FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

Development Studio
Fortran使用手引書 別冊 COARRAY

J2UL-2559-01Z0(01)
2020年11月

まえがき

本書の目的

共配列(coarray)は、Fortran 2008規格で定義されている文法の一部です。本書では、Fortran処理系(以降、本処理系と呼びます)で共配列を使用する方法について説明します。

本書では、以下のように定義します。

COARRAYプログラム

共配列を使ったプログラム

COARRAY機能

COARRAYプログラムを翻訳・実行する機能

本書の読者

本書は、本処理系でCOARRAY機能を使用する方を対象に記述しています。

本書を読むにあたっては、Fortranプログラム、Cプログラム、C++プログラム、OpenMP、MPI、およびLinuxのコマンド・ファイル操作・シェルプログラミングの基本的な知識が必要です。

本書の構成

本書は、以下の構成になっています。

- 第1章 COARRAY機能の概要と利用
- 第2章 COARRAYによる並列化
- 第3章 COARRAYプログラムからのMPIの使用
- 第4章 COARRAYプログラムのデバッグ
- 第5章 COARRAYプログラムのチューニング
- 第6章 注意事項

関連するマニュアル

関連するマニュアルを以下に示します。

- Fortran文法書
- Fortran使用手引書
- Fortran翻訳時メッセージ
- Fortran/C/C++実行時メッセージ
- C言語使用手引書
- C++言語使用手引書
- MPI使用手引書

上記以外に、以下の関連ソフトウェアのマニュアルも必要に応じて参照してください。

- ジョブ運用ソフトウェア
- FEFS/LLIO

本書の注意事項

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

本書の表記について

構文表記記号

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

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

輸出管理規制について

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

商標

- OpenMPは、OpenMP Architecture Review Boardの商標です。
- Linux(R)は米国及びその他の国におけるLinus Torvaldsの登録商標です。
- そのほか、本マニュアルに記載されている会社名および製品名は、それぞれ各社の商標または登録商標です。

出版年月および版数

版数	マニュアルコード
2020年11月 第1.1版	J2UL-2559-01Z0(01)
2020年 2月 初版	J2UL-2559-01Z0(00)

著作権表示

変更履歴

変更内容	変更箇所	版数
Cプログラムとの結合の説明文を見直しました。	1.2.2	第1.1版

目　次

第1章 COARRAY機能の概要と利用

この章では、本処理系が提供するCOARRAY機能の概要と利用について説明します。

1.1 COARRAY機能の概要

本処理系が提供するCOARRAY機能の概要を説明します。

本処理系のCOARRAY機能は、プロセス並列だけをサポートしています。スレッド並列については、OpenMPをご利用ください。OpenMP仕様による並列化については、“Fortran使用手引書”の“OpenMP仕様による並列化”を参照してください。

1.1.1 本処理系のCOARRAY機能の特徴

本処理系のCOARRAY機能は、MPIライブラリを利用しています。

MPI(Message Passing Interface)は、MPI Forumで定められた規格であり、分散メモリ型の並列計算機システムにおいて、FortranまたはC言語によるMPI並列プログラミングを可能とするライブラリインターフェースを規定しています。

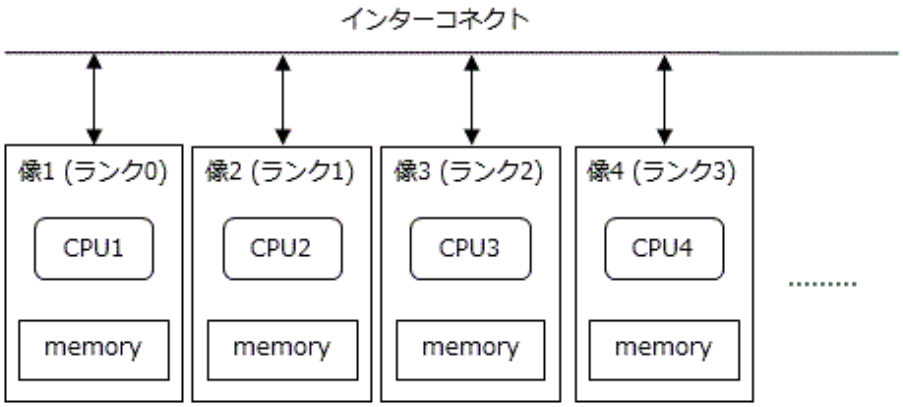
また、本処理系はMPIプログラムとリンクすることができます。

1.1.2 像とランク

COARRAYプログラムは、複製されて非同期に実行されます。それぞれの複製を“像(image)”と呼びます。像は1から始まる番号をもち、この番号を“像番号(image index)”と呼びます。

本処理系では、実行環境としてMPIをベースとしているため、像を“ランク”と呼ぶことがあります。ランクは0から始まる番号をもち、この番号を“ランク番号”と呼びます。

像番号とランク番号の関係は、「像番号＝ランク番号＋1」となります。



1.2 COARRAY機能の利用

COARRAYプログラムの翻訳と実行について説明します。

1.2.1 翻訳方法

COARRAY機能に関連する翻訳時オプションの形式と意味を説明します。

オプションの詳細は、“Fortran使用手引書”の“翻訳時オプション”を参照してください。

-N{coarray|nocoarray}

COARRAY仕様を有効にするかどうかを指示します。

指定しない場合は、-Nnocoarrayが指定されたとみなします。

coarray

COARRAY仕様を有効にします。

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

例

```
$ frt a.f90 -Ncoarray -c
$ frt a.o -Ncoarray
```

nocoarray

COARRAY仕様を無効にします。

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

1.2.2 C/C++プログラムとの言語間結合

COARRAY機能を利用しているFortranプログラムとC/C++プログラムをリンクして実行することができます。

言語間結合の詳細は、各言語の使用手引書を参照してください。

Cプログラムとの言語間結合

Fortranコンパイラ(frtまたはfrtpx)を使用します。リンク時に-Ncoarrayオプションを指定してください。

もしくは、Cコンパイラ(fccまたはfccpx)を使用します。リンク時に--linkcoarrayオプションを指定してください。

C++プログラムとの言語間結合

C++コンパイラ(FCCまたはFCCpx)を使用します。リンク時に--linkcoarrayオプションを指定してください。

1.2.3 実行方法

-Ncoarrayオプションを指定して翻訳・リンクした実行可能プログラムは、計算ノード上でmpiexecコマンドを使用して実行します。ただし、利用者が計算ノード上で実行するのではなく、実行可能プログラムを実行するジョブの投入を、ジョブ運用ソフトウェアに依頼します。

ジョブの投入方法については、ジョブ運用ソフトウェアのマニュアルを参照してください。mpiexecコマンドの詳細は、“MPI使用手引書”を参照してください。

ジョブ運用ソフトウェアのマニュアルでは、プロセス並列に関する説明はMPIプロセスを前提に記載されています。適宜、COARRAY向けに読み替えてください。

1.2.3.1 実行コマンドの形式

MPIプログラムを実行するmpiexecコマンドを使用して、COARRAYプログラムを実行します。

本書では代表的な使用方法を示します。

mpiexecコマンドの形式

コマンド名	オペランド
mpiexec	*global_options local_options execfile execfile_arguments*

mpiexecコマンドのランクについては、“1.1.2 像とランク”を参照してください。

例

```
$ mpiexec -of-proc procfile ./a.out
```

global_options

*global options*に指定できるオプションを以下に示します。

{ -h | --help }

本コマンドのヘルプメッセージを表示してmpiexecコマンドを終了します。

{ -of | --of | -std | --std } *FILE*

並列プロセスの標準出力と標準エラー出力をファイルに出力します。出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

出力先は、ジョブ運用ソフトウェアのジョブACL(Access Control List)機能で設定されている制限によって異なります。
本オプションが許可されている場合、*FILE*で指定したファイル名のファイルに出力します。また、*FILE*の指定にはメタ文字を指定することもできます。
本オプションが許可されていない場合、出力先はジョブ運用ソフトウェアのジョブACL機能の設定に従います。

メタ文字の指定およびジョブACL機能の設定については、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。

{ -oferr | --oferr | -stderr | --stderr } *ERR_FILE*

並列プロセスの標準エラー出力をファイルに出力します。出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

出力先は、ジョブ運用ソフトウェアのジョブACL機能で設定されている制限によって異なります。
本オプションが許可されている場合、*ERR_FILE*で指定したファイル名のファイルに出力します。また、*ERR_FILE*の指定にはメタ文字を指定することもできます。
本オプションが許可されていない場合、出力先はジョブ運用ソフトウェアのジョブACL機能の設定に従います。

メタ文字の指定およびジョブACL機能の設定については、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。

{ -oferr-proc | --oferr-proc | -stderr-proc | --stderr-proc } *ERR_PROC_FILE*

並列プロセスの標準エラー出力をプロセスごとに“*ERR_PROC_FILE.mpiexec.rank*”というファイル名のファイルに出力します。また、*ERR_PROC_FILE*の指定にはメタ文字を指定することもできます。メタ文字の指定については、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。

出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -ofout | --ofout | -stdout | --stdout } *OUT_FILE*

並列プロセスの標準出力をファイルに出力します。出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

出力先は、ジョブ運用ソフトウェアのジョブACL機能で設定されている制限によって異なります。
本オプションが許可されている場合、*OUT_FILE*で指定したファイル名のファイルに出力します。また、*OUT_FILE*の指定にはメタ文字を指定することもできます。
本オプションが許可されていない場合、出力先はジョブ運用ソフトウェアのジョブACL機能の設定に従います。

メタ文字の指定およびジョブACL機能の設定については、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。

{ -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc } *OUT_PROC_FILE*

並列プロセスの標準出力をプロセスごとに“*OUT_PROC_FILE.mpiexec.rank*”というファイル名のファイルに出力します。また、*OUT_PROC_FILE*の指定にはメタ文字を指定することもできます。メタ文字の指定については、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。

出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -of-proc | --of-proc | -std-proc | --std-proc } *PROC_FILE*

並列プロセスの標準出力と標準エラー出力をプロセスごとに“*PROC_FILE.mpiexec.rank*”というファイル名のファイルに出力します。また、*PROC_FILE*の指定にはメタ文字を指定することもできます。メタ文字の指定については、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。

出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -ofprefix | --ofprefix | -stdprefix | --stdprefix } *PREFIX*

並列プロセスの標準出力と標準エラー出力の行の先頭に、*PREFIX*に指定したキーワードに対応する内容の文字列を出力します。

*PREFIX*に指定できるキーワードは、date、rank、およびnidです。キーワードはコンマ(,)で区切って複数指定できます(例: date, rank, nid)。キーワードを複数指定した場合は、date、rank、nid の順に対応する内容を出力します。それぞれのキーワードの指定によって、出力する文字列の形式は次のとおりです。

date

出力する文字列の先頭に出力時刻を付加します。

rank

出力する文字列の先頭にMPI_COMM_WORLDでのランクを付加します。

nid

出力する文字列の先頭にノードIDを付加します。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

ノードIDは、並列プロセスが割り当てられた計算ノードを識別する番号です。ノードID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。

{ -stdin | --stdin } *STDIN_FILE*

COARRAYプログラムの実行によって生成されるすべての並列プロセスの標準入力を、*STDIN_FILE*に指定したファイル名のファイルから読み込みます。ファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -V | --version }

本コマンドのバージョン情報を表示してmpiexecコマンドを終了します。

プロセスごとに出力される標準出力ファイル名と標準エラー出力ファイル名の文字列について説明します。
“*mpiexec*”の文字列は、ジョブスクリプト内で何回目のmpiexecコマンドの実行であるかを表したものです。“*rank*”の文字列は、実際のMPI_COMM_WORLDでのランクを表したものです。

local_options

*local_options*に指定できるオプションを以下に示します。

-x *NAME=VALUE*

COARRAYプログラムを実行する時の環境変数を指定します。

*NAME*は環境変数名、*VALUE*は*NAME*に設定する値です。

指定に空白を含めたい場合は、次のように引用符で括ってください。

```
"NAME=VALUE"
```

環境変数を複数指定したい場合は、本オプションを複数指定してください。

例

```
-x OMP_NUM_THREADS=8 -x THREAD_STACK_SIZE=4096
```

同じ環境変数を複数指定した場合は、最後に指定した内容が優先されます。

{ -c | -np | --np | -n | --n } *N*

対応するCOARRAYプログラムの並列プロセス数(整数)を指定します。

本オプションを省略した場合、生成できる最大値が指定されたものとみなします。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

1.2.3.2 プロセス数

並列に動作するプロセスの数は、以下の優先順位で決定されます。

1. mpiexecコマンドのオプションによる指定
2. ジョブスクリプトによる指定

例

ジョブスクリプト例

・ 4ノードで1コアに1プロセスで実行する場合

```
#!/bin/sh
#PJM -L node=4
#PJM --mpi proc=192
PATH=/製品インストールパス/bin:$PATH; export PATH
LD_LIBRARY_PATH=/製品インストールパス/lib64:$LD_LIBRARY_PATH; export LD_LIBRARY_PATH
mpiexec -n 192 ./a.out
```

“*製品インストールパス*”は、システム管理者にお問い合わせください。

1.2.3.3 実行時プロフィルファイル

通常のFortranプログラムと同様に、COARRAYプログラムでも実行時プロフィルファイル機能を利用することができます。

1.2.3.4 実行時コマンドの標準入出力および標準エラー出力

mpiexecコマンドの標準入力が自動的に各像の標準入力にはなりません。COARRAYプログラムが標準入力を期待する場合は、標準入力にしたい内容をファイルにしてmpiexecコマンドの--stdinオプションを指定する必要があります。同様に標準出力、標準エラー出力先を指定するオプションにより、これらの出力先を指定することができます。

