

XcalableMP講習会 初級編

実習 1

理化学研究所 計算科学研究機構
岩下英俊、中尾昌広

目標

- 逐次プログラムをXcalableMPの指示文を使って並列化する。

サンプルプログラムの準備

		C	Fortran
課題 1	逐次版 並列化途中	init.c xmp_init.c	init.f90 xmp_init.f90
課題 2	逐次版 並列化途中 (回答例	laplace.c xmp_laplace.c xmp_laplace_ans.c	laplace.f90 xmp_laplace.f90 xmp_laplace_ans.f90)

課題

1. 簡単なプログラムの並列化
2. 2次元差分法計算の並列化

簡単なプログラムの並列化

- ループ文を各ノードで並列実行するために、配列aを分散する

```
#include <stdio.h>

int a[10];

int main() {
    int i;
    for (i=0; i<10; i++)
        a[i] = i+1;

    for (i=0; i<10; i++)
        printf("%d¥n", a[i]);

    return 0;
}
```

```
program init
    integer :: a(10)
    integer :: i

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

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

end program init
```

プログラムの並列実行

- xmp_init.c, xmp_init.f90 に指示文を追加して並列化し、実行する。xmp_node_num() はノード番号を取得する関数。

```
#pragma xmp nodes p[2]
#pragma xmp template t[10]
#pragma xmp distribute t[block] onto p
int a[10];
[align指示文]

int main() {
    int i;

    [loop指示文]
    for (i=0; i<10; i++)
        a[i] = i+1;

    [loop指示文]
    for (i=0; i<10; i++)
        printf(" [%d] %d¥n", xmp_node_num(), a[i]);

    return 0;
}
```

```
program init
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p
    integer :: a(10)
    [align指示文]
    integer :: i

    [loop指示文]
    do i=1, 10
        a(i)=i
    end do

    [loop指示文]
    do i=1, 10
        print *, xmp_node_num(), a(i)
    end do

end program init
```

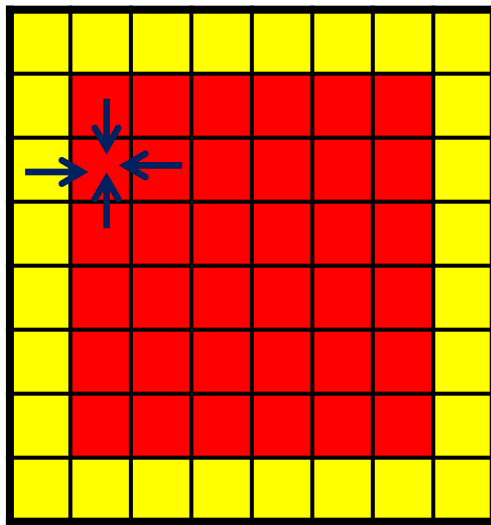
プログラムの並列実行

- コンパイル
 - `xmpcc xmp_init.c`
 - `xmpf90 xmp_init.f90`
- 実行 (2プロセスの場合)
 - `mpirun -np 2 ./a.out`

課題

1. 簡単なプログラムの並列化
2. 2次元差分法計算の並列化

2次元の差分法計算



初期値0（境界条件）



初期値

[C] $\sin((\text{double}) j / N2 * M_PI) + \cos((\text{double}) i / N1 * M_PI)$


[F] $\sin(\text{dble}(i-1) / N1 * PI) + \cos(\text{dble}(j-1) / N2 * PI)$



に対して、ラプラス方程式の差分法による
時間発展アルゴリズムを実行する。

時間発展ループ

```
for(j = 1; j < N2-1; j++)
  for(i = 1; i < N1-1; i++)
    u[j][i] = (uu[j-1][i] + uu[j+1][i] + uu[j][i-1] + uu[j][i+1])/4.0;
```

各  は上下左右の要素を使い更新される。

逐次コンパイルと実行

- `laplace.c`または`laplace.f90`をコンパイルする。
 - `gcc laplace.c -lm`
 - `gfortran laplace.f90`
- 実行する
 - `./a.out`
- 検証値 (Verification) が以下の値であることを確認せよ。

5.548855...

