FUJITSU

FUJITSU Software

FUJITSU

SSL II/MPI 使用手引書

(科学用サブルーチンライブラリ)

J2UL-2574-01Z0(02)
2021年3月

ま　え　が　き

本マニュアルは、科学用サブルーチンライブラリ SSL II/MPI(Scientific Subroutine Library II/MPI)の機能と使用方法について記述しています。
SSL II/MPI は、分散メモリ型並列計算機システム上で大規模な問題を効率良く計算するための計算機能を、並列処理向けのアルゴリズムを採用して提供しています。
本書の構成は次のとおりです。

第 **I** 部　概説

SSL II/MPI を利用する上での注意事項を示します。

第 **II** 部　サブルーチンの使用方法

個々のサブルーチンの使用方法を述べます。各サブルーチンはアルファベット順に編集されています。

SSL II/MPI を利用するためには、MPI の知識を前提としています。MPI に関する詳細は、MPI 使用手引書を参照してください。

本書は旧マニュアル（初版）を元に作成しました．記載内容が旧マニュアルと異なる箇所には，目次および SSL II/MPI 機能一覧表に＊印を付けてあります．

輸出管理規制について

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

出版年月および版数

版数	マニュアルコード
2021 年 3 月 第 1.2 版	J2UL-2574-01Z0(02)
2020 年 6 月 第 1.1 版	J2UL-2574-01Z0(01)
2020 年 2 月 初版	J2UL-2574-01Z0(00)

著作権表示

変更履歴

変更内容	変更箇所	版数
単精度機能を追加。	SS_V3DCFT2X, SS_V3DRCF2X	第 1.2 版
製品のレベルアップに伴い、体裁を変更。	-	第 1.1 版
誤字の訂正	DS_V3DCFT2X, DS_V3DRCF2X	

お　願　い

目　　次

SSL II/MPI 機能一覧表 …… 巻1

第 **I** 部　概説 1

第 1 章　SSL II/MPI の概要 …… 3

第 2 章　SSL II/MPI の一般規約 …… 5
2.1　サブルーチンの精度…… 5
2.2　サブルーチン名…… 5
2.3　パラメタ …… 5
2.4　SSL II/MPI の利用方法…… 5
2.5　コンディションコード…… 8

第 **II** 部　サブルーチンの使用方法 9
DS_V3DCFT …… 11
DS_V3DCFT2X…… 16
DS_V3DCFT3…… 24
DS_V3DRCF …… 30
DS_V3DRCF2X …… 36
DS_V3DRCF3…… 44
SS_V3DCFT2X …… 51*
SS_V3DRCF2X …… 59*

付録 1　参考文献一覧表 …… 67

索引　…… 69

SSL II/MPI 機能一覧表

フーリエ変換

サブルーチン名	項　　目	ページ
DS_V3DCFT	3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底)	11
DS_V3DCFT2X	3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散)	16
DS_V3DCFT3	3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底, 3 軸分散)	24
DS_V3DRCF	3 次元離散型実フーリエ変換 (実数から複素数へ) (2、3、5 および 7 の混合基底)	30
DS_V3DRCF2X	3 次元離散型実フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散)	36
DS_V3DRCF3	3 次元離散型実フーリエ変換 (2、3、5 および 7 の混合基底, 3 軸分散)	44
SS_V3DCFT2X *	3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散, 単精度)	51
SS_V3DRCF2X *	3 次元離散型実フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散, 単精度)	59

第I部　概説

第1章　SSL II/MPI の概要

SSL II/MPI は、分散メモリ型並列計算機上で並列動作する数値計算ライブラリであり、大規模な問題を並列処理することで効率よく計算するためのルーチンを提供しています。

SSL II/MPI の各機能はサブルーチンの形で提供されており、CALL 文により利用できます。

なお、SSL II/MPI は単一プロセッサ用および SMP 並列向けの数値計算ライブラリ SSL II とは機能範囲、サブルーチン名、および呼出し方法が異なります。

第2章　SSL II/MPI の一般規約

2.1　サブルーチンの精度

SSL II/MPI は、倍精度で演算を行うサブルーチンおよび特定ルーチンについての単精度版をサをサポートしています。

2.2　サブルーチン名

サブルーチン名の先頭 4 文字が DS_V または SS_V である利用者が CALL できる利用者サブルーチンと、先頭 4 文字が DS_U または SS_U であるスレーブサブルーチンから構成されています。

本マニュアルでは、利用者サブルーチンの使用方法についてのみ記述します。

2.3　パラメタ

(1)　パラメタの並びの順

パラメタの並びの順序は SSL II と同じく、一般に以下の順に従っています。
(入出力パラメタの並び、入力パラメタの並び、出力パラメタの並び、ICON)

(2)　パラメタの型

先頭が I、J、K、L、M、N で始まるものは整数型です。その他の文字で始まるものは、実数、あるいは複素数の浮動小数点型です。整数型のバイト数および単倍精度の区別については各ルーチンのパラメタ説明を確認してください。

2.4　SSL II/MPI の利用方法

① SSL II/MPI のサブルーチンは、MPI 環境の初期化後(MPI_INIT の呼出し後)に利用することができます。
サブルーチンには、引数としてコミュニケータを指定します。
サブルーチンは、コミュニケータに属するプロセスを利用し、並列計算を行います。
データはコミュニケータに属する各プロセスに分散されて配置されている必要があります。

② 以下に SSL II/MPI の利用例を記述します。

a.　3 次元離散型複素フーリエ変換の例を考えます。
プロセスごとに、X(KX1, KX2, KX3P)の倍精度複素数型 3 次元配列を割り当てます。変

換の対象となる 3 次元複素データを行列 D(N1, N2, N3)と仮想的に考えます。MPI のサブルーチン MPI_COMM_RANK で取得した *rank*(0, ... , $p - 1$) (p はプロセス総数)で決まるプロセスに割り当てられる配列 X に、D の 3 次元目を巾 KX3P で分割した *rank* + 1 番目の部分行列を格納します。

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

ここで、N3S = KX3P × *rank* + 1, N3E = MIN(N3, KX3P × (*rank* + 1)), N3P = MAX(0, N3E – N3S + 1)とします。このように、各プロセスに分散されたデータを利用して並列に計算します。

```
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
```

例 2.1 SSL II/MPI のルーチンの利用例

512×512×512 の 3 次元複素数データのフーリエ変換(正変換･逆変換)を 8 プロセス並列で行います。

b.　各プロセスの計算のスレッドによる並列計算

各プロセスの計算はスレッドにより並列計算することが可能です。

スレッド数は、環境変数 OMP_NUM_THREADS で指定することができます。

2.5　コンディションコード

SSL II/MPI の実行後の状態を示すパラメタ ICON が用意されています。

コードは 0 から 90000 の値をとりますが、結果が保証されているか否かに応じて表 2.1 のように区分されています。

表 2.1　コンディションコードの区分

コード	意　　味	結果の保障	区分
0	正常に処理が終了している。	結果は保障される。	正常
1～9999	正常に処理が終了しているが、なんらかの補助的な情報を含んでいる。		
10000～19999	処理の過程で、内部的な制限を加えることにより、一応の処理は終了している。	制限付きで結果は保障される。	警告
20000～29999	処理の過程で異常が生じ、処理が打ち切られた。	結果は保障されない。	異常
30000～39999	入力パラメタにエラーがあったため、処理が打ち切られた。		
90000	本ルーチンはライブラリから削除しました。より良いルーチンがあるので、そちらを使ってください。この旨は、両ルーチン名と共に標準エラー出力ファイルにメッセージ出力します。メッセージは下記参照。	―	―

ICON＝90000 時のメッセージ：

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

ここで、

AAAAA：削除されたルーチン　　BBBBB：より良いルーチン

第II部　サブルーチンの使用方法

DS_V3DCFT

3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底)
CALL DS_V3DCFT(X, KX1, KX2, KX3P, N1, N2, N3, W, KW2P, KW3, ISN, COMM, ICON)

