Fortran 90/VPP の紹介

渡部 善隆 *

本稿では，新スーパーコンピュータ VPP700/56 のベクトル並列 Fortran コンパイラである Fortran 90/VPP の機能を紹介します．内容は『VPP700/56 利用の手引 (第 1.0 版)』第 10 章の抜粋となっています．

具体的なプログラムの実行方法やオプション機能などは利用の手引を参照願います[1]．

なお， Fortran 90/VPP による並列プログラミングのためには，文法書である [1] をお読み下さい．

1　並列化のイメージ

1.1　XOCL 行

Fortran 90/VPP では，ソースプログラムに「拡張最適化制御行」と呼ばれる並列化に関する指示行を埋め込み，翻訳時オプション -Wx を付加し翻訳，編集結合することにより並列化を行います．

```
      SUBROUTINE SETDAT(A,B)
      INTEGER LA,N,NPE,NMAX,NBPE
      PARAMETER (LA=18000,N=LA,NPE=32)
      PARAMETER (NMAX=((LA-1)/NPE+1)*NPE,NBPE=100*NPE,LD=NMAX)
      DOUBLE PRECISION A(LD,NMAX),B(LD,NMAX),PI
      INTEGER I,J
!XOCL PROCESSOR P_MAIN(NPE)                                  ! 拡張最適化制御行 1
!XOCL SUBPROCESSOR P(NPE)=P_MAIN(1:NPE)                      ! 拡張最適化制御行 2
!XOCL INDEX PARTITION IP= (PROC=P,INDEX=1:NMAX,PART=BAND)    ! 拡張最適化制御行 3
!XOCL LOCAL   A(:,/IP),B(:,/IP)                              ! 拡張最適化制御行 4
      PI=4.0D0*ATAN(1.0D0)
!XOCL SPREAD DO /IP                                          ! 拡張最適化制御行 5
      DO 10 J=1,N
      DO 20 I=1,N
        A(I,J)=SQRT(2.0D0/(N+1))*SIN(I*J*PI/(N+1))
        B(I,J)=A(I,J)
   20 CONTINUE
   10 CONTINUE
!XOCL END SPREAD                                             ! 拡張最適化制御行 6
      RETURN
      END
```

拡張最適化制御行は **XOCL** 行ともいいます[2]． XOCL 行の記述方法は，第 1 桁から第 5 桁までが《必ず》“!XOCL” で始まります．固定形式では *XOCL も許します．また，小文字で !xocl と書いても構いません．以下はすべて !XOCL で統一します．

*九州大学大型計算機センター・研究開発部　E-mail : watanabe@cc.kyushu-u.ac.jp

[1]『VPP700/56 利用の手引 (第 1.0 版)』の入手を希望される方は，連絡所経由で共同利用掛までお申し込みください．また，最新のバージョンは kyu-cc の /usr/local/doc/VPP700guide.ps で PostScript ファイルとして公開しています．

[2]“eXtended Optimization Control Line” の略です． Fortran 90/VPP 独自の用語です．また，ベクトル化に関する指示行である OCL 行もあります ([2])．

```
!234567890                                  ! これは単なるコメント
!XOCL PROCESSOR P_MAIN(NPE)                 ! XOCL 行
```

!XOCL に続く第 6 桁目は空白です．

第 7 桁以降に並列化に関する構文を記述します[3]．!XOCL の継続は，継続する行の最後に & を記述します．

```
!XOCL INDEX PARTITION IP= (PROC=P,INDEX=1:NMAX, &     ! 次の行へ継続
!XOCL                     PART=BAND)
```

翻訳時オプション -Wx を指定しないと，!XOCL はコメント行と見なされます[4]．従って，単一 PE 版の Fortran プログラム (Fortran 90/VP) や他のワークステーションのコンパイラとの互換性は保たれます[5]．具体的な拡張最適化制御行の機能は次節で紹介します．

1.2 変数の配置

並列処理でもっとも重要なのが，配列に代表される変数を各 PE にどのように割り振るかです．

Fortran 90/VPP における変数の形態は，グローバル変数とローカル変数とに分類されます．

●グローバル変数

VPP700/56 は各 PE がそれぞれ独立したメモリとプロセッサを持つベクトル計算機だと見ることができます．

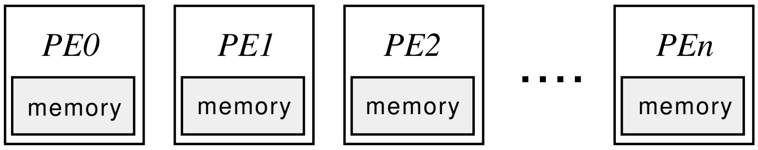


図 1：VPP700/56 のイメージ

Fortran 90/VPP では，これら物理的に分散したメモリをあたかも一つのメモリ空間であるかのように見せる機能があります．論理的にメモリが 1 つのように見える空間をグローバル空間[6]と呼び，グローバル空間に配置された変数をグローバル変数 (*global variable*) と呼びます．

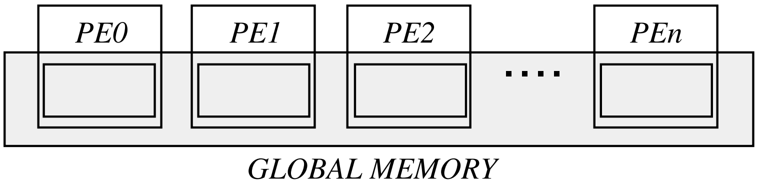


図 2：グローバル変数によるメモリの共有 I

図 2 はすべての PE のメモリを一つのグローバル空間として見た極端な例で，実際の計算では各 PE のメモリの一部をグローバル変数として宣言します．

[3]並列化 Fortran は，FORTRAN 77 を強く意識した仕様となっており，Fortran 90 の「自由形式プログラミング」の精神から逸脱しています．しかし，現在のところ，プログラムの見やすさを考えると通常のプログラムも第 7 桁から書き始めた方がいいでしょう．

[4]Fortran 90 では，“!” 以降は注釈と見なされ無視されます．なお，この機能は FORTRAN77 EX/VP でも使用できます．

[5]とはいっても，あまり「見栄え」がいいプログラムとは言えなくなります．

[6]あくまでもソフトウェアの力によって実現された空間ですので，「仮想グローバル空間」ともいいます．

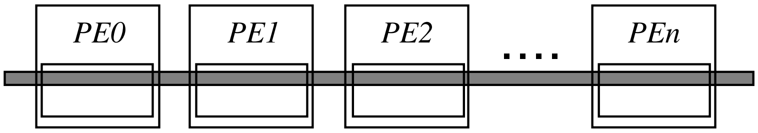


図 3 : グローバル変数によるメモリの共有 II

グローバル変数は，すべての PE から共通に参照することができます．つまり， VPP700/56 は物理的には「分散メモリ型計算機」でありながら，論理的にはグローバル変数によって「共有メモリ型計算機」であるかのように見ることもできます[7]．ただし，

グローバル変数はデータアクセスに時間がかかり，配列演算もベクトル化できません．

PE 間のデータ通信はネットワークを経由して行います．異なる PE(例えば PE1 と PE2) のデータ転送は PE 内の転送 (例えば PE1 の中だけ) に比べて大量の時間を要します．また DO ループなどのベクトル化もできませんので，ベクトル計算機としての性能を発揮することができません．つまり，実際問題としてグローバル配列を用いた直接の演算は使いものになりません．

グローバル変数は，通常分割ローカル変数と `EQUIVALENCE` 文により記憶域を共有し，異なる PE 間のデータ交換を仲介するために利用されます (cf.2.9節)．

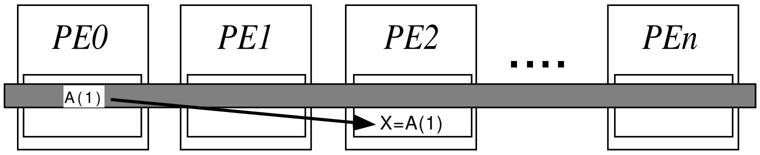


図 4 : グローバル変数によるデータの参照

図 4 は PE0 の `A(1)` をグローバル変数として宣言することで，異なる PE から参照する例です．

•ローカル変数

各 PE のメモリに配置された変数をローカル変数 (*local variable*) と呼びます．ローカル配列[8]は，他の PE から参照できないかわりに， Fortran 90/VP の自動ベクトル化機能がすべて使えます．異なる PE からローカル変数を参照するためには，グローバル空間を介して行います．

ローカル変数は，データの分割方法によって，さらに重複ローカル変数と分割ローカル配列に分かれます．

○重複ローカル変数

`!XOCL` で何も指定しない変数は すべての PE に重複して 割り当てられます．このローカル変数を重複ローカル変数 (*duplicate local variable*) と呼びます．例えば 8PE を用いたプログラム中，`DOUBLE PRECISION A(100)` で宣言された配列 `A` は，各 PE で重複して

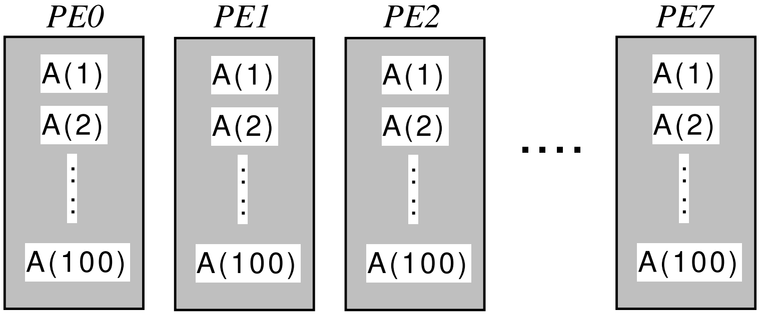


図 5 : 重複ローカル変数

[7]マニュアルにはこの構造を「階層メモリ型」と名付けています．

[8]配列宣言されたローカル変数の集まりのことです．

と配置されます．また，プログラムでこれまた `!XOCL` の指定がない場合は，各 PE でまったく同じ演算が実行されます．

○分割ローカル配列

1 つの PE のメモリでは収まりきらない大きな配列を使用するプログラムでは，配列を複数の PE に分割して配置する必要があります．この配列を分割ローカル配列 (*partitioned local array*) と呼びます．