ただし、READ文のUNIT指定子に“*”を指定した場合、像番号が1の像だけ実行することができます。他の像で実行すると、実行時エラーになります。

1.2.3.5 入出力処理時のファイル名

COARRAY機能を使用すると、一般的に多数のノードで同様な動作をするプログラムを走行することになります。このため、入出力を行う場合には、同名のファイルの実体が同じファイルになるのか別のファイルになるのかを意識しなければなりません。共用ファイルシステム上の同じ名前のファイルを参照した場合は同じファイルになり、ノードにローカルなファイルは同じ名前でもノードごとに別のファイルになります。

共用ファイルシステムにファイルを作成する場合には、ファイル名に像番号を付加すれば、像ごとに別々のファイルを作成することができます。このようなファイルの扱いはすべて利用者の責任で行う必要があります。

事前接続されたファイルのファイル名は、通常のFortranプログラムではfort.{unit番号}となりますが、COARRAY機能を使用する場合には、像ごとに別々のファイルとするためにfort.{unit番号}.{像番号}となります。

第2章 COARRAYによる並列化

この章では、COARRAYを用いた並列化について説明します。

2.1 COARRAYプログラムの構成

例題を用いてCOARRAYプログラムの構成を説明します。

例示するプログラムは、ヤコビ反復法などによく使われるステンシル演算を模擬したプログラムです。

```
 1  PROGRAM TOY_STENCIL_SOLVER_CAF_1D_SIMPLE
 2    USE ISO_FORTRAN_ENV
 3    IMPLICIT NONE
 4    INTEGER::ITER,I,J,IMAX,JMAX,JMAX_ALL,JACCMAX,JLOC,NIMG,ID
 5    REAL(8)::DIFF
 6    REAL(8),SAVE::RSD[*]
 7    REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::FLD_A
 8    REAL(8),DIMENSION(:,:),ALLOCATABLE::FLD_B
 9    INTEGER,PARAMETER::SIZE=1026,NN=10
10    REAL(8),PARAMETER::PARAM=0.1666666666
11
12    NIMG=NUM_IMAGES()
13    ID=THIS_IMAGE()
14
15    IF(NIMG/=16)THEN
16      IF(ID==1)THEN
17        PRINT *,' THE NUMBER OF IMAGES IS NOT 16. '
18      END IF
19      STOP
20    END IF
21
22    IMAX=SIZE
23    JMAX_ALL=SIZE
24    JACCMAX=(JMAX_ALL-2)/NIMG
25    JMAX=JACCMAX+2
26
27    ALLOCATE(FLD_A(IMAX,JMAX)[*],FLD_B(IMAX,JMAX))
28
20    JLOC=JACCMAX*(ID-1)
30    DO J=1,JMAX
31      FLD_A(:,J)=1.0+DBLE(MOD((JLOC + J),16))/DBLE(JMAX_ALL)
32    END DO
33    FLD_B=0.0
34
35    RSD=0.0
36
37    DO ITER=1,NN
38
39      DO J=2,JMAX-1
40        DO I=2,IMAX-1
41          FLD_B(I,J)=PARAM*(FLD_A(I-1,J)+FLD_A(I+1,J)+FLD_A(I,J-1)+FLD_A(I,J+1))
42          DIFF=FLD_B(I,J)-FLD_A(I,J)
43          RSD=RSD+DIFF*DIFF
44        END DO
45      END DO
46
47      FLD_A(2:IMAX-1,2:JMAX-1)=FLD_B(2:IMAX-1,2:JMAX-1)
48
49      IF(NIMG>1)THEN
50        SYNC ALL
51        IF(ID==1)THEN
52          FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
```

```
53        ELSE IF(ID==NIMG)THEN
54           FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
55        ELSE
56           FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
57           FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
58        END IF
50        SYNC ALL
60      END IF
61
62    END DO
63
64    SYNC ALL
65    CALL CO_SUM(RSD,RESULT_IMAGE=1)
66    IF(ID==1)THEN
67      PRINT *,' RESIDUAL :',RSD
68    END IF
69
70    DEALLOCATE(FLD_A,FLD_B)
71
72    STOP
73  END PROGRAM TOY_STENCIL_SOLVER_CAF_1D_SIMPLE
```

2.1.1 共配列の宣言

プログラム中のすべての変数は、各像(image)に実体があります。

共配列は、以下のように宣言します。これらは像間で定義・参照が可能です。

```
6    REAL(8),SAVE::RSD[*]
7    REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::FLD_A
```

共配列以外の変数は、それぞれの像にローカルな変数です。

```
4    INTEGER::ITER,I,J,IMAX,JMAX,JMAX_ALL,JACCMAX,JLOC,NIMG,ID
5    REAL(8)::DIFF
 :
8    REAL(8),DIMENSION(:,:),ALLOCATABLE::FLD_B
```

2.1.2 像番号(image index)

各々の像には像番号(image index)があります。また、全体の像の数を獲得することもできます。

全体の像の数は、プログラムの途中で変更することはできません。

例題の12行目のNUM_IMAGESは全体の像の数を獲得し、13行目のTHIS_IMAGEは処理中の像番号を獲得します。

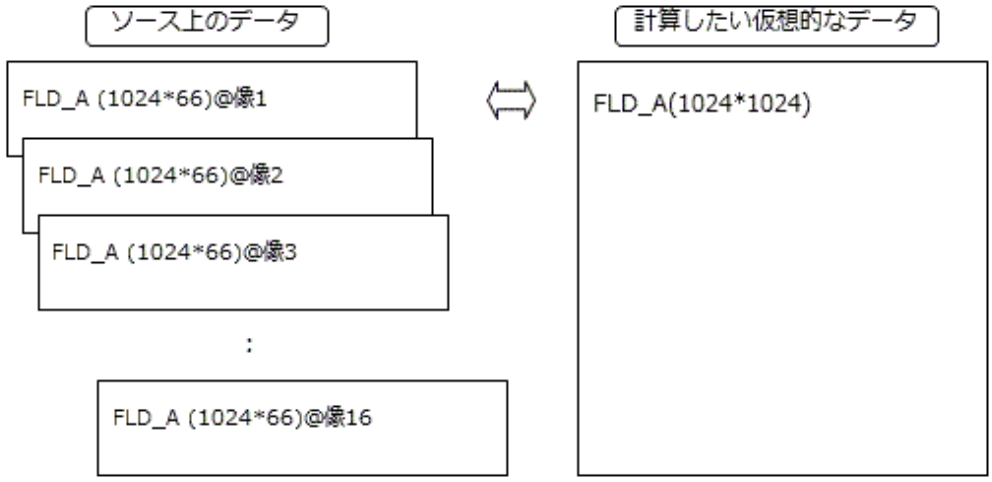
```
12    NIMG=NUM_IMAGES()
13    ID=THIS_IMAGE()
```

2.1.3 ローカル作業域の獲得とデータの作成

例題のデータ構造について説明します。プログラム全体で計算したい領域は1024*1024の配列です。これを16像で分割します。

この計算では配列の上下左右の隣接データが必要なので上下に1要素、合計2要素重ねることにします。

よって1つの像でのデータ域は1024*66の大きさとなります。



最初に分割したローカル作業域を各々の像で獲得します。

SIZEとIMAXは隣接を含めて1026が入っています。JACCMAXは1つの像が計算する1024/16=64が入っています。JMAXは隣接を含めて64+2=66が入ります。

1つの像で必要な領域を獲得します。1024*66の配列を獲得します。

```
22    IMAX=SIZE
23    JMAX_ALL=SIZE
24    JACCMAX=(JMAX_ALL-2)/NIMG
25    JMAX=JACCMAX+2
26
27    ALLOCATE(FLD_A(IMAX,JMAX)[*],FLD_B(IMAX,JMAX))
```

計算の入力となるデータを設定します。

```
29    JLOC=JACCMAX*(ID-1)
30    DO J=1,JMAX
31      FLD_A(:,J)=1.0+DBLE(MOD((JLOC+J),16))/DBLE(JMAX_ALL)
32    END DO
33    FLD_B=0.0
```

JLOCは像1では、(64-1)*(1-1)なので0から開始します。

像2では、(64-1)*(2-1)なので63から開始します。あとは63ずつ増えていきます。

上記データ配置で、重なりのある部分に同じデータが入るようにします。

ここで、共配列FLD_Aは像選択子(“[”)が付かないため、各像にローカルな変数へ代入します。すなわち、ここでは各像が必要なデータをそれぞれの像が設定します。

2.1.4 各像のローカルな計算

各像が担当する計算を行います。この例では1026*66の配列を入力として、1024*64の範囲の計算を行います。

FLD_Aを入力として1024*64の範囲の計算結果をFLD_Bに格納します。そののちにFLD_BをFLD_Aに書き戻します。

```
39      DO J=2,JMAX-1
40        DO I=2,IMAX-1
41          FLD_B(I,J)=PARAM*(FLD_A(I-1,J)+FLD_A(I+1,J)+FLD_A(I,J-1)+FLD_A(I,J+1))
42          DIFF=FLD_B(I,J)-FLD_A(I,J)
43          RSD=RSD+DIFF*DIFF
44        END DO
45      END DO
46
47      FLD_A(2:IMAX-1,2:JMAX-1)=FLD_B(2:IMAX-1,2:JMAX-1)
```

2.1.5 計算結果の隣接への反映

各々の像が計算結果を他の像へ反映します。

49行目は像の数が1の場合の考慮なので、本ケースでは常に真です。

50行目で各像のローカルな計算の完了を待ちます。COARRAYではSYNC ALL等の像制御文(image control statement)を記述しない限り共配列への代入は保証されません。

像1では、FLD_A(2:1025,66)に像2のFLD_A(2:1025,2)を代入します。

像16では、FLD_A(2:1025,1)に像15のFLD_A(2:1025,65)を代入します。

像2から15では各々のローカル配列へ、隣の像の端を代入します。

59行目では各像間の代入が完了するのを待ちます。

```
49      IF(NIMG>1)THEN
50        SYNC ALL
51        IF(ID==1)THEN
52           FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
53        ELSE IF(ID==NIMG)THEN
54           FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
55        ELSE
56           FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
57           FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
58        END IF
59        SYNC ALL
60      END IF
```

2.1.6 計算結果の集計

ここまでに各像は各担当部分の計算を行い、64行目で完了を待ちます。

CO_SUM組込みサブルーチンは各像での計算結果の総和を求めます。総和は像1に返されます。

像1が代表して結果を出力します。

```
64    SYNC ALL
65    CALL CO_SUM(RSD,RESULT_IMAGE=1)
66    IF(ID==1)THEN
67      PRINT *,' RESIDUAL :',RSD
68    END IF
```

2.2 COARRAYプログラムの開始と終了

COARRAYプログラムの開始と終了について説明します。

2.2.1 プログラムの開始

- COARRAYプログラムが起動されると、起動時に決定された数の像が非同期に起動されます。
- 各像は、それぞれのプロセスとして起動され、プロセスとして固有の環境をもちます。
- 像を識別する像番号は、1から「起動時に決定された像の数」までの整数値です。

2.2.2 プログラムの終了

- プログラムの実行の終了には、“正常終了”と“誤り終了”があります。
- ある像で誤り終了が開始されると、すべての像で誤り終了が開始されます。
- すべての像の実行が終了したことを、プログラムの実行が終了したといいます。
- STOP文やEND PROGRAM文が実行されると、その像で正常終了が開始されます。

- ERROR STOP文が実行されるか、誤り条件が発生すると、その像で誤り終了が開始されます。
- 次のいずれかの場合、その像で正常終了が開始されます。
  - 富士通拡張サービスサブルーチンSETRCDを実行した場合
  - 富士通拡張サービスサブルーチンEXITを実行した場合
  - C言語exit(3)関数を実行した場合(C/C++言語間結合時)
- 次のいずれかの場合、その像で誤り終了が開始されます。
  - 富士通拡張サービスサブルーチンABORTを実行した場合
  - C言語abort(3)関数を実行した場合(C/C++言語間結合時)
  - 実行時オプション-aを指定して実行が終了した場合

2.2.3 復帰コード

複数の像で異なった復帰コードを指定した場合は、mpiexecコマンドの振る舞いに依存します。

詳細は、“MPI使用手引書”を参照してください。

2.2.4 共配列への定義と参照

共配列への定義または参照で、以下のエラーが発生する場合があります。

返却値	意味
1711	共配列領域のデータ転送処理でエラーが発生しました。
1721	像番号は像の数以下の正の整数でなければなりません。
1790	転送先または転送元のいずれかが自像でない像間の転送は、サポートされていません。