(1) 機能

3 次元複素フーリエ変換の正変換または逆変換を行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

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

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

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

(2) パラメタ

X.....入力。複素数データ。

3 次元複素データを行列 D(N1, N2, N3)と見なします。MPI のサブルーチン MPI_COMM_RANK で取得した *rank*(0, ... , *p* − 1) (*p* はプロセス総数)で決まるプロセスに割り当てられる配列 X に、D の 3 次元目を巾 KX3P で分割した *rank* + 1 番目の部分行列を格納します。

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

ここで、N3S = KX3P × *rank* + 1, N3E = MIN(N3, KX3P × (*rank* + 1)), N3P =

MAX(0, N3E – N3S + 1)とします。
出力。変換された複素数データ。
行列 D(N1, N2, N3)の 3 次元データの変換された結果が、入力と同じ配列 X に同じように分散されて格納されます。
X(KX1, KX2, KX3P)なる倍精度複素数型 3 次元配列。

KX1.....入力。配列 X、W の 1 次元目の大きさ。(≥ N1)
4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。
KX2 = KW2P × p、p はプロセス総数。(≥ N2)
4 バイト整数型。

KX3P .....入力。配列 X の 3 次元目の大きさ。
3 次元データ D の 3 次元目を分割する巾。(KX3P × p ≥ N3、p はプロセス総数.)
4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。
4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。
4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。
4 バイト整数型。

W.....作業領域。W(KX1, KW2P, KW3)なる倍精度複素数型 3 次元配列。

KW2P.....入力。配列 W の 2 次元目の大きさ。(KW2P × p ≥ N2、p はプロセス総数。)
4 バイト整数型。

KW3.....入力。配列 W の 3 次元目の大きさ。
KW3 = KX3P × p、p はプロセス総数。
4 バイト整数型。

ISN.....入力。変換か逆変換かを指定。
変換　：ISN = 1
逆変換：ISN = –1
4 バイト整数型。

COMM.....入力。データを分散配置し、並列計算を行うプロセッサーの集合を表すコミュニケータ。
4 バイト整数型。

ICON.....出力。コンディションコード。
表 DS_V3DCFT-1 参照。
4 バイト整数型。

表 DS_V3DCFT-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
30000	n_1, n_2, n_3 が 0 または 0 以下であった。 KX1 < N1, KX2 ≠ KW2P × p, KX3P × p ≠ KW3, KX2 < N2, KW3 < N3 または ISN の値が適切でない。	処理を打ち切る。
30008	変換数が 2、3、5、7 を基底としていない。	

(3)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義されます。

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

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

ここで、$\omega_{n1} = \exp(2\pi i / n_1)$, $\omega_{n2} = \exp(2\pi i / n_2)$, $\omega_{n3} = \exp(2\pi i / n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

②　配列 X、W と性能について

KX1 は、N1 に近い値、KX2 = KW2P × p は、KX2 ≥ N2 を満たす最小の値、KW3 = KX3P × p は、KW3 ≥ N3 を満たす最小の値を設定するとき、最も通信量が少なくなります。p はプロセス総数。

b.　使用例

3 次元 FFT を 8 プロセス並列で計算します。

```
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

3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散)
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) 機能

3 次元複素フーリエ変換の正変換または逆変換を行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

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

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

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

本ルーチンでは3次元データを2次元とみなしたプロセス形状に合わせて分散格納します。各プロセスのローカルな入力配列と出力配列は、グローバルデータを異なる方向で柱状に分割することにより異なる形状を格納します。これにより、元のデータ並びに戻す通信を省略することができます。

(2) パラメタ

以降の説明において、3 次元複素データ全体を仮想的に配列 D(N1, N2, N3)と見なし、プロセス形状を ND2 × ND3 とします。

X.....IDIR = 1 のとき入力。複素数データ。

プロセス形状に合わせて、仮想配列 D の 2 次元目と 3 次元目を ND2 と ND3

でそれぞれ分割した柱状の部分配列を配列 X に格納します。
N*m* が ND*m*(*m* = 2, 3)でそれぞれ割り切れる場合、各次元の分割幅は KX*m*P = N*m* / ND*m* に設定することを薦めます。このとき、各プロセスに割り当てるデータは(N1, KX2P, KX3P)の大きさに等分配されます。
N*m* が ND*m* で割り切れない場合、KX*m*P = N*m* / ND*m* +1 に設定することを薦めます。このとき、各次元の端に相当するプロセスでは KX*m*P 未満のサイズの部分配列を格納することになります。
プロセスごとに配列 X に格納する D の部分配列は、以下で示されます。
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)
ここで *rank2*,*rank3* は、それぞれ MPI のコミュニケータ COMM2,COMM3 を用いて MPI_COMM_RANK で取得するランク ID を示します。
演算後の内容は保証されません。

.....IDIR = –1 のとき出力。変換された複素数データ。
配列 Z に格納された D(N1, N2, N3)の 3 次元データを変換した結果が、IDIR=1 での分散方式と同じように配列 X に格納されます。
X(KX1, KX2, NX3P)なる倍精度複素数型 3 次元配列。

KX1.....入力。配列 X の 1 次元目の大きさ。(≥ N1)
4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。(≥ NX2P)
4 バイト整数型。

KX2P .....入力。配列 X に格納するときに、3 次元データ D の 2 次元目を分割する巾。
(KX2P × ND2 ≥ N2)
4 バイト整数型。

KX3P .....入力。配列 X に格納するときに、3 次元データ D の 3 次元目を分割する巾。
(KX3P × ND3 ≥ N3)
4 バイト整数型。

Z.....IDIR = 1 のとき出力。変換された複素数データ。
プロセス形状に合わせて、仮想配列 D の 1 次元目と 2 次元目を ND2 と ND3 でそれぞれ分割した柱状の部分配列が配列 Z に格納されます。
プロセスごとに配列 Z に格納する D の部分配列は、以下で示されます。
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)

.....IDIR = –1 のとき入力。複素数データ。
配列 D(N1, N2, N3)の 3 次元データを、配列 Z に IDIR=1 での分散方式と同

じように格納します。

Z(KZ1, KZ2, KZ3)なる倍精度複素数型 3 次元配列。

KZ1 .....NW=0 のとき出力。配列 Z の 1 次元目の大きさの推奨値。

NW≠0 のとき入力。配列 Z の 1 次元目の大きさ。(≥ NZ1E–NZ1B+1)

4 バイト整数型。

KZ2 .....NW=0 のとき出力。配列 Z の 2 次元目の大きさの推奨値。

NW≠0 のとき入力。配列 Z の 2 次元目の大きさ。(≥ NZ2E–NZ2B+1)

4 バイト整数型。

KZ3 .....NW=0 のとき出力。配列 Z の 3 次元目の大きさの推奨値。

NW≠0 のとき入力。配列 Z の 3 次元目の大きさ。(≥ N3)

4 バイト整数型。

NZ1B .....出力。配列 Z に格納する、仮想配列 D の 1 次元目の開始インデックス。

4 バイト整数型。

NZ1E.....出力。配列 Z に格納する、仮想配列 D の 1 次元目の終了インデックス。

4 バイト整数型。

NZ2B .....出力。配列 Z に格納する、仮想配列 D の 2 次元目の開始インデックス。

4 バイト整数型。

NZ2E.....出力。配列 Z に格納する、仮想配列 D の 2 次元目の終了インデックス。

4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。

n_1 は 2、3、5、7 の巾の積として表わされる数。

4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。

n_2 は 2、3、5、7 の巾の積として表わされる数。

4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。

n_3 は 2、3、5、7 の巾の積として表わされる数。

4 バイト整数型。

W.....作業領域。大きさ NW なる倍精度複素数型 1 次元配列。

NW.....入力／出力。作業領域の大きさ。

NW = 0 が入力されたとき、NW, KZ1, KZ2, KZ3 に推奨サイズをそれぞれ出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力します。

8 バイト整数型。((3)使用上の注意 a.注意④参照)

ISN.....入力。変換か逆変換かを指定します。

変換　：ISN = 1

逆変換：ISN = –1

4 バイト整数型。

IDIR .....入力。配列の入出力方向を指定します。

配列 X を入力とし、配列 Z に出力　：IDIR = 1

配列 Z を入力とし、配列 X に出力　：IDIR = –1

4 バイト整数型。

COMM2.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND2 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意②参照)
4 バイト整数型。

COMM3.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND3 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意②参照)
4 バイト整数型。

ICON.....出力。コンディションコード。
表 DS_V3DCFT2X-1 参照。
4 バイト整数型。

表 DS_V3DCFT2X-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
1000	NW=0 が入力された。	NW,KZ1,KZ2,KZ3 に推奨サイズを出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力する。 配列 X または配列 Z への出力は行われない。
25000	作業領域が不足した。	処理を打ち切る。
30000	n_1, n_2, n_3 が 0 以下であった。 KX1 < N1, KX2 < NX2P, N2 > KX2P × ND2, N3 > KX3P × ND3 または ISN, IDIR の値が適切でない。	
30008	変換数が 2、3、5、7 を基底としていない。	
30100	COMM2 または COMM3 が適切でない。	

(3)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義されます。

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

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

ここで、$\omega_{n1}= \exp(2\pi i/n_1)$, $\omega_{n2}= \exp(2\pi i/n_2)$, $\omega_{n3}= \exp(2\pi i/n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1n_2n_3\alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

② プロセス形状 ND2 × ND3 について

実行プロセスの形状の指定が可能なシステムにおいて、ND2, ND3 の設定値が実行プロセスの形状に適合していない場合には性能が悪くなる可能性があります。プロセスの形状指定方法の有無については、ジョブ運用ソフトウェアのマニュアルを参照してください。

③ プロセス間でのパラメタ整合性について

パラメタ KX2P, KX3P, N1, N2, N3, NW, ISN, IDIR は、全プロセスで同じ値を設定する必要があります。異なっていた場合には結果は保証されません。

④ 作業領域サイズについて

ルーチン内部で MPI 通信の送受信バッファとして利用するため、配列 X または配列 Z の２倍程度のサイズを必要とします。作業領域の大きさを示すパラメタ NW は 8 バイト整数型であることにご注意ください。

b. 使用例

3 次元 FFT を 2 × 3 プロセス並列で計算します。

