の場合は、複数プロセスでCPU資源を共有します。

各プロセスで同時に計算する頻度が低いような特殊な用途の場合、実行性能の改善が期待できます。一般的に使用した場合、実行性能の改善は期待できません。

以下の機能を併用してください。

- 翻訳時およびリンク時に、-Kopenmpオプションを使用する。
- 実行時に、環境変数FLIB_FASTOMP=FALSEを定義する。
- 実行時に、環境変数FLIB_SPINWAIT=0を定義する。

これらの機能を使用しない場合、著しく性能劣化しデッドロックのような状態になる可能性があります。-Kopenmpオプション、環境変数FLIB_FASTOMPおよびFLIB_SPINWAITについては、“J.3 OpenMP仕様による並列化”を参照してください。

*procnum_on_node*が1の場合は、TRUEとFALSEで同じ結果になります。*procnum_on_node*が1でない場合は、コア間ハードウェアバリアおよびセクタキャッシュは使用できません。

*cpunum_on_node*および*procnum_on_node*については、“J.5.1 CPU資源の管理”の“ジョブで利用できるCPU数”を、コア間ハードウェアバリアについては“J.5.3コア間ハードウェアバリア”を、セクタキャッシュについては“J.5.4 セクタキャッシュ”を参照してください。

FALSE

ジョブで利用できるCPU数を、“J.5.1 CPU資源の管理”の“ジョブで利用できるCPU数”の値にします。

MPIプログラムを実行する場合、VCOORDファイルでプロセスのCPU数を設定しても無視されます。MPIの詳細については、“MPI使用手引書”を参照してください。

FLIB_USE_CPURESOURCE

環境変数FLIB_USE_CPURESOURCEで、ジョブで利用できるCPU資源の管理を制御できます。

指定できる値とその意味は以下のとおりです。デフォルトは、TRUEです。

TRUE

ジョブで利用できるCPU資源を管理します。

FALSE

ジョブで利用できるCPU資源を管理しません。CPU数の制御はできなくなります。また、CPU資源管理を前提とした、以下の機能も無効になります。

- “J.5.2 スレッドのCPUへのバインド”
- “J.5.3 コア間ハードウェアバリア”
- “J.5.4 セクタキャッシュ”

J.5.2 スレッドのCPUへのバインド

自動並列化またはOpenMPを使用したプログラムをジョブで実行する場合、スレッドをCPUにバインドすることができます。“J.5.1 CPU資源の管理”のとおり、使用するCPU資源が厳密に管理されるので、バインドにより性能向上が期待できます。

自動並列化については“J.2 自動並列化”を、OpenMP仕様による並列化については“J.3 OpenMP仕様による並列化”を参照してください。

FLIB_CPUBIND

環境変数FLIB_CPUBINDでスレッドのCPUへのバインドを制御できます。FLIB_CPUBINDには、以下のいずれかを指定します。指定がない場合は、chip_packと同じになります。

chip_pack	1つのCPUチップに集中するようにスレッドをバインドします。
chip_unpack	多くのCPUチップに分散するようにスレッドをバインドします。
off	スレッドをCPUへバインドしません。独自にバインドする場合などに使用します。

1CPUチップしか使用できない環境では、chip_packとchip_unpackは等価になります。

1CPUチップあたり1CPUしか使用できない環境では、chip_packとchip_unpackは等価になります。

J.5.3 コア間ハードウェアバリア

ジョブでは、FXシステムのCPUで実装されているコア間ハードウェアバリアをスレッド間バリアとして使用することができます。コア間ハードウェアバリアとは、スレッド並列処理における同期処理を高速にし、実行性能を向上させるハードウェア機能です。

ジョブの作成方法およびジョブの投入方法については、ジョブ運用ソフトウェアのマニュアルを参照してください。

翻訳

コア間ハードウェアバリアをスレッド間バリアとして利用するプログラムを作成するには、-Kparallelオプションまたは-Kopenmpオプションを指定します。

コア間ハードウェアバリアを使用できる環境でプログラムを実行すると、自動的にコア間ハードウェアバリアが使用されます。使用できない場合は、ソフトウェアバリアが使用されます。

実行

コア間ハードウェアバリアは、ジョブでのみ利用することができます。コア間ハードウェアバリアは、コア間のスレッド間同期専用機能です。

FLIB_CNTL_BARRIER_ERR

コア間ハードウェアバリアを使用できない場合には、以下の診断メッセージを出力しソフトウェアバリアを使用して実行を続けます。

```
jwe1050i-w The hardware barrier couldn't be used and continues processing using the software barrier.
```

環境変数FLIB_CNTL_BARRIER_ERRを使用することにより診断メッセージjwe1050i-wのエラーを制御することができます。指定できる値とその意味は、以下のとおりです。デフォルトは、TRUEです。

TRUE

診断メッセージjwe1050i-wのエラーを検出します。

コア間ハードウェアバリアを使用できない場合、診断メッセージjwe1050i-wを出力し、ソフトウェアバリアを使用して実行を続けます。

FALSE

診断メッセージjwe1050i-wのエラーを検出しません。

コア間ハードウェアバリアを使用できない場合は、ソフトウェアバリアを使用して実行を続けます。

FLIB_NOHARDBARRIER

環境変数FLIB_NOHARDBARRIERを設定した場合、コア間ハードウェアバリアを使用せず、ソフトウェアバリアを使用します。常に、診断メッセージjwe1050i-wを出力します。