例えば，`DOUBLE PRECISION A(1000)` で宣言した `A` を 8 つの PE で均等に分割すると，配列のイメージは

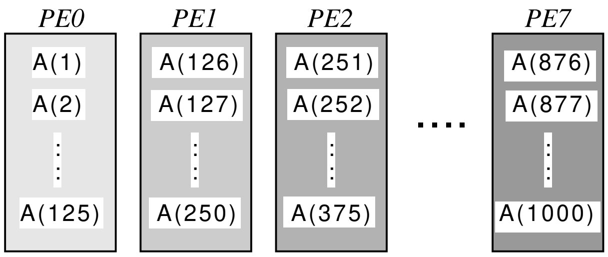


図 6：分割ローカル配列

となります[9]．

1.3 実行方式

VPP700/56 の実行方式には，逐次，冗長，並列の 3 つの実行方式があります．

逐次実行	1 台の PE を使ってプログラムを実行します
冗長実行	すべての PE で同一処理を行う方式です
並列実行	複数の PE が処理を分担して並列に実行することです

マルチプロセッサの効果を引き出すためには，並列実行の部分の割合を高める必要があります．たとえば，1PE では A, B, C, D の処理を順番に実行する必要があり，もっともコストのかかる部分は C だとします．この C の部分を手続きの分割 (C1 ~ Cn) によって複数の PE で並列に実行することができれば，C の時間の短縮が全体の処理時間の短縮につながります．

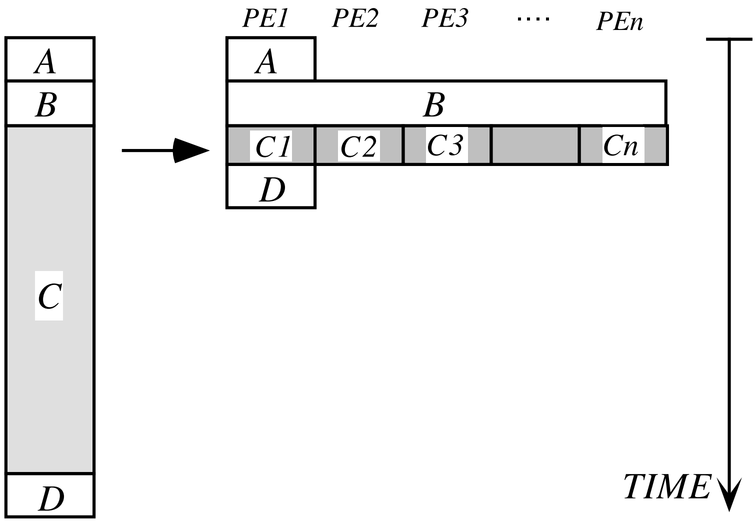


図 7：並列処理による実行時間の短縮イメージ

[9] 例では配列が PE 数できれいに割り切れていますが，割り切れなくても後ろのプロセッサで自動的に調整されますので，特に意識する必要はありません．

図 7 の A, D は 1 台の PE で実行されている逐次実行です．B はすべての PE で実行される冗長実行です．C の部分は各 PE に処理が分担されて処理されている並列実行です．

冗長実行は並列処理の意味では「無駄」な部分といえます．しかし，実際のプログラムではどうしても出てくるようです[10]．

そのため，センターでは並列に実行される各 PE の CPU 時間の最長のものに対して課金する方式を採用しています．

[10]『冗長』は「繁雑でわずらわしく長いこと」ですので，もちろん，いい意味ではありません．「冗長な会議」「冗長な話」「冗長な文章」で無駄な労力を使うのは世の常です．

2 拡張最適化制御行の機能

この節では，拡張最適化制御行 (XOCL 行) の機能を紹介します．XOCL 行は宣言構文 (declaration construct) と実行構文 (executable construct) に分類できます．

2.1 宣言構文一覧

宣言構文は，INTEGER I,J,K や REAL A(10,10) などの宣言文のある場所に記述します．

表 1: 宣言構文一覧

制御文	機能
PROCESSOR SUBPROCESSOR	使用する PE の数と名前の宣言 サブルーチン内で使用する PE の数と名前の宣言
INDEX PARTITION	分割指定の定義と名前の宣言
LOCAL GLOBAL	分割指定をしたローカル配列の宣言 グローバル配列の宣言
PROC ALIAS	プロセッサグループの別名を宣言

2.2 実行構文一覧

実行構文は DO 文や IF 文などのある場所に記述します．

表 2: 実行構文一覧

制御文	機能
PARALLEL REGION END PARALLEL REGION	並列実行の開始 並列実行の終了
SPREAD REGION REGION END SPREAD REGION	文の並びを分割し，分割部分を該当プロセッサで実行することを指定 SPREAD REGION の補助として区切りを示す SPREAD REGION の終了
SPREAD DO END SPREAD DO	DO ループを分割し，並列実行する SPREAD DO の終了
SPREAD MOVE END SPREAD MOVE	ローカル変数とグローバル変数の代入を一括転送する SPREAD MOVE の終了
SPREAD ASSIGNMENT END SPREAD ASSIGNMENT	代入文の実行を，PE が分担して実行する SPREAD ASSIGNMENT の終了
BROADCAST	指定された PE のデータを転送する
OVERLAPFIX	袖付き分割ローカル配列の袖部分にデータを転送する
UNIFY	各 PE で定義された重複ローカル配列の値を揃える
MOVEWAIT	転送の完了を待つことを指示
BARRIER	PE 間でバリア同期を取ることを指示
LOCKON END LOCKON	リージョン間での排他制御の区間の開始 リージョン間での排他制御の区間の終了

2.3 PROCESSOR 文

PROCESSOR 文は，プログラムで使用する PE の数と名前を宣言します．何はともあれ，この指定がないとプログラムが始まりません．

```
!XOCL PROCESSOR P(16)          ! プロセッサの定義
```

例では，16 個の PE を使用することを宣言しています．PROCESSOR 文によって定義された “P” をプロセッサグループと呼びます．プロセッサグループの名前は任意です．“PRO” であっても “MAIN_P” であっても構いません．しかし，プログラム内の変数名とプロセッサグループ名を 重複させることはできません．この記事では，一番安直な名前として “P” とします．

PE の数は，パラメータを用いて

```
      PARAMETER (NP=8)
!XOCL PROCESSOR P(NP)
```

と書くこともできます．Fortran 90 なら

```
      INTEGER,PARAMETER :: NP=8
!XOCL PROCESSOR P(NP)
```

となります[11]．プロセッサグループを 2 次元 (P(3,3)) や 3 次元 (P(2,3,4)) で定義することもできます[12]が，これは上級者用のテクニックです．

2.4　PROC ALIAS 文

既存のプロセッサグループを分割して別の名前のプロセッサグループとして定義することもできます．別名は PROC ALIAS 文で宣言します．

```
!XOCL PROCESSOR P(32)                          ! プロセッサの定義
!XOCL PROC ALIAS P1(8)=P(1:8),P2(24)=P(9:32)  ! P を二つに分け，別名をつける
```

例では，プロセッサグループ P を二つに分割して，8 つの PE から成る P1 と，24 の PE から成る P2 に分割しています．PROC ALIAS 文は異なるプロセッサグループを駆使するため，初心者はまず使いません．

2.5　SUBPROCESSOR 文

サブルーチンに XOCL 行が出てくる場合は，メインプログラムで宣言したプロセッサグループの情報を引き継ぐ必要があります．

まず，メインプログラムで，プロセッサグループが次のように宣言してあるとします．

```
      PARAMETER (NP=32)
!XOCL PROCESSOR P(NP)
```

サブルーチンでのプロセッサグループの引き継ぎは SUBPROCESSOR 文で行います．

```
      SUBROUTINE SETDAT(A,B)
      PARAMETER (NPE=32)
!XOCL PROCESSOR P(NPE)                          ! プロセッサグループの宣言
!XOCL SUBPROCESSOR PS(NPE)=P(1:NPE)             ! プロセッサグループの引き継ぎ
```

SUBPROCESSOR 文によって，メインプログラムのプロセッサグループがサブルーチン内のプロセッサグループ PS に引き継がれました．“1:NPE” は，1 から NPE までの PE を意味します．

また，メインプログラムとサブルーチンとは変数が独立ですので，メインプログラムと同じプロセッサグループ名 (ここでは P) を使いたければ，

[11] Fortran 90 のパラメータ属性，および暗黙の型宣言追放の観点からは，“INTEGER,PARAMETER::NP=8” と書くべきですが，現状 Fortran 90 がさほど浸透していないことからこの後は FORTRAN 77 の記法を踏襲します．

[12] 上限は 7 次元です．

```
      SUBROUTINE SETDAT(A,B)
      PARAMETER (NPE=32)
!XOCL PROCESSOR P_MAIN(NPE)                       ! プロセッサグループの宣言
!XOCL SUBPROCESSOR P(NPE)=P_MAIN(1:NPE)           ! プロセッサグループの引き継ぎ
```

と記述しても差し支えありません．大事なことは，メインプログラムで宣言したプロセッサグループの情報がサブルーチンに引き継がれることです[13]．

2.6 INDEX PARTITION 文

INDEX PARTITION 文は，分割方法を定義し，名前をつけます．指定できるものは，プロセッサグループの名前，次元，インデックスの範囲，分割方法，PE にオーバーラップを持たせるかどうかの 5 つです．具体的な配列の分割のイメージは次の LOCAL 文で紹介します．

```
!XOCL PROCESSOR P(32)                                  ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:100,PART=BAND) ! 分割方法を定義
```

“IP” は分割指定名です．プロセッサグループ名と同じで，名前は何でも構いません．“PROC=P は”，すぐ上で定義したプロセッサグループを分割に使用することを意味します．“INDEX=1:100” は分割する大きさの定義です．“PART=BAND” は，分割を均等に行うことを指定します．

分割方法は PART=BAND の他に循環分割 (PART=CYCLIC) および不均等分割があります[14]．

```
      PARAMETER(NMAX=10000)
!XOCL PROCESSOR P(16)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND,OVERLAP=(1,1))
```

分割指定では，分割された左右の部分を重ねて持つことができます．この重なった部分を「袖」と呼びます．

OVERLAP=(1,1) は，左右に 1 個袖を持つことを指定します．

2.7 LOCAL 文

LOCAL 文は INDEX PARTITION 文で分割を指定したローカル配列を宣言します．

●分割例 (1 次元)

