3　ループの並列化

この章は，Loop 指示文を使ったループの並列化について解説します.

3.1　Loop 指示文の使い方

Loop 指示文は，対象のループを指示通りに強制的に並列化する指示行です．そのため，Loop 指示文を使うときには以下の 2 つの条件を満たす必要があります.

1. （並列化可能）対象ループは，繰り返しを跨ぐデータ依存や制御依存がないこと．つまり，ループの繰り返しは，どのような順番で実行しても正しい結果となるようなループであること.

2. （通信なし）ループ中でのデータのアクセスは，通信なしで行えること．分散されている配列は，その配列要素をアクセスするノード（使用者）と，その配列要素を持っているノード（所有者）が一致していなければならない.

ループ内でアクセスされる分散配列と重複変数について，それぞれ詳しく見ていきましょう.

3.1.1　分散配列

図 8 は，許される Loop 指示文の例です．ループ内でアクセスされる変数 a，b の添え字は i だけなので，条件 1（並列化可能）をクリアします．条件 2（通信なし）のチェックには，データの所有者と使用者のテンプレート上の位置を比較します．ループインデックス i に対して,

- データ a(i) の使用者は t(i) の分散先（赤い実線の関係）,
- データ a(i) の所有者も t(i) の分散先（青い点線の関係）

ですから，一致するので，条件 2 をクリアします．データ b(i) についても同様です.

同じプログラムで，もしループが do i=1,10 でなく do i=2,9 と書かれていたら，どのように並列化できるでしょうか．データを使用する範囲が変わりましたが，使用者に変わりはないので，Loop 指示文は同じです.

では，ループ内の a と b の添え字式が i から i+1 に変わったらどうでしょうか．この場合には，Loop 指示文の On 節に同じ式を使って t(i+1) とすることで，Loop 指示文による並列化が可能になります．原則として，Loop 指示文の On 節の式は，ループ内の添え字式の形に合わせればよいと考えてください．こうすることで，a(i) の使用者は t(i) とできます．Align 指示文でシ

```
program program6
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

real a(10),b(10),c(10)
!$xmp align a(i) with t(i)
!$xmp align b(i) with t(i)

!$xmp loop on t(i)
do i=1,10
   a(i)=a(i)+1.0
   b(i)=a(i)
end do

end program
```

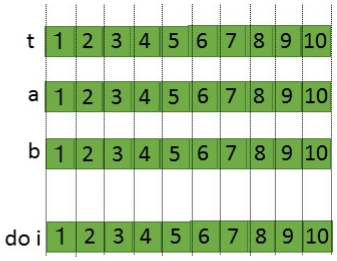

図 8: プログラム 6

```
program program7
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

real a(10),b(10),c(10)
!$xmp align a(i) with t(i)
!$xmp align b(i) with t(i)

!$xmp loop on t(i)
do i=2,9
   a(i)=a(i)+1.0
   b(i)=a(i)
end do

end program
```

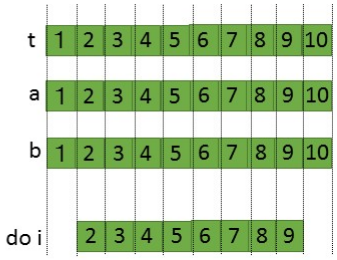

図 9: プログラム 7

フトがない場合（(i) with t(i) の形の場合），a(i) の所有者は t(i) なので，条件 2（通信なし）をクリアできます．

ループ中に現れる添え字式の形が 1 つでない場合はどうでしょうか．同じ変数に対して，a(i) と a(i-1) のように複数の添え字式の形でアクセスされている場合には，条件 2 が満たされません．一般には並列ループの前後で通信を起こす必要がありますが，データ分散の側を一工夫すれば，少ないプログラム変更で Loop 指示文が使える場合もあります．これについては 4 章に譲ります．

```
program program8
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p

real a(10),b(10),c(10)
!$xmp align a(i) with t(i)
!$xmp align b(i) with t(i)

!$xmp loop on t(i)
do i=2,9
    a(i+1)=a(i+1)+1.0
    b(i+1)=a(i+1)
end do

end program
```

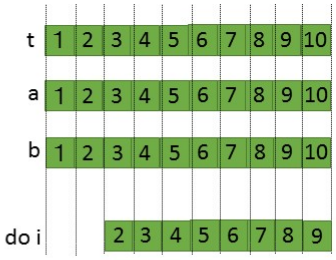

図 10: プログラム 8

3.1.2 重複変数

スカラ変数（配列でない変数）と，Align 指示文で宣言されていない配列は，重複変数となります．重複変数は全実行ノードに割付けられているので，並列化の条件 2（通信なし）は Loop 指示文の On 節の書き方に関わりなく常に満たされます．ループ内での重複変数の使い方は，以下の 3 通りが典型的です．(a) は使用するだけで定義されない変数で，ループ実行後も変数の値は不変です．(b) はループの各繰り返しの中だけで使用される変数であり，ループ実行後にこの変数が参照されることはありません．(a),(b) の変数は，Loop 指示文で指定された並列ループの中で利用できます．これらに対し，(c) は集計計算（reduction）と呼ばれるパターンで，集計変数について繰り返しを跨いだデータ依存関係があるため，ここまでに説明した Loop 指示文だけでは並列化できません．

```
program program9a
integer v,k,i
integer a(10),b(10)

v=2
k=1

do i=0,9
  a(i)=b(i+k)*v
end do
end program program9a
```

