FUJITSU Software

FUJITSU SSL II/MPI User's Guide (Scientific Subroutine Library)

J2UL-2488-02ENZ0(01)
March 2021

Preface

This manual describes the functions and usage of the Scientific Subroutine Library II/MPI (SSL II/MPI).

SSL II/MPI provides the computational functionality to efficiently compute large-scale problems on a parallel computer with distributed memory system. The algorithms for parallel processing have been adopted.

This manual consists of two parts.

Part I General Description

General rules which should be kept in mind when using SSL II/MPI are outlined.

Part II Usage of Subroutines

The functions and usage of each subroutine are described in alphabetical order of their subroutine names.

Readers of this manual are assumed to be familiar with the MPI system.

For details, refer to the user’s guide for MPI.

The asterisks in the table of contents and subroutine list of this manual indicate items added or changed from the previous version.

Export Controls

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

Date of Publication and Version

Version	Manual code
March 2021, Version 6.1	J2UL-2488-02ENZ0(01)
March 2020, 6th Version	J2UL-2488-02ENZ0(00)
January 2020, 5th Version	J2UL-2488-01ENZ0(00)
October 2014, 4th Version	J2UL-1909-01ENZ0(00)
March 2014, 3rd Version	J2UL-1796-02ENZ0(00)
February 2012, 2nd Version	—
April 2007, 1st Version	—

Copyright

Update History

Changes	Location	Version
The following routine was added. • DS_V3DRCF3	SSL II/MPI Subroutine List, Usage of Subroutines	3rd Version
• The parameter NW's complemental explanation was added. • Description of Notes 2) was revised.	DS_V3DCFT3	
The parameter X’s explanation was corrected.	DS_V3DRCF	
Rework format	Cover, Preface	4th Version
The following routine was added. • DS_V3DCFT2X • DS_V3DRCF2X	SSL II/MPI Subroutine List, Usage of Subroutines	5th Version
Changed the look according to product upgrades.	-	6th Version
Single precision routines are added.	SS_V3DCFT2X, SS_V3DRCF2X	Version 6.1
Correction of a wrong word.	DS_V3DCFT2X, DS_V3DRCF2X	

SSL II/MPI Subroutine List

Fourier transforms

Subroutine name	Item	Page
DS_V3DCFT	Three-dimensional discrete complex Fourier transforms (mixed radix of 2, 3, 5 and 7, slab decomposition)	8
DS_V3DCFT2X	Three-dimensional discrete complex Fourier transforms (mixed radix of 2, 3, 5 and 7, pencil decomposition)	13
DS_V3DCFT3	Three-dimensional discrete complex Fourier transforms (mixed radix of 2, 3, 5 and 7, volumetric decomposition)	21
DS_V3DRCF	Three-dimensional discrete real Fourier transforms (mixed radix of 2, 3, 5 and 7, slab decomposition)	27
DS_V3DRCF2X	Three-dimensional discrete real Fourier transforms (mixed radix of 2, 3, 5 and 7, pencil decomposition)	33
DS_V3DRCF3	Three-dimensional discrete real Fourier transforms (mixed radix of 2, 3, 5 and 7, volumetric decomposition)	41
SS_V3DCFT2X *	Three-dimensional discrete complex Fourier transforms (mixed radix of 2, 3, 5 and 7, pencil decomposition, single precision)	48
SS_V3DRCF2X *	Three-dimensional discrete real Fourier transforms (mixed radix of 2, 3, 5 and 7, pencil decomposition, single precision)	56

Contents

Part I
General Description

Chapter 1
Outline

SSL II/MPI is a mathematical software library for parallel execution in a parallel computer with a distributed memory system. It provides the subroutines needed to efficiently compute large-scale problems by parallel processing.

Each SSL II/MPI function is supplied as a subroutine in Fortran. Every subroutine can be referred with a CALL statement.

The functional range, subroutine names, and calling mode of SSL II/MPI are different from those used in the mathematical software library SSL II of the uni-processor version.

Chapter 2
General Rules

2.1 Precision of Subroutines

SSL II/MPI provides subroutines that compute in double-precision, and some routines support the corresponding single-precision.

2.2 Subroutine Names

SSL II/MPI consists of user subroutines, that are callable by users and slave subroutines. Each user subroutine has DS_V or SS_V in the first four characters of its subroutine name. Each slave subroutine name has DS_U or SS_U in the first four characters.

This manual describes how to use user subroutines.

2.3 Parameters

(1) Order of parameters sequence

In general, the order of parameters sequence is the same as that in SSL II:

(input and output parameter list, input parameter list, output parameter list, ICON)

(2) Parameter types

Parameters beginning with I, J, K, L, M, or N are integer type. Parameters beginning with other characters are of the real or complex floating point type. Please refer to the parameter description of each subroutine for the number of bytes of the integer type and the single or double precision of the floating point type.

2.4 How to Use SSL II/MPI

(1) The subroutines in SSL II/MPI can be available after the initialization of MPI environment by calling MPI_INIT.

The communicator is specified in the argument of the subroutines. The subroutines make parallel computation in use of the processes belong to the communicator.

The data must be distributed among the processes belong to the communicator.

(2) An example how to use SSL II/MPI

a. Three dimensional complex Fourier transform is considered as an example.

A double precision complex three dimensional array X(KX1, KX2, KX3P) is allocated in each process. The complex three dimensional data to be transformed is assumed as a matrix of D(N1, N2, N3).

The (*rank*+1)-th sub matrix, into which D is partitioned in the third dimension equally by the size KX3P, is stored in the array X on the process of the rank(0, ..., *p*-1) (*p* is the total number of processes) obtained by MPI_RANK of an MPI subroutine.

X(1:N1, 1:N2, 1:N3P) ← D(1:N1, 1:N2, N3S:N3E) where N3S = KX3P×*rank*+1, N3E = MIN(N3, KX3P×(*rank*+1)), N3P = MAX(0, N3E-N3S+1).

The computation is done in parallel using the distributed data along processes as above.

```
c     ** example program **
      use mpi
      implicit  real*8 (a-h,o-z)
      parameter (mpn=8)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (kx1=n1+1)
      parameter (kw2p=((n2+mpn-1)/mpn),kx2=kw2p*mpn)
      parameter (kx3p=((n3+mpn-1)/mpn),kw3=kx3p*mpn)
      parameter (nwork=388)
      complex*16 x(kx1,kx2,kx3p),w(kx1,kw2p,kw3),
     $           wc(kx1,kx2,kx3p)
      real*8 dwork(nwork)
c
      call mpi_init( ierr )
      call mpi_comm_size( mpi_comm_world, nump, ierr )
      call mpi_comm_rank( mpi_comm_world, nop, ierr )
      nop=nop+1
c
      ix=1000
c
      ix=ix*nump+nop      ! different seed
      do i1=1,kx3p
      call dvrau4(ix,x(1,1,i1),
     $               2*kx1*kx2,
     &               dwork, nwork, icon)
      enddo
c
      do i3=1, kx3p
      do i2=1, n2
      do i1=1, n1
      wc(i1,i2,i3)=x(i1,i2,i3)
      enddo
      enddo
      enddo
c
      isn=1
      call ds_v3dcft(x,kx1,kx2,kx3p,
     $           n1,n2,n3,w,kw2p,kw3,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
cc
      print*,'icon=',icon
cc
c
      isn=-1
      call ds_v3dcft(x,kx1,kx2,kx3p,
     $           n1,n2,n3,w,kw2p,kw3,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
cc
      print*,'icon=',icon
c
      errorx=0
      iof3=(nop-1)*kx3p
      do i1=1,n1
      do i2=1,n2
      do i3=1,min(kx3p,max(n3-iof3,0))
      errorx=max(dabs(dble(wc(i1,i2,i3))-
     $       dble(x(i1,i2,i3))/n1/n2/n3),errorx)
      errorx=max(dabs(dimag(wc(i1,i2,i3))-
     $       dimag(x(i1,i2,i3))/n1/n2/n3),errorx)
      enddo
      enddo
      enddo
c
      call mpi_allreduce(errorx,errormax,1,mpi_double_precision,
     $                   mpi_max,mpi_comm_world,ierr)
c
      if(nop.eq.1)then
c
      print*,'-----(',n1,',',n2,',',n3,')-----'
      print*,'error=',errormax
c
      endif
 9000 continue
c
      call mpi_finalize( ierr )
c
      stop
      end
```

Example 2.1

Example of using an SSL II/MPI routine

Fourier transforms (normal and inverse transforms) for three-dimensional complex data are computed in 8 processes.

b. Parallel computation in threads within each process

Computation in each process can be parallelized in use of threads. The number of threads can be specified in the environment variable OMP_NUM_THREADS.

2.5 Condition Codes

The ICON parameter is prepared to indicate the status after the execution of SSL II/MPI.

A value between 0 and 90000 is set as the condition code. The values are classified as shown below depending on whether the result is guaranteed.

Table 2.1 Condition codes

Code	Meaning	Integrity of the result	Classification
0	Processing has ended normally.	The results are correct.	Normal
1 to 9999	Processing has ended normally, but auxiliary information was included.		
10000 to 19999	Processing has ended with the placing of internal restrictions during execution.	The results are correct on the restrictions.	Warning
20000 to 29999	Processing was discontinued due to abnormal conditions which had occurred during execution.	The results are not correct.	Abnormal
30000 to 39999	Processing was discontinued due to invalid input parameter.		
90000	This routine was deleted from the library. Use the currently supported, superior routine. This comment is also indicated in a message output together with these two routine names to the standard error output file. This message is shown below.	—	—

Message output at ICON=90000:

```
JNO0001-S : The SSL II/MPI routine ‘AAAAA’ no longer available, the
better one ‘BBBBB’ now recommended for use.
```

Where,

AAAAA	: Name of deleted routine
BBBBB	: Name of superior routine

Part II
Usage of Subroutines

DS_V3DCFT

Three-dimensional discrete complex Fourier transforms. (mixed radices of 2, 3, 5 and 7, slab decomposition)
CALL DS_V3DCFT (X, KX1, KX2, KX3P, N1, N2, N3, W, KW2P, KW3, ISN, COMM, ICON)

(1) Function

The subroutine DS_V3DCFT performs a three-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3}$$
$$\begin{aligned}&, k_1 = 0, 1, ..., n_1 - 1\\ &, k_2 = 0, 1, ..., n_2 - 1\\ &, k_3 = 0, 1, ..., n_3 - 1\\ &, \omega_{n1} = \exp(2\pi i/n_1)\\ &, \omega_{n2} = \exp(2\pi i/n_2)\\ &, \omega_{n3} = \exp(2\pi i/n_3)\end{aligned} \tag{1.1}$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3}$$
$$\begin{aligned}&, j_1 = 0, 1, ..., n_1 - 1\\ &, j_2 = 0, 1, ..., n_2 - 1\\ &, j_3 = 0, 1, ..., n_3 - 1\\ &, \omega_{n1} = \exp(2\pi i/n_1)\\ &, \omega_{n2} = \exp(2\pi i/n_2)\\ &, \omega_{n3} = \exp(2\pi i/n_3)\end{aligned} \tag{1.2}$$

(2) Parameters

X .... Input. Complex data.

Three dimensional complex data is regarded as a matrix data of D(N1, N2, N3).

The matrix data D is equally partitioned in the third dimension by the size of KX3P into submatrices.

This (*rank*+1)-th submatrix is stored in an array X on a process of the rank (0, ... , $p-1$) (p is the total number of processes), which is known in use of the subroutine MPI_COMM_RANK of MPI.

X(1:N1, 1:N2, 1:N3P) ← D(1:N1, 1:N2, N3S:N3E), where N3S = KX3P×*rank*+1, N3E = MIN(N3, KX3P×(*rank*+1)), N3P = MAX(0, N3E - N3S+1).

Output. Transformed complex data.

Resultant transformed three dimensional data of D(N1, N2, N3) are stored into an array X in the same distributed way.

This is a double precision complex three-dimensional array X(KX1, KX2, KX3P).

KX1 .... Input. The size of the first dimension of arrays X and W(≥ N1).

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of an array X.

KX2 = KW2P×p (≥ N2), and p is the total number of processes.

Integer(INTEGER*4).

KX3P .... Input. The size of the third dimension of an array X.

The size by which the third dimension is equally partitioned (KX3P×p≥ N3, and p is the total number of processes.).

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 .... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 .... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

W .... Work area. This is double precision complex three dimensional array W(KX1, KW2P, KW3).