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX)
!XOCL PROCESSOR P(4)                                    ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND) ! 分割方法を定義
!XOCL LOCAL A(/IP)                                      ! 分割ローカル配列の宣言
```

1 次元の配列 A を 4 つのプロセッサに均等に割り当てます．“INDEX=1:NMAX” は配列の上下限を指定します．LOCAL 文の “/” は，「データ分割をするよ」という意味です．IP は，すぐ上で定義した分割方法の名前です．この分割指定により．配列 A は図 8 のように各 PE に配置されます．

[13] SUBPROCESSOR 文によるプロセッサの引き継ぎには他にも方法があります (cf.[1])．また，部分的にプロセッサを引き継ぐこともできます．現在のところ，並列化の機能を引き出すためには，サブルーチン単位で使用する PE 数を記述する必要があります．実は，PE 台数に依存しないプログラムの記述方法があるのですが，メーカーの正式サポートの回答を得ておりませんので， ここでは紹介できません．

[14] 循環分割, 不均等分割とも初級レベルでは使いませんので，説明を省略します．

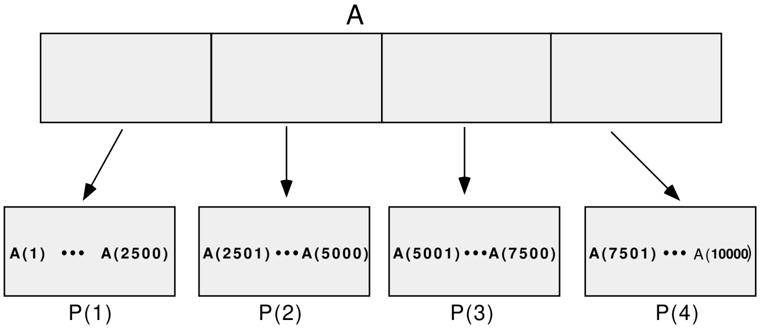


図 8： 分割ローカル配列 (1 次元)

PROCESSOR 文で定義された 4 つの PE(P(1) ~P(4)) にそれぞれ担当となる配列データが割り付けられます．

配列 A を例えば A(-500,500) で宣言した場合も上下限を設定すれば大丈夫です．

```
      PARAMETER(NMAX=500)
      DOUBLE PRECISION A(-NMAX,NMAX)
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=-NMAX:NMAX,PART=BAND)
!XOCL LOCAL A(/IP)
```

分割方法は直接 LOCAL 文に書き込むことも可能です．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX)
!XOCL PROCESSOR P(4)
!XOCL LOCAL A(/(P,INDEX=-NMAX:NMAX,PART=BAND))
```

例の 2 つの LOCAL 文は同じ意味を持ちます．が，プログラムの見やすさや拡張性を考えると INDEX PARTITION 文で名前を定義した方がいいでしょう．

また，LOCAL 文に “/” がないと，重複ローカル配列と見なされます．

•分割例 (2 次元)

Fortran で多次元のデータを分割する場合，第 1 添字でベクトル化し，最後の添字で並列化をする方法が，メモリアクセスの観点から見て有効です．従って，特別な事情がないかぎり，

2 次元配列の分割は 2 番目の添字を分割する

ようにして下さい．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX,NMAX)                       ! 2 次元配列
!XOCL PROCESSOR P(4)                                      ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)  ! 分割方法を定義
!XOCL LOCAL A(:,/IP)                                      ! 分割ローカル配列の宣言
```

10000×10000 の正方行列 A を第 2 添字で均等に分割しました．メモリアクセスの良い行方向はそのままです．“:” は「この方向には分割をしないよ」という意味です．この分割指定により，配列 A は図 9 のように各 PE に配置されます．

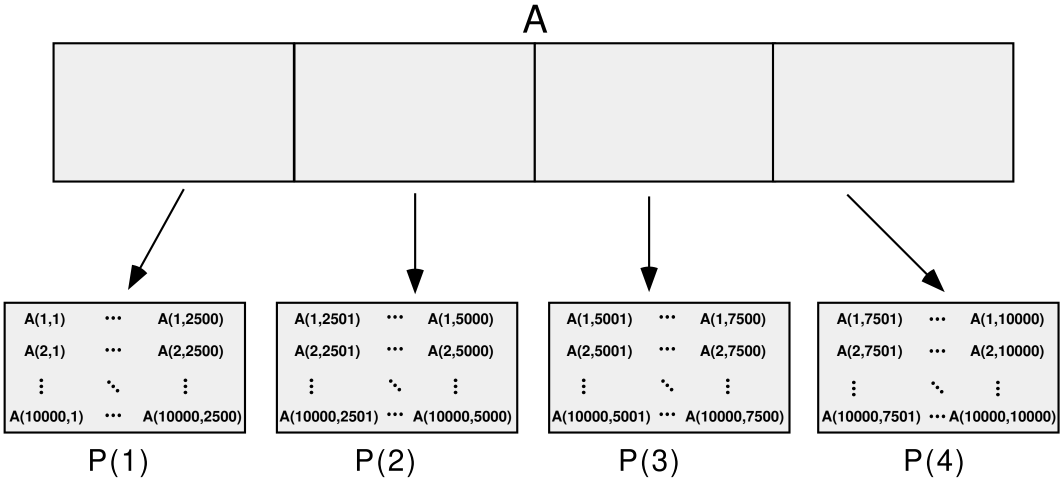


図 9： 分割ローカル配列 (2 次元)

なお，各添字を LOCAL A(/IP1,/IP2) などと同時に分割することも可能です[15]．

●分割例 (3 次元)

3 次元の流体の数値シミュレーションでは，X, Y, Z 方向の格子を切りますので，3 次元配列がよく登場します．

```
      PARAMETER(NX=150,NY=100,NZ=80)
      DOUBLE PRECISION A(NX,NY,NZ)                            ! 3 次元配列
!XOCL PROCESSOR       P(4)                                    ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NZ,PART=BAND)        ! 分割方法を定義
!XOCL LOCAL           A(:,:,/IP)                              ! 分割ローカル配列の宣言
```

$150 \times 100 \times 80$ の 3 次元配列 A を第 3 添字で均等に分割しました．この分割指定により，配列 A は図 10 のように各 PE に配置されます．

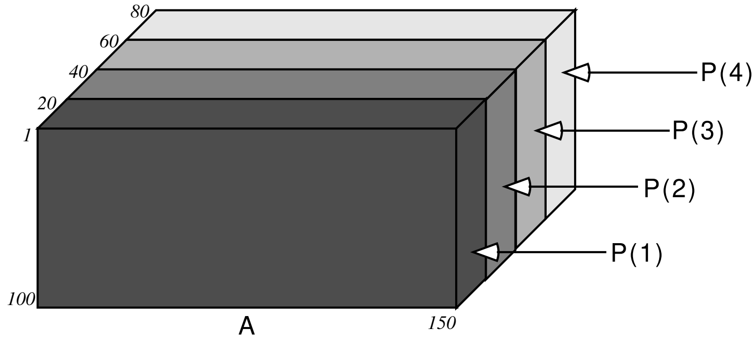


図 10： 分割ローカル配列 (3 次元)

2 番目の添字を分割する場合も同様です．

```
      PARAMETER(NX=150,NY=100,NZ=80)
      DOUBLE PRECISION A(NX,NY,NZ)                            ! 3 次元配列
!XOCL PROCESSOR       P(4)                                    ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NY,PART=BAND)        ! 分割方法を定義
!XOCL LOCAL           A(:,/IP,:)                              ! 分割ローカル配列の宣言
```

[15] もちろんデータの管理が複雑になります．

●分割例 (袖付き 2 次元)

分割指定により各 PE に割り当てられた分割ローカル配列は，割り当てられた当の PE からは高速にアクセスできますが，他の PE からはアクセスできません．しかし，プログラムでは他の PE の値がよく必要になります．差分法のアルゴリズムを持ち出すまでもなく，参照したい他の PE の値の多くは分割されたメモリの「境界」の部分です．

均等分割では，左右の添字の値を重なって持つことができます．具体的には，OVERLAP パラメータを指定し，隣の PE の境界の値を重複して持つことにより，前後の添字の指す配列を利用することができます．この重なった添字のことを袖 (*overlap*) と呼びます．

袖

OVERLAP(l,r) は，分割後の各添字の範囲に対して，小さい方向に l 個，大きい方向に r 個伸ばすことを意味します[16]．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX,NMAX)
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND,OVERLAP=(1,1))
!XOCL LOCAL A(:,/IP)
```

10000×10000 の正方行列 A を第 2 添字で前後 1 個の袖つきで均等に分割しました．この分割指定により，配列 A は図 11 ように袖つきで各 PE に配置されます．

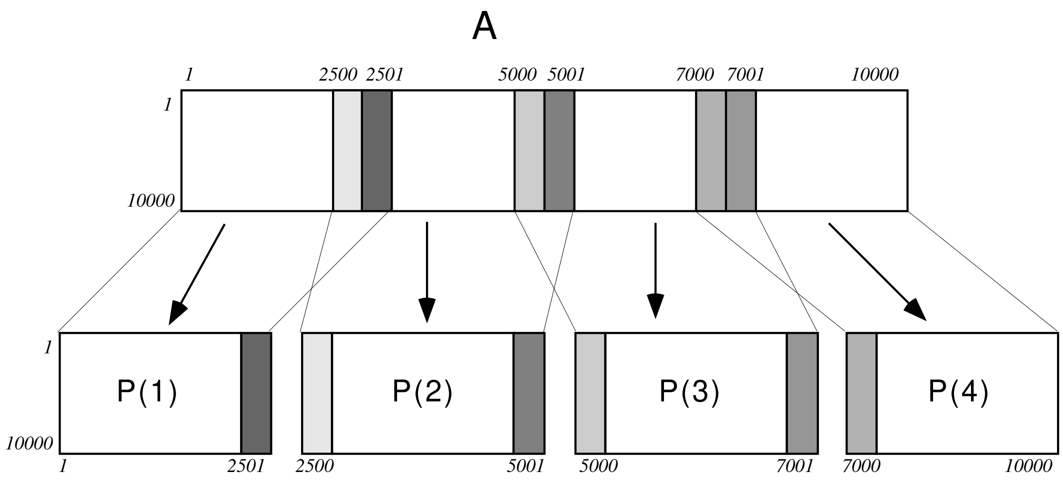


図 11： 袖付き分割ローカル配列 (2 次元)