2.3 像制御文と組込みサブルーチン

像制御文と組込みサブルーチンについて説明します。

注意

説明の中で実行時エラーメッセージを引用する場合、わかりやすさのためにメッセージ本文を和文で表記していますが、実際の実行時エラーメッセージは英文で出力されます。

2.3.1 SYNC ALL文

SYNC ALL文は、すべての像と同期をとります。

以下は、SYNC ALL文の使い方の例です。

像1は、データを読み込んで、他の像に配信します。

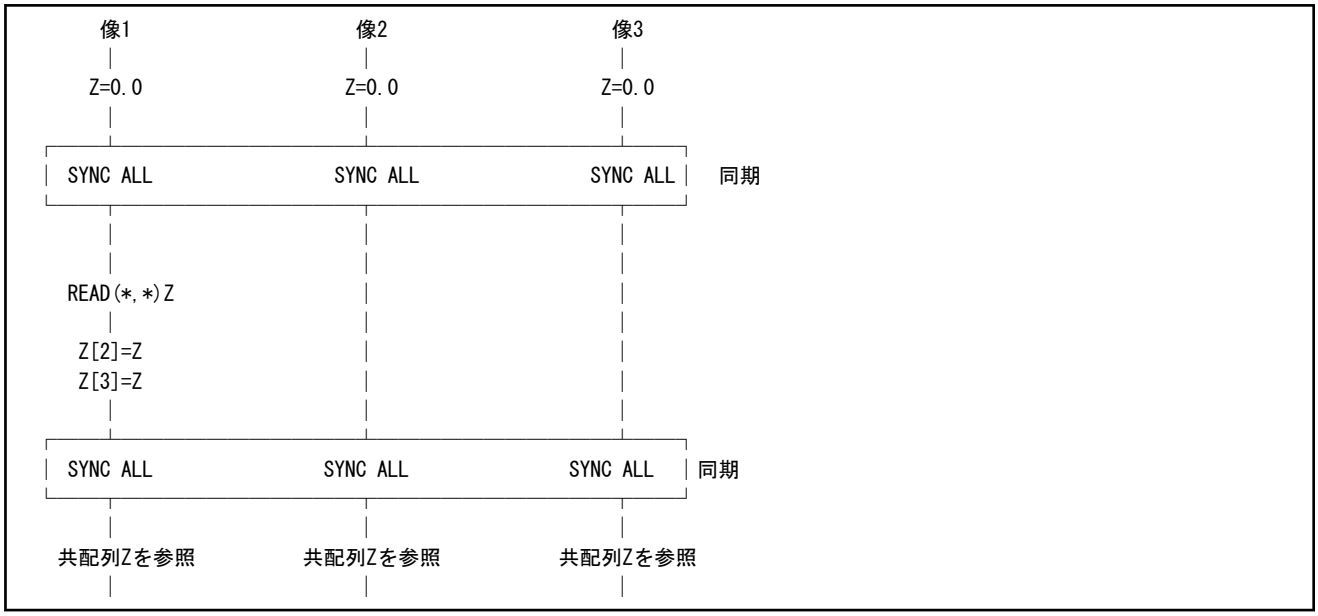
1番目のSYNC ALL文は、像1が共配列ZにREAD文で入力した後に、他の像によって共配列Zへの初期化がされないように制御します。

2番目のSYNC ALL文は、像1が共配列ZにREAD文で入力した新しい値を設定する前に、別の像が共配列Zを参照しないように制御します。

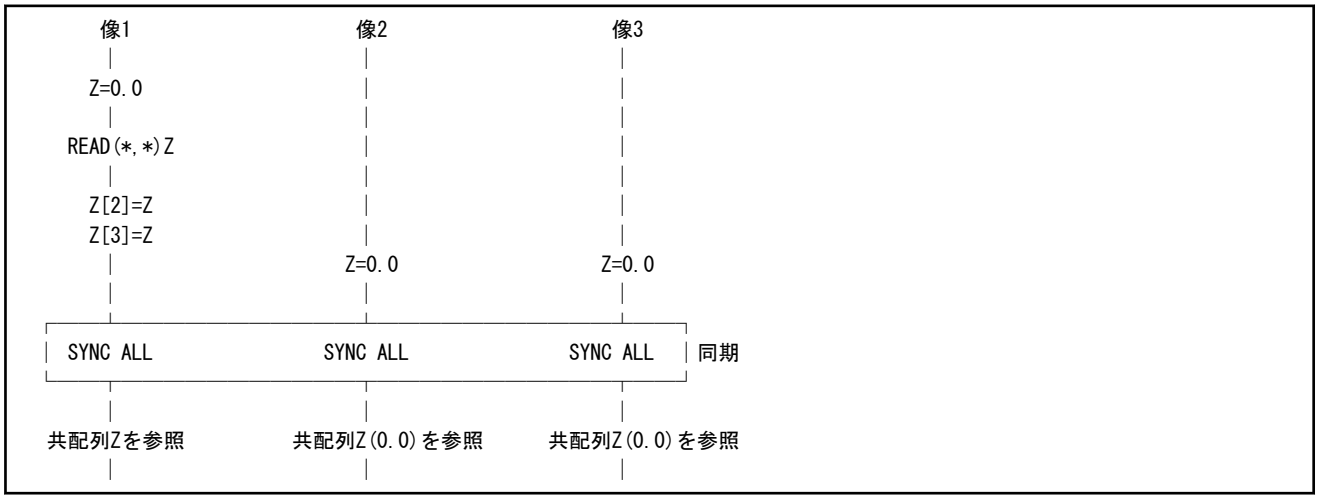
```
REAL(8),SAVE::Z[*]
 :
Z=0.0                    ! 各像で共配列Zを値0.0で初期化
SYNC ALL                 ! 1番目のSYNC ALL文
IF(THIS_IMAGE()==1)THEN
```

```
   READ(*,*)Z

   Z[2]=Z
   Z[3]=Z

END IF
SYNC ALL                ! 2番目のSYNC ALL文
 :
                        ! 共配列Zを参照
```

以下は、各像の動作イメージです。

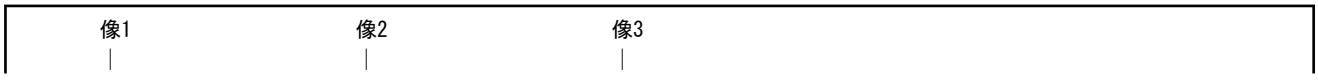


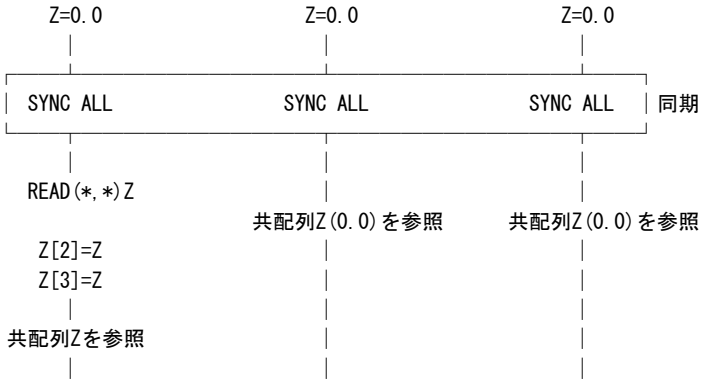
以下は、1番目のSYNC ALL文がなかった場合の各像の動作イメージです。



上図のような順序で実行される可能性があるため、像1がREAD文で新しい値をZに代入し、像2、像3のZに新しいデータを設定した後、像2および像3で変数Zに値0.0で初期化してしまう可能性があります。

以下は、2番目のSYNC ALL文がなかった場合の各像の動作イメージです。





像1が新しい値を他像(2,3)のZに設定する前に、像2および像3が共配列Zを参照してしまう例です。

SYNC ALL文のSTAT指定子

SYNC ALL文にSTAT指定子を指定して呼び出した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
1713	CRITICAL構文内で像制御文を実行することはできません。
1723	この像でデッドロックを検出しました。
1731	他の像で実行の終了が開始されたため、SYNC ALL文を終了しました。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_STOPPED_IMAGEです。)
1751	SYNC ALL文の同期処理でエラーが発生しました。
1752	SYNC ALL文の通信処理でエラーが発生しました。

2.3.2 SYNC IMAGES文

SYNC IMAGES文は、指定された像と同期をとります。

下例では、像1は他の像がデータを使用できるようになることを待たせています。

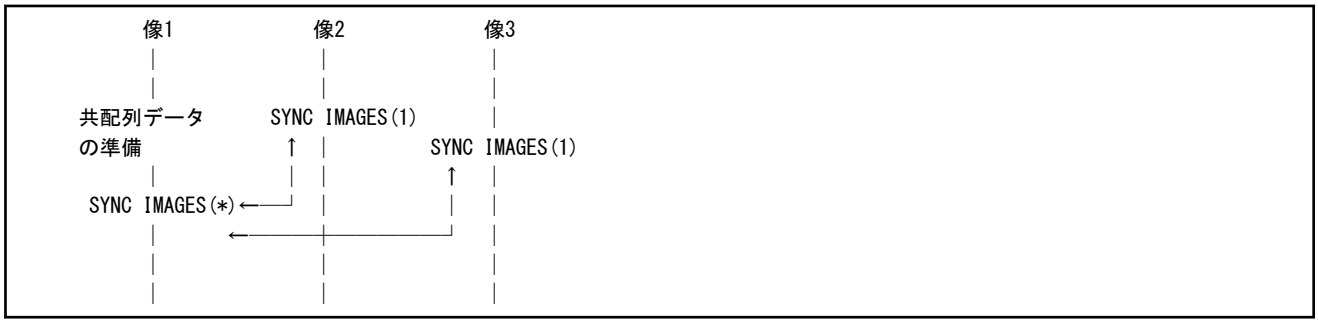
他の像は、像1がデータをセットアップするのを待っていますが、他のいかなる(像1以外の)像は待っていません。よって、像1以外のすべての像は、像1と同期後、実行を継続します。

像1は、像1以外のすべての像と同期が完了したときに実行を継続します。

```
REAL(8),SAVE::Z[*]
 :
IF(THIS_IMAGE()==1)THEN
   !
   ! 像2,3のデータを準備する
   !
   READ(*,*)Z

   Z[2]=Z
   Z[3]=Z
   SYNC IMAGES(*)   ！像1は像2,3と同期をとる。
ELSE
   SYNC IMAGES(1)   ！像2,3は像1と同期をとる。
END IF
   !
```

```
! 像1によって準備されたデータを使用する
!
```

以下は、各像の動作イメージです。



データを準備する像1とそれを使用する像2または像3の間で同期がとられればよいので、像2と像3の同期をとらなくても正常な結果を得ることができます。

SYNC IMAGES文のSTAT指定子

SYNC IMAGES文にSTAT指定子を指定して呼び出した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
1713	CRITICAL構文内で像制御文を実行することはできません。
1720	SYNC IMAGES文に同じ像番号を2回以上指定することはできません。
1722	SYNC IMAGES文に指定する像番号は像の数以下の正の整数でなければなりません。
1723	この像でデッドロックを検出しました。
1731	他の像で実行の終了が開始されたため、SYNC IMAGES文を終了しました。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_STOPPED_IMAGEです。)
1752	SYNC IMAGES文の通信処理でエラーが発生しました。

2.3.3 SYNC MEMORY文

SYNC MEMORY文は、セグメントを分割し、その像内で先行するセグメント内で実行されるメモリ操作を完了させます。

SYNC MEMORY文によって分割された異なる像にあるセグメントの実行順序を順序付けるためには、利用者がアトミックサブルーチンを利用して同期機構を構築するなどして実現しなければなりません。

SYNC MEMORY文のSTAT指定子

SYNC MEMORY文にSTAT指定子を指定して呼び出した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
1713	CRITICAL構文内で像制御文を実行することはできません。
1752	SYNC MEMORY文の通信処理でエラーが発生しました。

2.3.4 ALLOCATE文とDEALLOCATE文

ALLOCATE文およびDEALLOCATE文に単一または複数の共配列を指定した場合、すべての像で同じ共配列を、同じ大きさ、同じ順番で同じ個数だけ指定しなければなりません。

共配列を指定したALLOCATE文およびDEALLOCATE文を実行すると、すべての像が暗黙のうちに同期します。

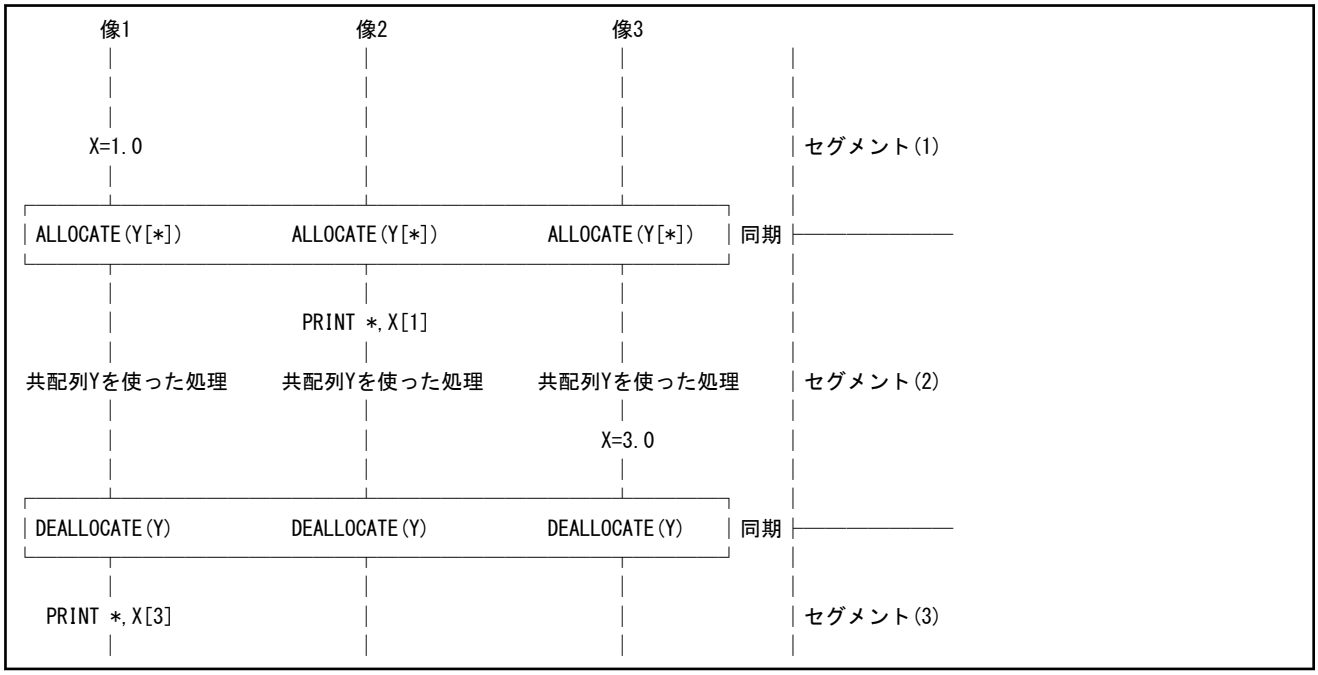
また、SYNC MEMORY文を実行するという効果を含んでいます。

ロックされているLOCK_TYPE型の共配列(ロック変数)をUNLOCK文でアンロックせずに、DEALLOCATE文や暗黙の割付け解除を行ってはなりません。必ず、UNLOCK文でアンロックしてから割付け解除してください。