```
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

3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底, 3 軸分散)
CALL DS_V3DCFT3(X, KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, W, NW, ISN, COMM, ICON)

(1) 機能

3 次元複素フーリエ変換の正変換または逆変換を行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

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

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

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

本ルーチンでは3次元データを3次元とみなしたプロセス形状に合わせて分散させることができます。1 次元のみの分割では処理を分配できないような多数のプロセスを使用した場合でも高並列に実行できます。

(2) パラメタ

X.....入力。複素数データ。

3 次元複素データ全体を仮想的に配列 D(N1, N2, N3)と見なします。

プロセス形状に合わせて分割した D の部分配列を配列 X に格納します。

N*m* が ND*m*(*m* = 1, 2, 3)でそれぞれ割り切れる場合、各次元の分割幅は KX*m*P = N*m* / ND*m* に設定することを薦めます。このとき、各プロセスに割り当て

るデータは(KX1P, KX2P, KX3P)の大きさに等分配されます。
N*m* が ND*m* で割り切れない場合、KX*m*P = N*m* / ND*m* +1 に設定することを薦めます。このとき、各次元の端に相当するプロセスでは KX*m*P 未満のサイズの部分配列を格納することになります。
プロセスごとに配列 X に格納する D の部分配列は、以下で示されます。
X(1:N1P, 1:N2P, 1:N3P) ← D(N1S:N1E, N2S:N2E, N3S:N3E)
N1S = KX1P × *rank1* + 1
N1E = MIN(N1, KX1P × (*rank1* + 1))
N1P = MAX(0, N1E - N1S + 1)
N2S = KX2P × *rank2* + 1
N2E = MIN(N2, KX2P × (*rank2* + 1))
N2P = MAX(0, N2E - N2S + 1)
N3S = KX3P × *rank3* + 1
N3E = MIN(N3, KX3P × (*rank3* + 1))
N3P = MAX(0, N3E – N3S + 1)
ここで *rank1*,*rank2*,*rank3* は、MPI のサブルーチン MPI_COMM_RANK で取得した *rank* から以下のように計算される各次元方向のプロセス座標を示します。
rank1=mod(*rank*, ND1)
rank2=mod(*rank* / ND1, ND2)
rank3=*rank* / (ND1 × ND2)
出力。変換された複素数データ。
配列 D(N1, N2, N3)の 3 次元データの変換された結果が、入力と同じ配列 X に同じように分散されて格納されます。
X(KX1, KX2, KX3P)なる倍精度複素数型 3 次元配列。

KX1.....入力。配列 X の 1 次元目の大きさ。(≥ KX1P)
4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。(≥ KX2P)
4 バイト整数型。

KX1P .....入力。3 次元データ D の 1 次元目を分割する巾。(KX1P × ND1 ≥ N1)
4 バイト整数型。

KX2P .....入力。3 次元データ D の 2 次元目を分割する巾。(KX2P × ND2 ≥ N2)
4 バイト整数型。

KX3P .....入力。3 次元データ D の 3 次元目を分割する巾。(KX3P × ND3 ≥ N3)
4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。
n_1 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。
n_2 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。
n_3 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

ND1.....入力。3 次元データ D の 1 次元目を分割するプロセス数。
4 バイト整数型。
((3)使用上の注意 a.注意③参照)

ND2.....入力。3 次元データ D の 2 次元目を分割するプロセス数
4 バイト整数型。
((3)使用上の注意 a.注意③参照)

ND3.....入力。3 次元データ D の 3 次元目を分割するプロセス数。
4 バイト整数型。
((3)使用上の注意 a.注意③参照)

W.....作業領域。大きさ NW なる倍精度複素数型 1 次元配列。

NW.....入力。作業領域の大きさ(NW ≥ MAX(KX1 × ND1, KX2P × ND2, KX3P × ND3) × 3)。効率的な処理のためには十分大きなサイズを指定することをすすめます。((3)使用上の注意 a.注意②参照)
4 バイト整数型。

ISN.....入力。変換か逆変換かを指定。
変換　：ISN = 1
逆変換：ISN = –1
4 バイト整数型。

COMM.....入力。データを分散配置し、並列計算を行うプロセッサーの集合を表すコミュニケータ。
4 バイト整数型。

ICON.....出力。コンディションコード。
表 DS_V3DCFT3-1 参照。
4 バイト整数型。

表 DS_V3DCFT3-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
25000	作業領域が不足した。	処理を打ち切る。
30000	n_1, n_2, n_3 が 0 以下であった。 KX1 < KX1P, KX2 < KX2P, N1 > KX1P × ND1, N2 > KX2P × ND2, N3 > KX3P × ND3 または ISN の値が適切でない。	
30008	変換数が 2、3、5、7 を基底としていない。	
30100	ND1 × ND2 × ND3 がコミュニケータ COMM に属するプロセス総数と等しくない。	

(3) 使用上の注意

a. 注意

① 一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義されます。

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

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

ここで、$\omega_{n1} = \exp(2\pi i / n_1)$, $\omega_{n2} = \exp(2\pi i / n_2)$, $\omega_{n3} = \exp(2\pi i / n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

② 作業領域の大きさについて

作業域サイズに応じて、ルーチン内部で行うデータ転送および各ノードの計算処理の分割サイズが決まります。作業域 W のサイズ NW は、max(KX1 × ND1, KX2P × ND2, KX3P × ND3) × 3 × (プロセスあたりのスレッド数)よりも十分大きな領域を確保することを薦めます。例えば、プロセスあたりに割り当てられるキャッシュ容量が 8MB で配列 X をそのサイズで分割処理できる場合には、NW=500,000 以上を与えると効率的です。

③ パラメタ ND1, ND2, ND3 について

3 軸分散の場合、ND1, ND2, ND3 のそれぞれがプロセス総数(ND1 × ND2 × ND3)の立方根に近い値となるよう設定することを奨めます。また、実行プロセスの形状の指定が可能なシステムにおいて、ND1, ND2, ND3 の設定値が実行プロセスの形状に適合していない場合には性能が悪くなる可能性があります。プロセスの形状指定方法の有無については、ジョブ運用ソフトウェアのマニュアルを参照してください。

