XcalableACCハンズオン

理化学研究所　計算科学研究センター
中尾昌広

目標

逐次プログラムをXcalableACCを使って並列化する。

```
$ wget http://xcalablemp.org/ja/download/lecture/XACC/sample.tar.bz2
$ tar xfj sample.tar.bz2
$ cd sample
```

課題 1	逐次版 並列化途中 回答例	loop.c xacc_loop.c answer/xacc_loop.c
課題 2	逐次版 並列化途中 回答例	laplace.c xacc_laplace.c answer/xacc_laplace.c

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

loop.c

```
int a[10], i;

int main(){

  for(i=0;i<10;i++)
    a[i] = i;

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

  return 0;
}
```

プログラムの並列実行

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

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

int main(){
[OpenACC parallel loop指示文 with copy節]
[XMP loop指示文]
  for(i=0;i<10;i++)
    a[i] = i;

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

  return 0;
}
```

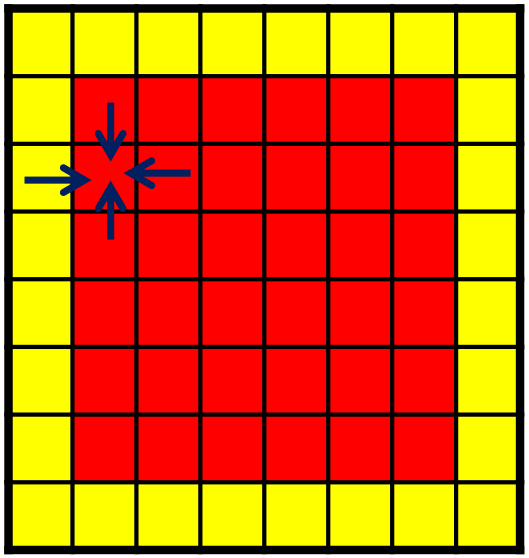
プログラムの並列実行

- コンパイル
  - xmpcc xacc_loop.c -xacc --device=Pascal
- 実行（Reedbush-Hのバッチジョブ）
