2　最初の例

この章では，１つの簡単な例を使って，並列プログラミングを進めながらXMP の主な機能を紹介します.

2.1　逐次プログラムから始める

以下の簡単なプログラムから始めます.

```
program program1
integer a(10)

do i=1, 10
   a(i)=i**2
end do

write (*,*) a
end program program1
```

図 1: サンプルプログラム 1

このプログラムは，そのまま XMP コンパイラで翻訳し，実行することができます．1 ノードで実行すれば，普通の Fortran プログラムと同じで，結果はこの通りです.

```
 1  4  9  16  25  36  49  64  81  100
```

複数のノードで実行すればどうでしょうか．2 ノードで実行した結果はこの通りです.

```
 1  4  9  16  25  36  49  64  81  100
 1  4  9  16  25  36  49  64  81  100
```

2 つのノードが同じプログラムを実行し，それぞれの結果が出力されています[1]．XMP では，並列化の指示を書かない限り，すべてのノードが同じプログラムを実行します．この実行を冗長実行（redundant execution）と呼びます．冗長実行を並列実行に変えるためには，1) データの分散配置と 2) 計算の負荷分散を行い，それによって必要となる 3) 通信・同期を記述します.

では，図 1 のプログラムを「並列化」していきましょう.

[1]結果は，2 プロセスから標準出力に対して同時に出力を行った場合のシステムの動作に依存します．システムによっては 2 つの出力が混ざり合うかもしれませんし，1 プロセスからの出力だけが有効になるかもしれません.

2.2 テンプレートを分散する

最初に記述するのは，Nodes 指示文，Template 指示文，Distribute 指示文の 3 つです．これらは宣言指示文で，プログラムの宣言部に書きます．この

```
program program2
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

integer a(10)

do i=1, 10
   a(i)=i**2
end do

write (*,*) a
end program program2
```

図 2: サンプルプログラム 2

例では，サイズ 10 のテンプレート t を，サイズ 2 のノード配列 p に対して，block 状に分散することを宣言しています．つまり，テンプレート要素 t(1) から t(5) がノード p(1) に対応付けられ，t(6) から t(10) がノード p(2) に対応付けられます．

Nodes 指示文は，ノードの名前と形状を宣言します．この例では，2 ノードから成る 1 次元ノード配列に p という名前を付けました．2 ノードは全実行ノードかもしれないし，その一部かもしれません．ノード数を“ * ”とすれば，全ノードが p となります．全ノードの数はプログラムの実行開始時に決まります．

Template 指示文は，テンプレートの名前と形状を宣言します．テンプレートとは，配列変数のノードへの分散配置と，計算ループのノードへの負荷分散を表現するために，仲介として使われる抽象的な配列です．このプログラムでは，配列のサイズも，配列が出現するループの回転数も 10 なので，これらに合わせてテンプレートはサイズ 10 の配列にするのがよいでしょう．

Distribute 指示文は，テンプレートの分散を宣言します．分散の種別には，**block**，**cyclic**，**block-cyclic**，および **gblock**（不均等分割）があり，アプリケーションの性質によって使い分けることができます．多くの場合 block が使われます．

ここまでで，ノード配列 p とテンプレート t が宣言されました．これは，プログラムの並列化の戦略を決めたことになりますが，変数 a の宣言や DO

ループや WRITE 文について何の指示もまだ加えていませんので，コンパイルして実行しても逐次プログラムと同じで冗長実行になるだけです.

2.3 データと計算をマップする

図 3 では，テンプレートの分散が宣言されました．次に，配列変数と計算ループをテンプレートに対応付けることによって，配列変数の分散と，計算ループの並列化を指示します．配列変数をテンプレートに対応付けるのは Align 宣言指示文，計算ループをテンプレートに対応付けるのは Loop 実行指示文です.

```
program program3
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

integer a(10)
!$xmp align a(i) with t(i)

!$xmp loop on t(i)
do i=1, 10
   a(i)=i**2
end do

write (*,*) a
end program program3
```

図 3: サンプルプログラム 3

Align 指示文はプログラムの宣言部に記述し，変数とテンプレートのマッピングを指示します．この例では「ずれ」のないマッピング i→i を表現していますので，テンプレート t を介して，a(1) から a(5) の 5 要素がノード p(1)，a(6) から a(10) の 5 要素がノード p(2) に分散配置されます．Align 指示文でマッピングを指示された配列を分散配列と呼びます．指示されない配列は，全実行ノードに全体配列を割付けるので，重複配列と呼びます．スカラ変数は全実行ノードに割付けます．重複配列とスカラ変数を合わせて重複変数と呼びます．データ分散についてのより詳しい説明は，4 章にあります.