なお、ユーザープログラムのデータ分散方法やシステムで利用可能なプロセス形状の都合によっては、ND1, ND2, ND3 のいずれかに 1 を設定することで 1 軸または 2 軸分散として本ルーチンを使用することも可能です。

④ プロセス間でのパラメタ整合性について

パラメタ KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, NW, ISN は、全プロセスで同じ値を設定する必要があります。異なっていた場合には結果は保証されません。

b. 使用例

3 次元 FFT を $2 \times 2 \times 2$ プロセス並列で計算します。

```
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

3 次元離散型実フーリエ変換 (実数から複素数へ)(2、3、5 および 7 の混合基底)
CALL DS_V3DRCF(X, KX1, KX2, KX3P, N1, N2, N3, W, KW2P, KW3, ISIN, ISN, COMM, ICON)

(4) 機能

3 次元実フーリエ変換の正変換または逆変換を、混合基底 FFT により行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5 および 7 の巾の積で表すことのできる数です。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = &\sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r} \\
&, k_1 = 0,1,\ldots,n_1-1 \\
&, k_2 = 0,1,\ldots,n_2-1 \\
&, k_3 = 0,1,\ldots,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3) \\
&, r = 1 \text{ または } r = -1
\end{aligned}
\tag{1.1}
$$

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

$$
\begin{aligned}
x_{j1j2j3} = &\sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r} \\
&, j_1 = 0,1,\ldots,n_1-1 \\
&, j_2 = 0,1,\ldots,n_2-1 \\
&, j_3 = 0,1,\ldots,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3) \\
&, r = 1 \text{ または } r = -1
\end{aligned}
\tag{1.2}
$$

(5) パラメタ

X.....入力／出力。3 次元実データ。

3 次元実データを行列 D(N1, N2, N3)と見なします。MPI のサブルーチン MPI_COMM_RANK で取得した *rank*(0, ... , p − 1) (p はプロセス総数)で決まるプロセスに割り当てられる配列 X に、D の 3 次元目を巾 KX3P で分割した *rank* + 1 番目の部分行列を格納します。

X(1:N1, 1:N2, 1:N3P) ← D(1:N1, 1:N2, N3S:N3E)
ここで、N3S = KX3P × *rank* + 1, N3E = MIN(N3, KX3P × (*rank* + 1)), N3P = MAX(0, N3E – N3S + 1)とします。
実数から複素数への変換(ISN=1)のとき入力、複素数から実数への変換(ISN = –1)のとき出力となります。
出力／入力。変換された複素データの実部および虚部。
実数から複素数への変換(ISN=1)のとき出力、複素数から実数への変換(ISN = –1)のとき入力となります。
実データの行列 D(N1, N2, N3)の変換された結果の複素データ CD(N1, N2, N3)には、共役関係があり約半分が 1 次元目を約半分(1～N1/2+1)にして配列 X に格納されます。((3)使用上の注意 a. 注意②参照)
X を X(2, KX1/2, KX2, KX3P)なる配列と見なして、実部が X(1,1:N1/2+1, 1:N2, 1:N3P)に、虚部が X(2,1:N1/2+1, 1:N2, 1:N3P)に各プロセスで格納されます。
X(KX1, KX2, KX3P)なる倍精度実数型 3 次元配列。

KX1.....入力。配列 X、W の 1 次元目の大きさ($\geq 2 \times (n_1/2+1)$)。偶数。
4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。
KX2 = KW2P × *p*、*p* はプロセス総数。(≥ N2)
4 バイト整数型。

KX3P .....入力。配列 X の 3 次元目の大きさ。
3 次元データ D の 3 次元目を分割する巾。(KX3P × *p* ≥ N3、*p* はプロセス総数.)
4 バイト整数型。

N1.....入力。変換する実データの 1 次元目の大きさ n_1。
n_1 は、2、3、5 および 7 の巾の積で表すことのできる数。
4 バイト整数型。

N2.....入力。変換する実データの 2 次元目の大きさ n_2。
n_2 は、2、3、5 および 7 の巾の積で表すことのできる数。
4 バイト整数型。

N3.....入力。変換する実データの 3 次元目の大きさ n_3。
n_3 は、2、3、5 および 7 の巾の積で表すことのできる数。
4 バイト整数型。

W.....作業領域。W(KX1, KW2P, KW3)なる倍精度実数型 3 次元配列。

KW2P.....入力。配列 W の 2 次元目の大きさ。(KW2P × *p* ≥ N2、*p* はプロセス総数。)
4 バイト整数型。

KW3.....入力。配列 W の 3 次元目の大きさ。
KW3 = KX3P × *p*、*p* はプロセス総数。
4 バイト整数型。

ISIN.....入力。変換の方向を表します。
1 のとき　 $r = 1$
–1 のとき　$r = -1$
4 バイト整数型。

ISN.....入力。変換か逆変換かを指定。
変換　：ISN = 1
逆変換：ISN = –1
4 バイト整数型。

COMM.....入力。データを分散配置し、並列計算を行うプロセッサーの集合を表すコミュニケータ。
4 バイト整数型。

ICON.....出力。コンディションコード。
表 DS_V3DRCF-1 参照。
4 バイト整数型。

表 DS_V3DRCF-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし	ー
30000	KX1 < 2 × (N1/2+1), KX1 が偶数でない。 KX2 ≠ KW2P × *p*, KX3P × *p* ≠ KW3, KX2 < N2, KW3 < N3, N1 < 1, N2 < 1, N3 < 1, ISIN ≠ 1, –1, ISN ≠ 1, –1。	処理を打ち切る。
30008	変換数が 2, 3, 5, 7 を基底としていない。	

(6)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1), (3.2)で定義されます。

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

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

ここで、$\omega_{n1} = \exp(2\pi i / n_1)$, $\omega_{n2} = \exp(2\pi i / n_2)$, $\omega_{n3} = \exp(2\pi i / n_3)$

本サブルーチンでは、(3.1) 、(3.2)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

②　3 次元実データのフーリエ変換の結果には、次の複素共役(¯ で示します)の関係があります。

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1n2-k2n3-k3}} \tag{3.3}$$

つまり、k_1=0, ... , n_1/2, k_2=0, ... , n_2 - 1, k_3=0, ... , n_3 - 1
のデータから残りのデータは求めることができます。

b. 使用例

3 次元実 FFT を 8 プロセス並列で計算します。

```
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

3 次元離散型実フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散)
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) 機能

3 次元実フーリエ変換の正変換または逆変換を行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \qquad (1.1)$$

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

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\cdot\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \qquad (1.2)$$

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

本ルーチンでは3次元データを2次元とみなしたプロセス形状に合わせて分散格納します。各プロセスのローカルな入力配列と出力配列は、グローバルデータを異なる方向で柱状に分割することにより異なる形状を格納します。これにより、元のデータ並びに戻す通信を省略することができます。

(2) パラメタ

以降の説明において、3 次元実データ全体を仮想的に配列 DR(N1, N2, N3)、変換後の 3 次元複素データを配列 DC(N1/2+1, N2, N3)と見なし、プロセス形状を ND2 × ND3 とします。

X.....IDIR = 1 のとき入力。3 次元実データ。

プロセス形状に合わせて、仮想配列 DR の 2 次元目と 3 次元目を ND2 と ND3

でそれぞれ分割した柱状の部分配列を配列 X に格納します。
N*m* が ND*m*(*m* = 1, 2, 3)でそれぞれ割り切れる場合、各次元の分割幅は KX*m*P = N*m* / ND*m* に設定することを薦めます。このとき、各プロセスに割り当てるデータは基本的には(N1, KX2P, KX3P)の大きさに等分配されます。
N*m* が ND*m* で割り切れない場合、KX*m*P = N*m* / ND*m* +1 に設定することを薦めます。このとき、各次元の端に相当するプロセスでは KX*m*P 未満のサイズの部分配列を格納することになります。
プロセスごとに配列 X に格納する DR の部分配列は、以下で示されます。
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)
ここで *rank2*,*rank3* は、それぞれ MPI のコミュニケータ COMM2, COMM3 を用いて MPI_COMM_RANK で取得するランク ID を示します。
演算後の内容は保証されません。