KW2P .... Input. The size of the second dimension of an array W.

(KW2P×p≥ N2, and p is the total number of processes.)

Integer (INTEGER*4)

KW3 .... Input. The size of the third dimension of an array W.

KW3 = KX3P×*p*, and *p* is the total number of processes.

Integer(INTEGER*4).

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

COMM .... Input. The communicator indicating a set of processes on which data are distributed and by which computation is done in parallel.

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DCFT-1.

Table DS_V3DCFT-1 Condition codes

Code	**Meaning**	**Processing**
0	No error	
30000	N1<1, N2<1, N3<1, KX1<N1, KX2≠KW2P×*p*, KX3P×*p*≠KW3, KX2<N2, KW3<N3, the value of ISN is incorrect.	Processing is discontinued.
30008	The order of the transform is not radix 2/3/5/7.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1, \ldots, n_1 - 1$$
$$, k_2 = 0,\ 1, \ldots, n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1, \ldots, n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1, \ldots, n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0,\ 1, \ldots, n_2 - 1$$
$$, j_3 = 0,\ 1, \ldots, n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) The relation between the size of array X and W, and the performance.

The data amounts for communication become minimal when KX1 is close to N1, KX2=KW2P×p is the minimum value satisfying KX2≥N2 and KW3=KX3P×p is the minimum value satisfying KW3≥N3. p is the total number of processes.

b. Example

Three-dimensional FFT is computed in 8 processes.

```
c     ** example program **
      use mpi
      implicit  real*8 (a-h,o-z)
      parameter (mpn=8)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (kx1=n1+1)
      parameter (kw2p=((n2+mpn-1)/mpn),kx2=kw2p*mpn)
      parameter (kx3p=((n3+mpn-1)/mpn),kw3=kx3p*mpn)
      parameter (nwork=388)
      complex*16 x(kx1,kx2,kx3p),w(kx1,kw2p,kw3),
     $           wc(kx1,kx2,kx3p)
      real*8 dwork(nwork)
c
      call mpi_init( ierr )
      call mpi_comm_size( mpi_comm_world, nump, ierr )
      call mpi_comm_rank( mpi_comm_world, nop, ierr )
      nop=nop+1
c
      ix=1000
c
      ix=ix*nump+nop      ! different seed
      do i1=1,kx3p
      call dvrau4(ix,x(1,1,i1),
     $               2*kx1*kx2,
     &               dwork, nwork, icon)
      enddo
c
      do i3=1, kx3p
      do i2=1, n2
      do i1=1, n1
      wc(i1,i2,i3)=x(i1,i2,i3)
      enddo
      enddo
      enddo
c
      isn=1
      call ds_v3dcft(x,kx1,kx2,kx3p,
     $           n1,n2,n3,w,kw2p,kw3,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
cc
```

```
      print*,'icon=',icon
cc
c
      isn=-1
      call ds_v3dcft(x,kx1,kx2,kx3p,
     $           n1,n2,n3,w,kw2p,kw3,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
cc
      print*,'icon=',icon
c
      errorx=0
      iof3=(nop-1)*kx3p
      do i1=1,n1
      do i2=1,n2
      do i3=1,min(kx3p,max(n3-iof3,0))
      errorx=max(dabs(dble(wc(i1,i2,i3))-
     $       dble(x(i1,i2,i3))/n1/n2/n3),errorx)
      errorx=max(dabs(dimag(wc(i1,i2,i3))-
     $       dimag(x(i1,i2,i3))/n1/n2/n3),errorx)
      enddo
      enddo
      enddo
c
      call mpi_allreduce(errorx,errormax,1,mpi_double_precision,
     $                   mpi_max,mpi_comm_world,ierr)
c
      if(nop.eq.1)then
c
      print*,'-----(',n1,',',n2,',',n3,')-----'
      print*,'error=',errormax
c
      endif
 9000 continue
c
      call mpi_finalize( ierr )
c
      stop
      end
```

DS_V3DCFT2X

Three-dimensional discrete complex Fourier transforms. (mixed radices of 2, 3, 5 and 7, pencil decomposition)
CALL DS_V3DCFT2X (X, KX1, KX2, KX2P, KX3P, Z, KZ1, KZ2, KZ3, NZ1B, NZ1E, NZ2B, NZ2E, N1, N2, N3, W, NW, ISN, IDIR, COMM2, COMM3, ICON)

(1) Function

The subroutine DS_V3DCFT2X performs a three-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} &= \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\,\omega_{n1}^{-j1k1}\,\omega_{n2}^{-j2k2}\,\omega_{n3}^{-j3k3} \\
&, k_1 = 0,\ 1,\dots, n_1-1 \\
&, k_2 = 0,\ 1,\dots, n_2-1 \\
&, k_3 = 0,\ 1,\dots, n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3)
\end{aligned} \tag{1.1}
$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$
\begin{aligned}
x_{j1j2j3} &= \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\,\omega_{n1}^{j1k1}\,\omega_{n2}^{j2k2}\,\omega_{n3}^{j3k3} \\
&, j_1 = 0,\ 1,\dots, n_1-1 \\
&, j_2 = 0,\ 1,\dots, n_2-1 \\
&, j_3 = 0,\ 1,\dots, n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3)
\end{aligned} \tag{1.2}
$$

This subroutine provides an efficient and scalable 3D FFT functionality using pencil decomposition. The global data of the three-dimensional array is to be distributed among a two-dimensional process grid. The local input array and output array of each process store distinct shapes by dividing the global data in different directions of pencils. This

allows the routine to omit communication for transposing back to the original distribution shape.

(2) Parameters

In the following, the global three dimensional complex data is regarded as an array D(N1, N2, N3) virtually, and the process grid is regarded as ND2 × ND3 shape.

X .... Input when IDIR = 1. Complex data.

The local array X stores a subarray of D, which corresponds to a columnwise decomposed part of the global array by dividing the second and third dimensions by ND2 and ND3 respectively.

If each N*m* is divisible by ND*m* (*m*=2,3), setting a width parameter KX*m*P=N*m*/ND*m* is recommended, then the sizes of the subarray can be (N1,KX2P,KX3P) uniformly.

If N*m* is not divisible by ND*m*, setting KX*m*P=N*m*/ND*m*+1 is recommended, then the sizes of the subarrays that are assigned to the edge of the process grid are less than KX*m*P.

The array X in each process stores the subarray of D as follows:
X(1:N1, 1:NX2P, 1:NX3P) ← D(1:N1, NX2B:NX2E, NX3B:NX3E)
  NX2B = KX2P × *rank2* + 1
  NX2E = MIN(N2, KX2P × (*rank2* + 1))
  NX2P = MAX(0, NX2E - NX2B + 1)
  NX3B = KX3P × *rank3* + 1
  NX3E = MIN(N3, KX3P × (*rank3* + 1))
  NX3P = MAX(0, NX3E - NX3B + 1),

where the *rank2* and *rank3* are ranks of the process in the communicator COMM2 and COMM3 respectively, which are obtained by subroutine MPI_COMM_RANK of MPI.

The input values are not retained after the calculation.

..........Output when IDIR = -1. Transformed complex data.

Resultant transformed three dimensional data of D(N1, N2, N3) are stored into the array X in the same distributed way as stated above.

This is a double precision complex three-dimensional array X(KX1, KX2, NX3P).

KX1 .... Input. The size of the first dimension of the array X (≥ N1).

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of the array X (≥ NX2P).

Integer(INTEGER*4).

KX2P .... Input. The size by which the second dimension of D is equally partitioned when the data are stored in the array X. (KX2P×ND2≥ N2)

Integer(INTEGER*4).

KX3P .... Input. The size by which the third dimension of D is equally partitioned when the data are stored in the array X. (KX3P×ND3≥ N3)

Integer(INTEGER*4).

Z .... Output when IDIR = 1. Transformed complex data.

The local array Z stores a subarray of D, which corresponds to a columnwise decomposed part of the global array by dividing the first and second dimensions by ND2 and ND3 respectively.

The array Z in each process stores the subarray of D as follows:
Z(1:NZ1P, 1:NZ2P, 1:N3) ← D(NZ1B:NZ1E, NZ2B:NZ2E, 1:N3)
  NZ1P = MAX(0, NZ1E − NZ1B + 1)
  NZ2P = MAX(0, NZ2E − NZ2B + 1).

.... Input when IDIR = -1. Complex data.

The local array Z stores the subarray of D in the same distributed way as stated above.

This is a double precision complex three-dimensional array Z(KZ1, KZ2, KZ3).

KZ1 .... Output when NW = 0. The recommended size for the first dimension of the array Z.

Input when NW ≠ 0. The size of the first dimension of the array Z (≥ NZ1E-NZ1B+1).

Integer(INTEGER*4).

KZ2 .... Output when NW = 0. The recommended size for the second dimension of the array Z.

Input when NW ≠ 0. The size of the second dimension of the array Z (≥ NZ2E-NZ2B+1).

Integer(INTEGER*4).

KZ3 .... Output when NW = 0. The recommended size of the third dimension of the array Z.

Input when NW ≠ 0. The size of the third dimension of the array Z (≥ N3).

Integer(INTEGER*4).

NZ1B .... Output. The starting index for the first dimension of the global array D.

Integer(INTEGER*4).

NZ1E .... Output. The ending index for the first dimension of the global array D.
NZ1B and NZ1E indicate which portion of the first dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

NZ2B .... Output. The starting index for the second dimension of the global array D.

Integer(INTEGER*4).

NZ2E .... Output. The ending index for the second dimension of the global array D.
NZ2B and NZ2E indicate which portion of the second dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 .... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 .... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

W .... Work area. This is double precision complex one-dimensional array W(NW).

NW .... Input / Output. The size of the work array W.

When NW = 0 specified, the recommended sizes of NW, KZ1, KZ2, and KZ3 are set respectively, and index information is set to NZ1B, NZ1E, NZ2B, and NZ2E.

Integer (INTEGER*8). (See note 4) in (3), “Comments on use.”)

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

IDIR .... Input. The direction of transform between arrays is indicated.

IDIR = 1 for the transform from the array X to the array Z.

IDIR = -1 for the transform from the array Z to the array X.

Integer (INTEGER*4).

COMM2 ...... Input. The MPI communicator that represents a set of processes whose size of the process group is ND2, which is obtained by MPI_COMM_SIZE, in the process shape ND2 × ND3. (See note 2) in (3), “Comments on use.”)

Integer (INTEGER*4).

COMM3 ...... Input. The MPI communicator that represents a set of processes whose size of the process group is ND3, which is obtained by MPI_COMM_SIZE, in the process shape ND2 × ND3. (See note 2) in (3), “Comments on use.”)

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DCFT2X-1.

Integer (INTEGER*4).

Table DS_V3DCFT2X-1 Condition codes

Code	Meaning	Processing
0	No error	

Table DS_V3DCFT2X-1 Condition codes

Code	Meaning	Processing
1000	NW = 0 is specified.	The recommended sizes of NW, KZ1, KZ2, and KZ3 are set. Index information is set to NZ1B, NZ1E, NZ2B, and NZ2E. There is no output to array X or Z.
25000	Too small work area.	Processing is discontinued.
30000	N1<1, N2<1, N3<1, KX1 < N1, KX2 < NX2P, N2 > KX2P × ND2, N3 > KX3P × ND3, or the value of ISN or IDIR is incorrect.	
30008	The order of the transform is not radix 2/3/5/7.	
30100	COMM2 or COMM3 is incorrect.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1, ..., n_1 - 1$$
$$, k_2 = 0,\ 1, ..., n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1, ..., n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0,\ 1, ..., n_2 - 1$$
$$, j_3 = 0,\ 1, ..., n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) Process shape ND2 and ND3

Note that the performance of this routine may deteriorate when ND2 and ND3 do not match the shape of the executing process grid on a system which can specify the shape of the process grid. Refer to the Job Operation Software manual whether the system can assign a shape of the process grid.

3) Consistency of parameters among processes

The parameters KX2P,KX3P,N1,N2,N3,NW,ISN, and IDIR needs to have same value respectively among all processes, otherwise the result is not guaranteed.