[16]袖の長さは INDEX の範囲を超えなければいくらでも伸ばすことができますが，その分，分割ローカル配列のメモリが増えます．

2.8 GLOBAL 文

GLOBAL 文は，各 PE で共有したい配列を指定します．次の EQUIVALENCE 文と対でよく使われます．配列を分割する要領は LOCAL 文と同じです．ただし，グローバル配列は袖を持つことができません．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX,NMAX),B(NMAX)
!XOCL PROCESSOR P(4)                                      ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)  ! 分割方法を定義
!XOCL GLOBAL A(:,/IP)                                     ! グローバル配列の宣言
```

2.9 EQUIVALENCE 文

GLOBAL 文で宣言したグローバル配列は，各 PE で共有のデータとして扱うことができます．しかし，アクセスに時間がかかるうえにベクトル化もできません．一方，LOCAL 文で宣言した分割ローカル配列は，割り当てられた PE 内ならば (同じプロセッサ内のデータですので) 高速にアクセスできますし，ベクトル化も可能です．Fortran の EQUIVALENCE 文を用いてグローバル配列と分割ローカル配列を結合すると，両方の性格をもったデータを宣言することができます．

EQUIVALENCE 文で結合する場合のグローバル配列は分割指定をしないか，袖以外のすべてがローカル配列と一致した分割でなくてはなりません．プログラムの見やすさからは，分割指定を書かない方がよいでしょう．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX),AG(NMAX)
!XOCL PROCESSOR P(8)                                      ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)  ! 分割方法を定義
!XOCL GLOBAL AG                                           ! グローバル配列の宣言
!XOCL LOCAL  A(/IP)                                       ! ローカル配列の宣言
      EQUIVALENCE(AG,A)                                   ! 記憶単位の共有
```

EQUIVALENCE 文による結合によって，例えばプロセッサ P(1) からはグローバル配列 AG の AG(1) ~ AG(10000) すべてが，また，分割ローカル配列 A の A(1) ~ A(1250) が参照可能です．さらに，A(1) ~ A(1250) と AG(1) ~ AG(1250) が結合されています[17]．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX),AG(NMAX)
!XOCL PROCESSOR P(8)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND,OVERLAP=(1,1))
!XOCL GLOBAL AG
!XOCL LOCAL  A(/IP)
      EQUIVALENCE(AG,A)
```

こちらは 1 次元配列の袖付きの例です．A と AG の結合は図 12 のようになります．

[17]Fortran 90 では「使わない方がいい機能」([3]) に入っている EQUIVALENCE 文が Fortran 90/VPP の大きな特徴となっているのは悲しいことです．

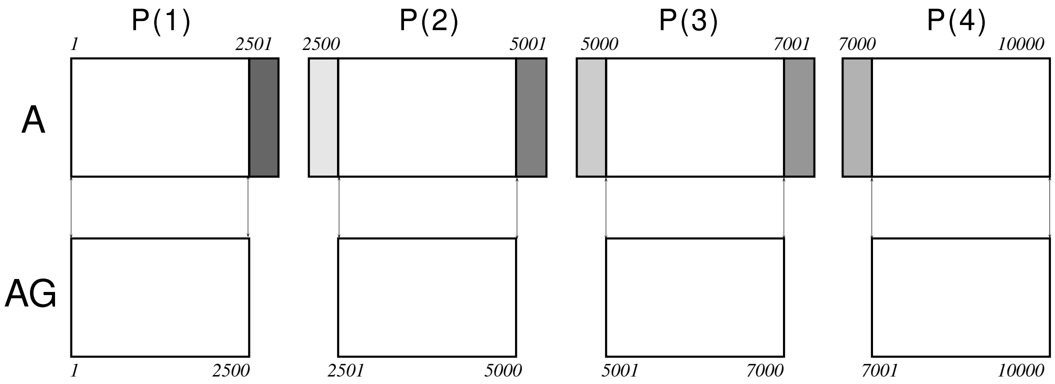


図 12 : 分割ローカル配列とグローバル配列の結合

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX),AG(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL GLOBAL AG
!XOCL LOCAL A(:,/IP)
      EQUIVALENCE(AG,A)
```

2 次元のグローバル配列と分割ローカル配列との結合例です．

2.10 分割指定の省略形

これまで見てきた宣言構文で，分割にかかわるパラメータはそれぞれ省略形 (デフォルト) を持っています．最初はきちんと INDEX PARTITION 文で分割指定を定義すべきですが，慣れてくるとだんだん不精な構文が書けるようになります．

- “PROC=” は省略できます．ただし項目の最初にプロセッサグループ名を書く必要があります．
- INDEX の下限の省略値は 1 です．
- 分割指定で INDEX を省略すると，配列の下限・上限が採用されます．
- PART の省略値は “PART=BAND” です．

つまり，

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL LOCAL A(:,/IP)
```

は

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(P,INDEX=NMAX)
!XOCL LOCAL A(:,/IP)
```

と書くことができます．さらに INDEX PARTITION を削ってしまい，プロセッサグループだけを使って，

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL LOCAL A(:,/(P))
```

と書くこともできます．SSL II/VPP のマニュアル ([4]) にはよくこの記述がでてきます．

2.11 PARALLEL REGION 構文

並列実行の開始と終了は PARALLEL REGION，END PARALLEL REGION で指示します．この構文は必ずプログラムのどこかに書く必要があります．END PARALLEL REGION は "END PARALLEL" と省略することができ，手引書の例では省略形しか記述してありませんので，本稿でも END PARALLEL と書くことにします．

PARALLEL REGION 文の前までは，1 つの PE による逐次実行が行われています．PARALLEL REGION 文に到達すると，実行ファイルとデータが他の PE に複写され並列実行が開始されます．END PARALLEL により並列実行が終了すると，再び 1 つの PE による逐次処理に戻ります．

```
      PROGRAM TEST
      PARAMETER (LA=18000,N=LA,NPE=32)
      PARAMETER (NMAX=((LA-1)/NPE+1)*NPE,NBPE=100*NPE,LD=NMAX)
      REAL*8  A(LD,NMAX),B(LD,NMAX),C(LD,NMAX),W(LD,NBPE)
      REAL*8  AG(LD,NMAX),BG(LD,NMAX),CG(LD,NMAX)
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION IP= (PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL GLOBAL          AG(:,/IP),BG(:,/IP),CG(:,/IP)
!XOCL LOCAL           A(:,/IP),B(:,/IP),C(:,/IP)
      EQUIVALENCE    (AG,A),(BG,B),(CG,C)
!XOCL PARALLEL REGION                             !-----+
        CALL SETDAT(A,B)                          !     |
        CALL MAMUL(AG,BG,CG,LA,LA,LA,N,N,N)       !     |--- 並列実行
        CALL EST(C,ERROR)                         !     |
!XOCL END PARALLEL                                !-----+
      STOP
      END
```

なお，1 回のPARALLEL REGION 構文の実行にはものすごいオーバヘッドがかかるため，プログラムの中に幾つも PARALLEL REGION 構文を記述すると，いつまで経っても計算が終らない事態を招きます．従って，

PARALLEL REGION 構文はプログラムの中に 1 回しか使ってはいけません．

PARALLEL REGION 構文は，主プログラムまたは副プログラムの中に (全体で)1 回だけ記述するようにして下さい．

2.12 SPREAD REGION 構文

SPREAD REGION[18]文は，実行文を分割してプロセッサに処理を割り振ることを指示します．

終了は END SPREAD REGION 文です．END SPREAD REGION は "END SPREAD" と省略可能です．
"REGION" は区切りを表します．

[18] "SPREAD" は「広める」「ばらまく」という意味です．

```
!XOCL PROCESSOR P(16)                                      ! プロセッサ数の宣言
!XOCL PROC ALIAS P1(8)=P(1:8),P2(4)=P(9:12),P3(4)=P(13:16) ! 別名の定義
         :
!XOCL SPREAD REGION / P1
         :                  ! P(1) ~ P(8) での処理
!XOCL REGION / P2
         :                  ! P(9) ~ P(12) での処理
!XOCL REGION / P3
         :                  ! P(13) ~ P(16) での処理
!XOCL END SPREAD REGION
```

SPREAD REGION 構文では入れ子[19]も可能です．SPREAD REGION 構文は，プログラムを複数の PE に分割して割り振るため，どの PE で何が実行されているかをきちんと管理する必要があります．そのため，初心者は手を出さない方がいいでしょう．

実行文の分割よりも，次に紹介する DO ループの分割の方が簡単かつ並列化効果が期待できます．

2.13 SPREAD DO 構文

SPREAD DO 構文は，DO ループを分割し，並列に実行することを指示します．DO ループの分割の終りを意味する END SPREAD DO は “END SPREAD” と省略可能です．また，総和や最大・最小などのよく使われる演算を並列に実行する関数もサポートされています．

●分割ローカルデータの演算例 I

```
      PARAMETER (N=10000)
      DOUBLE PRECISION A(N)
!XOCL PROCESSOR P(4)                                  ! プロセッサ数の宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND) ! 分割方法の指定
!XOCL LOCAL A(/IP)                                    ! 分割ローカル配列の宣言
            :
!XOCL PARALLEL REGION                                 ! 並列実行の開始
            :
!XOCL SPREAD DO /IP                                   !---+
       DO I=1,N                                       !   |
        A(I)=SQRT(2.0D0*I)                            !   |---DO ループの分割
       END DO                                         !   |
!XOCL END SPREAD DO                                   !---+
            :
!XOCL END PARALLEL                                    ! 並列実行の終了
            :
```

4 つの PE を用いて DO ループを分割し，並列処理します．A は分割ローカルデータとして各 PE に割り当てられているので，それぞれの PE 上で

P(1)	P(2)	P(3)	P(4)
DO I=1,2500 A(I)=SQRT(2.0D0*I) END DO	DO I=2501,5000 A(I)=SQRT(2.0D0*I) END DO	DO I=5001,7500 A(I)=SQRT(2.0D0*I) END DO	DO I=7501,10000 A(I)=SQRT(2.0D0*I) END DO

[19] SPREAD REGION 構文の中で SPREAD REGION 構文を使うこと．

とベクトル処理されます．

“/IP” は，INDEX PARTITION で指定した分割方法に従って DO ループを分割することを指示しています．分割方法 (“/IP”) を省略した場合は，「DO ループの添字の範囲をプロセッサ数で均等に分割する」仕様となります．従って，上の例では “/IP” は省略しても構わないのですが，複数の分割方法を使いこなす将来を見越して，きちんと書いた方が無難です．

●分割ローカルデータの演算例 II