下例は、ALLOCATE文およびDEALLOCATE文が暗黙のうちに同期し、SYNC MEMORY文を実行するという効果を示しています。

```
INTEGER::MY
REAL,SAVE::X[*]
REAL,ALLOCATABLE::Y[:]
 :                              |
MY=THIS_IMAGE()                 | セグメント(1)
IF(MY==1)X=1.0                  |
ALLOCATE(Y[*])                  |---------------
IF(MY==2)PRINT *,X[1]           |
   !                            |
   ! 共配列Yを使った処理        | セグメント(2)
   !                            |
IF(MY==3)X=3.0                  |
DEALLOCATE(Y)                   |---------------
IF(MY==1)PRINT *,X[3]           | セグメント(3)
 :                              |
```

以下は、各像の動作イメージです。



各像においてALLOCATE文とDEALLOCATE文によって3つのセグメントに分割され、セグメント(1)、セグメント(2)、セグメント(3)の順に順番付けられています。

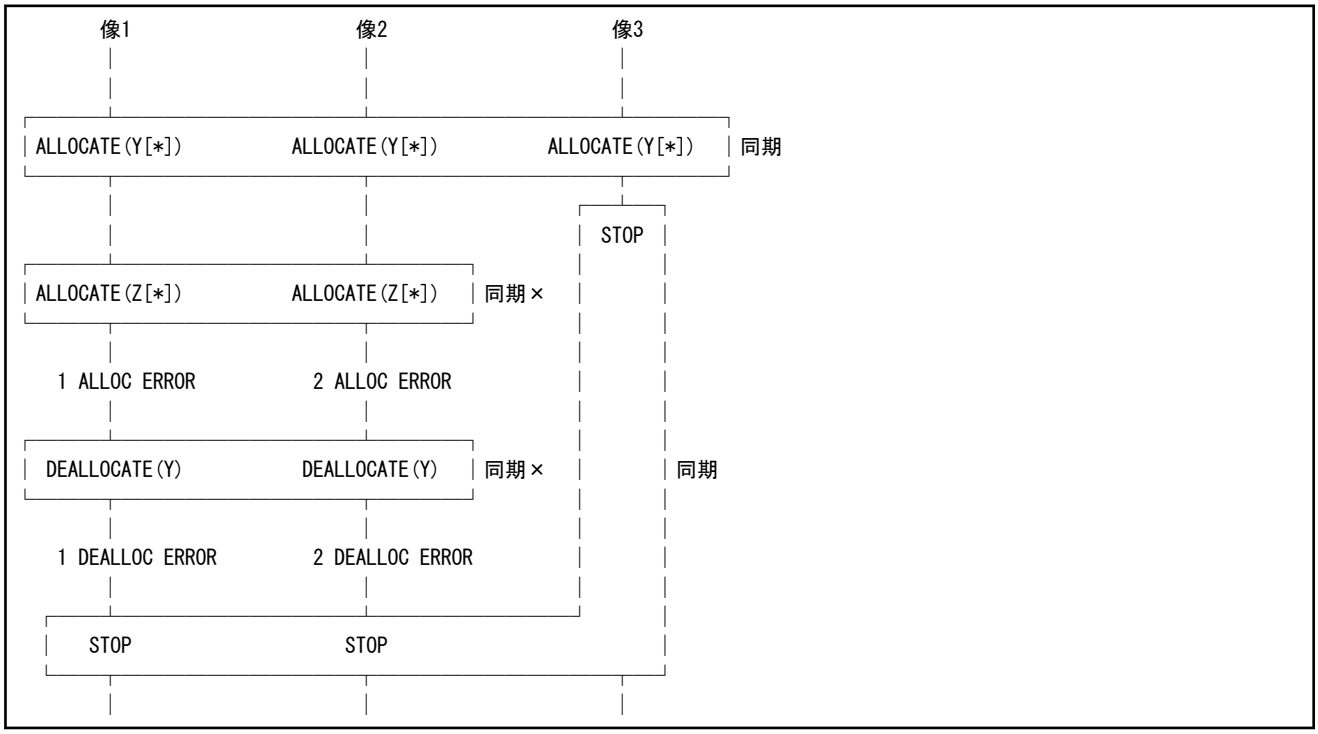
ALLOCATE文およびDEALLOCATE文は、それぞれ暗黙的にすべての像で同期されるため、すべての像の各セグメントはセグメント(1)、セグメント(2)、セグメント(3)の順に順番付けられます。

そのため、像2のセグメント(2)のPRINT文は「1.0」を表示し、像1のセグメント(3)のPRINT文は「3.0」を表示します。

下例では、ALLOCATE文またはDEALLOCATE文が実行され、誤って1つの像でSTOP文を実行してしまい、ALLOCATE文とDEALLOCATE文のSTAT指定子に、組込みモジュールISO_FORTRAN_ENVに定義されたSTAT_STOPPED_IMAGEが設定されます。

```
INTEGER::MY,ST
REAL,SAVE::X[*]=0.0
REAL,ALLOCATABLE::Y[:],Z[:]
 :
MY=THIS_IMAGE()
ALLOCATE(Y[*])
IF(MY==3)THEN
  STOP
END IF
ALLOCATE(Z[*],STAT=ST)
IF(ST.EQ.STAT_STOPPED_IMAGE)THEN
  PRINT *,MY,' ALLOC ERROR'
END IF
DEALLOCATE(Y,STAT=ST)
IF(ST.EQ.STAT_STOPPED_IMAGE)THEN
  PRINT *,MY,' DEALLOC ERROR'
END IF
 :
STOP
```

以下は、各像の動作イメージです。



最初のALLOCATE文は、すべての像で実行され正しく動作しますが、2番目のALLOCATE文は、像1と像2では実行されますが、像3ではSTOP文が実行されます。そのため、像1と像2のALLOCATE文のSTAT指定子にSTAT_STOPPED_IMAGEが設定されます。

DEALLOCATE文も同様に、像1と像2のDEALLOCATE文のSTAT指定子にSTAT_STOPPED_IMAGEが設定されます。

その後、像1と像2のSTOP文が実行され、既に実行していた像3のSTOP文と同期して、プログラムは終了します。

ALLOCATE文とDEALLOCATE文のSTAT指定子

共配列を指定したALLOCATE文とDEALLOCATE文は、STAT指定子を指定した場合、共配列でない領域の場合に加え、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味	
0	正常に実行されました。	
1713	CRITICAL構文内で像制御文を実行することはできません。	
1723	この像でデッドロックを検出しました。	
1731	他の像で実行の終了が開始されたため、ALLOCATE文またはDEALLOCATE文を終了しました。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_STOPPED_IMAGEです。)	
1705	ALLOCATE文	共配列領域の確保処理中に他の像で像制御文や異なったALLOCATE文を実行することはできません。
1706	ALLOCATE文	共配列領域の確保処理で同期エラーが発生しました。
1707	ALLOCATE文	共配列領域の確保処理でエラーが発生しました。
1741	DEALLOCATE文	共配列領域の解放処理中に他の像で像制御文や異なったDEALLOCATE文を実行することはできません。
1742	DEALLOCATE文	共配列領域の解放処理で同期エラーが発生しました。
1743	DEALLOCATE文	共配列領域の解放処理でエラーが発生しました。
1744	DEALLOCATE文	共配列領域の解放処理でエラーが発生しました。
1745	DEALLOCATE文	ロック変数をアンロックせずに解放することはできません。

2.3.5 LOCK文とUNLOCK文

LOCK文やUNLOCK文を使用して、排他制御を行うことができます。

ロック変数は、割当て時に、ロック変数ごとにロック資源を1つ割当てます。

利用可能ロック資源数は、プログラムの開始時に決定され、実行時に動的に変更することはできません。

あらかじめプログラムで利用するロック変数の数を把握し、環境変数FLIB_COARRAY_LOCKNOの省略値を超える場合に、指定可能な範囲内で環境変数FLIB_COARRAY_LOCKNOにロック資源数を指定しなければなりません。

ロック変数の割当て時に、ロック資源が不足した場合、以下の診断メッセージを出力して誤り終了を開始します。

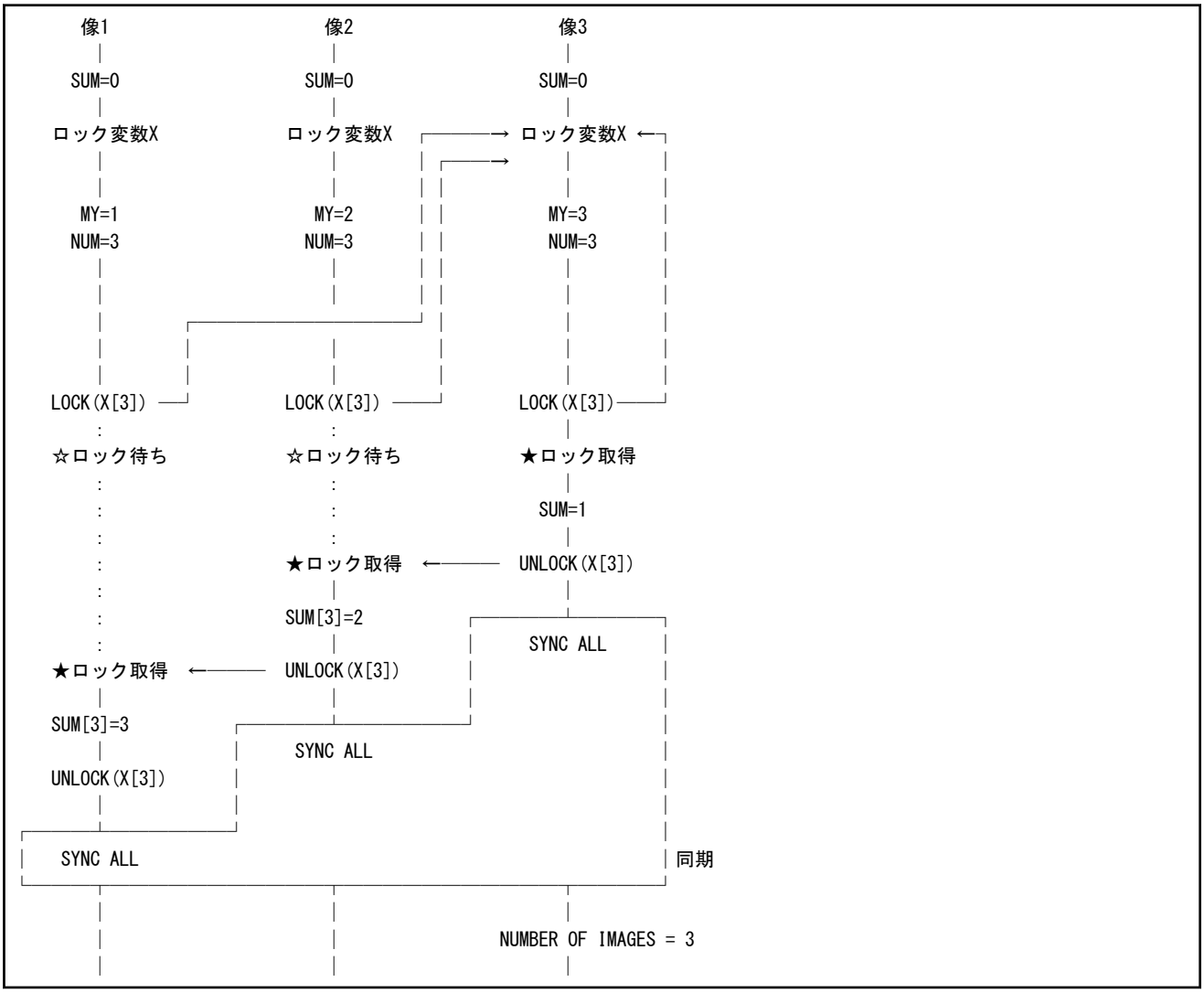
```
jwe1738i-s ロック資源が不足しました。
```

複数の像間で、1つのロック変数を使って、LOCK文やUNLOCK文によって排他制御する場合、ロック変数は共配列なので、排他したい像のすべてから、同じ像の同じロック変数を指定しなければなりません。各像のLOCK文やUNLOCK文で指定したロック変数が、同じ像のものでなくそれぞれが各像のものだった場合、それぞれ像間で排他制御はされません。

下例では、1つのロック変数で各像が排他制御されて動作します。

```
USE,INTRINSIC::ISO_FORTRAN_ENV
TYPE(LOCK_TYPE)::X[*]
INTEGER,SAVE::SUM[*]=0
MY=THIS_IMAGE()
NUM=NUM_IMAGES()

LOCK(X[NUM])
  SUM[NUM]=SUM[NUM]+1
UNLOCK(X[NUM])

SYNC ALL
IF(MY.EQ.NUM)THEN
  PRINT *,'NUMBER OF IMAGES = ',SUM
END IF
```