4) The size of work area W

The size of the work array NW needs to be about twice the size of array X or Z to be used as send/receive buffers for MPI communication inside the routine. Note that the parameter NW is an 8-byte integer type.

b. Example

Three-dimensional FFT is computed in 2×3 processes.

```
c     ** example program **
      use mpi
      implicit  real*8 (a-h,o-z)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (nd2=2,nd3=3)
      parameter (kx1=n1)
      parameter (kx2p=((n2+nd2-1)/nd2),kx2=kx2p)
      parameter (kx3p=((n3+nd3-1)/nd3))
      parameter (nwrand=388)
      real*8 dwork(nwrand)
      integer comm2,comm3
      integer*8 nw
      complex*16 x(kx1,kx2,kx3p),wc(kx1,kx2,kx3p)
      complex*16,allocatable :: z(:,:,:),w(:)
c     --- prepare sub-communicator ---
      call mpi_init(ierr)
      call mpi_comm_size(mpi_comm_world, nsize, ierr)
      call mpi_comm_rank(mpi_comm_world, nrank, ierr)
      ncolory=nrank/nd2
      call mpi_comm_split(mpi_comm_world,ncolory,nrank,
     &                    comm2,ierr)
      call mpi_comm_size(comm2, nsize2, ierr)
      call mpi_comm_rank(comm2, nrank2, ierr)
      ncolorz=mod(nrank,nd2)
      call mpi_comm_split(mpi_comm_world,ncolorz,nrank,
     &                    comm3,ierr)
      call mpi_comm_size(comm3, nsize3, ierr)
      call mpi_comm_rank(comm3, nrank3, ierr)
      if(nsize.ne.nd2*nd3 .or. nsize2.ne.nd2 .or.
     &   nsize3.ne.nd3) then
        print*,'nsize=',nsize,nsize2,nsize3
        go to 9000
      endif
c     --- prepare test-data ---
      nx2=min(kx2p,max(n2-nrank2*kx2p,0))
      nx3=min(kx3p,max(n3-nrank3*kx3p,0))
      ix=1000
      ix=ix+nrank      ! different seed
      do i3=1,nx3
        do i2=1,nx2
          call dvrau4(ix,x(1,i2,i3),2*n1,dwork,nwrand,icon)
          do i1=1,n1
            wc(i1,i2,i3)=x(i1,i2,i3)
          enddo
        enddo
      enddo
c     --- inquire necessary size ---
      nw=0
      call ds_v3dcft2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &           nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &           comm2,comm3,icon)
      if(icon.ne.1000) then
        print*,'icon=',icon
        go to 9000
      endif
      allocate (z(kz1,kz2,kz3),w(nw))
      print*,'nrank,nrank2,nrank3=',nrank,nrank2,nrank3,
     &       ' Z-pencil x-range=',nz1b,nz1e,
     &       ' y-range=',nz2b,nz2e,
     &       ' z-range=',1,n3
c     --- forward FFT ---
      idir=1
      isn=1
      call ds_v3dcft2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &           nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &           comm2,comm3,icon)
      if(icon.ne.0) then
        print*,'icon=',icon
        go to 9000
      endif
c     --- backward FFT ---
      idir=-1
      isn=-1
      call ds_v3dcft2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &           nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &           comm2,comm3,icon)
      if(icon.ne.0) then
        print*,'icon=',icon
        go to 9000
      endif
c     --- check result ---
      errorx=0
      do i3=1,nx3
        do i2=1,nx2
          do i1=1,n1
            errorx=max(cdabs(wc(i1,i2,i3)-
     &             x(i1,i2,i3)/n1/n2/n3),errorx)
          enddo
        enddo
      enddo
      call mpi_allreduce(errorx,errormax,1,mpi_double_precision,
     &                   mpi_max,mpi_comm_world,ierr)
      if(nrank.eq.0)then
        print*,'num proc=',nsize
        print*,'nd2,nd3=',nsize2,nsize3
        print*,'-----(',n1,',',n2,',',n3,')-----'
        print*,'error=',errormax
      endif
c
 9000 continue
      call mpi_comm_free(comm2,ierr)
      call mpi_comm_free(comm3,ierr)
      call mpi_finalize(ierr)
      stop
      end
```

DS_V3DCFT3

Three-dimensional discrete complex Fourier transforms. (mixed radices of 2, 3, 5 and 7, volumetric decomposition)
CALL DS_V3DCFT3 (X, KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, W, NW, ISN, COMM, ICON)

(1) Function

The subroutine DS_V3DCFT3 performs a three-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3}=\sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1}x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3}$$

$$\begin{aligned}&,k_1=0,\ 1,...,n_1-1\\&,k_2=0,\ 1,...,n_2-1\\&,k_3=0,\ 1,...,n_3-1\\&,\omega_{n1}=\exp(2\pi i/n_1)\\&,\omega_{n2}=\exp(2\pi i/n_2)\\&,\omega_{n3}=\exp(2\pi i/n_3)\end{aligned}\qquad(1.1)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3}=\sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1}\alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3}$$

$$\begin{aligned}&,j_1=0,\ 1,...,n_1-1\\&,j_2=0,\ 1,...,n_2-1\\&,j_3=0,\ 1,...,n_3-1\\&,\omega_{n1}=\exp(2\pi i/n_1)\\&,\omega_{n2}=\exp(2\pi i/n_2)\\&,\omega_{n3}=\exp(2\pi i/n_3)\end{aligned}\qquad(1.2)$$

This subroutine provides an efficient and scalable 3D FFT functionality on a massively parallel machine. The global data of the three-dimensional array can be distributed among processes which are regarded as a three-dimensional grid, therefore the volumetric decomposition allows the distribution of work more efficiently than a slabwise decomposition in an environment where massively parallel processes are available.

(2) Parameters

X .... Input. Complex data.

When the global three dimensional complex data is regarded as an array D(N1, N2, N3) virtually, the local array X stores subarray of D distributed along with the shape of the process grid.

If each N*m* is divisible by ND*m* (*m*=1,2,3), setting a width parameter KX*m*P=N*m*/ND*m* is recommended, then the sizes of the subarray can be (KX1P,KX2P,KX3P) uniformly.

If N*m* is not divisible by ND*m*, setting KX*m*P=N*m*/ND*m*+1 is recommended, then the sizes of the subarrays that are assigned to the edge of the process grid are less than KX*m*P.

The array X in each process stores the subarray of D as follows:
X(1:N1P, 1:N2P, 1:N3P) ← D(N1S:N1E, N2S:N2E, N3S:N3E),
$N1S = KX1P \times rank1 + 1$,
$N1E = MIN(N1, KX1P \times (rank1 + 1))$,
$N1P = MAX(0, N1E - N1S + 1)$,
$N2S = KX2P \times rank2 + 1$,
$N2E = MIN(N2, KX2P \times (rank2 + 1))$,
$N2P = MAX(0, N2E - N2S + 1)$,
$N3S = KX3P \times rank3 + 1$,
$N3E = MIN(N3, KX3P \times (rank3 + 1))$,
$N3P = MAX(0, N3E - N3S + 1)$.

The *rank1*,*rank2* and *rank3* are coordinates of the process grid calculated as follows from the *rank* value, which is obtained by subroutine MPI_COMM_RANK of MPI.
rank1 = mod(*rank*, ND1),
rank2 = mod(*rank*/ND1, ND2),
rank3 = *rank*/(ND1×ND2).

Output. Transformed complex data.

Resultant transformed three dimensional data of D(N1, N2, N3) are stored into the array X in the same distributed way.

This is a double precision complex three-dimensional array X(KX1, KX2, KX3P).

KX1 .... Input. The size of the first dimension of the array X (≥ KX1P).

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of the array X (≥ KX2P).

Integer(INTEGER*4).

KX1P .... Input. The size by which the first dimension is equally partitioned.(KX1P×ND1≥ N1)

Integer(INTEGER*4).

KX2P .... Input. The size by which the second dimension is equally partitioned.(KX2P×ND2≥ N2)

Integer(INTEGER*4).

KX3P .... Input. The size by which the third dimension is equally partitioned (KX3P×ND3≥ N3)

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 .... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 .... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

ND1............. Input. The number of processes by which the first dimension is partitioned.

Integer (INTEGER*4)
(See note 2) in (3), “Comments on use.”)

ND2............. Input. The number of processes by which the second dimension is partitioned.

Integer (INTEGER*4)
(See note 2) in (3), “Comments on use.”)

ND3............. Input. The number of processes by which the third dimension is partitioned.

Integer (INTEGER*4)
(See note 2) in (3), “Comments on use.”)

W .... Work area. This is double precision complex one-dimensional array W(NW).

NW .... Input. The size of the work array W (NW $\geq$ MAX(KX1 $\times$ ND1, KX2P $\times$ ND2, KX3P $\times$ ND3) $\times$ 3). It is recommended to specify a sufficiently large size for efficiency. (See note 2) in (2), “Comments on use.”)

Integer (INTEGER*4)

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

COMM .... Input. The communicator indicating a set of processes on which data are distributed and by which computation is done in parallel.

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DCFT3-1.

Integer (INTEGER*4).

Table DS_V3DCFT3-1 Condition codes

Code	Meaning	Processing
0	No error	
25000	Too small work area.	Processing is discontinued.
30000	N1<1, N2<1, N3<1, KX1 < KX1P, KX2 < KX2P, N1 > KX1P × ND1, N2 > KX2P × ND2, N3 > KX3P × ND3, or the value of ISN is incorrect.	
30008	The order of the transform is not radix 2/3/5/7.	
30100	ND1×ND2×ND3 is not equal to total processes.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1, \ldots, n_1 - 1$$
$$, k_2 = 0,\ 1, \ldots, n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1, \ldots, n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1, \ldots, n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0,\ 1, \ldots, n_2 - 1$$
$$, j_3 = 0,\ 1, \ldots, n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) The size of work area W

The size of the work array determines partition sizes for transferring data among nodes and calculation on each node in this routine. Setting the size NW of work area W much larger than MAX(KX1×ND1, KX2P×ND2, KX3P×ND3) × 3 × (number of threads in a process) is recommended. For example, setting NW > 500,000 is expected to be efficient when the assigned cache size to the process is 8MB and the array X is partitionable by that size of the work area.

3) Parameters ND1, ND2 and ND3

When using volumetric decomposition, it is recommended to adjust ND1,ND2 and ND3 to be about the comparable value of cube root of the number of total process ND1×ND2×ND3 for overall efficiency of transferring data. Note that the performance of this routine may deteriorate when ND1,ND2 and ND3 do not match the shape of the executing process grid on a system which can specify the shape of the process grid. Refer to the Job Operation Software manual whether the system can assign a shape of the process grid.

Additionally, this routine can be used for slabwise decomposition or 2-dimensional decomposition also by setting any of ND1,ND2 or ND3 to 1, when user's program exploits specific decomposition or available shape of the process grid on a system is limited.

4) Consistency of parameters among processes

The parameters KX1,KX2,KX1P,KX2P,KX3P,N1,N2,N3,ND1,ND2,ND3,NW and ISN needs to have same value respectively among all processes, otherwise the result is not guaranteed.

b. Example

Three-dimensional FFT is computed in 2×2×2 processes.

```
c     ** example program **
      use mpi
c
      implicit  real*8 (a-h,o-z)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (nd1=2,nd2=2,nd3=2)
      parameter (kx1p=((n1+nd1-1)/nd1),kx1=kx1p)
      parameter (kx2p=((n2+nd2-1)/nd2),kx2=kx2p)
      parameter (kx3p=((n3+nd3-1)/nd3))
      parameter (nw=kx1*kx2*kx3p)
      parameter (nwork=388)
      real*8 dwork(nwork)
      complex*16 x(kx1,kx2,kx3p),w(nw),
     $           wc(kx1,kx2,kx3p)
c
      call mpi_init( ierr )
      call mpi_comm_size( mpi_comm_world, nump, ierr )
      call mpi_comm_rank( mpi_comm_world, nop, ierr )
      nrank1=mod(nop,nd1)
      nrank2=mod(nop/nd1,nd2)
      nrank3=nop/(nd1*nd2)
c
      ix=1000
      ix=ix*nump+nop      ! different seed
      do i1=1,kx3p
      call dvrau4(ix,x(1,1,i1),
     $               2*kx1*kx2,
     &               dwork, nwork, icon)
      enddo
c
      do i3=1, kx3p
      do i2=1, kx2p
      do i1=1, kx1p
      wc(i1,i2,i3)=x(i1,i2,i3)
      enddo
      enddo
      enddo
c
      isn=1
      call ds_v3dcft3(x,kx1,kx2,kx1p,kx2p,kx3p,
     $           n1,n2,n3,nd1,nd2,nd3,w,nw,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
cc
      print*,'icon=',icon
c
      isn=-1
      call ds_v3dcft3(x,kx1,kx2,kx1p,kx2p,kx3p,
     $           n1,n2,n3,nd1,nd2,nd3,w,nw,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
cc
      print*,'icon=',icon
c
      errorx=0
      iof1=nrank1*kx1p
      iof2=nrank2*kx2p
      iof3=nrank3*kx3p
      do i1=1,min(kx1p,max(n1-iof1,0))
      do i2=1,min(kx2p,max(n2-iof2,0))
      do i3=1,min(kx3p,max(n3-iof3,0))
      errorx=max(dabs(dble(wc(i1,i2,i3))-
     $       dble(x(i1,i2,i3))/n1/n2/n3),errorx)
      errorx=max(dabs(dimag(wc(i1,i2,i3))-
     $       dimag(x(i1,i2,i3))/n1/n2/n3),errorx)
      enddo
      enddo
      enddo
c
      call mpi_allreduce(errorx,errormax,1,mpi_double_precision,
     $                  mpi_max,mpi_comm_world,ierr)
c
      if(nop.eq.1)then
      print*,'-----(',n1,',',n2,',',n3,')-----'
      print*,'error=',errormax
      endif
 9000 continue
c
      call mpi_finalize( ierr )
c
      stop
      end
```