.....IDIR = –1 のとき出力。変換された実データ。
配列 DC(N1/2+1, N2, N3)の 3 次元データを変換した結果が、IDIR=1 での分散方式と同じように配列 X に格納されます。
X(KX1, KX2, KX3P)なる倍精度実数型 3 次元配列。

KX1.....入力。配列 X の 1 次元目の大きさ。偶数。(≥ N1)
4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。(≥ NX2P)
4 バイト整数型。

KX2P .....入力。3 次元データ DR の 2 次元目を分割する巾。(KX2P × ND2 ≥ N2)
4 バイト整数型。

KX3P .....入力。3 次元データ DR の 3 次元目を分割する巾。(KX3P × ND3 ≥ N3)
4 バイト整数型。

Z.....IDIR = 1 のとき出力。変換された複素数データ。
実データの行列 DR(N1, N2, N3)の変換された結果の複素データには共役関係があり、1 次元目を約半分とした DC(N1/2+1, N2, N3)が配列 Z に格納されます。プロセス形状に合わせて、仮想配列 DC の 1 次元目と 2 次元目を ND2 と ND3 でそれぞれ分割した柱状の部分配列が配列 Z に格納されます。
プロセスごとに配列 Z に格納する DC の部分配列は、以下で示されます。
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)

.....IDIR = –1 のとき入力。複素数データ。
配列 DC の 3 次元データを、配列 Z に IDIR=1 での分散方式と同じように格納します。

Z(KZ1, KZ2, KZ3)なる倍精度複素数型 3 次元配列。
((3)使用上の注意 a. 注意②参照)。

KZ1 .....NW=0 のとき出力。配列 Z の 1 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 1 次元目の大きさ。(≥ NZ1E–NZ1B+1)
4 バイト整数型。

KZ2 .....NW=0 のとき出力。配列 Z の 2 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 2 次元目の大きさ。(≥ NZ2E–NZ2B+1)
4 バイト整数型。

KZ3 .....NW=0 のとき出力。配列 Z の 3 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 3 次元目の大きさ。(≥ N3)
4 バイト整数型。

NZ1B .....出力。配列 Z に格納する、仮想配列 DC の 1 次元目の開始インデックス。
4 バイト整数型。

NZ1E.....出力。配列 Z に格納する、仮想配列 DC の 1 次元目の終了インデックス。
4 バイト整数型。

NZ2B .....出力。配列 Z に格納する、仮想配列 DC の 2 次元目の開始インデックス。
4 バイト整数型。

NZ2E.....出力。配列 Z に格納する、仮想配列 DC の 2 次元目の終了インデックス。
4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。
n_1 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。
n_2 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。
n_3 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

W.....作業領域。大きさ NW なる倍精度実数型 1 次元配列。

NW.....入力／出力。作業領域の大きさ。
NW = 0 が入力されたとき、NW, KZ1, KZ2, KZ3 に推奨サイズをそれぞれ出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力します。
8 バイト整数型。((3)使用上の注意 a. 注意⑤参照)

ISN.....入力。変換か逆変換かを指定します。
変換　：ISN = 1
逆変換：ISN = –1
4 バイト整数型。

IDIR .....入力。配列の入出力方向を指定します。
実数型配列 X を入力とし、複素数型配列 Z に出力　：IDIR = 1
複素数型配列 Z を入力とし、実数型配列 X に出力　：IDIR = –1
4 バイト整数型。

COMM2.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND2 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意③参照)
4 バイト整数型。

COMM3.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND3 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意③参照)
4 バイト整数型。

ICON.....出力。コンディションコード。
表 DS_V3DRCF2X-1 参照。
4 バイト整数型。

表 DS_V3DRCF2X-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
1000	NW=0 が入力された。	NW, KZ1,KZ2,KZ3 に推奨サイズを出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力する。 配列 X または配列 Z への出力は行われない。
25000	作業領域が不足した。	処理を打ち切る。
30000	n_1, n_2, n_3 が 0 以下であった。 KX1 < N1, KX2 < NX2P, KX1 が偶数でない。 N2 > KX2P × ND2, N3 > KX3P × ND3, (N1/2+1) × 2 > KZ1 × ND2, ISN ≠ 1, –1, IDIR ≠ 1, –1。	
30008	変換数が 2、3、5、7 を基底としていない。	
30100	COMM2 または COMM3 が適切でない。	

(3)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義されます。

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

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

ここで、$\omega_{n1}= \exp(2\pi i/n_1)$, $\omega_{n2}= \exp(2\pi i/n_2)$, $\omega_{n3}= \exp(2\pi i/n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1n_2n_3\alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

② 3 次元実データのフーリエ変換の結果には、次の複素共役(¯ で示します)の関係があります。

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1n2-k2n3-k3}} \quad (3.3)$$

つまり、k_1=0, ... , n_1/2, k_2=0, ... , n_2 - 1, k_3=0, ... , n_3 - 1
のデータから残りのデータは求めることができます。

③ ND2, ND3 の値について
実行プロセスの形状の指定が可能なシステムにおいて、ND2, ND3 の設定値が実行プロセスの形状に適合していない場合には性能が悪くなる可能性があります。プロセスの形状指定方法の有無については、ジョブ運用ソフトウェアのマニュアルを参照してください。

④ プロセス間でのパラメタ整合性について
パラメタ KX2P, KX3P, N1, N2, N3, NW, ISN, IDIR は、全プロセスで同じ値を設定する必要があります。異なっていた場合には結果は保証されません。

⑤ 作業領域サイズについて
ルーチン内部で MPI 通信の送受信バッファとして利用するため、配列 X または配列 Z の２倍程度のサイズを必要とします。作業領域の大きさを示すパラメタ NW は 8 バイト整数型であることにご注意ください。