以下は、像の数を3とした時の各像の動作イメージです。



LOCK文やUNLOCK文で指定したロック変数は、最終像のロック変数X[NUM]を指定しているため、すべての像間で排他制御されて動作します。

3つの像が同時に1つのロックを要求しますが、どの像が最初にロックを取得できるかは不定です。

上記イメージでは、像3から像2、像1の順にロックが取得できたことを示しています。

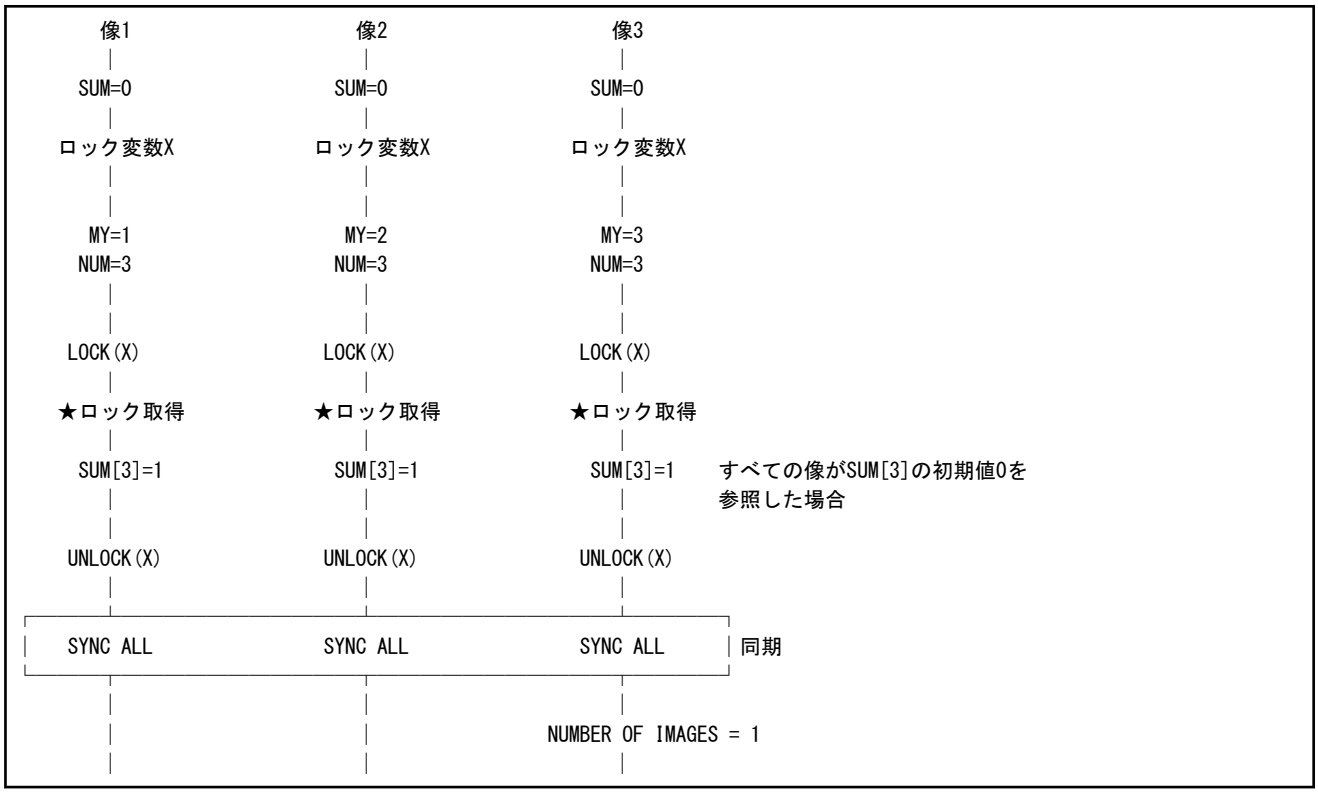
像3のPRINT文は「NUMBER OF IMAGES = 3」と表示します。

下例では、1つのロック変数で各像が排他制御されずに動作します。

```
USE,INTRINSIC::ISO_FORTRAN_ENV
TYPE(LOCK_TYPE)::X[*]
INTEGER,SAVE::SUM[*]=0
INTEGER::MY,NUM
MY=THIS_IMAGE()
NUM=NUM_IMAGES()

LOCK(X)
  SUM[NUM]=SUM[NUM]+1
UNLOCK(X)

SYNC ALL
```

```
IF(MY.EQ.NUM)THEN
  PRINT *,'NUMBER OF IMAGES = ',SUM
END IF
```

以下は、各像の動作イメージです。



LOCK文やUNLOCK文で指定したロック変数は、各像にローカルなロック変数Xを指定しているため、3つの像が同時にそれぞれのロックを取得できてしまいます。そのため、像間で排他制御が行われません。

期待される排他制御が行われないため、上記のように動作すると、像3のPRINT文は「NUMBER OF IMAGES = 1」と表示します。

LOCK文とUNLOCK文のSTAT指定子

LOCK文およびUNLOCK文にSTAT指定子を指定して呼び出した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味	
0	正常に実行されました。	
1713	CRITICAL構文内で像制御文を実行することはできません。	
1722	LOCK文またはUNLOCK文に指定する像番号は像の数以下の正の整数でなければなりません。	
1735	ロック変数でない変数をLOCK文またはUNLOCK文に指定することはできません。	
1732	LOCK文	ロック変数をロックした像によって再度ロックすることはできません。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_LOCKEDです。)
1736	LOCK文	ロック処理でエラーを検出しました。
1733	UNLOCK文	他の像がロックしたロック変数をアンロックすることはできません。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_LOCKED_OTHER_IMAGEです。)
1734	UNLOCK文	ロックされていないロック変数をアンロックすることはできません。

返却値	意味	
		(この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_UNLOCKEDです。)
1737	UNLOCK文	アンロック処理でエラーを検出しました。

2.3.6 CRITICAL構文

CRITICAL文とEND CRITICAL文によって囲まれたブロックは同時に2以上の像で実行されることはありません。

CRITICAL構文内で実行が禁止されている像制御文を実行しようとした場合、以下の診断メッセージを出力して誤り終了を開始します。

```
jwe1713i-s CRITICAL構文内で像制御文を実行することはできません。
```

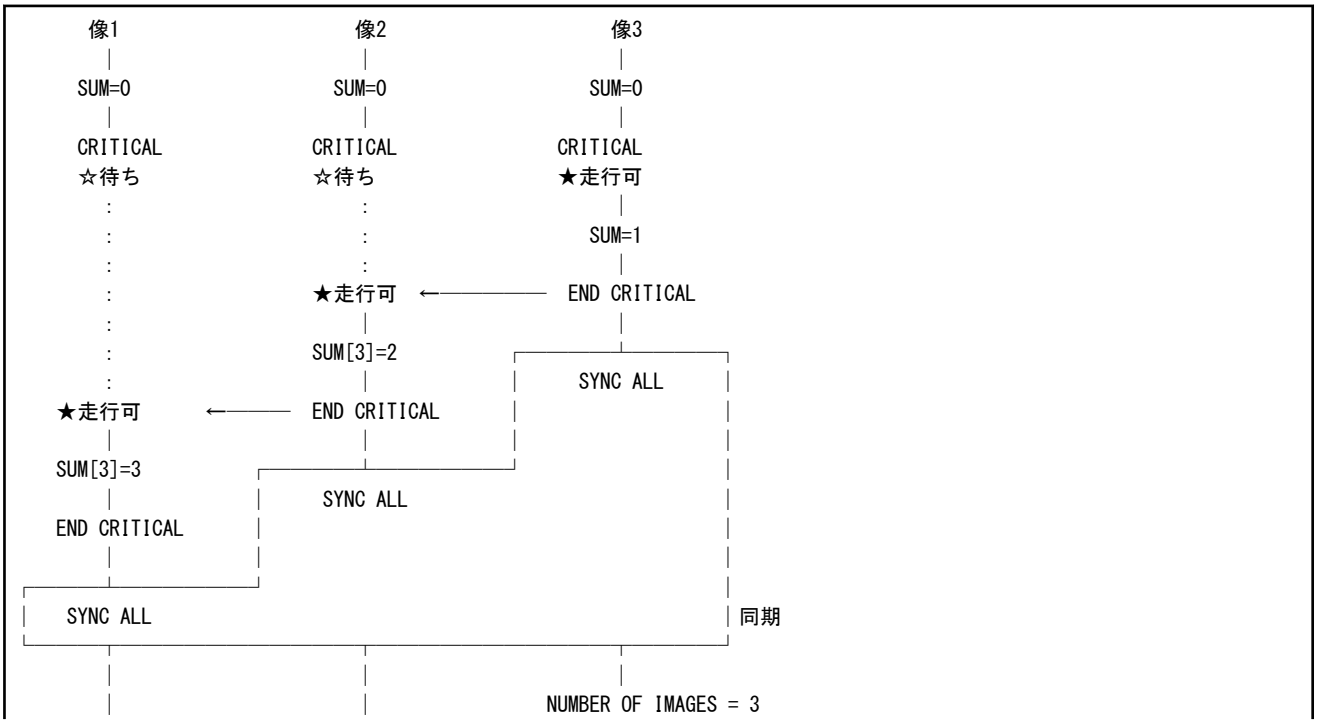
複数の像でCRITICAL構文が実行される場合、どの像からCRITICAL構文を実行されるかは順序不定ですが、最終的にCRITICAL構文を実行しようとしたすべての像で実行されます。

下例は、各像が像3の共変数に1を加算することによって動作している像の数を得るプログラムです。

ある像が加算を行っている間に他の像が加算を行うことを、CRITICAL構文を使用して防いでいます。

```
USE,INTRINSIC::ISO_FORTRAN_ENV
INTEGER,SAVE::SUM[*]=0
INTEGER::MY,NUM
MY=THIS_IMAGE()
NUM=NUM_IMAGES()

CRITICAL
  SUM[3]=SUM[3]+1
END CRITICAL

SYNC ALL
IF(MY.EQ.3)THEN
  PRINT *,'NUMBER OF IMAGES = ',SUM
END IF
```

以下は、各像の動作イメージです。



```
        |                       |                       |
        |                       |                       |
```

2.3.7 アトミックサブルーチン

アトミックサブルーチンは、変数をアトミックに定義または参照します。

アトミックサブルーチンのSTAT引数

アトミックサブルーチンにSTAT引数を指定して呼び出した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
1722	アトミックサブルーチンに指定する像番号は像の数以下の正の整数でなければなりません。

2.3.8 集団サブルーチン

以下の集団サブルーチンがあります。

- CO_MAX組込みサブルーチン
- CO_MIN組込みサブルーチン
- CO_SUM組込みサブルーチン

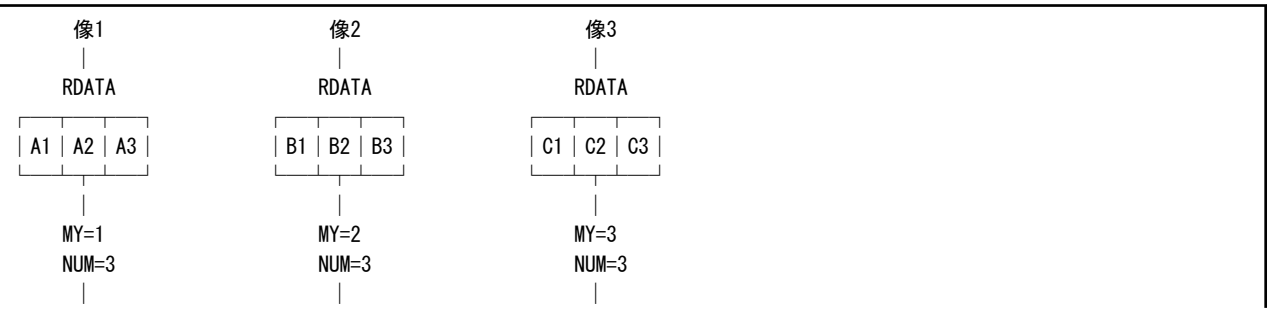
集団サブルーチンは、すべての像から同じ形式で呼び出さなければなりません。

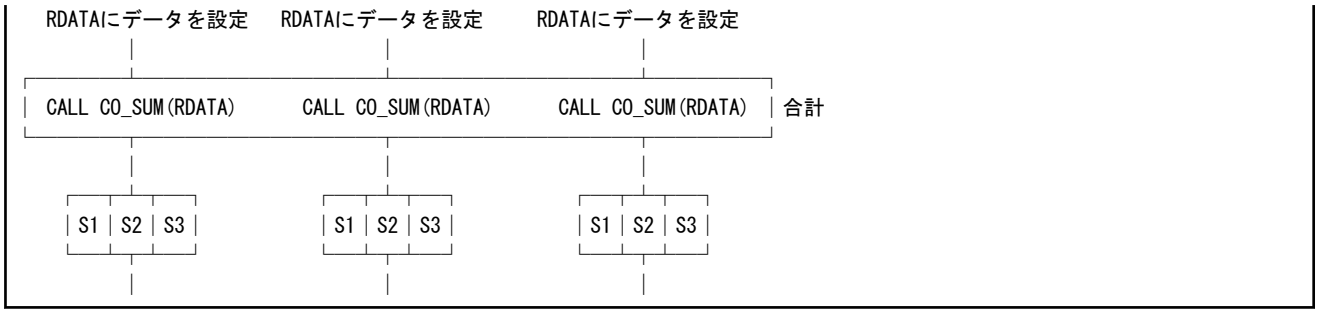
つまり、すべての像で、同じ名前、同じ型、同じ大きさをもつ共配列で、RESULT_IMAGE引数を指定している場合、同じ値でなければなりません。

下例では、CO_SUM組込みサブルーチンを使用して、共配列のすべての像の各要素の合計値が、すべての像に設定されます。

```
PROGRAM MAIN
  USE ISO_FORTRAN_ENV
  IMPLICIT NONE
  REAL(8),SAVE::RDATA(3)[*]
  INTEGER::MY,NUM,I,S

  MY=THIS_IMAGE()
  NUM=NUM_IMAGES()
        :       ! 共配列RDATAにデータを設定
  CALL CO_SUM(RDATA)
        :
END
```