DS_V3DRCF

Three-dimensional discrete real Fourier transforms. (mixed radices of 2, 3, 5 and 7, slab decomposition)
CALL DS_V3DRCF(X, KX1, KX2, KX3P, N1, N2, N3, W, KW2P, KW3, ISIN, ISN, COMM, ICON)

(1) Function

The subroutine DS_V3DRCF performs a three-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of the three-dimensional array (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3}=\sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1}x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r}$$

$$\begin{aligned}&,k_1=0,\ 1,...,n_1-1\\&,k_2=0,\ 1,...,n_2-1\\&,k_3=0,\ 1,...,n_3-1\\&,\omega_{n1}=\exp(2\pi i/n_1)\\&,\omega_{n2}=\exp(2\pi i/n_2)\\&,\omega_{n3}=\exp(2\pi i/n_3)\\&r=1\text{ or }r=-1\end{aligned}\qquad(1.1)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3}=\sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1}\alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r}$$

$$\begin{aligned}&,j_1=0,\ 1,...,n_1-1\\&,j_2=0,\ 1,...,n_2-1\\&,j_3=0,\ 1,...,n_3-1\\&,\omega_{n1}=\exp(2\pi i/n_1)\\&,\omega_{n2}=\exp(2\pi i/n_2)\\&,\omega_{n3}=\exp(2\pi i/n_3)\\&r=1\text{ or }r=-1\end{aligned}\qquad(1.2)$$

(2) Parameters

X .... Input/Output. Three-dimensional real data.

Three dimensional real data is regarded as a matrix data of D(N1, N2, N3).

The matrix data D is equally partitioned in the third dimension by the size of KX3P into submatrices.

This (*rank*+1)-th submatrix is stored in an array X on a process of the rank (0, ... , $p-1$) (p is the total number of processes), which is known in use of the subroutine MPI_COMM_RANK of MPI.

X(1:N1, 1:N2, 1:N3P) ← D(1:N1, 1:N2, N3S:N3E)

Where N3S = KX3P×*rank*+1, N3E = MIN(N3, KX3P×(*rank*+1)), N3P = MAX(0, N3E - N3S+1).

For the real to complex transform (ISN = 1), data is input; for the complex to real transform (ISN = -1), data is output.

Output/input. The real and imaginary parts of the transformed complex data.

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = -1), data is input.

The complex data CD(N1, N2, N3) obtained from real data D(N1, N2, N3) by Fourier transform has the complex conjugate relation so the about a half of the first dimension(1〜N1/2+1) is used to store the complex data.

(See note 2) in (3), "Comments on use.")

Regarding an array X as X(2, KX1/2, KX2, KX3P), the real and imaginary parts are stored in X(1, 1:N1/2+1, 1:N2, 1:N3) and X(2, 1:N1/2+1, 1:N2, 1:N3) respectively.

This is a double precision real three dimensional array X(KX1, KX2, KX3P).

KX1 .... Input. The size of the first dimension of array X and W(≥ 2×N1/2+1). KX1 must be even.

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of an array X.

KX2 = KW2P×p, where p is the total number of processes (≥ N2).

Integer(INTEGER*4).

KX3P .... Input. The size of the third dimension of an array X.

The size by which the third dimension is equally partitioned (KX3P×p≥ N3, and p is the total number of processes.).

Integer(INTEGER*4).

N1 ....Input. The length n_1 of real data in the first dimension to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 ....Input. The length n_2 of real data in the second dimension to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 ....Input. The length n_3 of real data in the third dimension to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

W .... Work area. This is double precision real three dimensional array W(KX1, KW2P, KW3).

KW2P .... Input. The size of the second dimension of an array W.

(KW2P×p≥ N2, and p is the total number of processes.)

Integer (INTEGER*4)

KW3 .... Input. The size of the third dimension of an array W.

KW3 = KX3P×p, and p is the total number of processes.

Integer(INTEGER*4).

ISIN .... Input. The direction of the transformation.

r = 1 for 1.

r = -1 for -1.

Integer (INTEGER*4)

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

COMM .... Input. The communicator indicating a set of processes on which data are distributed and by which computation is done in parallel.

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DRCF-1.

Table DS_V3DRCF-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	KX < 2×(N1/2+1), KX1 is not even, KX2≠KW2P×p, KX3P×p≠KW3, KX2<N2, KW3<N3, N1<1, N2<1, N3<1, ISIN≠1, -1, ISN≠1, -1.	Processing is discontinued.
30008	The order of the transform is not radix 2/3/5/7.	

(3) Comments on use

a Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1, ...,\ n_1 - 1$$
$$, k_2 = 0,\ 1, ...,\ n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1, ...,\ n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1, ..., n_1 - 1$$
$$, j_2 = 0,\ 1, ..., n_2 - 1 \qquad (3.2)$$
$$, j_3 = 0,\ 1, ..., n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left term of (3.1) or (3.2), respectively. The normalization of the results may be required.

2) The results of the three-dimensional real Fourier transform has the following complex conjugate relation (indicated by $^-$).

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\ n2-k2\ n3-k3}} \qquad (3.3)$$

The remainder of the data is obtained from data in $k_1 = 0, ..., n_1-1$, $k_2 = 0, ..., n_2-1$, and $k_3 = 0, ..., n_3/2$.

b Example

Three-dimensional real FFT is computed in 8 processes.

```
cc    ** example program **
      use mpi
      implicit  real*8 (a-h,o-z)
c
      parameter (mpn=8)
      parameter (n1=512,n2=n1,n3=n1)
      parameter (kx1=(n1/2+1)*2)
      parameter (kw2p=((n2+mpn-1)/mpn),kx2=kw2p*mpn)
      parameter (kx3p=((n3+mpn-1)/mpn),kw3=kx3p*mpn)
      parameter (nwork=388)
      real*8 x(kx1,kx2,kx3p),w(kx1,kw2p,kw3),
     $       wc(kx1,kx2,kx3p)
      real*8 dwork(nwork)
c
      call mpi_init( ierr )
      call mpi_comm_size( mpi_comm_world, nump, ierr )
      call mpi_comm_rank( mpi_comm_world, nop, ierr )
      nop=nop+1
c
      ix=1000
c
```

```
      ix=ix*nump+nop      ! different seed
      do i1=1,kx3p
      call dvrau4(ix,x(1,1,i1),
     $               kx1*kx2,
     &               dwork, nwork, icon)
      enddo
c
      do i3=1, kx3p
      do i2=1, n2
      do i1=1, n1
      wc(i1,i2,i3)=x(i1,i2,i3)
      enddo
      enddo
      enddo
c
      isin=1
      isn=1           ! real to complex
      call ds_v3drcf(x,kx1,kx2,kx3p,
     $           n1,n2,n3,w,kw2p,kw3,isin,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
      print*,'icon=',icon
c
      isin=1         ! same direction
      isn=-1         ! comprex to real
      call ds_v3drcf(x,kx1,kx2,kx3p,
     $           n1,n2,n3,w,kw2p,kw3,isin,isn,
     $           mpi_comm_world,icon)
      if(icon.ne.0) go to 9000
      print*,'icon=',icon
c
      iof3=(nop-1)*kx3p
      error=0
      do i1=1,n1
      do i2=1,n2
      do i3=1,min(kx3p,max(n3-iof3,0))
      error=max(dabs(wc(i1,i2,i3)-
     $       x(i1,i2,i3)/n1/n2/n3),error)
      enddo
      enddo
      enddo
      call mpi_allreduce(error,errormax,1,mpi_double_precision,
     $                   mpi_max,mpi_comm_world,ierr)
c
      if(nop.eq.1)then
c
      print*,'-----(',n1,',',n2,',',n3,')-----'
      print*,'error=',errormax
c
      endif
 9000 continue
c
      call mpi_finalize( ierr )
```

```
c
      stop
      end
```

DS_V3DRCF2X

Three-dimensional discrete real Fourier transforms. (mixed radices of 2, 3, 5 and 7, pencil decomposition)
CALL DS_V3DRCF2X (X, KX1, KX2, KX2P, KX3P, Z, KZ1, KZ2, KZ3, NZ1B, NZ1E, NZ2B, NZ2E, N1, N2, N3, W, NW, ISN, IDIR, COMM2, COMM3, ICON)

(1) Function

The subroutine DS_V3DRCF2X performs a three-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$\begin{aligned} n_1n_2n_3\alpha_{k1k2k3} &= \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \\ &, k_1 = 0,\ 1,\dots,n_1-1 \\ &, k_2 = 0,\ 1,\dots,n_2-1 \\ &, k_3 = 0,\ 1,\dots,n_3-1 \\ &, \omega_{n1} = \exp(2\pi i/n_1) \\ &, \omega_{n2} = \exp(2\pi i/n_2) \\ &, \omega_{n3} = \exp(2\pi i/n_3) \end{aligned} \tag{1.1}$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$\begin{aligned} x_{j1j2j3} &= \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \\ &, j_1 = 0,\ 1,\dots,n_1-1 \\ &, j_2 = 0,\ 1,\dots,n_2-1 \\ &, j_3 = 0,\ 1,\dots,n_3-1 \\ &, \omega_{n1} = \exp(2\pi i/n_1) \\ &, \omega_{n2} = \exp(2\pi i/n_2) \\ &, \omega_{n3} = \exp(2\pi i/n_3) \end{aligned} \tag{1.2}$$

This subroutine provides an efficient and scalable 3D FFT functionality using pencil decomposition. The global data of the three-dimensional array is to be distributed among a two-dimensional process grid. The local input array and output array of each process store distinct shapes by dividing the global data in different directions of pencils. This

allows the routine to omit communication for transposing back to the original distribution shape.

(2) Parameters

In the following, the global three dimensional real data is regarded as an array DR(N1, N2, N3), and the transformed global three dimensional complex data is regarded as an array DC(N1/2+1, N2, N3) virtually, and the process grid is regarded as ND2 × ND3 shape.

X .... Input when IDIR = 1. Real data.

The local array X stores a subarray of DR, which corresponds to a columnwise decomposed part of the global array by dividing the second and third dimensions by ND2 and ND3 respectively.

If each N*m* is divisible by ND*m* (*m*=2,3), setting width parameters KX*m*P=N*m*/ND*m* is recommended, then the data size settled in the subarray can be (N1,KX2P,KX3P) uniformly.

If N*m* is not divisible by ND*m*, setting KX*m*P=N*m*/ND*m*+1 is recommended, then the sizes of the subarrays that are assigned to the edge of the process grid are less than KX*m*P.

The array X in each process stores the subarray of DR as follows:
X(1:N1, 1:NX2P, 1:NX3P) ← DR(1:N1, NX2B:NX2E, NX3B:NX3E),
NX2B = KX2P × *rank2* + 1
NX2E = MIN(N2, KX2P × (*rank2* + 1))
NX2P = MAX(0, NX2E - NX2B + 1)
NX3B = KX3P × *rank3* + 1
NX3E = MIN(N3, KX3P × (*rank3* + 1))
NX3P = MAX(0, NX3E - NX3B + 1),

where, *rank2* and *rank3* are the ranks of the process in the communicator COMM2 and COMM3 respectively, which are obtained by subroutine MPI_COMM_RANK of MPI.

The input values are not retained after the calculation.

..........Output when IDIR = -1. Transformed real data.