```
      PARAMETER (N=18000)
      REAL*8  A(N+1,N),B(N+1,N)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
!XOCL LOCAL A(:,/IP),B(:,/IP)

!XOCL PARALLEL REGION
           :
       (A,B に関する演算)
           :
!XOCL SPREAD DO /IP
       DO J=1,N
         DO I=1,N
          A(I,J)=A(I,J)+B(I,J)/2.0D0
         END DO
       END DO
!XOCL END SPREAD DO
!XOCL END PARALLEL
      STOP
      END
```

多重ループの一番内側を分割した場合は，ベクトル性能が出ない場合がありますので，分割はできるだけ外側のループで分割します[20]．

○グローバル配列の演算は？

グローバル配列が DO ループの中に混入すると，ベクトル性能が著しく低下します．対策としては，DO ループの実行に先だってローカル配列に SPREAD MOVE 構文を使って転送するか，EQUIVALENCE 文によって結合された分割ローカル配列を使用するかします．

○重複ローカルデータの演算は？

各 PE にデータが重複して存在する重複ローカルデータも SPREAD DO で並列処理することができます．ただし，DO ループを並列化すると，DO ループ終了後の重複ローカルデータは PE の一部に正しい演算結果があり，その他は適当な数値が入っている状態となっています．そのため，PE に散らばったデータをきちんと集め直す必要があります．その機能として UNIFY 文が提供されています (cf.2.20節)．

●グローバル関数

SPREAD DO 構文には，数値計算でよく使われる処理をグローバル関数 (*global function*) として用意しています．グローバル関数は総和 (SUM)，最大 (MAX)，最小 (MIN)，論理積 (AND)，論理和演算 (OR)，最終値 (LAST) です．

[20]また，PART=CYCLIC で循環分割された DO ループはベクトル化されません．

```
      PARAMETER (N=10000,NPE=32)
      REAL*8  A(N+1,N),T
      INTEGER I,J,MAXI,MAXJ
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL LOCAL A(:,/IP)
         :
      (A の計算)
         :
      T=0.0D0
!XOCL SPREAD DO /IP
      DO J=1,N
       DO I=1,N
        IF ( T < ABS(A(I,J) ) ) THEN      !--+
          T = ABS(A(I,J)                  !  |
          MAXI = I                        !  |-- 最大値と添字の捜索
          MAXJ = J                        !  |
        END IF                            !--+
       END DO
      END DO
!XOCL END SPREAD DO MAX(T,MAXI,MAXJ)
```

分割した DO ループで最大値の計算を行うと，それぞれの PE が担当する局所的な最大値は求まりますが，全体の最大値はこれだけでは分かりません．グローバル関数 SUM は，各 PE の最大値を調べ，配列全体の最大値 (例では T) を計算します．また，添字の値 (例では MAXI, MAXJ) も調べることができます．添字を調べる必要がない場合は，(例では)MAX(T) とだけ記述します．

```
      DO I=1,N
       PIVOT(I) = 0
      END DO
!XOCL SPREAD DO /IP
      DO J=1,N
       DO I=1,N
        PIVOT(LIST(I,J)) = PIVOT(LIST(I,J)) + 1
       END DO
      END DO
!XOCL END SPREAD DO SUM(PIVOT)
```

2 次元配列 LIST の情報を PIVOT に格納する例です．SUM グローバル関数に記述する変数，配列 (この場合 PIVOT) は，SPREAD DO の前ですべての PE で同じ値にしておく必要があります．

●グローバル関数の注意事項

グローバル関数に配列を使用する場合，作業領域が不足して処理が遅くなる場合があります．その場合，実行時オプション “-Wl,-Pg” に続けて作業領域を Kbyte 単位で指定すると，処理が高速化されます．省略値は 1Kbyte です．例えば実行ファイルを a.out に対し，1MB を指定する場合は，

```
a.out -Wl,-Pg1024
```

と指定します．

2.14 SPREAD MOVE 構文

SPREAD MOVE 構文は，ローカル変数とグローバル変数の代入を一括して行うことを指定します．分割の方向が異なるような分割ローカルデータ相互の代入をグローバルデータを通して行う場合などに使います．

SPREAD DO を用いても同様の代入が可能ですが，SPREAD MOVE に比べてかなり遅くなります．SPREAD MOVE 構文は，あくまでも「代入文」のみに使える機能です．

一括代入の終了は END SPREAD MOVE です．“END SPREAD” と省略することもできます．
また，END SPREAD MOVE に続けて，データ転送の識別名をつけることで，その後の MOVEWAIT 文で転送の完了を確認することができます．

```
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:100000,PART=BAND)
      REAL*8  A(10000,10000),B(10000,10000)
!XOCL LOCAL  A(:,/IP)
!XOCL GLOBAL B(/IP,:)
          :
!XOCL SPREAD MOVE /IP,:          !---+
      DO J=1,N                   !   |
        DO I=1,N                 !   |
         A(I,J)=B(I,J)           !   |-- データの一括代入
        END DO                   !   |
      END DO                     !   |
!XOCL END SPREAD MOVE (ID)       !---+ 識別番号 ID を振る
!XOCL MOVEWAIT (ID)              !     転送の完了を確認
```

例では，分割ローカル配列 A にグローバル配列 B の値を全要素一括して代入しています．

SPREAD MOVE /IP,: の “/IP,:” は，DO ループの出現順 に分割方法を指定しています．つまり，第 2 添字を分割すること，第 1 添字は何もしないことを意味します[21]．分割指定は分割ローカル配列に合わせて指定します．

END SPREAD MOVE (ID) の “(ID)” は，転送の識別番号 ID (任意です) を振り当てることを意味します．直後の MOVE WAIT (ID) は，識別番号 ID の転送が完了したことを確認する指令です[22]．

2.15 SPREAD ASSIGNMENT 構文

SPREAD ASSIGNMENT 構文は，代入文の実行を PE が分担して実行することを指示します．記述できる文は代入文の他に，単純 WHERE 文，WHERE 文です．組み込み代入文の変数 (左辺) はベクトル添字を持たない配列である必要があります．

```
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:100,PART=BAND)
      REAL*8  A(100),B(100),C(100)
           :
!XOCL SPREAD ASSIGNMENT /IP
       A(1:50) = B(11:60) + C(21:70)
!XOCL END SPREAD ASSIGNMENT
```

[21] “,:” は省略できます．

[22] なぜ識別番号や転送確認の指示があるのかというと，実行時間の短縮のためには，データの転送中に他の処理を行ったり，複数の転送をうまく使いこなしたりするチューニングを想定しているためです．

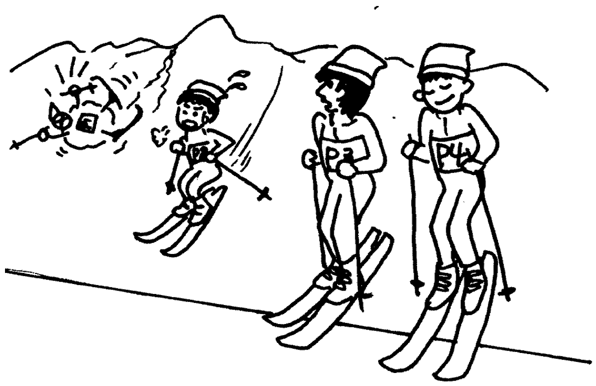
2.16　BROADCAST 文

BROADCAST 文は，LOGICAL が true となる重複ローカル変数の値を，他の PE に転送することを指定します．

```
      PARAMETER (N=10000,NPE=32)
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      REAL*8  A(N),B(N)
      LOGICAL FLAG                  ! 論理型の定義
          :
      FLAG=.FALSE.                  ! PE すべてを false にする
!XOCL SPREAD DO /IP
      DO J=1,1                      ! P(1) でのみ計算
        FLAG=.TRUE.                 ! P(1) の論理型を true にする
        DO I=1,N                    !--+
           A(I)=B(I)+2.0            !  |--P(1) での計算
        END DO                      !--+
      END DO
!XOCL END SPREAD DO
!XOCL BROADCAST(A) (FLAG)           ! 転送を行う
```

2.17　BARRIER 文

各 PE に異なる処理を割り振って「せーの」で実行に移しても，処理の形態によっては，ある PE の処理はすぐに終ってしまったのに，いつまでも処理が終らない PE があったりします．Fortran 90/VPP には，並列実行する PE にあるポイントで実行を停止し，すべての PE の処理がそのポイントまで到達した後，実行を再開するという同期機能があります．この機能をバリア同期と呼びます．BARRIER 文は，すべての PE でバリア同期を取ることを指定します．



バリア同期

通常，SPREAD DO，SPREAD MOVE 構文などは，特に指定しなければ BARRIER 文が指定されたと見なされます[23]．

2.18　LOCKON 構文

LOCKON 構文は，SPREAD REGION 構造で分割された複数のリージョン間のプログラムの実行を制御します．

LOCKON で振られた番号と同じものを持つ文は同時には実行されません (排他制御)．

[23]逆に BARRIER 文は，バリア同期をプログラマ自ら取り外して「ギンギン」のチューニングに手を染め出した時の歯止めだと言えます．

2.19 MOVEWAIT 文

MOVEWAIT 文は，SPREAD MOVE 構文，OVERLAPFIX 文，UNIFY 文で開始された PE 間データ転送の完了を確認します．

2.20 UNIFY 文

UNIFY 文は，各 PE で定義された重複ローカル配列の値を揃えることを指示します．

```
      PARAMETER (N=1000)
      DOUBLE PRECISION A(N,N),B(N,N)                      ! 重複ローカル配列の宣言
!XOCL PROCESSOR P(4)                                      ! プロセッサ数の宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)     ! 分割方法の指定
            :
!XOCL PARALLEL REGION                                     ! 並列実行の開始
!XOCL SPREAD DO /IP                                       !---+
       DO J=1,N                                           !   |
         DO I=1,N                                         !   |
           A(I,J)=SIN(B(I,J)+1.0D0)                       !   |---DO ループの分割
         END DO                                           !   |
       END DO                                             !   |
!XOCL END SPREAD DO                                       !---+
!XOCL UNIFY(A(:,/IP)) (ID)                                ! データを均一にする
!XOCL MOVEWAIT(ID)                                        ! データ転送の完了を確認
            :
!XOCL END PARALLEL
```

上の例で宣言された重複ローカル配列 A は，4 つの PE で全く同じ値を持ちます．SPREAD DO でのループの分割により，処理は各 PE で分担して