並列化可能なループの 3 パターン

1. ループインデックスが完全に揃っている場合

[C] $u[j][i] = uu[j][i];$

[F] $u(i, j) = uu(i, j)$

→ loop指示文で並列化（課題 1 で紹介済）

2. 隣接する配列要素の参照がある場合

[C] $u[j][i] = uu[j][i] + uu[j][i-1] + uu[j+1][i] + \dots;$

[F] $u(i, j) = uu(i, j) + uu(i-1, j) + uu(i, j+1) + \dots$

→ 袖通信+loop指示文（袖通信はこの後詳しく）

3. 総和など、ループ反復を横断する演算がある場合

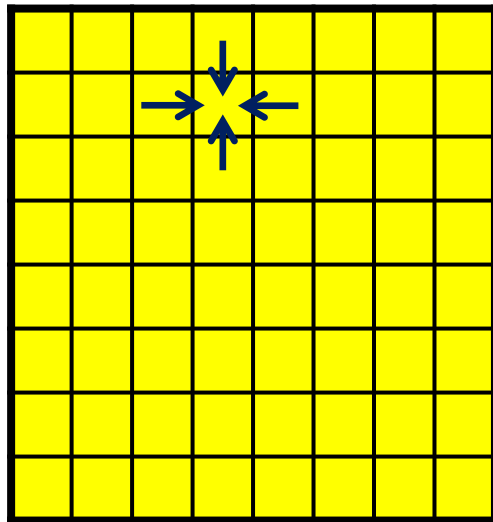
[C] $s += \text{abs}(uu[j][i] - u[j][i]);$