Resultant transformed three dimensional real data from DC(N1/2+1, N2, N3) are stored into the array X in the same distributed way as stated above.

This is a double precision real three-dimensional array X(KX1, KX2, KX3P).

KX1 ....Input. The size of the first dimension of the array X. KX1 must be even. (≥ N1).

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of the array X (≥NX2P).

Integer(INTEGER*4).

KX2P .... Input. The size by which the second dimension of DR is equally partitioned. (KX2P×ND2≥ N2)

Integer(INTEGER*4).

KX3P .... Input. The size by which the third dimension of DR is equally partitioned. (KX3P×ND3≥ N3)

Integer(INTEGER*4).

Z .... Output when IDIR = 1. Transformed complex data.

The complex data obtained from real data DR(N1, N2, N3) by Fourier transform has the complex conjugate relation so the about a half of the first dimension is used to store. The local array Z stores a subarray of DC(N1/2+1, N2, N3), which corresponds to a columnwise decomposed part of the global array by dividing the first and second dimensions by ND2 and ND3 respectively.

The array Z in each process stores the subarray of DC as follows:
Z(1:NZ1P, 1:NZ2P, 1:N3) ← DC(NZ1B:NZ1E, NZ2B:NZ2E, 1:N3)
  NZ1P = MAX(0, NZ1E − NZ1B + 1)
  NZ2P = MAX(0, NZ2E − NZ2B + 1).

.... Input when IDIR = −1. Complex data.

The local array Z stores the subarray of DC in the same distributed way as stated above.

This is a double precision complex three-dimensional array Z(KZ1, KZ2, KZ3).

(See note 2) in (3), “Comments on use.”)

KZ1 .... Output when NW = 0. The recommended size for the first dimension of the array Z.

Input when NW ≠ 0. The size of the first dimension of the array Z (≥ NZ1E−NZ1B+1).

Integer(INTEGER*4).

KZ2 .... Output when NW = 0. The recommended size for the second dimension of the array Z.

Input when NW ≠ 0. The size of the second dimension of the array Z (≥ NZ2E−NZ2B+1).

Integer(INTEGER*4).

KZ3 .... Output when NW = 0. The recommended size of the third dimension of the array Z.

Input when NW ≠ 0. The size of the third dimension of the array Z (≥ N3).

Integer(INTEGER*4).

NZ1B .... Output. The starting index for the first dimension of the global array DC.

Integer(INTEGER*4).

NZ1E .... Output. The ending index for the first dimension of the global array DC. NZ1B and NZ1E indicate which portion of the first dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

NZ2B .... Output. The starting index for the second dimension of the global array DC.

Integer(INTEGER*4).

NZ2E .... Output. The ending index for the second dimension of the global array DC. NZ2B and NZ2E indicate which portion of the second dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2	Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed. n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7. Integer (INTEGER*4)
N3	Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed. n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7. Integer (INTEGER*4)
W	Work area. This is double precision complex one-dimensional array W(NW).
NW	Input / Output. The size of the work array W. When NW = 0 specified, the recommended sizes of NW, KZ1, KZ2, and KZ3 are set respectively, and the index information is set to NZ1B, NZ1E, NZ2B, and NZ2E. Integer (INTEGER*8). (See note 5) in (3), “Comments on use.”)
ISN	Input. Either the transform or the inverse transform is indicated. ISN = 1 for the transform. ISN = -1 for the inverse transform. Integer (INTEGER*4).
IDIR	Input. The direction of transform between arrays is indicated. IDIR = 1: from the real array X to the complex array Z. IDIR = -1: from the complex array Z to the real array X. Integer (INTEGER*4).
COMM2	Input. The MPI communicator that represents a set of processes whose size of the process group is ND2, which is obtained by MPI_COMM_SIZE, in the process shape ND2 × ND3. (See note 3) in (3), “Comments on use.”) Integer (INTEGER*4).
COMM3	Input. The MPI communicator that represents a set of processes whose size of the process group is ND3, which is obtained by MPI_COMM_SIZE, in the process shape ND2 × ND3. (See note 3) in (3), “Comments on use.”) Integer (INTEGER*4).
ICON	Output. Condition code. See Table DS_V3DRCF2X-1. Integer (INTEGER*4).

Table DS_V3DRCF2X-1 Condition codes

Code	Meaning	Processing

Table DS_V3DRCF2X-1 Condition codes

Code	Meaning	Processing
0	No error	
1000	NW = 0 is specified	The recommended sizes of NW, KZ1, KZ2 and KZ3 are set. Index information is set to NZ1B, NZ1E, NZ2B, and NZ2E. There is no output to array X or Z.
25000	Too small work area.	Processing is discontinued.
30000	N1<1, N2<1, N3<1, KX1 < N1, KX1 is not even, KX2 < NX2P, N2 > KX2P × ND2, N3 > KX3P × ND3, (N1/2+1) × 2 > KZ1 × ND2, ISN≠1, -1, IDIR≠1, -1.	
30008	The order of the transform is not radix 2/3/5/7.	
30100	COMM2 or COMM3 is incorrect.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1, ..., n_1 - 1$$
$$, k_2 = 0,\ 1, ..., n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1, ..., n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0,\ 1, ..., n_2 - 1$$
$$, j_3 = 0,\ 1, ..., n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) The results of the three-dimensional real Fourier transform has the following complex conjugate relation (indicated by $^-$).

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\ n2-k2\ n3-k3}} \tag{3.3}$$

The remainder of the data is obtained from data in $k_1 = 0, ..., n_1–1$, $k_2 = 0, ..., n_2–1$, and $k_3 = 0, ..., n_3/2$.

3) Process shape ND2 and ND3

Note that the performance of this routine may deteriorate when ND1,ND2 and ND3 do not match the shape of the executing process grid on a system which can specify the shape of the process grid. Refer to the Job Operation Software manual whether the system can assign a shape of the process grid.

4) Consistency of parameters among processes

The parameters KX2P, KX3P, N1, N2, N3, NW, ISN and IDIR needs to have same value respectively among all processes, otherwise the result is not guaranteed.

5) The size of work area W

The size of the work array NW needs to be about twice the size of array X or Z to be used as send/receive buffers for MPI communication inside the routine. Note that the parameter NW is an 8-byte integer type.

b. Example

Three-dimensional FFT is computed in 2×3 processes.

```
c      ** example program **
       use mpi
       implicit  real*8 (a-h,o-z)
       parameter (n1=512,n2=n1,n3=n2)
       parameter (nd2=2,nd3=3)
       parameter (n1c=n1/2+1,kx1=n1c*2)
       parameter (kx2p=((n2+nd2-1)/nd2),kx2=kx2p)
       parameter (kx3p=((n3+nd3-1)/nd3))
       parameter (nwrand=388)
       real*8 dwork(nwrand)
       integer comm2,comm3
       integer*8 nw
       real*8 x(kx1,kx2,kx3p),wc(kx1,kx2,kx3p)
       complex*16,allocatable :: z(:,:,:)
       real*8,allocatable :: w(:)
c      --- prepare sub-communicator ---
       call mpi_init(ierr)
       call mpi_comm_size(mpi_comm_world, nsize, ierr)
       call mpi_comm_rank(mpi_comm_world, nrank, ierr)
       ncolory=nrank/nd2
       call mpi_comm_split(mpi_comm_world,ncolory,nrank,
      &                    comm2,ierr)
       call mpi_comm_size(comm2, nsize2, ierr)
       call mpi_comm_rank(comm2, nrank2, ierr)
       ncolorz=mod(nrank,nd2)
       call mpi_comm_split(mpi_comm_world,ncolorz,nrank,
      &                    comm3,ierr)
       call mpi_comm_size(comm3, nsize3, ierr)
       call mpi_comm_rank(comm3, nrank3, ierr)
       if(nsize.ne.nd2*nd3 .or. nsize2.ne.nd2 .or.
      &   nsize3.ne.nd3) then
         print*,'nsize=',nsize,nsize2,nsize3
         go to 9000
       endif
 c     --- prepare test-data ---
       nx2=min(kx2p,max(n2-nrank2*kx2p,0))
       nx3=min(kx3p,max(n3-nrank3*kx3p,0))
       ix=1000
       ix=ix+nrank      ! different seed
       do i3=1,nx3
         do i2=1,nx2
           call dvrau4(ix,x(1,i2,i3),n1,dwork,nwrand,icon)
           do i1=1,n1
             wc(i1,i2,i3)=x(i1,i2,i3)
           enddo
         enddo
       enddo
 c     --- inquire necessary size ---
       nw=0
       call ds_v3drcf2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
      &          nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
      &          comm2,comm3,icon)
       if(icon.ne.1000) then
         print*,'icon=',icon
         go to 9000
       endif
       allocate (z(kz1,kz2,kz3),w(nw))
       print*,'nrank,nrank2,nrank3=',nrank,nrank2,nrank3,
      &       ' Z-pencil x-range=',nz1b,nz1e,
      &       ' y-range=',nz2b,nz2e,
      &       ' z-range=',1,n3
 c     --- forward FFT ---
       idir=1
       isn=1
       call ds_v3drcf2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
      &          nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
      &          comm2,comm3,icon)
       if(icon.ne.0) then
         print*,'icon=',icon
         go to 9000
       endif
 c     --- backward FFT ---
       idir=-1
       isn=-1
       call ds_v3drcf2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
      &          nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
      &          comm2,comm3,icon)
```

```
      if(icon.ne.0) then
        print*,'icon=',icon
        go to 9000
      endif
c     --- check result ---
      errorx=0
      do i3=1,nx3
        do i2=1,nx2
          do i1=1,n1
            errorx=max(dabs(wc(i1,i2,i3)-
     &             x(i1,i2,i3)/n1/n2/n3),errorx)
          enddo
        enddo
      enddo
      call mpi_allreduce(errorx,errormax,1,mpi_double_precision,
     &                   mpi_max,mpi_comm_world,ierr)
      if(nrank.eq.0)then
        print*,'num proc=',nsize
        print*,'nd2,nd3=',nsize2,nsize3
        print*,'-----(',n1,',',n2,',',n3,')-----'
        print*,'error=',errormax
      endif
c
 9000 continue
      call mpi_comm_free(comm2,ierr)
      call mpi_comm_free(comm3,ierr)
      call mpi_finalize(ierr)
      stop
      end
```

DS_V3DRCF3

Three-dimensional discrete real Fourier transforms. (mixed radices of 2, 3, 5 and 7, volumetric decomposition)
CALL DS_V3DRCF3 (X, KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, W, NW, ISIN, ISN, COMM, ICON)

(1) Function

The subroutine DS_V3DRCF3 performs a three-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$.

$$n_1 n_2 n_3 \alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1r} \omega_{n2}^{-j2k2r} \omega_{n3}^{-j3k3r}$$

$$\begin{aligned} &, k_1 = 0,\ 1, ..., n_1 - 1 \\ &, k_2 = 0,\ 1, ..., n_2 - 1 \\ &, k_3 = 0,\ 1, ..., n_3 - 1 \\ &, \omega_{n1} = \exp(2\pi i / n_1) \\ &, \omega_{n2} = \exp(2\pi i / n_2) \\ &, \omega_{n3} = \exp(2\pi i / n_3) \\ &r = 1 \text{ or } r = -1 \end{aligned} \qquad (1.1)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1r} \omega_{n2}^{j2k2r} \omega_{n3}^{j3k3r}$$

$$\begin{aligned} &, j_1 = 0,\ 1, ..., n_1 - 1 \\ &, j_2 = 0,\ 1, ..., n_2 - 1 \\ &, j_3 = 0,\ 1, ..., n_3 - 1 \\ &, \omega_{n1} = \exp(2\pi i / n_1) \\ &, \omega_{n2} = \exp(2\pi i / n_2) \\ &, \omega_{n3} = \exp(2\pi i / n_3) \\ &r = 1 \text{ or } r = -1 \end{aligned} \qquad (1.2)$$

This subroutine provides an efficient and scalable 3D FFT functionality on a massively parallel machine. The global data of the three-dimensional array can be distributed among processes which are regarded as a three-dimensional grid, therefore the volumetric decomposition allows the distribution of work more efficiently than a slabwise decomposition in an environment where massively parallel processes are available.

(2) Parameters

X .... Input/Output. Real data.

When the global three dimensional real data is regarded as an array D(N1, N2, N3) virtually, the local array X stores subarray of D distributed along with the shape of the process grid.