```
DO J=1,250
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(1)

```
DO J=251,500
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(2)

```
DO J=501,750
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(3)

```
DO J=751,1000
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(4)

と実行されます．ところが，重複ローカルデータは，各 PE で同じ値を持っていなければいけないデータですので，各 PE が分担した計算結果を他の PE に知らせる必要があります．

"ID" は識別番号です．直後の MOVEWAIT は，データ転送の完了を確認する文です．

2.21 OVERLAPFIX 文

OVERLAPFIX 文は，袖付き分割ローカル配列の袖部分にデータを転送することを指示します．

置き換えは，EQUIVALENCE 文によって結合しているグローバル配列の対応する範囲の値で行われます．

例えば，単一 PE で動く次のプログラムの並列化を企んだとします．

```
! ======================== STEP0 ==================================== !
      PARAMETER(N=1000000)
      DOUBLE PRECISION A(N,N),B(N,N)
           :
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
```

まず，PE32 台で A, B の第 2 添字を均等に分割します．また，A, B を分割ローカル配列として各 PE に割り振ります．さらに，SPREAD DO 構文で DO ループを分割します．

```
! ======================== STEP1 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N)
!XOCL LOCAL A(:,/IP),B(:,/IP)
           :
!XOCL PARALLEL REGION
           :
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

これで終りではありません．分割された DO ループでは，PE 上にあるデータ以外に隣の PE の境界のデータが必要です．

従って A を袖付きのローカル配列として定義し直します．方法は幾つかありますが，LOCAL 文の A の宣言に OVERLAP パラメータを追加するのがいいでしょう．

```
! ======================== STEP2 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N)
!XOCL LOCAL A(:,/IP(OVERLAP=(1,1))),B(:,/IP)
           :
!XOCL PARALLEL REGION
           :
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

まだあります．ローカル配列 A の袖へのデータの転送は，EQUIVALENCE 文で結合されたグローバル配列を介して行います．とにかく，グローバル配列を宣言し，結合させてみましょう．

```
! ======================== STEP3 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N),AG(N,N)
!XOCL LOCAL  A(:,/IP(OVERLAP=(1,1))),B(:,/IP)
!XOCL GLOBAL AG
      EQUIVALENCE (AG,A)
           :
!XOCL PARALLEL REGION
           :
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

これで OVERLAPFIX 文が登場する準備が整いました．OVERLAPFIX 文は，グローバル配列 AG から袖付き分割ローカル配列 A の袖にデータを転送することを指示します．

```
! ======================== STEP4 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N),AG(N,N)
!XOCL LOCAL  A(:,/IP(OVERLAP=(1,1))),B(:,/IP)
!XOCL GLOBAL AG
      EQUIVALENCE (AG,A)
           :
!XOCL PARALLEL REGION
           :
!XOCL OVERLAPFIX(A) (ID)    ! A の袖に AG の値を転送
!XOCL MOVEWAIT (ID)         ! 転送の完了を待つ
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

転送の識別名 ID は何でも構いません．OVERLAPFIX 文にグローバル配列 AG の姿は見えませんが，袖のデータを提供する大切な役目を果たしています．

3 サブルーチンの記述方法

3.1 プロセッサグループの受渡し

メインプログラムで宣言したプロセッサグループの情報をは SUBPROCESSOR 文でサブルーチンに引き渡します．

```
      PROGRAM MAIN                        !  メインプログラム
      PARAMETER (NPE=4)                   !  プロセッサ数
!XOCL PROCESSOR PE(NPE)                   !  プロセッサグループの宣言
             :
!XOCL PARALLEL REGION                     !  並列実行開始
             :
      CALL SUB(A)                         !  サブルーチンの CALL
             :
!XOCL PARALLEL REGION                     !  並列実行終了
             :
      SUBROUTINE SUB(A)                   !  サブルーチン
      PARAMETER (NPE=4)                   !  プロセッサ数
!XOCL PROCESSOR PE(NPE)                   !  プロセッサグループの宣言
!XOCL SUBPROCESSOR PES(NPE)=PE(1:NPE)     !  プロセッサグループの引き継ぎ
             :
```

現在の Fortran 90/VPP の文法で，プロセッサ数に依存しないサブルーチンを書くことは，できないことはないのですが，EQUVALENCE 文や OVERLAPFIX 文が使えないなどの強い制限があるため，普通はいちいちプロセッサ数を宣言することになります．

サブルーチンに XOCL 行を記述しない場合は，SUBPROCESSOR 文によるプロセッサグループの引き継ぎは不要です．ただし，そのサブルーチンはすべての PE で同じ処理がされます (冗長実行)．この部分が著しく足を引っ張るようでしたら，並列化を検討する必要があります．

3.2 データの受渡し

特に，グローバル，分割ローカル配列をサブルーチンに渡す場合は，実引数と仮引数は同じ分割をしたデータである必要があります．

例えば，メインプログラムが

```
      PROGRAM MAIN                             !  メインプログラム
      PARAMETER (NPE=32,MAXDIM=100001)         !  プロセッサ数，次元数
!XOCL PROCESSOR PE(NPE)                        !  プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)   !  分割方法の指定
      DOUBLE PRECISION A(MAXDIM,MAXDIM),AG(MAXDIM,MAXDIM) !  配列の宣言
!XOCL LOCAL  A(:,/IP)                          !  分割ローカル配列の宣言
!XOCL GLOBAL AG                                !  グローバル配列の宣言
      EQUIVALENCE (A,AG)                       !  記憶領域の共有
             :
!XOCL PARALLEL REGION                          !  並列実行開始
             :
      CALL MAKE_A(L,NU,U1,U2,A)                !  サブルーチンの CALL
             :
!XOCL PARALLEL REGION                          !  並列実行終了
      END
```

とすると，サブルーチンにも同様の分割指定が必要です．

```
      SUBROUTINE MAKE_A(L,NU,U1,U2,A)
      PARAMETER (NPE=32,MAXDIM=100001)       !  プロセッサ数，次元数
!XOCL PROCESSOR MAIN_PE(NPE)                 !  プロセッサグループの宣言
!XOCL SUBPROCESSOR PE(NPE)=MAIN_PE(1:NPE)    !  プロセッサグループの引き継ぎ
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM) !  分割方法の指定
      DOUBLE PRECISION A(MAXDIM,MAXDIM)      !  配列の宣言
!XOCL LOCAL  A(:,/IP)                        !  分割ローカル配列の宣言
            :
      RETURN
      END
```

慣れた人は，同様の手続き部分を別のファイルに記述し，include 文で読み込んだりしています．

また，サブルーチン内で，グローバル配列と分割ローカル配列を EQUIVALENCE 文で結合する必要がある場合は，グローバル配列を COMMON 文で渡し，サブルーチン内で分割ローカル配列を宣言し結合します．

```
      PROGRAM MAIN
      PARAMETER (NPE=8,MAXDIM=100001)
!XOCL PROCESSOR PE(NPE)
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)
      DOUBLE PRECISION AG(MAXDIM)
!XOCL GLOBAL AG
      COMMON/BLK/AG
            :
!XOCL PARALLEL REGION
            :
      CALL SUB                               !  サブルーチンの CALL
            :
!XOCL PARALLEL REGION
            :
      SUBROUTINE SUB
      PARAMETER (NPE=8,MAXDIM=100001)
!XOCL PROCESSOR MAIN_PE(NPE)
!XOCL SUBPROCESSOR PE(NPE)=MAIN_PE(1:NPE)
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)
      DOUBLE PRECISION A(MAXDIM),AG(MAXDIM)
!XOCL LOCAL  A(/IP)                          !  分割ローカル配列の宣言
!XOCL GLOBAL AG                              !  グローバル配列の宣言
      COMMON/BLK/AG
      EQUIVALENCE (A,AG)
            :
      RETURN
      END
```

なお，COMMON 文の共通ブロックに属する要素に，重複ローカル変数，分割ローカル配列，グローバル変数を混在させることはできません[24]．

4 ファイル入出力

分割ローカル配列の入出力をする場合の注意点です．

[24] Fortran 90 では「使わない方がいい機能」([3]) に入っている COMMON 文が Fortran 90/VPP の大きな特徴となっているのは，実に悲しいことです．

- ファイルは PARALLEL REGION 文の前にオープンする．
- EQUIVALENCE 文により結合したグローバル配列で行う．
- グローバル配列の読み書きは冗長実行部で行い，SPREAD DO 内には書かない．

重複ローカル配列は従来通りのプログラムで問題ありません．READ, WRITE には，DO 型並びも使用できますが，配列名でのアクセスが効率的です．

```
      PROGRAM MAIN
      PARAMETER (NPE=8,MAXDIM=100001)
!XOCL PROCESSOR PE(NPE)
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)
      DOUBLE PRECISION A(MAXDIM,MAXDIM),AG(MAXDIM,MAXDIM)
!XOCL LOCAL  A(:,/IP)
!XOCL GLOBAL AG
      EQUIVALENCE (A,AG)
      OPEN(33,FILE='input.data')            ! ファイルのオープン
      OPEN(15,FILE='output.data')           ! ファイルのオープン
!XOCL PARALLEL REGION
            :
      READ(33) AG
            :
      WRITE(15) AG
            :
!XOCL END PARALLEL REGION
      CLOSE(15,FILE='output.data')          ! ファイルのクローズ
      CLOSE(33,FILE='input.data')           ! ファイルのクローズ
      END
```

5　並列化例 (パラメトリックスタディ)

入力データやパラメータを変えることにより，何度も同じ計算を繰り返し実行するようなケースをパラメトリックスタディ (*parametric study*) と呼びます．

簡単な例で説明します．

```
      PROGRAM MAIN
      DOUBLE PRECISION A(1000),B(1000)
      INTEGER I
      READ(5) B                         ! データの読み込み
      DO I=1,1000                       !
       CALL SUB(A(I),B(I))              ! 同じサブルーチンを CALL
      END DO                            !
      WRITE(6) A                        ! 処理結果の書き出し
      END
!
      SUBROUTINE SUB(X,Y)
      DOUBLE PRECISION X,Y
       Y=SIN(X)
      RETURN
      END
```

5 番からデータを読み込み，B(I) をパラメータとして，次々に A(I) を求めるプログラムです．この場合，個々の I に対する SUB の処理は全く独立ですので，簡単に並列化することができます．