詳細は、“J.5.3 コア間ハードウェアバリア”の“FLIB_CNTL_BARRIER_ERR”を参照してください。

注意

- -Kopenmpオプションを指定して作成したプログラムでコア間ハードウェアバリアを使用する場合には、環境変数FLIB_FASTOMPがFALSEであってはいけません。FLIB_FASTOMPの意味と使用上の注意については、“J.3 OpenMP仕様による並列化”を参照してください。
- コア間ハードウェアバリアを使用する場合、スレッドのCPUへのバインドが必須になります。したがって、環境変数FLIB_CPUBINDにoffを指定した場合は、コア間ハードウェアバリアは、使用できません。スレッドのCPUへのバインドについては、“J.5.2 スレッドのCPUへのバインド”を参照してください。
- スレッド数が1の場合、コア間ハードウェアバリアは使用できません。スレッド数の詳細については、“J.2.2 実行の方法(自動並列化)”の“スレッド数”または“J.3.2 実行の方法(OpenMP仕様による並列化)”の“スレッド数”を参照してください。
- 1プロセスのCPU数が3以下の場合、コア間ハードウェアバリアを使用できないことがあります。1プロセスのCPU数を4以上に設定することをお勧めします。
- ジョブスワップ対象ジョブおよびノード共有ジョブで1プロセスのCPU数が3以下の場合、コア間ハードウェアバリアを使用できません。1プロセスのCPU数を4以上に設定することをお勧めします。
- FXシステムのコア間ハードウェアバリアは同じNUMAノード内でのみ使用できます。したがって、プロセスが複数のNUMAノードに跨る場合は、NUMAノード内はコア間ハードウェアバリアが使用されますがNUMAノード間はソフトウェアバリアが使用されるためスレッド間のバリア性能は劣化します。スレッド間のバリア性能を優先する場合は、1プロセスが1NUMAノード内に収まる使い方を推奨します。
- 以下の場合、1つのプロセスがコア間ハードウェアバリア資源を占有できません。そのためハードウェアバリアの動作は不定になります。診断メッセージjwe1044i-uまたはjwe1045i-uを出力しプログラムを終了する、または、プログラムが異常終了することがあります。

  ー プロセスを富士通のMPI機能以外で生成する場合

  例:

  - Fortran FORK/SYSTEM/SHサービス関数などを使用する場合
  - C forkシステムコール/system関数などを使用する場合

例

例1: スレッド並列ジョブで、コア間ハードウェアバリアを使用するプログラムを実行

```
$ cat job.sh
#!/bin/sh
./a.out
$ pjsub -L "node=1" job.sh
```

例2: ハイブリッド(プロセスおよびスレッド)並列ジョブ(1次元で1つのジョブを実行:プロセス数にノード数を設定)で、コア間ハードウェアバリアを利用するプログラムを実行

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=12"
#PJM --mpi "shape=12"
#PJM --mpi "rank-map-bynode"
mpiexec -n 12 a.out
$ pjsub job.sh
```

J.5.4　セクタキャッシュ

ジョブでは、FXシステムのCPUで実装されているセクタキャッシュを制御することができます。

セクタキャッシュの制御方法については、“9.17 セクタキャッシュのソフトウェア制御 ”を参照してください。

ここでは、実行上の注意のみを説明します。

実行上の注意

1次キャッシュと2次キャッシュでセクタキャッシュを利用できる実行環境は以下の通りです。

実行環境	1次キャッシュで セクタキャッシュ利用	2次キャッシュで セクタキャッシュ利用
NUMAノードを1プロセスで占有できる	利用可	利用可
NUMAノードを1プロセスで占有できない	利用可 (注)	利用不可

注) 環境変数FLIB_L1_SCCR_CNTL=FALSEを指定した場合、利用しません。環境変数FLIB_L1_SCCR_CNTLについては、“9.17.2.2 環境変数および最適化制御行によるソフトウェア制御”を参照してください。

また以下の場合、NUMAノードを1プロセスで占有できる場合を識別できません。そのためセクタキャッシュの動作は不定になります。性能が大幅に劣化する、診断メッセージjwe1048i-uを出力しプログラムを終了する、またはプログラムが異常終了することがあります。

- プロセスを富士通のMPI機能以外で生成する場合

  例:

  ー Fortran FORK/SYSTEM/SHサービス関数などを使用する場合

  ー C forkシステムコール/system関数などを使用する場合

- スレッドを富士通の自動並列/OpenMP機能以外で生成する場合

  例:

  ー pthread関数などでスレッドを制御する場合

実行環境に関わらず無条件にセクタキャッシュ制御機能を無効化したい場合は、環境変数FLIB_SCCR_CNTLにFALSEを指定します。詳細は、“9.17.2.2 環境変数および最適化制御行によるソフトウェア制御”を参照してください。

索　引

[あ]

[か]

[さ]