(a) 使用のみの変数v,k

```
program program9b
integer tmp,i
integer a(10),b(10)

do i=1,10
  tmp=a(i)
  a(i)=b(i)
  b(i)=tmp
end do
end program program9b
```

(b)一時変数tmp

```
program program9c
integer i
real s
real a(10)
s=0.0
do i=1,10
  s=s+a(i)*2.0
end do
end program program9c
```

(c)集計変数s

図 11: プログラム 9

集計計算については重要なので，3.2 節で詳しく説明します．

3.2 集計計算の使い方

次のプログラム例を見ていきましょう．図 12 の DO ループは，逐次実行で

```
real function asum(a,n,a0)
real a(10),a0

integer i
real asum

asum=a0
do i=1,n
   asum=asum+a(i)
enddo
return
end function asum

program program10

integer j,m
real b(10)
do j=1,10
   b(j)=j
enddo
m=10
print *,asum(b,m,0)

end program program10
```

図 12: プログラム 10

解釈すると，変数 asum について繰返しを跨いで値が継承されるので，ここまでに説明した Loop 指示文の機能だけでは並列化できません．しかし，このループが asum を**集計変数**とする**集計計算**であると分かれば，Loop 指示文に **Reduction 節**を加えることで次のように並列化できます．Reduction 節には，集計変数と共に**集計演算**を指定します．図 13 では加算の演算子を指定し，この集計計算がノードを跨ぐ総和を求めていることを表現しています．図**??**では，asum に a の要素を a(1) から a(n) の順序で足し込んでいく計算

```
real function asum0(a,n,a0)
!$xmp nodes p(*)
!$xmp template t(10)
!$xmp distribute t(block) onto p
real a(10),a0
integer n
!$xmp align a(i) with t(i)
real asum
asum=a0
!$xmp loop on t(i) reduction(+:asum)
do i=1,n
   asum=asum+a(i)
enddo

return
end function asum0

program program11
!$xmp nodes p(*)
!$xmp template t(10)
!$xmp distribute t(block) onto p
integer j,m
real b(10)
!$xmp align b(j) with t(j)

!$xmp loop on t(j)
do j=1,10
   b(j)=j
enddo
m=10
print *,asum0(b,m,0.0)

end program program11
```

図 13: プログラム 11

になっていますが，順序を守ったままでは並列化できません．図 13 のプログラムでは，Reduction 節により順序を崩して並列化することを指示しています．図 14 は 2 プロセッサのときの並列実行の様子を示しています．DO ルー

プは並列化され，変数 a の分散（cyclic）に合わせて，4 行目のようなループ処理の分担が行われています．集計変数に関わる生成コードは，コンパイラによって多少異なるでしょうが，概ね太字で示したようになります．tmp はコンパイラが自動生成した変数です．ここでは，

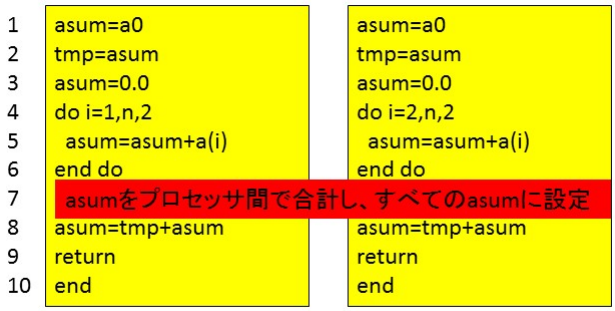

図 14: プログラム 12

1. 現在の値を退避し，0 で初期化（2，3 行目）
2. 並列ループ内で，自分の持つデータだけを合計
3. プロセッサ間で合計（7 行目)
4. 退避していた値を加えて終了（8 行目）

という実行順序に変更することで，ループを並列化しています．[4] このような集計計算を行うことができる演算は，結合則が成り立つ演算に限られます．XMP で使用できるのは，加算，乗算，論理和・積，最大・最小，ビット和・積などです．集計変数は，配列であっても構いません．その場合は，すべての配列要素に対して集計演算が行われます．

[4] 集計変数が浮動小数点型の場合には，計算順序の違いにより，逐次実行と並列実行で結果が異なる場合があります．