以下は、各像の動作イメージです。



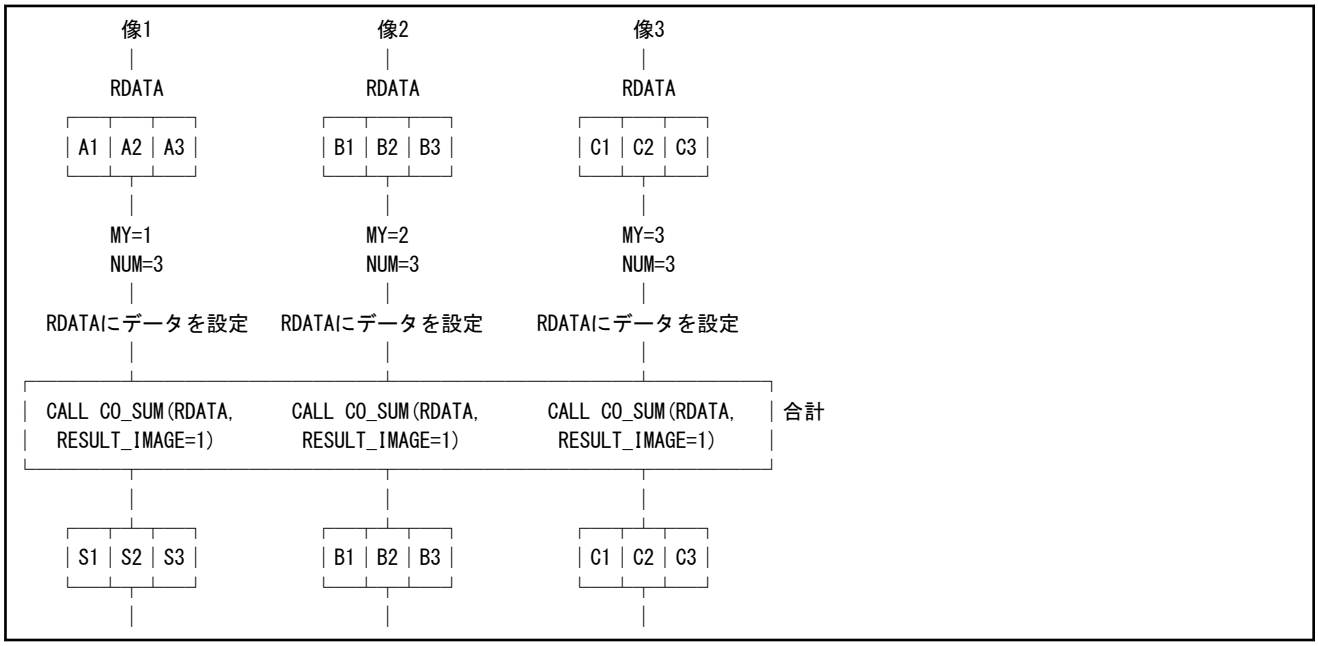


CO_SUM組込みサブルーチンの呼出し後、RDATAの各要素の合計は、すべての像のRDATAにも設定されます。

下例では、CO_SUM組込みサブルーチンを使用して、共配列の各要素の合計値が、RESULT_IMAGE引数に指定した像にだけ設定されます。

```
PROGRAM MAIN
  USE ISO_FORTRAN_ENV
  IMPLICIT NONE
  REAL(8),SAVE::RDATA(3)[*]
  INTEGER::MY,NUM,I,S

  MY=THIS_IMAGE()
  NUM=NUM_IMAGES()
        :       ! 共配列RDATAにデータを設定
  CALL CO_SUM(RDATA,RESULT_IMAGE=1)
        :
END
```

以下は、各像の動作イメージです。



CO_SUM組込みサブルーチンの呼出し後、RDATAの各要素の合計は、RESULT_IMAGE引数で指定した像1にだけ設定され、他の像2,3の共配列は変更されません。

集団サブルーチンのSTAT引数

集団サブルーチンにSTAT引数を指定して呼出した場合、以下の値が返却されます。

0以外の値は、実行時の診断メッセージの番号です。1バイトの整数型の変数を指定すると正しく値を受け取ることができないので注意が必要です。

返却値	意味
0	正常に実行されました。
1713	CRITICAL構文内で像制御文を実行することはできません。
1722	集団サブルーチンに指定する像番号は像の数以下の正の整数でなければなりません。
1723	この像でデッドロックを検出しました。
1731	他の像で実行の終了が開始されたため、集団サブルーチンを終了しました。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_STOPPED_IMAGEです。)
1751	集団サブルーチンの同期処理でエラーが発生しました。
1752	集団サブルーチンの通信処理でエラーが発生しました。

2.3.9 MOVE_ALLOC組込みサブルーチン

共配列を指定したMOVE_ALLOC組込みサブルーチンの呼出しは、すべての像で同じ文の呼び出しでなければなりません。また、FROM引数およびTO引数に指定した各共配列の状態もすべての像で同じ割付け状態でなければなりません。

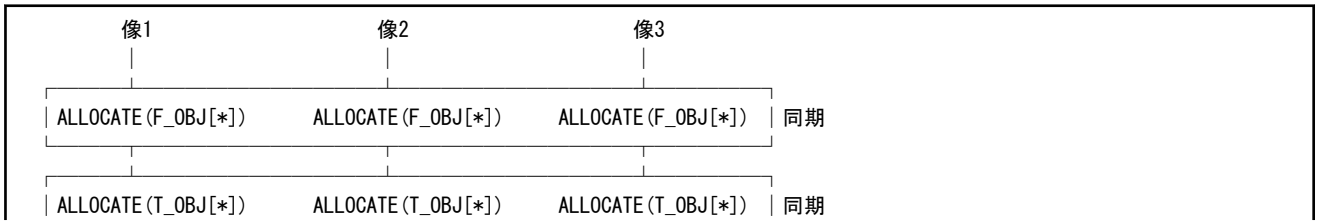
共配列を指定したMOVE_ALLOC組込みサブルーチンを呼び出すと、すべての像が暗黙のうちに同期します。

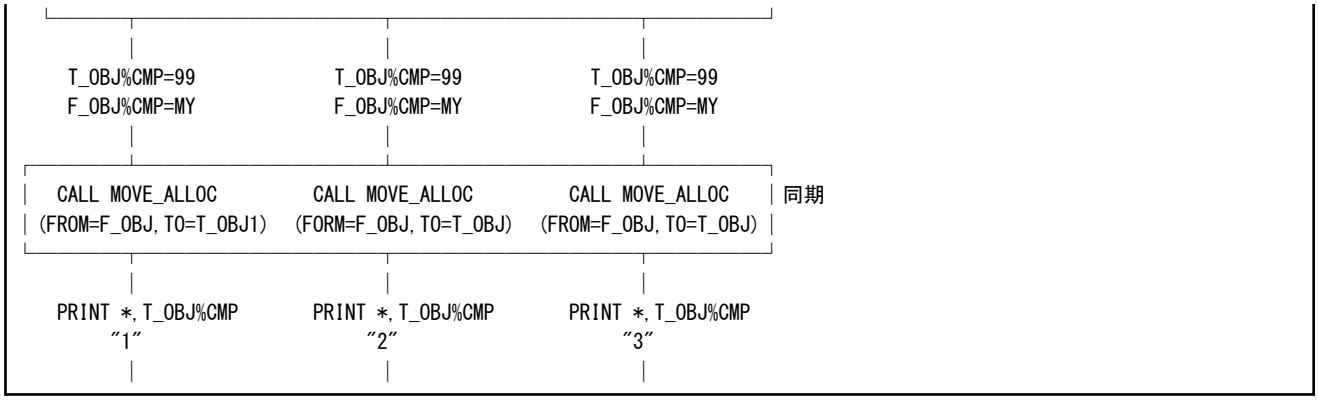
TO引数に指定した共配列がロックされているロック変数の場合、UNLOCK文でアンロックせずに、MOVE_ALLOC組込みサブルーチンを実行してはなりません。必ず、UNLOCK文でアンロックしてからMOVE_ALLOC組込みサブルーチンを実行してください。

下例では、MOVE_ALLOC組込みサブルーチンのFROM引数とTO引数へ、共に割付け実体がある共配列を指定して呼び出し、FROM引数からTO引数へ割付け実体を移動しています。

```
PROGRAM MAIN
  USE ISO_FORTRAN_ENV
  TYPE TDATA
    INTEGER::CMP
  END TYPE
  TYPE(TDATA),ALLOCATABLE::F_OBJ[:]
  TYPE(TDATA),ALLOCATABLE::T_OBJ[:]
  INTEGER::MY,NUM
  MY=THIS_IMAGE()
  NUM=NUM_IMAGES()

  ALLOCATE(F_OBJ[*])
  ALLOCATE(T_OBJ[*])

  T_OBJ%CMP=99
  F_OBJ%CMP=MY

  CALL MOVE_ALLOC(FROM=F_OBJ,TO=T_OBJ)

  PRINT *,T_OBJ%CMP
```

以下は、各像の動作イメージです。





各像は、MOVE_ALLOC組込みサブルーチンの呼び出し後、共配列F_OBJの割付け実体が共配列T_OBJに移動します。したがって、各像の「PRINT *,T_OBJ%CMP」は、それぞれ像番号を表示します。

注意

共配列を指定したMOVE_ALLOC組込みサブルーチンは、以下のエラーが発生する場合があります。

番号	意味
1306	組込みサブルーチンMOVE_ALLOCに指定する文字型の引数FROMとTOの長さは同じでなければなりません。
1713	CRITICAL構文内で像制御文を実行することはできません。
1723	この像でデッドロックを検出しました。
1731	他の像で実行の終了が開始されたため、MOVE_ALLOC組込みサブルーチンを終了しました。 (この返却値は組込みモジュールISO_FORTRAN_ENVのSTAT_STOPPED_IMAGEです。)
1743	共配列領域の解放処理でエラーが発生しました。
1744	共配列領域の解放処理でエラーが発生しました。
1745	ロック変数をアンロックせずに解放することはできません。
1751	MOVE_ALLOC組込みサブルーチンの同期処理でエラーが発生しました。
1752	MOVE_ALLOC組込みサブルーチンの通信処理でエラーが発生しました。

2.4 実行時の環境変数

COARRAYプログラムの実行時に指定可能な環境変数について説明します。

投入したジョブスクリプト内で設定した環境変数は、各プロセスに引き継がれます。また、mpiexecコマンドの-xオプションで環境変数を設定することもできます。

FLIB_COARRAY_DEADLOCK_TIMEOUT *time*

デッドロック検出までの時間を設定します。

*time*の単位は秒で、0≦*time*≦86,400です(86,400秒＝24時間)。*time*が0の場合、デッドロックを検出しません。

本環境変数が指定されていない場合、*time*に0(Fortranシステムの省略値)が設定されたとみなします。

例

環境変数FLIB_COARRAY_DEADLOCK_TIMEOUTの指定例

```
$ export FLIB_COARRAY_DEADLOCK_TIMEOUT=60
$ mpiexec ./a.out
```

この指定により、像間の同期で60秒間応答がない場合はデッドロックと判断します。

FLIB_COARRAY_LOCKNO *no*

プログラムで使用するロック変数の数(利用可能ロック資源数)を設定します。

*no*は、100≦*no*≦32,767です。

本環境変数が指定されていない場合、*no*に1024(Fortranシステムの省略値)が設定されたとみなします。

例

環境変数FLIB_COARRAY_LOCKNOの指定例

```
$ export FLIB_COARRAY_LOCKNO=3000
$ mpiexec ./a.out
```

FLIB_COARRAY_ERRMSG *out*

COARRAYプログラムの実行時にCOARRAYヘッダを追加出力するかどうかを指定します。

*out*は0または1です。*out*が0の場合、COARRAYヘッダを追加出力します。*out*が1の場合、COARRAYヘッダを出力しません。

環境変数が指定されていない場合、*out*に0(Fortranシステムの省略値)が設定されたとみなします。

例

環境変数FLIB_COARRAY_ERRMSGの指定例

```
$ export FLIB_COARRAY_ERRMSG=1
$ mpiexec ./a.out
```

この指定により、通常プログラムと同様にCOARRAYヘッダは出力されません。

2.4.1 実行コマンドの引数を設定する環境変数

COARRAYプログラムを実行する時の引数を環境変数で指定したい場合は、環境変数FORT90Lを使用します。ジョブスクリプト内で環境変数FORT90Lに設定してください。

第3章 COARRAYプログラムからのMPIの使用

本処理系のCOARRAYプログラムからMPIのサブルーチン・関数を使用することができます。

この章では、COARRAYプログラムでのMPIの使用について記述します。

3.1 リンクと実行

COARRAYプログラムでMPIを使用した場合の翻訳・リンク方法と 実行方法について説明します。

3.1.1 リンク方法

MPIを使用したCOARRAYプログラムを翻訳・リンクする場合は、mpifrt(またはmpifrtpx)に-Ncoarrayオプションを指定して翻訳・リンクしてください。

また、本処理系で翻訳時オプション-Ncoarrayを指定してfrt(またはfrtpx)によって作成したオブジェクトプログラムと、mpifrt(またはmpifrtpx)やmpifcc(またはmpifccpx)によって作成したオブジェクトプログラムとを、別々に作成した後にリンクする場合は、mpifrt(またはmpifrtpx)に-Ncoarrayオプションを指定してリンクしてください。