b. 使用例

3 次元 FFT を 2 × 3 プロセス並列で計算します。

```
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

3 次元離散型実フーリエ変換 (2、3、5 および 7 の混合基底, 3 軸分散)
CALL DS_V3DRCF3(X, KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, W, NW, ISIN, ISN, COMM, ICON)

(1) 機能

3 次元実フーリエ変換の正変換または逆変換を行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = &\sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r} \\
&, k_1 = 0,1,...,n_1-1 \\
&, k_2 = 0,1,...,n_2-1 \\
&, k_3 = 0,1,...,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3) \\
&, r = 1 \text{または} r = -1
\end{aligned}
\tag{1.1}
$$

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

$$
\begin{aligned}
x_{j1j2j3} = &\sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r} \\
&, j_1 = 0,1,...,n_1-1 \\
&, j_2 = 0,1,...,n_2-1 \\
&, j_3 = 0,1,...,n_3-1 \\
&, \omega_{n1} = \exp(2\pi i/n_1) \\
&, \omega_{n2} = \exp(2\pi i/n_2) \\
&, \omega_{n3} = \exp(2\pi i/n_3) \\
&, r = 1 \text{または} r = -1
\end{aligned}
\tag{1.2}
$$

本ルーチンでは3次元データを3次元とみなしたプロセス形状に合わせて分散させることができます。1 次元のみの分割では処理を分配できないような多数のプロセスを使用した場合でも高並列に実行できます。

(2) パラメタ

X.....入力／出力。3 次元実データ。

3 次元データ全体を仮想的に配列 D(N1, N2, N3)と見なします。

プロセス形状に合わせて分割した D の部分配列を配列 X に格納します。

N*m* が ND*m*(*m* = 1, 2, 3)でそれぞれ割り切れて N1/ND1 が偶数となる場合、各次元の分割幅は KX*m*P = N*m* / ND*m* に設定することができます。このとき、各プロセスに割り当てるデータは基本的には(KX1P, KX2P, KX3P)の大きさに等分配されますが、１次元目の端に相当するプロセスについては(KX1P+2, KX2P, KX3P)の出力データが格納されます。それを格納する配列サイズは全プロセスで共通に必要です。

m = 2, 3 について N*m* が ND*m* で割り切れない場合には、KX*m*P = N*m* / ND*m* +1 に設定することを薦めます。このとき、各次元の端に相当するプロセスでは KX*m*P 未満のサイズの部分配列を格納することになります。1 次元目(*m* = 1)については KX1P×ND1=N1 とならない場合、(N1/2+1)×2 ≤ KX1P×ND1 となる偶数 KX1P を設定してください。

プロセスごとに配列 X に格納する D の部分配列は、以下で示されます。

X(1:N1P, 1:N2P, 1:N3P) ← D(N1S:N1E, N2S:N2E, N3S:N3E)

N1S = KX1P × *rank1* + 1

N1E = MIN(N1, KX1P × (*rank1* + 1))

N1P = MAX(0, N1E - N1S + 1)

N2S = KX2P × *rank2* + 1

N2E = MIN(N2, KX2P × (*rank2* + 1))

N2P = MAX(0, N2E - N2S + 1)

N3S = KX3P × *rank3* + 1

N3E = MIN(N3, KX3P × (*rank3* + 1))

N3P = MAX(0, N3E – N3S + 1)

ここで *rank1*,*rank2*,*rank3* は、MPI のサブルーチン MPI_COMM_RANK で取得した *rank* から以下のように計算される各次元方向のプロセス座標を示します。

rank1=mod(*rank*, ND1)

rank2=mod(*rank* / ND1, ND2)

rank3=*rank* / (ND1 × ND2)

実数から複素数への変換(ISN=1)のとき入力、複素数から実数への変換(ISN = –1)のとき出力となります。

出力／入力。変換された複素数データ。

実データの行列 D(N1, N2, N3)の変換された結果の複素データ CD(N1, N2, N3)には、共役関係があり約半分が 1 次元目を約半分(1～N1/2+1)にして配列 X に格納されます。((3)使用上の注意 a. 注意②参照)

X を X(2, KX1/2, KX2, KX3P)なる配列と見なした時、

N1S = KX1P/2 × *rank1* + 1

N1E = MIN(N1/2+1, KX1P/2 × (*rank1* + 1))

N1P = MAX(0, N1E - N1S + 1)

として、実部が X(1,1:N1P, 1:N2P, 1:N3P)に、虚部が X(2,1:N1P, 1:N2P, 1:N3P)に各プロセスで格納されます。ただし KX1P × ND1 = N1 の場合、*rank1*=ND1–1 のプロセスでは X(1:2,1:N1P+1, 1:N2P, 1:N3P)に格納されます。

実数から複素数への変換(ISN=1)のとき出力、複素数から実数への変換

(ISN=–1)のとき入力となります。

X(KX1, KX2, KX3P)なる倍精度実数型 3 次元配列。

KX1.....入力。配列 X の 1 次元目の大きさ。偶数。

KX1P × ND1 = N1 のとき、KX1 ≥ KX1P+2。それ以外のとき、KX1 ≥ KX1P。

4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。(≥ KX2P)

4 バイト整数型。

KX1P .....入力。3 次元データ D の 1 次元目を分割する巾(KX1P × ND1 ≥ N1)。偶数。

4 バイト整数型。

KX2P .....入力。3 次元データ D の 2 次元目を分割する巾。(KX2P × ND2 ≥ N2)

4 バイト整数型。

KX3P .....入力。3 次元データ D の 3 次元目を分割する巾。(KX3P × ND3 ≥ N3)

4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。

n_1 は 2、3、5、7 の巾の積として表わされる数。

4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。

n_2 は 2、3、5、7 の巾の積として表わされる数。

4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。

n_3 は 2、3、5、7 の巾の積として表わされる数。

4 バイト整数型。

ND1.....入力。3 次元データ D の 1 次元目を分割するプロセス数。

4 バイト整数型。

((3)使用上の注意 a.注意④参照)

ND2.....入力。3 次元データ D の 2 次元目を分割するプロセス数

4 バイト整数型。

((3)使用上の注意 a.注意④参照)

ND3.....入力。3 次元データ D の 3 次元目を分割するプロセス数。

4 バイト整数型。

((3)使用上の注意 a.注意④参照)

W.....作業領域。大きさ NW なる倍精度実数型 1 次元配列。

NW.....入力。作業領域の大きさ(NW ≥ MAX(KX1 × ND1, KX2P × ND2, KX3P × ND3) × 6)。効率的な処理のためには十分大きなサイズを指定することをすすめます。((3)使用上の注意 a.注意③参照)

4 バイト整数型。

ISIN.....入力。変換の方向を表します。

1 のとき　$r = 1$

–1 のとき　$r = -1$

4 バイト整数型。

ISN.....入力。変換か逆変換かを指定。

変換　：ISN = 1

逆変換：ISN = –1

4 バイト整数型。

COMM.....入力。データを分散配置し、並列計算を行うプロセッサーの集合を表すコミュニケータ。

4 バイト整数型。

ICON.....出力。コンディションコード。

表 DS_V3DRCF3-1 参照。

4 バイト整数型。

表 DS_V3DRCF3-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
25000	作業領域が不足した。	処理を打ち切る。
30000	n_1, n_2, n_3 が 0 以下であった。 KX1 < KX1P, KX2 < KX2P, KX1 または KX1P が偶数でない。 N1 > KX1P × ND1, N2 > KX2P × ND2, N3 > KX3P × ND3, (N1/2+1) × 2 > KX1 × ND1, ISIN ≠ 1, –1, ISN ≠ 1, –1。	
30008	変換数が 2、3、5、7 を基底としていない。	
30100	ND1 × ND2 × ND3 がコミュニケータ COMM に属するプロセス総数と等しくない。	

(3)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義されます。

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

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

ここで、$\omega_{n1} = \exp(2\pi i / n_1)$, $\omega_{n2} = \exp(2\pi i / n_2)$, $\omega_{n3} = \exp(2\pi i / n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

② 3 次元実データのフーリエ変換の結果には、次の複素共役(¯ で示します)の関係があります。

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1n2-k2n3-k3}} \tag{3.3}$$

つまり、k_1=0, ... , n_1/2, k_2=0, ... , n_2-1, k_3=0, ... , n_3-1
のデータから残りのデータは求めることができます。

③ 作業領域の大きさについて
作業域サイズに応じて、ルーチン内部で行うデータ転送および各ノードの計算処理の分割サイズが決まります。作業域 W のサイズ NW は、max(KX1 × ND1, KX2P × ND2, KX3P × ND3) × 6 × (プロセスあたりのスレッド数)よりも十分大きな領域を確保することを薦めます。例えば、プロセスあたりに割り当てられるキャッシュ容量が 8MB で配列 X をそのサイズで分割処理できる場合には、NW=1,000,000 以上を与えると効率的です。

④ パラメタ ND1, ND2, ND3 について
3 軸分散の場合、ND1, ND2, ND3 のそれぞれがプロセス総数(ND1 × ND2 × ND3)の立方根に近い値となるよう設定することを奨めます。また、実行プロセスの形状の指定が可能なシステムにおいて、ND1, ND2, ND3 の設定値が実行プロセスの形状に適合していない場合には性能が悪くなる可能性があります。プロセスの形状指定方法の有無については、ジョブ運用ソフトウェアのマニュアルを参照してください。
なお、ユーザープログラムのデータ分散方法やシステムで利用可能なプロセス形状の都合によっては、ND1, ND2, ND3 のいずれかに 1 を設定することで 1 軸または 2 軸分散として本ルーチンを使用することも可能です。

⑤ プロセス間でのパラメタ整合性について
パラメタ KX1, KX2, KX1P, KX2P, KX3P, N1, N2, N3, ND1, ND2, ND3, NW, ISIN, ISN は、全プロセスで同じ値を設定する必要があります。異なっていた場合には結果は保証されません。

b. 使用例

3 次元 FFT を 2 × 2 × 2 プロセス並列で計算します。

```
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

3 次元離散型複素フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散, 単精度)
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) 機能

3 次元複素フーリエ変換の正変換または逆変換を単精度で行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \quad (1.1)$$

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

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

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

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

本ルーチンでは3次元データを2次元とみなしたプロセス形状に合わせて分散格納します。各プロセスのローカルな入力配列と出力配列は、グローバルデータを異なる方向で柱状に分割することにより異なる形状を格納します。これにより、元のデータ並びに戻す通信を省略することができます。本ルーチンは DS_V3DCFT2X の単精度版です。