If each N*m* is divisible by ND*m* (*m*=1,2,3) and N1/ND1 is an even number, setting width parameters KX*m*P=N*m*/ND*m* is acceptable, then the data size settled in the subarray can be (KX1P,KX2P,KX3P) in most processes, except the edge processes of the first dimension of the process grid, which have the data size of (KX1P +2,KX2P,KX3P). Note that the array with the slightly-increased area is necessary for every process.

If N*m* is not divisible by ND*m* for *m*=2,3, setting KX*m*P=N*m*/ND*m*+1 is recommended, then the sizes of the subarrays that are assigned to the edge of the process grid are less than KX*m*P. As for *m*=1, set an even number to KX1P such that $(N1/2+1) \times 2 \le KX1P \times ND1$, when $KX1P \times ND1 = N1$ is not adoptable.

The array X in each process stores the subarray of D as follows:
$X(1{:}N1P, 1{:}N2P, 1{:}N3P) \leftarrow D(N1S{:}N1E, N2S{:}N2E, N3S{:}N3E)$,
$N1S = KX1P \times rank1 + 1$,
$N1E = MIN(N1, KX1P \times (rank1 + 1))$,
$N1P = MAX(0, N1E - N1S + 1)$,
$N2S = KX2P \times rank2 + 1$,
$N2E = MIN(N2, KX2P \times (rank2 + 1))$,
$N2P = MAX(0, N2E - N2S + 1)$,
$N3S = KX3P \times rank3 + 1$,
$N3E = MIN(N3, KX3P \times (rank3 + 1))$,
$N3P = MAX(0, N3E - N3S + 1)$,

where, *rank1*,*rank2* and *rank3* are coordinates of the process grid calculated as follows from the *rank* value, which is obtained by subroutine MPI_COMM_RANK of MPI:
$rank1 = mod(rank, ND1)$,
$rank2 = mod(rank/ND1, ND2)$,
$rank3 = rank/(ND1 \times ND2)$.

For the real to complex transform (ISN = 1), data is input; for the complex to real transform (ISN = -1), data is output.

Output/input. The real and imaginary parts of the transformed complex data.

The complex data CD(N1, N2, N3) obtained from real data D(N1, N2, N3) by Fourier transform has the complex conjugate relation so the about a half of the first dimension(1～N1/2+1) is used to store the complex data.

(See note 2) in (3), “Comments on use.”)

Regarding an array X as X(2, KX1/2, KX2, KX3P), the real and imaginary parts are stored in X(1, 1:N1P, 1:N2P, 1:N3P) and X(2, 1:N1P, 1:N2P, 1:N3P) respectively, where,

$N1S = KX1P/2 \times rank1 + 1$,
$N1E = MIN(N1/2+1, KX1P/2 \times (rank1 + 1))$,
$N1P = MAX(0, N1E - N1S + 1)$.

Additionally, the edge processes of the first dimension of the process grid have the increased data in X(1:2, N1P+1, 1:N2P, 1:N3P), when the condition KX1P × ND1 = N1 is satisfied.

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = -1), data is input.

This is a double precision real three-dimensional array X(KX1, KX2, KX3P).

KX1 .... Input. The size of the first dimension of the array X. KX1 must be even.

The condition KX1 ≥ KX1P + 2 must be satisfied if KX1P × ND1 = N1. Otherwise, KX1 ≥ KX1P.

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of the array X (≥ KX2P).

Integer(INTEGER*4).

KX1P........... Input. The size by which the first dimension is equally partitioned (KX1P×ND1≥ N1). KX1P must be even.

Integer(INTEGER*4).

KX2P........... Input. The size by which the second dimension is equally partitioned.(KX2P×ND2≥ N2)

Integer(INTEGER*4).

KX3P .... Input. The size by which the third dimension is equally partitioned (KX3P×ND3≥ N3)

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 .... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 .... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

ND1............. Input. The number of processes by which the first dimension is partitioned.

Integer (INTEGER*4)
(See note 2) in (3), “Comments on use.”)

ND2............. Input. The number of processes by which the second dimension is partitioned.

Integer (INTEGER*4)
(See note 2) in (3), “Comments on use.”)

ND3............. Input. The number of processes by which the third dimension is partitioned.

Integer (INTEGER*4)
(See note 2) in (3), “Comments on use.”)

W .... Work area. This is double precision complex one-dimensional array W(NW).

NW .... Input. The size of the work array W (NW ≥ MAX(KX1 × ND1, KX2P × ND2, KX3P × ND3) × 6). It is recommended to specify a sufficiently large size for efficiency. (See note 2) in (2), “Comments on use.”)

Integer (INTEGER*4)

ISIN .... Input. The direction of the transformation.

ISIN = 1 for r = 1.

ISIN = -1 for r = -1.

Integer (INTEGER*4).

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

COMM .... Input. The communicator indicating a set of processes on which data are distributed and by which computation is done in parallel.

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DRCF3-1.

Integer (INTEGER*4).

Table DS_V3DRCF3-1 Condition codes

Code	Meaning	Processing
0	No error	
25000	Too small work area.	Processing is discontinued.
30000	N1<1, N2<1, N3<1, KX1 < KX1P, KX1 or KX1P is not even, KX2 < KX2P, N1 > KX1P × ND1, N2 > KX2P × ND2, N3 > KX3P × ND3, (N1/2+1) × 2 > KX1 × ND1, ISIN≠1, -1, ISN≠1, -1.	
30008	The order of the transform is not radix 2/3/5/7.	
30100	ND1×ND2×ND3 is not equal to total processes.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

$$, k_1 = 0,\ 1,\ldots, n_1 - 1$$

$$, k_2 = 0,\ 1,\ldots, n_2 - 1$$

$$, k_3 = 0,\ 1,\ldots, n_3 - 1 \qquad (3.1)$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$

$$, j_1 = 0,\ 1,\ldots, n_1 - 1 \qquad (3.2)$$

$$, j_2 = 0,\ 1,\ldots, n_2 - 1$$

$$, j_3 = 0,\ 1,\ldots, n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) The results of the three-dimensional real Fourier transform has the following complex conjugate relation (indicated by ¯).

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\ n2-k2\ n3-k3}} \qquad (3.3)$$

The remainder of the data is obtained from data in $k_1 = 0, \ldots, n_1-1$, $k_2 = 0, \ldots, n_2-1$, and $k_3 = 0, \ldots, n_3/2$.

3) The size of work area W

The size of the work array determines partition sizes for transferring data among nodes and calculation on each node in this routine. Setting the size NW of work area W much larger than MAX(KX1×ND1, KX2P×ND2, KX3P×ND3) × 6 × (number of threads in a process) is recommended. For example, setting NW > 1,000,000 is expected to be efficient when the assigned cache size to the process is 8MB and the array X is partitionable by that size of the work area.

4) Parameters ND1, ND2 and ND3

When using volumetric decomposition, it is recommended to adjust ND1,ND2 and ND3 to be about the comparable value of cube root of the number of total process ND1×ND2×ND3 for overall efficiency of transferring data. Note that the performance of this routine may deteriorate when ND1,ND2 and ND3 do not match the shape of the executing process grid on a system which can specify the shape of the process grid. Refer to the Job Operation Software manual whether the system can assign a shape of the process grid.

Additionally, this routine can be used for slabwise decomposition or 2-dimensional decomposition also by setting any of ND1,ND2 or ND3 to 1, when user's program exploits specific decomposition or available shape of the process grid on a system is limited.

5) Consistency of parameters among processes

The parameters KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, NW, ISIN and ISN needs to have same value respectively among all processes, otherwise the result is not guaranteed.

b. Example

Three-dimensional FFT is computed in 2×2×2 processes.

```
c     ** example program **
      use mpi
c
      implicit  real*8 (a-h,o-z)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (nd1=2,nd2=2,nd3=2)
      parameter (kx1p=((n1+nd1-1)/nd1+1)/2*2)
      parameter (kx1=((n1/2+1)+nd1-1)/nd1*2)
      parameter (kx2p=(n2+nd2-1)/nd2,kx2=kx2p)
      parameter (kx3p=(n3+nd3-1)/nd3)
      parameter (nw=kx1*kx2p*kx3p)
      parameter (nwork=388)
      real*8 dwork(nwork)
      real*8 x(kx1,kx2,kx3p),w(nw),wc(kx1,kx2,kx3p)
c
      call mpi_init( ierr )
      call mpi_comm_size( mpi_comm_world, nump, ierr )
      call mpi_comm_rank( mpi_comm_world, nop, ierr )
      nrank1=mod(nop,nd1)
      nrank2=mod(nop/nd1,nd2)
      nrank3=nop/(nd1*nd2)
c
      ix=1000
      ix=ix*nump+nop      ! different seed
      do i1=1,kx3p
      call dvrau4(ix,x(1,1,i1),
     $              kx1*kx2,
     &              dwork, nwork, icon)
      enddo
c
      do i3=1, kx3p
      do i2=1, kx2p
      do i1=1, kx1p
      wc(i1,i2,i3)=x(i1,i2,i3)
      enddo
      enddo
      enddo
c
      isin=1
      isn=1
      call ds_v3drcf3(x,kx1,kx2,kx1p,kx2p,kx3p,
     $          n1,n2,n3,nd1,nd2,nd3,w,nw,isin,isn,
     $          mpi_comm_world,icon)
      print*,'icon=',icon
      if(icon.ne.0) go to 9000
```

```
c
      isn=-1
      call ds_v3drcf3(x,kx1,kx2,kx1p,kx2p,kx3p,
     $           n1,n2,n3,nd1,nd2,nd3,w,nw,isin,isn,
     $           mpi_comm_world,icon)
      print*,'icon=',icon
      if(icon.ne.0) go to 9000
c
      errorx=0
      iof1=nrank1*kx1p
      iof2=nrank2*kx2p
      iof3=nrank3*kx3p
      do i1=1,min(kx1p,max(n1-iof1,0))
      do i2=1,min(kx2p,max(n2-iof2,0))
      do i3=1,min(kx3p,max(n3-iof3,0))
      errorx=max(dabs(wc(i1,i2,i3)-
     $       x(i1,i2,i3)/n1/n2/n3),errorx)
      enddo
      enddo
      enddo
c
      call mpi_allreduce(errorx,errormax,1,mpi_double_precision,
     $                   mpi_max,mpi_comm_world,ierr)
c
      if(nop.eq.0)then
      print*,'-----(',n1,',',n2,',',n3,')-----'
      print*,'error=',errormax
      endif
 9000 continue
c
      call mpi_finalize( ierr )
c
      stop
      end
```

SS_V3DCFT2X

Three-dimensional discrete complex Fourier transforms. (mixed radices of 2, 3, 5 and 7, pencil decomposition, single precision)
CALL SS_V3DCFT2X (X, KX1, KX2, KX2P, KX3P, Z, KZ1, KZ2, KZ3, NZ1B, NZ1E, NZ2B, NZ2E, N1, N2, N3, W, NW, ISN, IDIR, COMM2, COMM3, ICON)

(1) Function

The subroutine SS_V3DCFT2X performs a three-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT with single-precision.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$\begin{aligned} n_1n_2n_3\alpha_{k1k2k3} &= \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \\ &, k_1 = 0,\ 1,\dots,n_1-1 \\ &, k_2 = 0,\ 1,\dots,n_2-1 \\ &, k_3 = 0,\ 1,\dots,n_3-1 \\ &, \omega_{n1} = \exp(2\pi i/n_1) \\ &, \omega_{n2} = \exp(2\pi i/n_2) \\ &, \omega_{n3} = \exp(2\pi i/n_3) \end{aligned} \tag{1.1}$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$\begin{aligned} x_{j1j2j3} &= \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \\ &, j_1 = 0,\ 1,\dots,n_1-1 \\ &, j_2 = 0,\ 1,\dots,n_2-1 \\ &, j_3 = 0,\ 1,\dots,n_3-1 \\ &, \omega_{n1} = \exp(2\pi i/n_1) \\ &, \omega_{n2} = \exp(2\pi i/n_2) \\ &, \omega_{n3} = \exp(2\pi i/n_3) \end{aligned} \tag{1.2}$$

This subroutine provides an efficient and scalable 3D FFT functionality using pencil decomposition. The global data of the three-dimensional array is to be distributed among a two-dimensional process grid. The local input array and output array of each process store distinct shapes by dividing the global data in different directions of pencils. This

allows the routine to omit communication for transposing back to the original distribution shape. This subroutine is the single-precision version of the DS_V3DCFT2X.