C++のオブジェクトプログラムを含む場合には、mpiFCC(またはmpiFCCpx)に--linkcoarrayオプションを指定してリンクしてください。

mpifrt、mpifrtpx、mpiFCC、およびmpiFCCpxの詳細は、“MPI使用手引書”を参照してください。

3.1.2 実行方法

mpiexecコマンドで実行してください。詳細は、“MPI使用手引書”を参照してください。

3.2 注意事項

COARRAYプログラムでMPIを使用した場合の注意事項を説明します。

- 本処理系のCOARRAYプログラムでは、MPI規格に定義されたコミュニケータを共有しないグループ間での通信の確立および動的プロセス生成の機能を使用することはできません。
- MPI規格では、MPI_COMM_SET_ATTR、MPI_TYPE_SET_ATTR、MPI_WIN_SET_ATTR、MPI_ATTR_PUTサブルーチンによって、定義済みのコミュニケータ、データ型、ウィンドウにも削除コールバックサブルーチンを設定できることになっています。本処理系のCOARRAYプログラムでは、定義済みコミュニケータ、データ型、ウィンドウに削除コールバックサブルーチンを設定することはできません。設定した場合は、本来とは異なるタイミングで削除コールバックサブルーチンが呼ばれることがあります。
- MPI規格では、MPI_IS_THREAD_MAINサブルーチンは、呼び出しスレッドがMPI_INITサブルーチンまたはMPI_INIT_THREADサブルーチンを呼んだスレッドと同じかどうかのフラグを返すことになっています。本処理系のCOARRAYプログラムでは、フラグとして正しい値を返さないことがあります。
- MPIプロファイリング・インターフェースでは、本処理系が呼び出したMPIのサブルーチン・関数もプロファイル機能の対象になります。
- MPI統計情報は、MPI_FINALIZEサブルーチンが呼ばれたタイミングで出力されることになっています。しかし、本処理系のCOARRAYプログラムでは、COARRAYプログラムが終了するときに出力されます。
- MPI統計情報では、本処理系が呼び出したMPIのサブルーチン・関数の情報も加算されて出力されます。

第4章 COARRAYプログラムのデバッグ

Fortranプログラムで利用可能なデバッグ機能を、COARRAYプログラムでも利用することができます。

デバッグ機能については、“Fortran使用手引書”の“プログラムのデバッグ”も併せて参照してください。

4.1 実行時のMPIのエラーメッセージ

本処理系のCOARRAY機能は、MPIライブラリを利用しています。そのため、MPIのエラーメッセージが出ることがあります。詳細は“6.3.1 実行時のエラーメッセージ”を参照してください。

4.2 デバッグ機能の出力

通常、デバッグ機能の出力先は標準エラー出力です。これを、mpiexecコマンドのオプションで、像ごとのファイルに変更することができます。

4.2.1 実行時の出力情報について

COARRAYプログラムの実行時にエラーが発生した場合、プログラムの診断メッセージ、トレースバックマップ、およびエラー修正情報の前に、それぞれ“COARRAYヘッダ”を追加出力します。

COARRAYヘッダを出力するかどうかは、環境変数FLIB_COARRAY_ERRMSGで指定することができます。詳細は、“2.4 実行時の環境変数”を参照してください。

COARRAYヘッダは、以下の形式です。

```
タイムスタンプ[像番号/像数:プロセスID]
```

タイムスタンプ	現在時刻です。 現在時刻をgettimeofday(2)で取得し、紀元(1970-01-01 00:00:00 (UTC))からの秒数を10桁の秒と6桁のマイクロ秒で表します。 診断メッセージを出力するときのその計算ノードでの現在時刻です。
像番号	診断メッセージを出力した像の像番号です。
像数	診断メッセージを出力したCOARRAYプログラムが実行した総像数です。
プロセスID	診断メッセージを出力した像が割当てられたプロセスIDです。

注意

複数の像で診断メッセージが出力される場合、出力される順番は、mpiexecコマンドに依存します。

また、タイムスタンプは像が実行した各計算ノードの現在時刻なので、各計算ノード間で時刻差がある場合は注意が必要です。

例

像数2で実行し、像1について抜粋した出力例

```
1429142357.004211[1/2:21294] jwe1744i-s line 25 An error was detected in the deallocation process for the COARRAY area.
1429142357.319930[1/2:21294]  error occurs at MAIN__   line 25 loc 0000000000400df5 offset 0000000000000125
1429142357.319948[1/2:21294]  MAIN__       at loc 0000000000400cd0 called from o.s.
1429142357.319964[1/2:21294] jwe0903i-u Error number 1744 was detected. Maximum error count exceeded.
1429142357.320249[1/2:21294] error summary (Fortran)
1429142357.320259[1/2:21294] error number  error level  error count
1429142357.320268[1/2:21294]   jwe1744i         s           1
1429142357.320272[1/2:21294] total error count = 1
```

4.2.2 COARRAY機能固有のエラー制御表の標準値

本処理系におけるCOARRAY機能固有のエラー識別番号に対するエラー項目の標準値を“表4.1 COARRAY機能固有のエラー制御表の標準値”に示します。

表4.1 COARRAY機能固有のエラー制御表の標準値

番号	エラー打切り回数	メッセージ最大出力回数	エラー項目の修正	バッファの内容出力	トレースバックマップの出力	標準修正	エラーレベル
1701～1707	1	1	可能	出力しません	出力します	あり	s
1711～1713	1	1	可能	出力しません	出力します	あり	s
1719～1722	1	1	可能	出力しません	出力します	あり	s
1723	1	1	不可能	出力しません	出力します	なし	u
1724	1	1	可能	出力しません	出力します	あり	s
1731～1739	1	1	可能	出力しません	出力します	あり	s
1741～1745	1	1	可能	出力しません	出力します	あり	s
1751～1752	1	1	可能	出力しません	出力します	あり	s
1790	1	1	可能	出力しません	出力します	あり	s

4.2.3 COARRAY固有機能に関するエラーの処理

COARRAY固有機能の実行中に発生するエラーに対するシステムの標準修正処理、および利用者が修正可能な処理を“表4.2 COARRAY固有機能に関するエラー処理”に示します。

利用者修正プログラムに渡される引数は次のとおりです。

- 復帰コード
- エラー識別番号

表4.2 COARRAY固有機能に関するエラー処理

番号	標準修正	利用者の修正処理	利用者修正プログラムに渡すデータ
1701～1707	プログラムを終了します	修正は不可能	なし
1711～1713	プログラムを終了します	修正は不可能	なし
1719～1722	プログラムを終了します	修正は不可能	なし
1724	プログラムを終了します	修正は不可能	なし
1731～1739	プログラムを終了します	修正は不可能	なし
1741～1745	プログラムを終了します	修正は不可能	なし
1751～1752	プログラムを終了します	修正は不可能	なし
1790	プログラムを終了します	修正は不可能	なし

4.3 デバッグ機能の注意事項

共添字付き変数は、翻訳時オプション-Hのデバッグ機能の対象になりません。

第5章 COARRAYプログラムのチューニング

この章では、COARRAYプログラムのチューニングについて説明します。

5.1 COARRAY機能の有効な利用

性能を考えたCOARRAY機能の有効な利用方法を説明します。

COARRAYプログラムでは、データ転送時間や同期を意識したプログラミングが重要です。この考慮をしないと、COARRAY機能を利用しない場合より、実行性能が大幅に遅くなることがあります。

5.1.1 共配列による他像のアクセス

共配列は各々の像に、ローカルなデータ実体(共添字付き実体)が存在します。像選択子(“[”)を利用せずにローカルなデータ実体を利用すれば、従来とほぼ同等の実行性能を得ることができます。

逆に、像選択子を利用して他の像のデータを利用する場合には、インターコネクトを経由するために多大なコストが伴います。

5.1.1.1 配列記述と他の像へのアクセス

他の像へのアクセスは、1要素ずつ引用するのではなく、配列記述を利用してください。

例

- 望ましくない例

```
REAL,SAVE::AA(1000)[*]
 :
DO I=1,1000
  AA(I)[N]=0
END DO
 :
```

- 望ましい例

  配列記述を利用することによって、インターコネクトでの転送コストを軽減することが期待できます。
  ただし、配列記述であっても小さなデータを隙間を空けて転送する場合には性能が期待できません。転送元で連続域にパッキングし転送先で展開するなどの工夫が必要です。

```
REAL,SAVE::AA(1000)[*]
 :
AA(:)[N]=0
 :
```

一般的に、異なる像へのアクセスは、配列記述であっても配列全体が一度に転送されるわけではなく、いくつかの大きさに分割されてインターコネクトを通じて転送されます。ランタイムシステムが適度に分割するため、転送の大きさを気にする必要はありません。

5.1.1.2 データのアクセスの集中

全ての像から同じ像に同時に書き込むような通信を避けてください。

1例を“5.2.5 全対全通信”に示します。

5.1.1.3 CO_SUM、CO_MAX、CO_MIN組込みサブルーチン

すべての像にあるデータの総和はCO_SUM組込みサブルーチンを利用してください。最大値はCO_MAX組込みサブルーチンを、最小値はCO_MIN組込みサブルーチンを利用してください。

すべての像にあるデータの総和を求める例を以下に示します。

例

- 望ましくない例

```
SYNC ALL
IF(ID==1)THEN
  DO I=2,NIMG
    VAL=VAL+VAL[I]
  END DO
END IF
SYNC ALL
```

- 望ましい例(CO_SUM組込みサブルーチンを利用)

```
SYNC ALL
CALL CO_SUM(VAL,RESULT_IMAGE=1)
```

注意

- 演算の順序は保証されません。
- 集団サブルーチンはすべての像で利用してください。
- 組込みモジュールISO_FORTRAN_ENVを参照する必要があります。

5.1.2 像制御文の使い方

像制御文は、SYNC ALL文ではなく、SYNC IMAGES文を使用することをお勧めします。

詳細は、“2.3.2 SYNC IMAGES文”を参照してください。

5.2 集団通信の記述例

MPIの集団通信に相当するCOARRAY機能の記述例を示します。

なお、システム向けにチューニングされた集団サブルーチンやMPIの集団通信を使用すると、本記述例よりも高速な実行が期待できます。

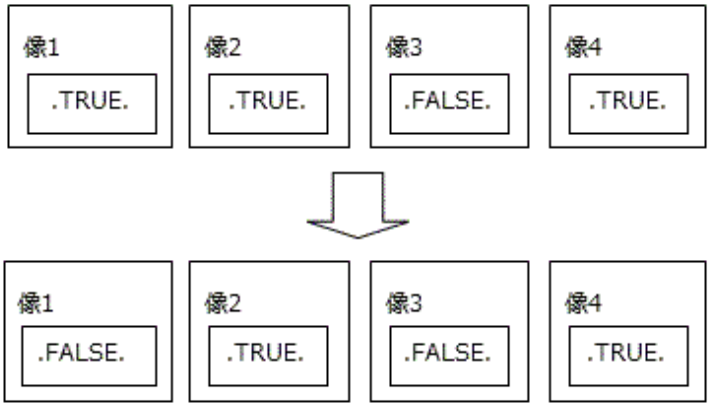
5.2.1 リダクション演算

全像の総和、最大値、最小値は集団サブルーチンCO_SUM、CO_MAX、CO_MINを利用してください。

リダクション演算のLOGICAL ANDの記述例を以下に示します。

例

```
LOGICAL,SAVE::VAL[*]
 :
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
 :
SYNC ALL
IF(ID==1)THEN
  DO I=2,NIMG
    VAL=VAL.AND.VAL[I]
  END DO
END IF
SYNC ALL
 :
```



配列を対象としたLOGICAL ANDの記述例を以下に示します。

配列の場合には、通信を一括化するために、共配列を各像にローカルな配列に転送するようにしています。

例

```
LOGICAL,DIMENSION(:),CODIMENSION[:],ALLOCATABLE::VAL
LOGICAL,DIMENSION(:),ALLOCATABLE::TMPVAL
 :
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
 :
ALLOCATE(VAL(MSGSIZE)[*])
ALLOCATE(TMPVAL(MSGSIZE))
 :
SYNC ALL
IF(ID==1)THEN
  DO I=2,NIMG
    TMPVAL(1:MSGSIZE)=VAL(1:MSGSIZE)[I]
    VAL(1:MSGSIZE)=VAL(1:MSGSIZE).AND.TMPVAL(1:MSGSIZE)
  END DO
END IF
SYNC ALL
 :
```

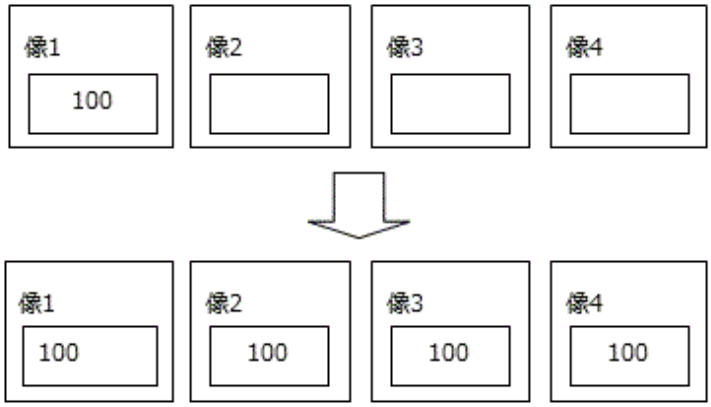
5.2.2 ブロードキャスト

ブロードキャストの記述例を以下に示します。

例

```
INTEGER :: NIMG,ID,I,MSGSIZE
REAL(8),DIMENSION(:),CODIMENSION[:],ALLOCATABLE :: ARRAY
 :
NIMG= NUM_IMAGES()
ID= THIS_IMAGE()
 :
ALLOCATE(ARRAY(MSGSIZE)[*])
 :
SYNC ALL
IF(ID /= 1) THEN
ARRAY(1:MSGSIZE) = ARRAY(1:MSGSIZE)[1]
```

```
END IF
SYNC ALL
 :
```