(2) パラメタ

以降の説明において、3 次元複素データ全体を仮想的に配列 D(N1, N2, N3)と見なし、プロセス形状を ND2 × ND3 とします。

X.....IDIR = 1 のとき入力。複素数データ。

プロセス形状に合わせて、仮想配列 D の 2 次元目と 3 次元目を ND2 と ND3 でそれぞれ分割した柱状の部分配列を配列 X に格納します。

N*m* が ND*m*(*m* = 2, 3)でそれぞれ割り切れる場合、各次元の分割幅は KX*m*P = N*m* / ND*m* に設定することを薦めます。このとき、各プロセスに割り当てるデータは(N1, KX2P, KX3P)の大きさに等分配されます。

N*m* が ND*m* で割り切れない場合、KX*m*P = N*m* / ND*m* +1 に設定することを薦めます。このとき、各次元の端に相当するプロセスでは KX*m*P 未満のサイズの部分配列を格納することになります。

プロセスごとに配列 X に格納する D の部分配列は、以下で示されます。

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)

ここで *rank2*,*rank3* は、それぞれ MPI のコミュニケータ COMM2,COMM3 を用いて MPI_COMM_RANK で取得するランク ID を示します。

演算後の内容は保証されません。

.....IDIR = –1 のとき出力。変換された複素数データ。

配列 Z に格納された D(N1, N2, N3)の 3 次元データを変換した結果が、IDIR=1 での分散方式と同じように配列 X に格納されます。

X(KX1, KX2, NX3P)なる単精度複素数型 3 次元配列。

KX1.....入力。配列 X の 1 次元目の大きさ。(≥ N1)

4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。(≥ NX2P)

4 バイト整数型。

KX2P .....入力。配列 X に格納するときに、3 次元データ D の 2 次元目を分割する巾。(KX2P × ND2 ≥ N2)

4 バイト整数型。

KX3P .....入力。配列 X に格納するときに、3 次元データ D の 3 次元目を分割する巾。(KX3P × ND3 ≥ N3)

4 バイト整数型。

Z.....IDIR = 1 のとき出力。変換された複素数データ。

プロセス形状に合わせて、仮想配列 D の 1 次元目と 2 次元目を ND2 と ND3 でそれぞれ分割した柱状の部分配列が配列 Z に格納されます。

プロセスごとに配列 Z に格納する D の部分配列は、以下で示されます。

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)

.....IDIR = –1 のとき入力。複素数データ。
配列 D(N1, N2, N3)の 3 次元データを、配列 Z に IDIR=1 での分散方式と同じように格納します。
Z(KZ1, KZ2, KZ3)なる単精度複素数型 3 次元配列。

KZ1 .....NW=0 のとき出力。配列 Z の 1 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 1 次元目の大きさ。(≥ NZ1E–NZ1B+1)
4 バイト整数型。

KZ2 .....NW=0 のとき出力。配列 Z の 2 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 2 次元目の大きさ。(≥ NZ2E–NZ2B+1)
4 バイト整数型。

KZ3 .....NW=0 のとき出力。配列 Z の 3 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 3 次元目の大きさ。(≥ N3)
4 バイト整数型。

NZ1B .....出力。配列 Z に格納する、仮想配列 D の 1 次元目の開始インデックス。
4 バイト整数型。

NZ1E.....出力。配列 Z に格納する、仮想配列 D の 1 次元目の終了インデックス。
4 バイト整数型。

NZ2B .....出力。配列 Z に格納する、仮想配列 D の 2 次元目の開始インデックス。
4 バイト整数型。

NZ2E.....出力。配列 Z に格納する、仮想配列 D の 2 次元目の終了インデックス。
4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。
n_1 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。
n_2 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。
n_3 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

W.....作業領域。大きさ NW なる単精度複素数型 1 次元配列。

NW.....入力／出力。作業領域の大きさ。
NW = 0 が入力されたとき、NW, KZ1, KZ2, KZ3 に推奨サイズをそれぞれ出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力します。
8 バイト整数型。((3)使用上の注意 a.注意④参照)

ISN.....入力。変換か逆変換かを指定します。
変換　：ISN = 1
逆変換：ISN = –1
4 バイト整数型。

IDIR .....入力。配列の入出力方向を指定します。
配列 X を入力とし、配列 Z に出力　：IDIR = 1

配列 Z を入力とし、配列 X に出力　：IDIR = –1

4 バイト整数型。

COMM2.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND2 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意②参照)

4 バイト整数型。

COMM3.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND3 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意②参照)

4 バイト整数型。

ICON.....出力。コンディションコード。

表 DS_V3DCFT2X-1 参照。

4 バイト整数型。

表 SS_V3DCFT2X-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
1000	NW=0 が入力された。	NW,KZ1,KZ2,KZ3 に推奨サイズを出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力する。 配列 X または配列 Z への出力は行われない。
25000	作業領域が不足した。	処理を打ち切る。
30000	n_1, n_2, n_3 が 0 以下であった。 KX1 < N1, KX2 < NX2P, N2 > KX2P × ND2, N3 > KX3P × ND3 または ISN, IDIR の値が適切でない。	
30008	変換数が 2、3、5、7 を基底としていない。	
30100	COMM2 または COMM3 が適切でない。	

(3)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義さ

れます。

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

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

ここで、$\omega_{n1} = \exp(2\pi i / n_1)$, $\omega_{n2} = \exp(2\pi i / n_2)$, $\omega_{n3} = \exp(2\pi i / n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

② プロセス形状 ND2 × ND3 について

実行プロセスの形状の指定が可能なシステムにおいて、ND2, ND3 の設定値が実行プロセスの形状に適合していない場合には性能が悪くなる可能性があります。プロセスの形状指定方法の有無については、ジョブ運用ソフトウェアのマニュアルを参照してください。

③ プロセス間でのパラメタ整合性について

パラメタ KX2P, KX3P, N1, N2, N3, NW, ISN, IDIR は、全プロセスで同じ値を設定する必要があります。異なっていた場合には結果は保証されません。

④ 作業領域サイズについて

ルーチン内部で MPI 通信の送受信バッファとして利用するため、配列 X または配列 Z の２倍程度のサイズを必要とします。作業領域の大きさを示すパラメタ NW は 8 バイト整数型であることにご注意ください。