(2) Parameters

In the following, the global three dimensional complex data is regarded as an array D(N1, N2, N3) virtually, and the process grid is regarded as ND2 × ND3 shape.

X .... Input when IDIR = 1. Complex data.

The local array X stores a subarray of D, which corresponds to a columnwise decomposed part of the global array by dividing the second and third dimensions by ND2 and ND3 respectively.

If each N*m* is divisible by ND*m* (*m*=2,3), setting a width parameter KX*m*P=N*m*/ND*m* is recommended, then the sizes of the subarray can be (N1,KX2P,KX3P) uniformly.

If N*m* is not divisible by ND*m*, setting KX*m*P=N*m*/ND*m*+1 is recommended, then the sizes of the subarrays that are assigned to the edge of the process grid are less than KX*m*P.

The array X in each process stores the subarray of D as follows:
X(1:N1, 1:NX2P, 1:NX3P) ← D(1:N1, NX2B:NX2E, NX3B:NX3E)
  NX2B = KX2P × *rank2* + 1
  NX2E = MIN(N2, KX2P × (*rank2* + 1))
  NX2P = MAX(0, NX2E - NX2B + 1)
  NX3B = KX3P × *rank3* + 1
  NX3E = MIN(N3, KX3P × (*rank3* + 1))
  NX3P = MAX(0, NX3E - NX3B + 1),

where the *rank2* and *rank3* are ranks of the process in the communicator COMM2 and COMM3 respectively, which are obtained by subroutine MPI_COMM_RANK of MPI.

The input values are not retained after the calculation.

.... Output when IDIR = -1. Transformed complex data.

Resultant transformed three dimensional data of D(N1, N2, N3) are stored into the array X in the same distributed way as stated above.

This is a single precision complex three-dimensional array X(KX1, KX2, NX3P).

KX1 .... Input. The size of the first dimension of the array X (≥ N1).

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of the array X (≥ NX2P).

Integer(INTEGER*4).

KX2P .... Input. The size by which the second dimension of D is equally partitioned when the data are stored in the array X. (KX2P×ND2≥ N2)

Integer(INTEGER*4).

KX3P .... Input. The size by which the third dimension of D is equally partitioned when the data are stored in the array X. (KX3P×ND3≥ N3)

Integer(INTEGER*4).

Z .... Output when IDIR = 1. Transformed complex data.

The local array Z stores a subarray of D, which corresponds to a columnwise decomposed part of the global array by dividing the first and second dimensions by ND2 and ND3 respectively.

The array Z in each process stores the subarray of D as follows:
Z(1:NZ1P, 1:NZ2P, 1:N3) ← D(NZ1B:NZ1E, NZ2B:NZ2E, 1:N3)
  NZ1P = MAX(0, NZ1E − NZ1B + 1)
  NZ2P = MAX(0, NZ2E − NZ2B + 1).

..........Input when IDIR = -1. Complex data.

The local array Z stores the subarray of D in the same distributed way as stated above.

This is a single precision complex three-dimensional array Z(KZ1, KZ2, KZ3).

KZ1 .... Output when NW = 0. The recommended size for the first dimension of the array Z.

Input when NW ≠ 0. The size of the first dimension of the array Z (≥ NZ1E-NZ1B+1).

Integer(INTEGER*4).

KZ2 .... Output when NW = 0. The recommended size for the second dimension of the array Z.

Input when NW ≠ 0. The size of the second dimension of the array Z (≥ NZ2E-NZ2B+1).

Integer(INTEGER*4).

KZ3 .... Output when NW = 0. The recommended size of the third dimension of the array Z.

Input when NW ≠ 0. The size of the third dimension of the array Z (≥ N3).

Integer(INTEGER*4).

NZ1B .... Output. The starting index for the first dimension of the global array D.

Integer(INTEGER*4).

NZ1E .... Output. The ending index for the first dimension of the global array D.
NZ1B and NZ1E indicate which portion of the first dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

NZ2B .... Output. The starting index for the second dimension of the global array D.

Integer(INTEGER*4).

NZ2E .... Output. The ending index for the second dimension of the global array D.
NZ2B and NZ2E indicate which portion of the second dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 .... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 .... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

W .... Work area. This is single precision complex one-dimensional array W(NW).

NW .... Input / Output. The size of the work array W.

When NW = 0 specified, the recommended sizes of NW, KZ1, KZ2, and KZ3 are set respectively, and index information is set to NZ1B, NZ1E, NZ2B, and NZ2E.

Integer (INTEGER*8). (See note 4) in (3), "Comments on use.")

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

IDIR .... Input. The direction of transform between arrays is indicated.

IDIR = 1 for the transform from the array X to the array Z.

IDIR = -1 for the transform from the array Z to the array X.

Integer (INTEGER*4).

COMM2 ..... Input. The MPI communicator that represents a set of processes whose size of the process group is ND2, which is obtained by MPI_COMM_SIZE, in the process shape ND2 $\times$ ND3. (See note 2) in (3), "Comments on use.")

Integer (INTEGER*4).

COMM3 ..... Input. The MPI communicator that represents a set of processes whose size of the process group is ND3, which is obtained by MPI_COMM_SIZE, in the process shape ND2 $\times$ ND3. (See note 2) in (3), "Comments on use.")

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DCFT2X-1.

Integer (INTEGER*4).

Table SS_V3DCFT2X-1 Condition codes

Code	Meaning	Processing
0	No error	

Table SS_V3DCFT2X-1 Condition codes

Code	Meaning	Processing
1000	NW = 0 is specified.	The recommended sizes of NW, KZ1, KZ2, and KZ3 are set. Index information is set to NZ1B, NZ1E, NZ2B, and NZ2E. There is no output to array X or Z.
25000	Too small work area.	Processing is discontinued.
30000	N1<1, N2<1, N3<1, KX1 < N1, KX2 < NX2P, N2 > KX2P × ND2, N3 > KX3P × ND3, or the value of ISN or IDIR is incorrect.	
30008	The order of the transform is not radix 2/3/5/7.	
30100	COMM2 or COMM3 is incorrect.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1,\ldots, n_1 - 1$$
$$, k_2 = 0,\ 1,\ldots, n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1,\ldots, n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1,\ldots, n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0,\ 1,\ldots, n_2 - 1$$
$$, j_3 = 0,\ 1,\ldots, n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) Process shape ND2 and ND3

Note that the performance of this routine may deteriorate when ND2 and ND3 do not match the shape of the executing process grid on a system which can specify the shape of the process grid. Refer to the Job Operation Software manual whether the system can assign a shape of the process grid.

3) Consistency of parameters among processes

   The parameters KX2P,KX3P,N1,N2,N3,NW,ISN, and IDIR needs to have same value respectively among all processes, otherwise the result is not guaranteed.

4) The size of work area W

   The size of the work array NW needs to be about twice the size of array X or Z to be used as send/receive buffers for MPI communication inside the routine. Note that the parameter NW is an 8-byte integer type.

b. Example

Three-dimensional FFT is computed in 2×3 processes.

```
c     ** example program **
      use mpi
      implicit  real (a-h,o-z)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (nd2=2,nd3=3)
      parameter (kx1=n1)
      parameter (kx2p=((n2+nd2-1)/nd2),kx2=kx2p)
      parameter (kx3p=((n3+nd3-1)/nd3))
      integer comm2,comm3
      integer*8 nw
      complex x(kx1,kx2,kx3p),wc(kx1,kx2,kx3p)
      complex,allocatable :: z(:,:,:),w(:)
c     --- prepare sub-communicator ---
      call mpi_init(ierr)
      call mpi_comm_size(mpi_comm_world, nsize, ierr)
      call mpi_comm_rank(mpi_comm_world, nrank, ierr)
      ncolory=nrank/nd2
      call mpi_comm_split(mpi_comm_world,ncolory,nrank,
     &                    comm2,ierr)
      call mpi_comm_size(comm2, nsize2, ierr)
      call mpi_comm_rank(comm2, nrank2, ierr)
      ncolorz=mod(nrank,nd2)
      call mpi_comm_split(mpi_comm_world,ncolorz,nrank,
     &                    comm3,ierr)
      call mpi_comm_size(comm3, nsize3, ierr)
      call mpi_comm_rank(comm3, nrank3, ierr)
      if(nsize.ne.nd2*nd3 .or. nsize2.ne.nd2 .or.
     &   nsize3.ne.nd3) then
        print*,'nsize=',nsize,nsize2,nsize3
        go to 9000
      endif
c     --- prepare test-data ---
      nx2=min(kx2p,max(n2-nrank2*kx2p,0))
      nx3=min(kx3p,max(n3-nrank3*kx3p,0))
      ix=1000
      ix=ix+nrank      ! different seed
      do i3=1,nx3
```

```
          do i2=1,nx2
            call ranu2(ix,x(1,i2,i3),2*n1,icon)
            do i1=1,n1
              wc(i1,i2,i3)=x(i1,i2,i3)
            enddo
          enddo
        enddo
c     --- inquire necessary size ---
        nw=0
        call ss_v3dcft2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &         nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &         comm2,comm3,icon)
        if(icon.ne.1000) then
          print*,'icon=',icon
          go to 9000
        endif
        allocate (z(kz1,kz2,kz3),w(nw))
        print*,'nrank,nrank2,nrank3=',nrank,nrank2,nrank3,
     &        ' Z-pencil x-range=',nz1b,nz1e,
     &        ' y-range=',nz2b,nz2e,
     &        ' z-range=',1,n3
c     --- forward FFT ---
        idir=1
        isn=1
        call ss_v3dcft2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &         nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &         comm2,comm3,icon)
        if(icon.ne.0) then
          print*,'icon=',icon
          go to 9000
        endif
c     --- backward FFT ---
        idir=-1
        isn=-1
        call ss_v3dcft2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &         nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &         comm2,comm3,icon)
        if(icon.ne.0) then
          print*,'icon=',icon
          go to 9000
        endif
c     --- check result ---
        errorx=0
        do i3=1,nx3
          do i2=1,nx2
            do i1=1,n1
              errorx=max(cabs(wc(i1,i2,i3)-
     &              x(i1,i2,i3)/n1/n2/n3),errorx)
            enddo
          enddo
        enddo
        call mpi_allreduce(errorx,errormax,1,mpi_real,
     &                   mpi_max,mpi_comm_world,ierr)
```

```
      if(nrank.eq.0)then
        print*,'num proc=',nsize
        print*,'nd2,nd3=',nsize2,nsize3
        print*,'-----(',n1,',',n2,',',n3,')-----'
        print*,'error=',errormax
      endif
c
 9000 continue
      call mpi_comm_free(comm2,ierr)
      call mpi_comm_free(comm3,ierr)
      call mpi_finalize(ierr)
      stop
      end
```

SS_V3DRCF2X

Three-dimensional discrete real Fourier transforms. (mixed radices of 2, 3, 5 and 7, pencil decomposition, single precision)
CALL SS_V3DRCF2X (X, KX1, KX2, KX2P, KX3P, Z, KZ1, KZ2, KZ3, NZ1B, NZ1E, NZ2B, NZ2E, N1, N2, N3, W, NW, ISN, IDIR, COMM2, COMM3, ICON)

(1) Function

The subroutine SS_V3DRCF2X performs a three-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT with single-precision.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3}$$

$$\begin{aligned}&, k_1 = 0,\ 1,\ldots, n_1 - 1\\ &, k_2 = 0,\ 1,\ldots, n_2 - 1\\ &, k_3 = 0,\ 1,\ldots, n_3 - 1\\ &, \omega_{n1} = \exp(2\pi i/n_1)\\ &, \omega_{n2} = \exp(2\pi i/n_2)\\ &, \omega_{n3} = \exp(2\pi i/n_3)\end{aligned} \qquad (1.1)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3}$$

$$\begin{aligned}&, j_1 = 0,\ 1,\ldots, n_1 - 1\\ &, j_2 = 0,\ 1,\ldots, n_2 - 1\\ &, j_3 = 0,\ 1,\ldots, n_3 - 1\\ &, \omega_{n1} = \exp(2\pi i/n_1)\\ &, \omega_{n2} = \exp(2\pi i/n_2)\\ &, \omega_{n3} = \exp(2\pi i/n_3)\end{aligned} \qquad (1.2)$$

This subroutine provides an efficient and scalable 3D FFT functionality using pencil decomposition. The global data of the three-dimensional array is to be distributed among a two-dimensional process grid. The local input array and output array of each process store distinct shapes by dividing the global data in different directions of pencils. This

allows the routine to omit communication for transposing back to the original distribution shape. This subroutine is the single-precision version of the DS_V3DRCF2X.