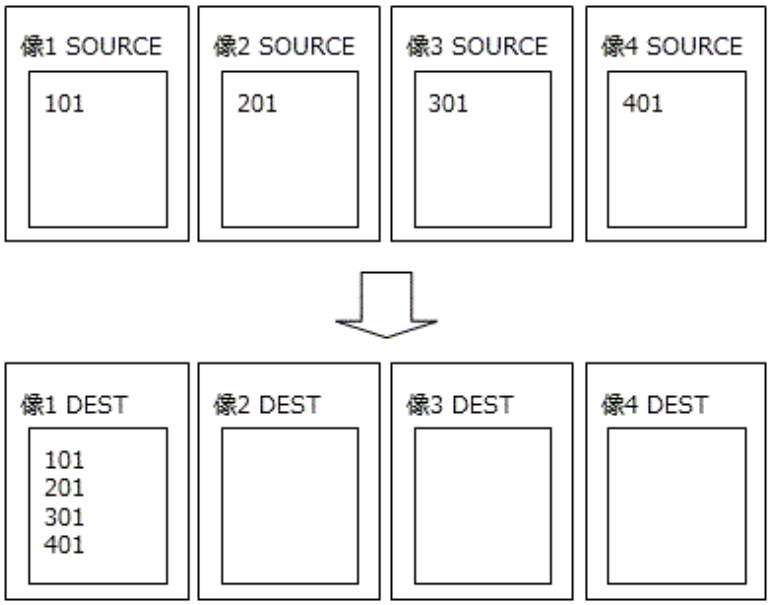


5.2.3 ギャザー

ギャザーの記述例を以下に示します。

例

```
INTEGER::NIMG,ID,MSGSIZE
REAL(8),DIMENSION(:),ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::SOURCE
 :
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
 :
ALLOCATE(DEST(MSGSIZE))
ALLOCATE(SOURCE(MSGSIZE,NIMG)[*])
 :
SYNC ALL
DEST(1:MSGSIZE,ID)[1] = SOURCE(1:MSGSIZE)
SYNC ALL
 :
```

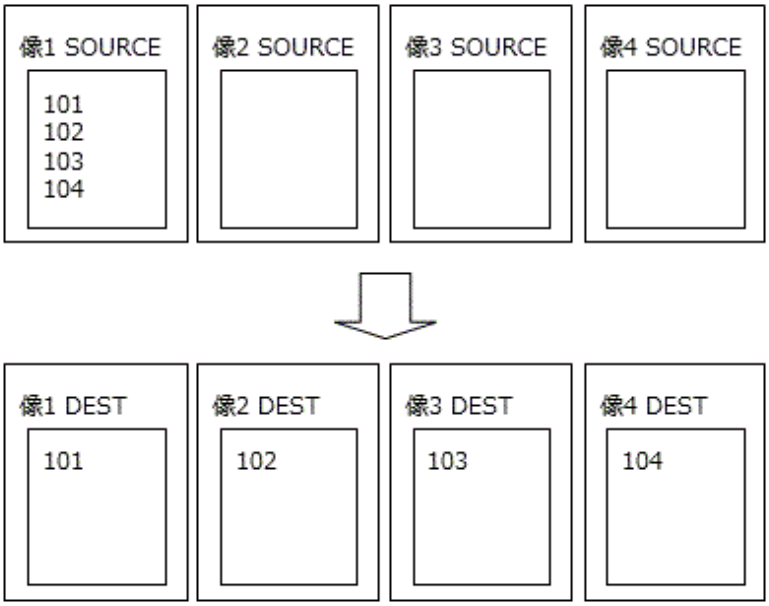


5.2.4 スキャタ

スキャタの記述例を以下に示します。

例

```
INTEGER::NIMG,ID,MSGSIZE
REAL(8),DIMENSION(:),ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::SOURCE
 :
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
 :
ALLOCATE(DEST(MSGSIZE))
ALLOCATE(SOURCE(MSGSIZE,NIMG)[*])
 :
SYNC ALL
DEST(1:MSGSIZE)=SOURCE(1:MSGSIZE,ID)[1]
SYNC ALL
 :
```

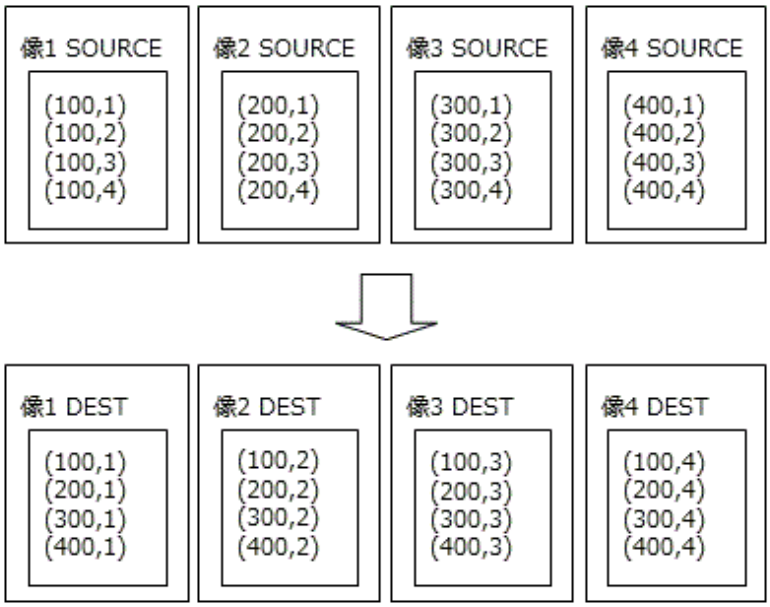


5.2.5 全対全通信

全対全通信の記述例を以下に示します。

例

```
INTEGER::NIMG,ID,I,MSGSIZE
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),ALLOCATABLE::SOURCE
 :
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
 :
ALLOCATE(DEST(MSGSIZE,NIMG)[*])
ALLOCATE(SOURCE(MSGSIZE,NIMG))
 :
SYNC ALL
DO I=1,NIMG
  DEST(1:MSGSIZE,ID)[I]=SOURCE(1:MSGSIZE,I)
END DO
SYNC ALL
 :
```



通信相手をずらして通信の衝突を避けると、上例より高速に実行することが期待できます。以下に例を示します。

例

```
INTEGER::NIMG,ID,I,ISHIFT,MSGSIZE
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),ALLOCATABLE::SOURCE
 :
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
 :
ALLOCATE(DEST(MSGSIZE,NIMG)[*])
ALLOCATE(SOURCE(MSGSIZE,NIMG))
 :
SYNC ALL
ISHIFT=ID
DO I=1,NIMG
  DEST(1:MSGSIZE,ID)[ISHIFT]=SOURCE(1:MSGSIZE,ISHIFT)
  ISHIFT=ISHIFT+1
  IF(ISHIFT>NIMG)THEN
     ISHIFT=1
  END IF
END DO
SYNC ALL
 :
```

5.3 並列入出力

ファイルの入出力方法について説明します。

複数の像が像ごとのローカルファイルを作成して並列に実行する方法の例

像番号をファイル名に含めることで像ごと別々のファイルを作成します。

```
  INTEGER,PARAMETER::NMAX=1000
  INTEGER::ID,NIMG
```

```
  CHARACTER(100)::FNAME_W,FNAME_R
  REAL(4),SAVE::A(NMAX)[*],B(NMAX)[*]
   :
  NIMG=NUM_IMAGES()
  ID=THIS_IMAGE()
   :
  WRITE(FNAME_W,'("A", I4.4, ".DAT")')ID
  OPEN(10,FILE=FNAME_W,FORM='UNFORMATTED')
  WRITE(10)A
  CLOSE(10)
   :
  WRITE(FNAME_R,'("A", I4.4, ".DAT")')ID
  OPEN(20,FILE=FNAME_R,FORM='UNFORMATTED')
  READ(20)B
  CLOSE(20)
   :
```

複数の像が1つの共用ファイルを並列にアクセスする方法の例

各像は対応する記録番号に書き込みます。

CLOSE(30,STATUS='FSYNC')文の実行によって、メモリ上に存在するファイルの内容を外部記憶装置上のファイルと同期させます。これにより、SYNC ALL文実行後にその共用ファイルの同期されたデータをアクセスすることができます。

```
  INTEGER,PARAMETER::NMAX=1000
  INTEGER::ID,NIMG
  REAL(4),SAVE::A(NMAX)[*]
  REAL(4),DIMENSION(:,:),ALLOCATABLE::D
   :
  NIMG=NUM_IMAGES()
  ID=THIS_IMAGE()
   :
  SYNC ALL
  OPEN(30,FILE='A.DAT',ACCESS='DIRECT',&
       FORM='UNFORMATTED',RECL=NMAX*4)
        :
    WRITE(30,REC=ID)A
     :
! FLUSH AND SYNC
  CLOSE(30,STATUS='FSYNC')
   :
  SYNC ALL
  IF(ID==1)THEN
    ALLOCATE(D(NMAX,NIMG))
    OPEN(40,FILE='A.DAT',FORM='BINARY')
    READ(40)D
    CLOSE(40)
     :
  END IF
   :
```

第6章 注意事項

この章では、本処理系のCOARRAYを使用する場合の注意事項を説明します。

6.1 COARRAYプログラムの注意事項

COARRAYプログラムの仕様に関する注意事項を説明します。

6.1.1 LOCK文のACQUIRED_LOCK指定子

LOCK文のACQUIRED_LOCK指定子は利用できません。指定した場合、実行時に診断メッセージ(jwe1739i-s)を出力して誤り終了を開始します。

6.1.2 共配列間の代入文

転送先または転送元のいずれかが自像でない像間の転送は利用できません。実行した場合、診断メッセージ(jwe1790i-s)を出力して誤り終了を開始します。

6.1.3 手続名と共通ブロック名の予約

以下の名前から始まる手続名や共通ブロック名は予約されています。

- fjmpi_
- mpi_
- ompi_
- pmpi_

翻訳時オプション-AUが指定されていないとき、原始プログラム中の英大文字と英小文字は区別されません。

翻訳時オプション-AUが指定されているとき、手続名と共通ブロック名は原始プログラムで指定したつづりになります。

外部手続名の加工方法については、“Fortran使用手引書”を参照してください。

BIND文を使用したCプログラムとの相互利用または手続言語束縛指定子を利用した場合、以下の名前から始まる束縛ラベルは予約されています。

- ARMCII_
- ARMCIX_
- ARMCI_
- FJMPI_
- MPI_
- OMPI_
- PARMCI_
- PFJMPI_
- PMPI_
- armci_
- fjmpi_
- mca_

- mpi_
- ompi_
- opal_
- orte_
- parmci_
- pmpi_

6.1.4 OpenMP仕様の使用上の注意

本処理系の-Kopenmpオプションは、-Ncoarrayオプションと同時に指定可能です。

COARRAY機能はマスタースレッドで実行してください。COARRAY機能をマスタースレッド以外で実行した場合、STAT指定子やSTAT引数の指定の有無に関わらず、以下の診断メッセージを出力して誤り終了を開始します。

```
jwe1704i-s COARRAY機能を使用する文はマスタースレッドでしか実行できません。
```

6.1.4.1 共配列

- 共配列は、THREADPRIVATE指示文に指定することはできません。
- 共添字付き変数は、OpenMP指示節に指定できません。
- 以下の指示節に指定した共配列は、その有効範囲でその共配列に共添字を指定することはできません。
  - PRIVATE
  - FIRSTPRIVATE
  - LASTPRIVATE
  - REDUCTION
  - DEPEND
  - LINEAR
  - ALIGNED
- 以下の指示節に指定した共配列は、その有効範囲で共配列として使用できません。
  - PRIVATE
  - FIRSTPRIVATE
  - LASTPRIVATE
  - REDUCTION
  - DEPEND
  - LINEAR
  - ALIGNED
- ALLOCATABLE属性をもつ共配列は、次の指示節に指定することはできません。
  - PRIVATE
  - FIRSTPRIVATE
  - LASTPRIVATE
  - COPYPRIVATE
  - REDUCTION
  - DEPEND

- LINEAR
- ALIGNED

6.1.5 プロセス生成する場合の注意

COARRAY機能を利用するプログラムにおいて、サービスルーチン(FORKなど)、言語間結合によるシステムコール(forkなど)、言語間結合によるライブラリ関数(systemなど)、などを使用してプロセス生成を行う場合、以下の制約があります。

- プロセス生成のタイミングで共配列が存在している場合、その共配列を含むページは子プロセスに引き継がれないことがあります。そのため、子プロセスはその共配列を含むページにアクセス(定義・参照)できないことがあります。

この制約のため、例えば、FORKサービス関数によって生成された子プロセスが、共配列への定義または参照を行うと、子プロセスにおいてuレベル(異常)のエラーが発生することがあります。

アクセスの可否はOSの管理するページの単位で判定されるため、あるメモリ領域が使用中の場合にその周辺のアドレスのメモリ領域がアクセスできないこともありますので、注意が必要です。

6.1.6 定数領域を他像へ転送する場合の注意

定数領域を他の像へ転送しようとした場合、定数領域の大きさが32バイトを超えると以下のインターコネクトのエラーが発生します。

```
[mpi::common-tofu::tofu-stag-error] Failed to query/register Tofu STag. [詳細情報]
```

このエラーは、以下のどちらかで回避できます。

- -Nnouse_rodataオプションを指定して翻訳する。
- 定数領域を書き込み可能な領域に代入し、その書き込み可能な領域を他の像へ転送する。

6.1.7 フック機能の使用上の注意

フック機能のユーザ定義サブルーチン内で、直接的および間接的に以下を実行することはできません。実行した場合の動作は保証されません。

- COARRAY機能
- STOP文、ERROR STOP文
- 富士通拡張サービスサブルーチンEXIT、富士通拡張サービスサブルーチンSETRCD
- C言語exit(3)関数

6.1.8 実行時情報出力機能の使用上の注意

-Ncoarrayオプションを指定して翻訳・リンクした実行可能プログラムが誤り終了した場合、実行時情報は出力されません。

6.2 インライン展開の制約事項

共配列を含む手続は、インライン展開されません。その手続を含む親のプログラム単位に共配列を含む場合も、インライン展開されません。

6.3 実行時の注意事項

COARRAYプログラムの実行時の注意事項について説明します。

6.3.1 実行時のエラーメッセージ

本処理系のCOARRAY機能はMPIライブラリを利用しているため、MPIのエラーメッセージが出ることがあります。

MPIのエラーメッセージは、「[mpi::」から始まります。MPIのエラーメッセージの詳細は、“MPI使用手引書”を参照してください。また、メッセージ中のランクは、像に読み替えてください。像番号はランク番号に1を加えたものです(“1.1.2 像とランク”参照)。