[F] $s = s + \text{abs}(uu(i, j) - u(i, j))$

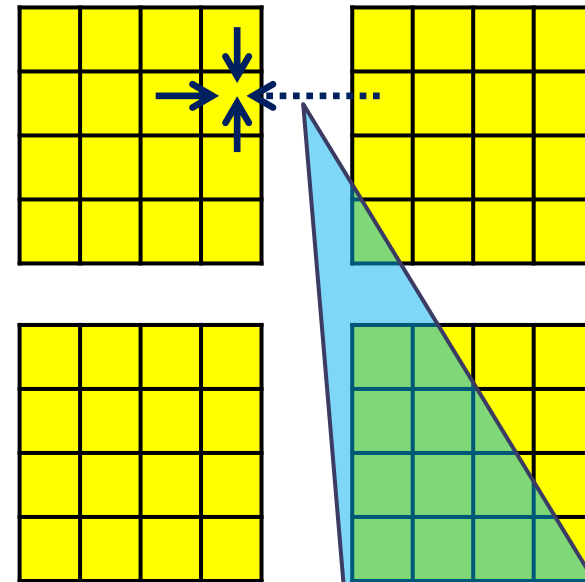
→ loop指示文に「reduction節」を付加

2次元分散における隣接要素の参照

逐次実行のときのデータ参照

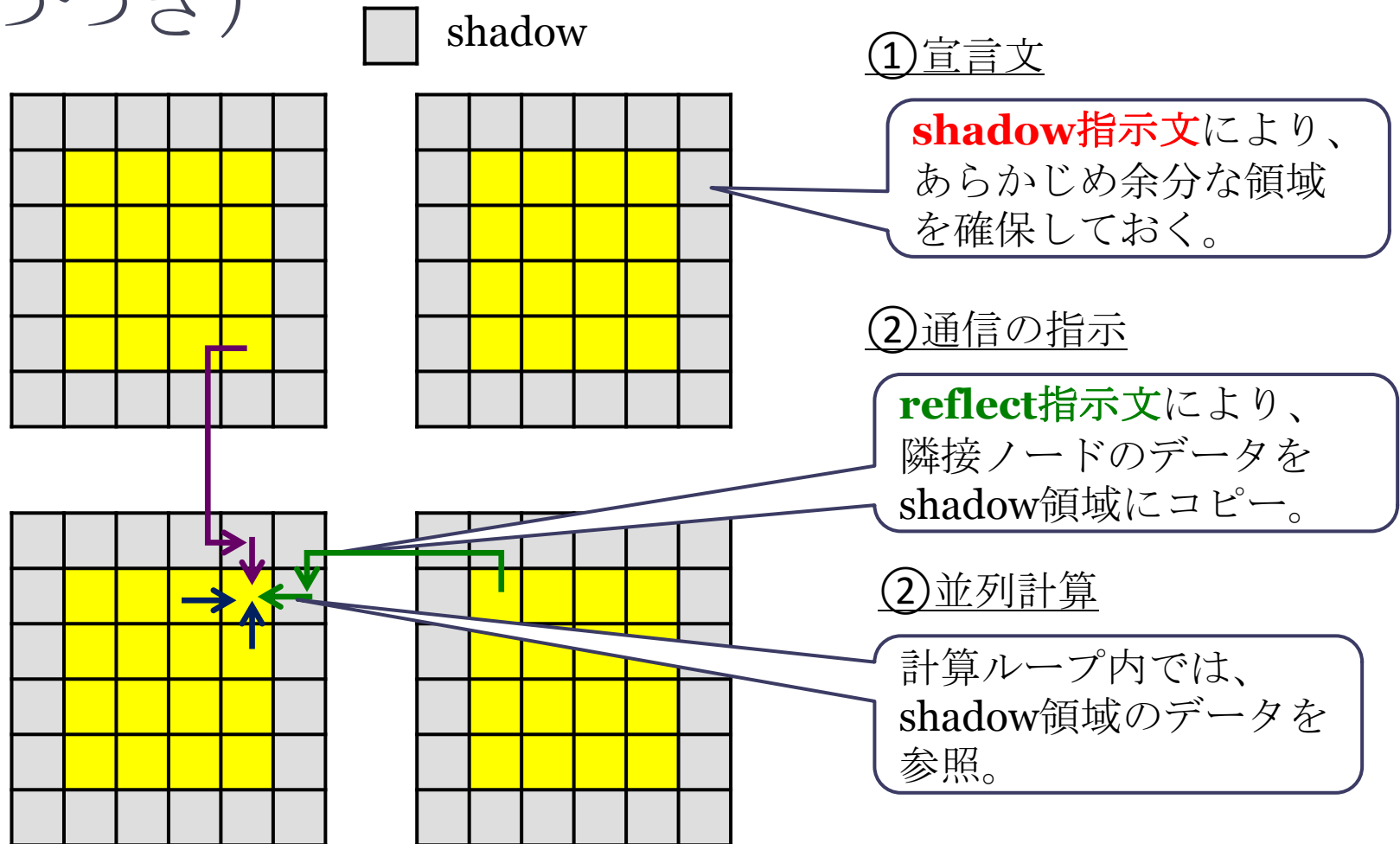


並列実行のときのデータ参照



ノードをまたぐデータの参照が必要
→ XMPは、この通信パターンをサポート

2次元分散における隣接要素の参照 (つづき)



2次元ブロック分散による並列化

- XMP指示文を用いて、`laplace.c`または`laplace.f90`を2次元ブロック分散で並列化せよ。
 - ベースプログラムは、`xmp_laplace.c`または`xmp_laplace.f90`
 - 各ループが、「ループ並列化の3つのパターン」のどれに該当するかを考える。
- 4ノードと8ノードで実行し、検証値が逐次プログラムと同程度であることを確認せよ。

2次元ブロック分散+ループ並列化3パターン

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define N1 64
#define N2 64
double u[N2][N1], uu[N2][N1];
```

```
#pragma xmp nodes p[*][4]
#pragma xmp template t[N2][N1]
[distribute指示文]
[align指示文]
[align指示文]
[shadow指示文]
```

```
int main(int argc, char **argv)
{
    int j, i, k, niter = 100;
    double value = 0.0;
```

```
#pragma xmp loop (j, i) on t[j][i]
    for(j = 0; j < N2; j++) {
        for(i = 0; i < N1; i++) {
            u[j][i] = 0.0;
            uu[j][i] = 0.0;
        }
    }
```

パターン 1

[**loop**指示文]

```
for(j = 1; j < N2-1; j++)
    for(i = 1; i < N1-1; i++)
        u[j][i] = sin((double)i/N1*M_PI)
            + cos((double)j/N2*M_PI);
```

パターン 1

```
for(k = 0; k < niter; k++) {
    /* old <- new */
```

[**loop**指示文]

```
for(j = 1; j < N2-1; j++)
    for(i = 1; i < N1-1; i++)
        uu[j][i] = u[j][i];
```

パターン 1

[**reflect**指示文]

[**loop**指示文]

```
for(j = 1; j < N2-1; j++)
    for(i = 1; i < N1-1; i++)
        u[j][i] = (uu[j-1][i] + uu[j+1][i] +
            uu[j][i-1] + uu[j][i+1])/4.0;
}
```

パターン 2

```
/* check value */
value = 0.0;
```

#pragma xmp loop (j, i) on t[j][i] [**reduction**節]

```
for(j = 1; j < N2-1; j++)
    for(i = 1; i < N1-1; i++)
        value += fabs(uu[j][i]-u[j][i]);
```

パターン 3

#pragma xmp task on p[0][0]

```
printf("Verification = %g¥n", value);
```

```
return 0;
```

```
}
```