(2) Parameters

In the following, the global three dimensional real data is regarded as an array DR(N1, N2, N3), and the transformed global three dimensional complex data is regarded as an array DC(N1/2+1, N2, N3) virtually, and the process grid is regarded as ND2 × ND3 shape.

X .... Input when IDIR = 1. Real data.

The local array X stores a subarray of DR, which corresponds to a columnwise decomposed part of the global array by dividing the second and third dimensions by ND2 and ND3 respectively.

If each N*m* is divisible by ND*m* (*m*=2,3), setting width parameters KX*m*P=N*m*/ND*m* is recommended, then the data size settled in the subarray can be (N1,KX2P,KX3P) uniformly.

If N*m* is not divisible by ND*m*, setting KX*m*P=N*m*/ND*m*+1 is recommended, then the sizes of the subarrays that are assigned to the edge of the process grid are less than KX*m*P.

The array X in each process stores the subarray of DR as follows:
X(1:N1, 1:NX2P, 1:NX3P) ← DR(1:N1, NX2B:NX2E, NX3B:NX3E),
NX2B = KX2P × *rank2* + 1
NX2E = MIN(N2, KX2P × (*rank2* + 1))
NX2P = MAX(0, NX2E - NX2B + 1)
NX3B = KX3P × *rank3* + 1
NX3E = MIN(N3, KX3P × (*rank3* + 1))
NX3P = MAX(0, NX3E - NX3B + 1),

where, *rank2* and *rank3* are the ranks of the process in the communicator COMM2 and COMM3 respectively, which are obtained by subroutine MPI_COMM_RANK of MPI.

The input values are not retained after the calculation.

.... Output when IDIR = -1. Transformed real data.

Resultant transformed three dimensional real data from DC(N1/2+1, N2, N3) are stored into the array X in the same distributed way as stated above.

This is a single precision real three-dimensional array X(KX1, KX2, KX3P).

KX1 .... Input. The size of the first dimension of the array X. KX1 must be even. (≥ N1).

Integer(INTEGER*4).

KX2 .... Input. The size of the second dimension of the array X (≥NX2P).

Integer(INTEGER*4).

KX2P........... Input. The size by which the second dimension of DR is equally partitioned. (KX2P×ND2≥ N2)

Integer(INTEGER*4).

KX3P .... Input. The size by which the third dimension of DR is equally partitioned. (KX3P×ND3≥ N3)

Integer(INTEGER*4).

Z .... Output when IDIR = 1. Transformed complex data.

The complex data obtained from real data DR(N1, N2, N3) by Fourier transform has the complex conjugate relation so the about a half of the first dimension is used to store. The local array Z stores a subarray of DC(N1/2+1, N2, N3), which corresponds to a columnwise decomposed part of the global array by dividing the first and second dimensions by ND2 and ND3 respectively.

The array Z in each process stores the subarray of DC as follows:
Z(1:NZ1P, 1:NZ2P, 1:N3) ← DC(NZ1B:NZ1E, NZ2B:NZ2E, 1:N3)
NZ1P = MAX(0, NZ1E − NZ1B + 1)
NZ2P = MAX(0, NZ2E − NZ2B + 1).

..........Input when IDIR = -1. Complex data.

The local array Z stores the subarray of DC in the same distributed way as stated above.

This is a single precision complex three-dimensional array Z(KZ1, KZ2, KZ3).

(See note 2) in (3), "Comments on use.")

KZ1 .... Output when NW = 0. The recommended size for the first dimension of the array Z.

Input when NW ≠ 0. The size of the first dimension of the array Z (≥ NZ1E-NZ1B+1).

Integer(INTEGER*4).

KZ2 .... Output when NW = 0. The recommended size for the second dimension of the array Z.

Input when NW ≠ 0. The size of the second dimension of the array Z (≥ NZ2E-NZ2B+1).

Integer(INTEGER*4).

KZ3 .... Output when NW = 0. The recommended size of the third dimension of the array Z.

Input when NW ≠ 0. The size of the third dimension of the array Z (≥ N3).

Integer(INTEGER*4).

NZ1B .... Output. The starting index for the first dimension of the global array DC.

Integer(INTEGER*4).

NZ1E .... Output. The ending index for the first dimension of the global array DC. NZ1B and NZ1E indicate which portion of the first dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

NZ2B .... Output. The starting index for the second dimension of the global array DC.

Integer(INTEGER*4).

NZ2E .... Output. The ending index for the second dimension of the global array DC. NZ2B and NZ2E indicate which portion of the second dimension within the global array is stored in the local array Z.

Integer(INTEGER*4).

N1 .... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 .... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 .... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

W .... Work area. This is a single precision complex one-dimensional array W(NW).

NW .... Input / Output. The size of the work array W.

When NW = 0 specified, the recommended sizes of NW, KZ1, KZ2, and KZ3 are set respectively, and the index information is set to NZ1B, NZ1E, NZ2B, and NZ2E.

Integer (INTEGER*8). (See note 5) in (3), "Comments on use.")

ISN .... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

Integer (INTEGER*4).

IDIR .... Input. The direction of transform between arrays is indicated.

IDIR = 1: from the real array X to the complex array Z.

IDIR = -1: from the complex array Z to the real array X.

Integer (INTEGER*4).

COMM2 ..... Input. The MPI communicator that represents a set of processes whose size of the process group is ND2, which is obtained by MPI_COMM_SIZE, in the process shape ND2 × ND3. (See note 3) in (3), "Comments on use.")

Integer (INTEGER*4).

COMM3 ..... Input. The MPI communicator that represents a set of processes whose size of the process group is ND3, which is obtained by MPI_COMM_SIZE, in the process shape ND2 × ND3. (See note 3) in (3), "Comments on use.")

Integer (INTEGER*4).

ICON .... Output. Condition code.

See Table DS_V3DRCF2X-1.

Integer (INTEGER*4).

Table SS_V3DRCF2X-1 Condition codes

Code	Meaning	Processing

Table SS_V3DRCF2X-1 Condition codes

Code	Meaning	Processing
0	No error	
1000	NW = 0 is specified	The recommended sizes of NW, KZ1, KZ2 and KZ3 are set. Index information is set to NZ1B, NZ1E, NZ2B, and NZ2E. There is no output to array X or Z.
25000	Too small work area.	Processing is discontinued.
30000	N1<1, N2<1, N3<1, KX1 < N1, KX1 is not even, KX2 < NX2P, N2 > KX2P × ND2, N3 > KX3P × ND3, (N1/2+1) × 2 > KZ1 × ND2, ISN≠1, -1, IDIR≠1, -1.	
30008	The order of the transform is not radix 2/3/5/7.	
30100	COMM2 or COMM3 is incorrect.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1, \ldots, n_1 - 1$$
$$, k_2 = 0,\ 1, \ldots, n_2 - 1 \qquad (3.1)$$
$$, k_3 = 0,\ 1, \ldots, n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1, \ldots, n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0,\ 1, \ldots, n_2 - 1$$
$$, j_3 = 0,\ 1, \ldots, n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

2) The results of the three-dimensional real Fourier transform has the following complex conjugate relation (indicated by $^{-}$).

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\ n2-k2\ n3-k3}} \tag{3.3}$$

The remainder of the data is obtained from data in $k_1 = 0, ..., n_1–1$, $k_2 = 0, ..., n_2–1$, and $k_3 = 0, ..., n_3/2$.

3) Process shape ND2 and ND3

Note that the performance of this routine may deteriorate when ND1,ND2 and ND3 do not match the shape of the executing process grid on a system which can specify the shape of the process grid. Refer to the Job Operation Software manual whether the system can assign a shape of the process grid.

4) Consistency of parameters among processes

The parameters KX2P, KX3P, N1, N2, N3, NW, ISN and IDIR needs to have same value respectively among all processes, otherwise the result is not guaranteed.

5) The size of work area W

The size of the work array NW needs to be about twice the size of array X or Z to be used as send/receive buffers for MPI communication inside the routine. Note that the parameter NW is an 8-byte integer type.

b. Example

Three-dimensional FFT is computed in 2×3 processes.

```
c     ** example program **
      use mpi
      implicit  real (a-h,o-z)
      parameter (n1=512,n2=n1,n3=n2)
      parameter (nd2=2,nd3=3)
      parameter (n1c=n1/2+1,kx1=n1c*2)
      parameter (kx2p=((n2+nd2-1)/nd2),kx2=kx2p)
      parameter (kx3p=((n3+nd3-1)/nd3))
      integer comm2,comm3
      integer*8 nw
      real x(kx1,kx2,kx3p),wc(kx1,kx2,kx3p)
      complex,allocatable :: z(:,:,:)
      real,allocatable :: w(:)
c     --- prepare sub-communicator ---
      call mpi_init(ierr)
      call mpi_comm_size(mpi_comm_world, nsize, ierr)
      call mpi_comm_rank(mpi_comm_world, nrank, ierr)
      ncolory=nrank/nd2
      call mpi_comm_split(mpi_comm_world,ncolory,nrank,
     &                    comm2,ierr)
      call mpi_comm_size(comm2, nsize2, ierr)
      call mpi_comm_rank(comm2, nrank2, ierr)
      ncolorz=mod(nrank,nd2)
      call mpi_comm_split(mpi_comm_world,ncolorz,nrank,
     &                    comm3,ierr)
      call mpi_comm_size(comm3, nsize3, ierr)
      call mpi_comm_rank(comm3, nrank3, ierr)
      if(nsize.ne.nd2*nd3 .or. nsize2.ne.nd2 .or.
     &   nsize3.ne.nd3) then
        print*,'nsize=',nsize,nsize2,nsize3
        go to 9000
      endif
c     --- prepare test-data ---
      nx2=min(kx2p,max(n2-nrank2*kx2p,0))
      nx3=min(kx3p,max(n3-nrank3*kx3p,0))
      ix=1000
      ix=ix+nrank      ! different seed
      do i3=1,nx3
        do i2=1,nx2
          call ranu2(ix,x(1,i2,i3),n1,icon)
          do i1=1,n1
            wc(i1,i2,i3)=x(i1,i2,i3)
          enddo
        enddo
      enddo
c     --- inquire necessary size ---
      nw=0
      call ss_v3drcf2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &        nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &        comm2,comm3,icon)
      if(icon.ne.1000) then
        print*,'icon=',icon
        go to 9000
      endif
      allocate (z(kz1,kz2,kz3),w(nw))
      print*,'nrank,nrank2,nrank3=',nrank,nrank2,nrank3,
     &       ' Z-pencil x-range=',nz1b,nz1e,
     &       ' y-range=',nz2b,nz2e,
     &       ' z-range=',1,n3
c     --- forward FFT ---
      idir=1
      isn=1
      call ss_v3drcf2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &        nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &        comm2,comm3,icon)
      if(icon.ne.0) then
        print*,'icon=',icon
        go to 9000
      endif
c     --- backward FFT ---
      idir=-1
      isn=-1
      call ss_v3drcf2x(x,kx1,kx2,kx2p,kx3p,z,kz1,kz2,kz3,
     &        nz1b,nz1e,nz2b,nz2e,n1,n2,n3,w,nw,isn,idir,
     &        comm2,comm3,icon)
      if(icon.ne.0) then
        print*,'icon=',icon
```

```
          go to 9000
        endif
c     --- check result ---
        errorx=0
        do i3=1,nx3
          do i2=1,nx2
            do i1=1,n1
              errorx=max(abs(wc(i1,i2,i3)-
     &              x(i1,i2,i3)/n1/n2/n3),errorx)
            enddo
          enddo
        enddo
        call mpi_allreduce(errorx,errormax,1,mpi_real,
     &                    mpi_max,mpi_comm_world,ierr)
        if(nrank.eq.0)then
          print*,'num proc=',nsize
          print*,'nd2,nd3=',nsize2,nsize3
          print*,'-----(',n1,',',n2,',',n3,')-----'
          print*,'error=',errormax
        endif
c
 9000 continue
      call mpi_comm_free(comm2,ierr)
      call mpi_comm_free(comm3,ierr)
      call mpi_finalize(ierr)
      stop
      end
```

Appendix

Appendix A References

[1] Markus Hegland

Block Algorithms for FFTs on Vector and Parallel Computers. PARCO 93, Grenoble, 1993.

[2] Charles Van Loan

Computational Frameworks for the Fast Fourier Transform, SIAM, 1992.

Index