b. 使用例

3 次元 FFT を 2 × 3 プロセス並列で計算します。

```
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

3 次元離散型実フーリエ変換 (2、3、5 および 7 の混合基底, 2 軸分散, 単精度)
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) 機能

3 次元実フーリエ変換の正変換または逆変換を単精度で行います。

3 次元データ(n_1, n_2, n_3)の各次元の大きさは、2、3、5、7 の巾の積として表わされる数でなければなりません。

a. 3 次元フーリエ変換

$\{x_{j1j2j3}\}$を入力し、(1.1)で定義する変換を行い、$\{n_1n_2n_3\alpha_{k1k2k3}\}$を求めます。

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3} \qquad (1.1)$$

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

b. 3 次元フーリエ逆変換

$\{\alpha_{k1k2k3}\}$を入力し、(1.2)で定義する変換を行い、$\{x_{j1j2j3}\}$を求めます。

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\cdot\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3} \qquad (1.2)$$

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

本ルーチンでは3次元データを2次元とみなしたプロセス形状に合わせて分散格納します。各プロセスのローカルな入力配列と出力配列は、グローバルデータを異なる方向で柱状に分割することにより異なる形状を格納します。これにより、元のデータ並びに戻す通信を省略することができます。本ルーチンは DS_V3DRCF2X の単精度版です。

(2) パラメタ

以降の説明において、3 次元実データ全体を仮想的に配列 DR(N1, N2, N3)、変換後の 3 次元複素データを配列 DC(N1/2+1, N2, N3)と見なし、プロセス形状を ND2 × ND3 とします。

X.....IDIR = 1 のとき入力。3 次元実データ。

プロセス形状に合わせて、仮想配列 DR の 2 次元目と 3 次元目を ND2 と ND3

でそれぞれ分割した柱状の部分配列を配列 X に格納します。
N*m* が ND*m*(*m* = 1, 2, 3)でそれぞれ割り切れる場合、各次元の分割幅は KX*m*P = N*m* / ND*m* に設定することを薦めます。このとき、各プロセスに割り当てるデータは基本的には(N1, KX2P, KX3P)の大きさに等分配されます。
N*m* が ND*m* で割り切れない場合、KX*m*P = N*m* / ND*m* +1 に設定することを薦めます。このとき、各次元の端に相当するプロセスでは KX*m*P 未満のサイズの部分配列を格納することになります。
プロセスごとに配列 X に格納する DR の部分配列は、以下で示されます。
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)
ここで *rank2*,*rank3* は、それぞれ MPI のコミュニケータ COMM2, COMM3 を用いて MPI_COMM_RANK で取得するランク ID を示します。
演算後の内容は保証されません。

.....IDIR = –1 のとき出力。変換された実データ。
仮想配列 DC(N1/2+1, N2, N3)の 3 次元データを変換した結果が、IDIR=1 での分散方式と同じように配列 X に格納されます。
X(KX1, KX2, KX3P)なる単精度実数型 3 次元配列。

KX1.....入力。配列 X の 1 次元目の大きさ。偶数。(≥ N1)
4 バイト整数型。

KX2.....入力。配列 X の 2 次元目の大きさ。(≥ NX2P)
4 バイト整数型。

KX2P .....入力。3 次元データ DR の 2 次元目を分割する巾。(KX2P × ND2 ≥ N2)
4 バイト整数型。

KX3P .....入力。3 次元データ DR の 3 次元目を分割する巾。(KX3P × ND3 ≥ N3)
4 バイト整数型。

Z.....IDIR = 1 のとき出力。変換された複素数データ。
実データの行列 DR(N1, N2, N3)の変換された結果の複素データには共役関係があり、1 次元目を約半分とした DC(N1/2+1, N2, N3)が配列 Z に格納されます。プロセス形状に合わせて、仮想配列 DC の 1 次元目と 2 次元目を ND2 と ND3 でそれぞれ分割した柱状の部分配列が配列 Z に格納されます。
プロセスごとに配列 Z に格納する DC の部分配列は、以下で示されます。
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)

.....IDIR = –1 のとき入力。複素数データ。
仮想配列 DC の 3 次元データを、配列 Z に IDIR=1 での分散方式と同じように格納します。

Z(KZ1, KZ2, KZ3)なる単精度複素数型 3 次元配列。
((3)使用上の注意 a. 注意②参照)。

KZ1 .....NW=0 のとき出力。配列 Z の 1 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 1 次元目の大きさ。(≥ NZ1E–NZ1B+1)
4 バイト整数型。

KZ2 .....NW=0 のとき出力。配列 Z の 2 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 2 次元目の大きさ。(≥ NZ2E–NZ2B+1)
4 バイト整数型。

KZ3 .....NW=0 のとき出力。配列 Z の 3 次元目の大きさの推奨値。
NW≠0 のとき入力。配列 Z の 3 次元目の大きさ。(≥ N3)
4 バイト整数型。

NZ1B .....出力。配列 Z に格納する、仮想配列 DC の 1 次元目の開始インデックス。
4 バイト整数型。

NZ1E.....出力。配列 Z に格納する、仮想配列 DC の 1 次元目の終了インデックス。
4 バイト整数型。

NZ2B .....出力。配列 Z に格納する、仮想配列 DC の 2 次元目の開始インデックス。
4 バイト整数型。

NZ2E.....出力。配列 Z に格納する、仮想配列 DC の 2 次元目の終了インデックス。
4 バイト整数型。

N1.....入力。変換する 3 次元データの 1 次元目の大きさ n_1。
n_1 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N2.....入力。変換する 3 次元データの 2 次元目の大きさ n_2。
n_2 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

N3.....入力。変換する 3 次元データの 3 次元目の大きさ n_3。
n_3 は 2、3、5、7 の巾の積として表わされる数。
4 バイト整数型。

W.....作業領域。大きさ NW なる単精度実数型 1 次元配列。

NW.....入力／出力。作業領域の大きさ。
NW = 0 が入力されたとき、NW, KZ1, KZ2, KZ3 に推奨サイズをそれぞれ出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力します。
8 バイト整数型。((3)使用上の注意 a. 注意⑤参照)

ISN.....入力。変換か逆変換かを指定します。
変換　：ISN = 1
逆変換：ISN = –1
4 バイト整数型。

IDIR .....入力。配列の入出力方向を指定します。
実数型配列 X を入力とし、複素数型配列 Z に出力　：IDIR = 1
複素数型配列 Z を入力とし、実数型配列 X に出力　：IDIR = –1
4 バイト整数型。

COMM2.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND2 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意③参照)
4 バイト整数型。

COMM3.....入力。ND2 × ND3 のプロセス形状において、MPI_COMM_SIZE で取得するプロセスグループのサイズが ND3 となるプロセスの集合を表す MPI のコミュニケータ。((3)使用上の注意 a.注意③参照)
4 バイト整数型。

ICON.....出力。コンディションコード。
表 SS_V3DRCF2X-1 参照。
4 バイト整数型。

表 SS_V3DRCF2X-1 コンディションコード

コード	意　　味	処理内容
0	エラーなし。	ー
1000	NW=0 が入力された。	NW, KZ1,KZ2,KZ3 に推奨サイズを出力し、NZ1B, NZ1E, NZ2B, NZ2E にインデックス情報を出力する。 配列 X または配列 Z への出力は行われない。
25000	作業領域が不足した。	処理を打ち切る。
30000	n_1, n_2, n_3 が 0 以下であった。 KX1 < N1, KX2 < NX2P, KX1 が偶数でない。 N2 > KX2P × ND2, N3 > KX3P × ND3, (N1/2+1) × 2 > KZ1 × ND2, ISN ≠ 1, –1, IDIR ≠ 1, –1。	
30008	変換数が 2、3、5、7 を基底としていない。	
30100	COMM2 または COMM3 が適切でない。	

(3)　使用上の注意

a.　注意

①　一般的なフーリエ変換の定義

3 次元離散型複素フーリエ変換および、逆変換は一般的に(3.1) 、(3.2)で定義されます。

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

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

ここで、$\omega_{n1}= \exp(2\pi i/n_1)$, $\omega_{n2}= \exp(2\pi i/n_2)$, $\omega_{n3}= \exp(2\pi i/n_3)$

本サブルーチンでは、(3.1)、(3.2)の左辺に対応して$\{n_1n_2n_3\alpha_{k1k2k3}\}$または$\{x_{j1j2j3}\}$を求めます。したがって、結果の正規化は必要に応じて行って下さい。

② 3 次元実データのフーリエ変換の結果には、次の複素共役(¯ で示します)の関係があります。

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1n2-k2n3-k3}} \quad (3.3)$$

つまり、k_1=0, ... , n_1/2, k_2=0, ... , n_2-1, k_3=0, ... , n_3-1
のデータから残りのデータは求めることができます。

③ ND2, ND3 の値について
実行プロセスの形状の指定が可能なシステムにおいて、ND2, ND3 の設定値が実行プロセスの形状に適合していない場合には性能が悪くなる可能性があります。プロセスの形状指定方法の有無については、ジョブ運用ソフトウェアのマニュアルを参照してください。

④ プロセス間でのパラメタ整合性について
パラメタ KX2P, KX3P, N1, N2, N3, NW, ISN, IDIR は、全プロセスで同じ値を設定する必要があります。異なっていた場合には結果は保証されません。

⑤ 作業領域サイズについて
ルーチン内部で MPI 通信の送受信バッファとして利用するため、配列 X または配列 Z の２倍程度のサイズを必要とします。作業領域の大きさを示すパラメタ NW は 8 バイト整数型であることにご注意ください。

b. 使用例

3 次元 FFT を 2 × 3 プロセス並列で計算します。

```
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
```

付録1　参考文献一覧表

[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.

索　　引

さ

3次元離散型実フーリエ変換 22, 28
3次元離散型複素フーリエ変換 11, 16

に

2、3、5および7の混合基底 11, 16, 22, 28