```
      PROGRAM MAIN
      PARAMETER (NPE=16)                     ! プロセッサ数の宣言
!XOCL PROCESSOR PE(NPE)                       ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PE,INDEX=1:1000) ! 分割方法の指定
      DOUBLE PRECISION A(1000),B(1000)
      INTEGER I
      READ(5) B                              ! データの読み込み
!XOCL PARALLEL REGION                         ! 並列実行の開始
!XOCL SPREAD DO /IP                           ! DO ループの分割
      DO I=1,1000
       CALL SUB(A(I),B(I))                   ! 同じサブルーチンを CALL
      END DO
!XOCL END SPREAD DO                           ! DO ループの分割
!XOCL UNIFY (A(/IP)) (ID)                     ! データを揃える
!XOCL MOVEWAIT(ID)                            ! データ転送の完了
!XOCL END PARALLEL REGION                     ! 並列実行の終了
      WRITE(6) A                             ! 処理結果の書き出し
      END
!
      SUBROUTINE SUB(X,Y)
      DOUBLE PRECISION X,Y
       Y=SIN(X)
      RETURN
      END
```

なお，パラメトリックスタディによる並列化プログラムのサンプル[25]を kyu-cc の /usr/local/doc 下に公開します．各自コピーして参照下さい．

```
kyu-cc% copy /usr/local/doc/parasta.txt . ↵     <--- 使用方法のテキスト
kyu-cc% copy /usr/local/doc/parasta.tar . ↵     <--- プログラム，データ（tar 形式）
```

6 並列化例 (SSL II/VPP)

数値計算でコストのかかる部分は，多くが連立 1 次方程式や固有値計算などの線形計算に集中するといってもいいでしょう．Fortran 90/VPP の賢い利用方法としては，並列計算用にチューニングされたサブルーチンライブラリ SSL II/VPP を積極的に利用することです．これらのライブラリは VPP700/56 の性能を最大限に発揮するようにチューニングされていますので，自分で組んだプログラムの方が速いということはまずありません[26]．

ここでは，需要が最もあると思われる連立 1 次方程式サブルーチン DP_VLAX の使用方法について，具体的な例をあげて説明します．

6.1 サブルーチンの引数の説明

vman dp_vlax で見ればわかるように，DP_VLAX の引数は 15 個もあります[27]．ただし，SSL II/VPP, SSL II/VP の引数の形式は一つ憶えてしまえばあとは似たりよったりなので，面倒でもすべて説明します．

[25]著作権は富士通に帰属しています．

[26]もし勝ったら是非センターまで御一報下さい．

[27]はっきり言って，富士通の日本語オンラインマニュアルの全角英数字は大いに間抜けです．

●呼び出し形式

```
CALL DP_VLAX(AG,KA,NA,N,B,IBLKS,EPSZ,ISW,IP,IS,WG,WA,WU,IWU,ICON)
```

●機能

実係数連立 1 次方程式 $Ax = b$ を，ブロック化した外積形の LU 分解法で解きます．A は $n \times n(n \geq 1)$ の正則な実行列，b は n 次元の実定数ベクトル，x は n 次元の解ベクトルです．

●引数の説明

整数は 4 バイトです．定数として入力できる引数もありますが，パラメータとしてきちんと宣言した方がデバッグに有利です．

表 3：DP_VLAX の引数一覧

引数	型	属性	入出力	内容
AG	倍精度実数	2 次元配列	入出力	入力として行列 A を AG(1:N,1:N) に格納．出力として行列 L と行列 U が AG に返される． AG は AG(KA,NA) で宣言されるグローバル配列で，このルーチンを呼び出すリージョンで 2 次元目を均等に分割されている．AG(1:N,1:N) 以外の AG の値は保証されない．
KA	整数	変数	入力	AG，WG の 1 次元目の大きさ (宣言した値)．KA $\geq$ N.
NA	整数	変数	入力	AG，WG の 2 次元目の大きさ (宣言した値)．NA $\geq$ N. NA はリージョンを構成する PE 数と IBLKS の積の倍数であること．
N	整数	変数	入力	行列 A の次数 n．
B	倍精度実数	1 次元配列	入出力	大きさ N のベクトル b を入力．解ベクトル x が出力される．
IBLKS	整数	変数	入力	ブロック化した外積形 LU 分解を行うブロックの大きさ．偶数でありかつ，30 $\leq$IBLKS$\leq$ 60 であること．
EPSZ	倍精度実数	変数	入力	ピボットの相対零判定値．EPSZ$\geq$0.0D0．0.0D0 のときは標準値が採用される．
ISW	整数	変数	入力	同一の係数行列 A をもつ方程式 $Ax = b_i$ に対し，ISW=1 のとき 1 組目の方程式を解く．ISW=2 のとき 2 組目以降の方程式を解く．ただし，このとき B の値だけを新しいベクトルに変え，それ以外の引数はそのままで使用する．
IP	整数	1 次元配列	出力	部分ピボット選択による行の入換えの履歴を示す大きさ n のベクトル．
IS	整数	変数	出力	行列 A の行列式を求めるための情報 (1 または -1)，演算後の配列 AG の n 個の対角要素と IS の値を掛け合わせると行列式が得られる．
WG	倍精度実数	2 次元配列	作業域	WG(KA,NA) なるグローバル配列で，このルーチンを呼び出すリージョンで 2 次元目を均等に分割されている．
WA	倍精度実数	2 次元配列	作業域	WA(N,IBLKS) なる配列．
WU	倍精度実数	1 次元配列	作業域	WU(IWU) で宣言される配列．
IWU	整数	変数	入力	作業用配列 WU の大きさ．NA をプロセッサ数で割った値．
ICON	整数	変数	出力	ICON=0　エラーなし． ICON=20000　行列 A のある行の要素がすべて零であったか，又はピボットが相対的に零となった．行列 A は非正則の可能性が強い． ICON=30000　NA がリージョンを構成する PE 数と IBLKS の積の倍数でない．またはNA<N, KA<N, N<1 である．または EPSZ<0.0D0 である．または IBLKS>60, IBLKS<30 である．または，グローバル配列が正しく分割されていない．

●使用上の注意

1. グローバル配列 AG は AG(1:N,1:N) の部分を渡します．通常 AG(KA,NA) と宣言していれば問題ありません．

2. SSL II/VP や NUMPAC のサブルーチンと異なり，2 次元配列の 2 次元目の大きさ NA も引数に必要です．また，NA は (IBLKS×PE 数) で割り切れる必要があります．

3. EPSZ に標準値以外の値を設定したとき，ピボットが[28] 設定した EPSZ 以下なら，そのピボットを零と見なし ICON=20000 として処理を打ち切ります．EPSZ の標準値は，丸め誤差の単位を u としたとき，EPSZ$=16 \times u$ です．なおピボットが小さくなっても計算を続行させる場合には，EPSZ へ極小の値を設定すればよいのですが，その結果は保証されません[29]．

4. 同一の係数行列 A をもつ方程式 $Ax = b_i$ を続けて解く場合は，2 回目以降を ISW=2 として解くと，もっともコストを要する LU 分解の過程を省略するため，実行時間が大幅 ($O(n)$ 違います) に短縮できます．

5. IP と IS は，LU 分解の過程で部分ピボット選択による行の入れ換えが生じた場合の情報を受け持ちます[30]．

6. DP_VLAX には，3 つ (も) の作業用配列が必要です．特に，WG は AG と全く同じ記憶領域が必要です．

6.2 単一 PE 版プログラム例

それでは，具体的な例に沿って DP_VLAX を利用するまでの過程を追いかけてみましょう．

まず，単一 PE で動作する Fortran プログラムを用意します．$n \times n$ 行列 $A = (a_{ij})$ を

$$a_{ij} = \sqrt{\frac{2}{n+1}}\ \sin\left(\frac{ij\pi}{n+1}\right),$$

$n \times 1$ ベクトル $b = (b_i)$ を

$$b_i = \sum_{j=1}^{n} a_{ij}$$

で定義します．すると連立 1 次方程式 $Ax = b$ の解は

$$x \equiv 1$$

と (理論的に) なります．プログラム Linear_Equation1 は，A, b を作成し，SSL II/VP のサブルーチン DVLAX に放りこんで得られた x(数値解) の誤差を

$$\max_{1 \le i \le n} |x_i - 1|$$

で測定するものです．

[28] 行列の消去段階で割り算を司る代表元です．ゼロになってしまうと計算が破綻してしまいます．

[29] 数値的に行列が非正則になったということは，もともとの問題に何らかの原因があるのではないかと疑って見るべきです．ただし，プログラムのチェックが先です．

[30]「部分ピボット選択」と言われてもピンと来ない人は数値計算の本を御覧下さい．

```
      program Linear_Equation1                    ! プログラムの名前
      implicit none                               ! 暗黙の型宣言の抑止
      integer,parameter            :: KA=501      ! 1 次元目の大きさのパラメータ
      integer,parameter            :: NA=500      ! 2 次元目の大きさのパラメータ
      real(kind=8),dimension(KA,NA) :: A          ! A の宣言
      real(kind=8),dimension(NA)    :: B,VW       ! B の宣言，作業領域 vw
      real(kind=8)                  :: error,s,Pi,EPSZ
      integer,dimension(NA)         :: IP
      integer                       :: N,i,j,IS,ICON,ISW
      N=KA-1                                      ! 次数の定義
      Pi=atan(1.0D0)*4.0D0                        ! πの定義
      B=0.0D0                                     ! B の初期化
       do j=1,N
        do i=1,N
         A(i,j)=sqrt(2.0D0/DBLE(N+1))*sin(dble(i)*dble(j)*Pi/dble(N+1))
           B(i)=B(i)+A(i,j)                       ! A,B の作成
        end do
       end do
! ------------------------------------------------------------------------ !
      EPSZ=0.0D0
      ISW=1
      call DVLAX(A,KA,N,B,EPSZ,ISW,IS,VW,IP,ICON)  ! 連立 1 次方程式を解く
! ------------------------------------------------------------------------ !
      error=0.0D0
      do i=1,N
       s=abs(B(i)-1.0D0)
       if(s.ge.error) error=s                     ! 誤差評価
      end do
      write(6,*) 'ICON=',ICON
      write(6,*) 'N=',N,' ERROR=',error
      end program Linear_Equation1
```

KA が N よりも一つだけ多いのは，バンクコンフリクトによる性能低下を避けるためで，それ以外の意味はありません．

6.3 SSL II/VPP のための変更

並列化に先だって，DP_VLAX のための変数を準備します．まず，多少メモリの無駄使いになるのは覚悟の上で，任意の IBLKS と PE 数に対応できるように NA の値を変更します．具体的には，Fortran で

$$\text{新 NA} = \left(\frac{\text{旧 NA}}{\text{PE 数} \times \text{IBLKS}} + 1\right) \times \text{PE 数} \times \text{IBLKS}$$

と宣言します．

次に，作業領域などのこまごまとした変数を宣言し，サブルーチン名を書き直します．グローバル配列となる AG での演算はベクトル化できませんので，A を分割ローカル配列としグローバル配列 AG と EQUIVALENCE 文で結合します．ここまでを Linear_Equation2 とします．