Loop 指示文は対象となる DO ループの直前に記述し，ループ反復とテンプレートとのマッピングを指示します．この例では「ずれ」のないマッピング i→i を表現していますので，テンプレート t を介して，イテレーション i=1

から i=5 が p(1) で実行され，i=6 から i=10 が p(2) で実行されます．ループの並列化についてのより詳しい説明は，2 章にあります．

このように，変数と計算を同じテンプレートに対してマップすることにより，並列ループでどのノードにどの配列要素の計算を担当させるかを正確に指示することができます．次に，WRITE 文による出力について見ていきましょう．このプログラムをコンパイルして 2 ノードで実行すると，2 つのノードからそれぞれ担当した 5 要素の出力が起こるので，例えば以下のような結果となります．

```
 1  4  9  16  25
 36  49  64  81  100
```

1 行目はノード p(1) が実行した WRITE 文の結果で，p(1) が担当する a(1) から a(5) を出力しています．2 行目はノード p(2) が実行した WRITE 文の結果で，a(6) から a(10) の出力です．WRITE 文は冗長実行されますので，どちらが先に出力されるかは決まっていません．このように行が逆転するかもしれません．

```
 36  49  64  81  100
 1  4  9  16  25
```

どちらになるかは，実行の度に違うかもしれません．システムによっては，行が混ざり合って出力されることもあり得ます．どちらのノードがどちらの行を出力しているかはっきりさせるために，WRITE 文を p(1) だけが実行するように書いてみます．実行するノードを限定するために，Task 指示構文を使います．

Task 指示構文は，Task 指示文と End Task 指示文で挟まれた実行区間について，On 節で指定されたノード群だけで実行することを指示します．図 4 の例では，p(1) の 1 ノードだけで実行されます．結果は以下のようになります．

```
 1  4  9  16  25
```

確かに，p(1) では a(1) から a(5) までを担当範囲として，正しく計算し，その結果を出力しているようです．Task 指示文の On 節の p(1) を p(2) に変えると，a(6) から a(10) までの結果が出力されます．これで，プログラムの並列化が正しく行われていることが確認できました．

2.4 ノード間通信を記述する

WRITE 文を冗長実行に任せるのではなく，プログラムしてみましょう．以下のように，逐次実行と同じ結果が出力されることを目指します．

```
program program4
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

integer a(10)
!$xmp align a(i) with t(i)

!$xmp loop on t(i)
do i=1, 10
   a(i)=i**2
end do

!$xmp task on p(1)
write (*,*) a
!$xmp end task
end program program4
```

図 4: サンプルプログラム 4

```
 1 4 9 16 25 36 49 64 81 100
```

XMP の入出力にはいくつかの方法が提供されていて，それを使う方が簡単なのですが [2] ，ここではあえて明示的にプログラムする方法を紹介します．配列 a と同じ型・形状をもつ配列 a1 を宣言します (1)．a は Align 指示行によって分散されていますが (2)，Align 指示行を書かないことによって，a1 は分散されない変数となります．つまり，すべてのノードが配列 a1 の全要素 a1(1) から a1(10) までを持ちます．a1 に分散配列 a の全要素を集めて (3)，p(1) だけが WRITE 文を実行すれば (4)，逐次実行と同じ出力になります．(3) では，通信を指示するために Gmove 構文が使われています．**Gmove 構文は**，Gmove 指示文と，それに続く配列代入文から成ります．これは Fortran90 仕様の配列代入文の形を借りていますが,「(通信先データ)=(通信元データ)」という形だけが許され，一般の式は書けません．この例では，分散配列 a の全要素から重複配列 a1 の全要素への通信を表現していて，これは図 6 のような通信パターンとなります．

Gmove 構文は，単純な配列代入文に指示文を付加するだけの簡単な仕様ですが，左辺と右辺の変数がどのように分散されているかによって，様々なパターンの通信を自動的に生成します．XMP では，Gmove 構文以外にも通信を指示する方法が数種類あります．詳しくは 4 章で説明します．XMP では，

[2]現在，言語仕様が検討されているところです．

```
program program5
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

integer a(10)
integer a1(10)                !(1)
!$xmp align a(i) with t(i)    !(2)

!$xmp loop on t(i)
do i=1, 10
   a(i)=i**2
end do

!$xmp gmove                   !(3)
         a1(1:10)=a(1:10)     !(3)

!$xmp task on p(1)            !(4)
write (*,*) a1                !(4)
!$xmp end task                !(4)
end program program5
```

図 5: サンプルプログラム 5

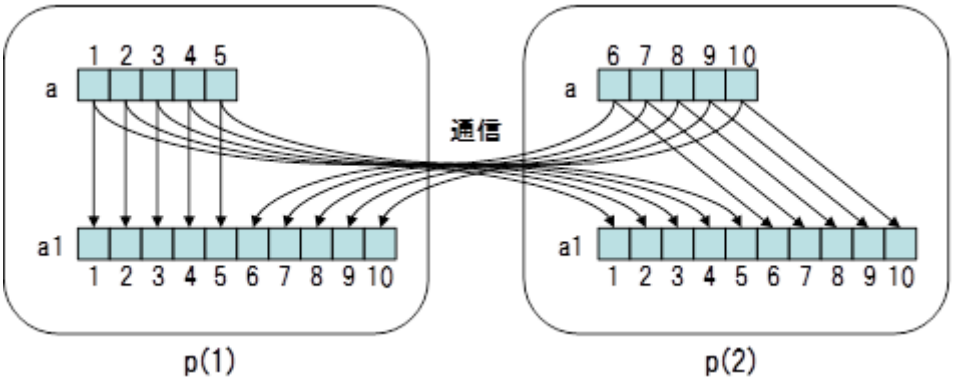

図 6: gmove の概念

ノード間の通信はすべて明示的に指示する必要がある，ということに注意してください．XMP のコンパイラは，簡単な記述で適切な通信を生成してくれますが，記述しない限り勝手に通信を生成することはありません．このよ

うな仕様とした理由は，利用者が予想できないところで通信が起こって予期せぬ性能ダウンとなり，それによってプログラムの性能チューニングが難しくなることを避けるためです．通信は利用者が意図した（記述した）場所だけで起こり，記述のない場所は全実行ノードが同じ実行を行う冗長実行となります．