```
      program Linear_Equation2
      implicit none
      integer,parameter              :: NPE=4,IBLKS=50,KA=501,LA=KA-1
      integer,parameter              :: NA=(LA/(NPE*IBLKS)+1)*NPE*IBLKS
      integer,parameter              :: IWU=NA/NPE
      real(kind=8),dimension(KA,NA)   :: A,AG,WG
      real(kind=8),dimension(LA,IBLKS):: WA
      real(kind=8),dimension(LA)      :: B
      real(kind=8),dimension(IWU)     :: WU
      real(kind=8)                    :: error,s,Pi,EPSZ
      integer,dimension(LA)           :: IP
      integer                         :: N,i,j,IS,ICON,ISW
      EQUIVALENCE (A,AG)
      N=KA-1
      Pi=atan(1.0D0)*4.0D0
      B=0.0D0
       do j=1,N
        do i=1,N
         A(i,j)=sqrt(2.0D0/DBLE(N+1))*sin(dble(i)*dble(j)*Pi/dble(N+1))
           B(j)=B(j)+A(i,j)
        end do
       end do
! ---------------------------------------------------------------------------- !
      EPSZ=0.0D0
      ISW=1
      call DP_VLAX(AG,KA,NA,N,B,IBLKS,EPSZ,ISW,IP,IS,WG,WA,WU,IWU,ICON)
! ---------------------------------------------------------------------------- !
      error=0.0D0
      do i=1,N
       s=abs(B(i)-1.0D0)
       if(s.ge.error) error=s
      end do
      write(6,*) 'ICON=',ICON
      write(6,*) '   N=',N,' ERROR=',error
      end program Linear_Equation2
```

6.4 並列化例

準備が整ったところで XOCL 行を挿入します．DP_VLAX の使用方法に従って AG(A), WG の 2 次元目を均等に分割し，グローバル配列を宣言します．二つの DO ループはともに SPREAD DO で分割処理します．最初のループでは，A が対称行列であることを用いて，B の添字を変更しています．B は重複ローカル配列ですので，分割処理の後は各 PE で値を均一にする UNIFY 文を実行します．

二つ目のループでは，最大値を検索するグローバル関数 MAX が必要です．この部分は無理して並列化しなくても冗長実行で計算可能です．

以上でとりあえずの並列化が完了しました．あとは，PE 数と次元のパラメータ NPE, KA だけを変更するだけで，自由に計算規模を変更できます．このプログラムを Linear_Equation3 とします．

現実の研究レベルとなると A, B の作成方法はこんなに楽ではありません．しかし，行列に i, j についての規則性があればそれほど苦労せずに並列化できるのではないでしょうか[31]．

[31]なにしろ，肝心な連立 1 次方程式の計算は SSL II/VPP まかせですから．

```
      program Linear_Equation3
      implicit none
      integer,parameter               :: NPE=4,IBLKS=50,KA=501,LA=KA-1
      integer,parameter               :: NA=(LA/(NPE*IBLKS)+1)*NPE*IBLKS
      integer,parameter               :: IWU=NA/NPE
      real(kind=8),dimension(KA,NA)   :: A,AG,WG
      real(kind=8),dimension(LA,IBLKS):: WA
      real(kind=8),dimension(LA)      :: B
      real(kind=8),dimension(IWU)     :: WU
      real(kind=8)                    :: error,s,Pi,EPSZ
      integer,dimension(LA)           :: IP
      integer                         :: N,i,j,IS,ICON,ISW
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION PX=(PROC=P,INDEX=1:NA,PART=BAND)
!XOCL LOCAL  A(:,/PX)
!XOCL GLOBAL AG, WG(:,/PX)
      EQUIVALENCE (A,AG)
!XOCL PARALLEL REGION
      N=KA-1
      Pi=atan(1.0D0)*4.0D0
      B=0.0D0
!XOCL SPREAD DO /PX
       do j=1,N
        do i=1,N
         A(i,j)=sqrt(2.0D0/DBLE(N+1))*sin(dble(i)*dble(j)*Pi/dble(N+1))
           B(j)=B(j)+A(i,j)
        end do
       end do
!XOCL END SPREAD DO
!XOCL UNIFY(B(/PX)) (ID)
!XOCL MOVEWAIT (ID)
! ------------------------------------------------------------------------------ !
      EPSZ=0.0D0
      ISW=1
      call DP_VLAX(AG,KA,NA,N,B,IBLKS,EPSZ,ISW,IP,IS,WG,WA,WU,IWU,ICON)
! ------------------------------------------------------------------------------ !
      error=0.0D0
!XOCL SPREAD DO /PX
      do i=1,N
       s=abs(B(i)-1.0D0)
       if(s.ge.error) error=s
      end do
!XOCL END SPREAD DO MAX(error)
!XOCL END PARALLEL
      write(6,*) 'ICON=',ICON
      write(6,*) '   N=',N,' ERROR=',error
      end program Linear_Equation3
```

6.5　測定データ

計算の大部分は DP_VLAX のコストですので，この部分のみ PEPA のサブルーチンで性能を測定しました．

●並列化性能

1PE でメモリーに入りきるように次数を N=10000 に固定して並列化効率を測定しました．計算部分の AG には約 763MB のメモリが必要です．

表 4 N=10000

PE 数	GFLOPS	ピーク比
1(DVLAX)	2.1	95%
1	1.8	81%
2	3.6	81%
4	6.9	78%
8	12.5	71%
16	21.0	59%
32	31.0	44%

PE 数が増えるに従って，だんだんピーク性能比が落ちていきます．これは，1 台の PE が担当する配列が小さくなったため，ベクトル計算機の性能が十分発揮できないためと思われます．

●最大性能

次は，複数の PE で確保できる最大の記憶領域近くまで N を大きくして測定してみました．

表 5 最大記憶領域近くの N

PE 数	N	AG のメモリ	GFLOPS	ピーク比
1(DVLAX)	10000	0.7GB	2.1	95%
1	10000	0.7GB	1.8	81%
2	14500	1.5GB	3.7	84%
4	19500	2.8GB	7.4	84%
8	29000	6.2GB	17.6	83%
16	40000	11.9GB	35.2	82%
32	58000	25.0GB	70.4	80%

次数を大きくすると 32PE でもピーク性能の 80% をなんとか達成しました[32]．

[32] 同じ容量の作業領域をとっているのだから，このくらいの性能は出てくれないと困るという気もします．また，最後の N=58000 の連立 1 次方程式を解くのに約 25 分要します．

7　メッセージパッシングライブラリの利用

Fortran 90/VPP 以外の並列化として，メッセージパッシングライブラリ (MPL) を用いた方法がプログラミングが可能です．

以下，MPL の概要と参考文献を紹介します．

7.1　PVM

PVM(Parallel Virtual Machine) は，アメリカオークリッジ国立研究所で開発されたメッセージパッシングライブラリです．

通常はネットワークを介した分散環境上で各ノード間の通信のために利用されますが，VPP700 上で PE をノードとしたデータ変換を行います．

サポート機能としては，プロセス生成，各種問い合わせなどのプロセス管理，同期／非同期送受信，広報送信などのメッセージ通信，複数メッセージの一括送受信，複数プロセス間にまたがる演算機能を提供します．

•マニュアル

富士通のマニュアル [5] を参照下さい[33]．

7.2　MPI

MPI Forum において業界標準を目指して仕様検討されているメッセージパッシングライブラリです．分散されたプロセス間の集配信機能，集積演算およびグルーピングなどの強力な機能をサポートしています．なお，Fortran の自由形式はサポートされていません．

•マニュアル

富士通のマニュアル [6] を参照下さい[34]．

7.3　PARMACS

プロセス管理，メッセージ送信，同期処理，各種問い合わせ，論理ネットワーク定義を C 言語関数または Fortran サブルーチンとして制御するメッセージパッシングライブラリです．VPP700 シリーズは分散メモリ並列コンピュータですが，高速クロスバ通信により共有メモリ型並列コンピュータ並の性能を実現します (とカタログに書いてあります)．

•マニュアル

[7] を御覧下さい．PostScript ファイルが kyu-cc の`/usr/local/doc/PM61-UserGuide.ps.gz` にあります．また，`README` ファイルを次のように参照することができます．

```
kyu-cc% rsh kyu-vpp cat /usr/lang/parmacs/doc/README.VPP700 | more ↵
```

[33] PVM の日本語マニュアルは `ftp://etlport.etl.go.jp/pub/pvm/new-jdoc/jpvmug-950205.ps.gz` にあります．

[34] MPI の日本語マニュアルは `ftp://et7alpha.et.u-tokai.ac.jp/pub/mpi/mpiprimj.ps.gz` にあります．

参考文献

[1] UXP/V Fortran 90/VPP 使用手引書 V10 用, J2U5-0080, 富士通株式会社 (1995).

[2] UXP/V Fortran 90/VP 使用手引書 V10 用, J2U5-0050, 富士通株式会社 (1995).

[3] M. Metcalf, J. Reid (西村 恕彦, 和田 英穂, 西村 和夫, 高田 正之 訳) : 詳解 Fortran 90, bit 別冊, 共立出版 (1993).

[4] FUJITSU SSL II/VPP 使用手引書 (科学用サブルーチンライブラリ)V11 用, J2X0-1370, 富士通株式会社 (1996) .

[5] UXP/V PVM 使用手引書 V10 用, J2U5–0140, 富士通株式会社 (1995).

[6] UXP/V MPI 使用手引書 V10 用, J2U5–0270, 富士通株式会社 (1996).

[7] PARMACS reference manual, user's manual, PALLAS GmbH (1995).