  - qsub sample.sh

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
sin((double)j/N2*M_PI)+cos((double)i/N1*M_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って更新する。

逐次コンパイルと実行

- laplace.cをコンパイルする。
  - gcc laplace.c -lm
- 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

   ```
   u[j][i] = uu[j][i];
   ```

   → loop指示文で並列化

2. 隣接する配列要素の参照がある場合

   ```
   u[j][i] = uu[j][i] + uu[j][i-1] + uu[j+1][i] + …;
   ```

   → 袖通信＋loop指示文（袖通信はこの後詳しく）

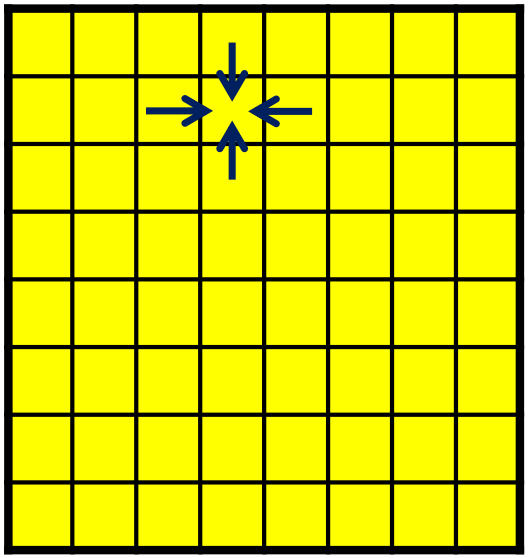
3. 総和など、ループ反復を横断する演算がある場合

   ```
   value += abs(uu[j][i]-u[j][i]);
   ```

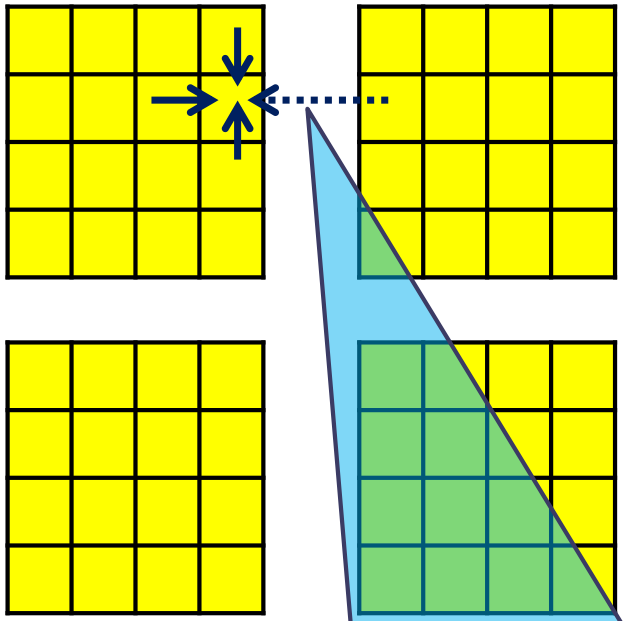
   loop指示文に「reduction節」を付加

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

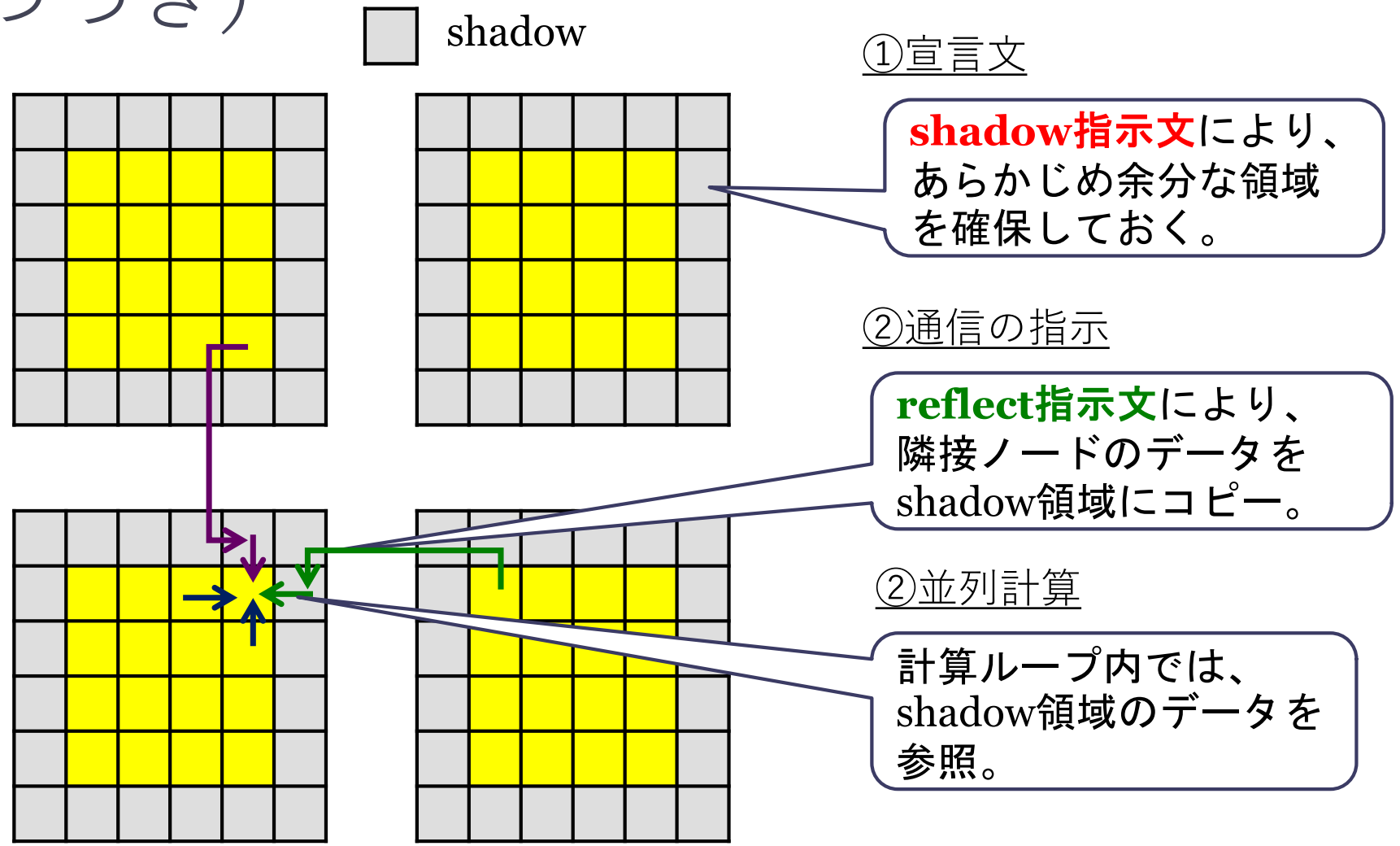


並列実行のときのデータ参照



ノードをまたぐデータの参照が必要
→ XACC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

- XACCを用いて、laplace.c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、xacc_laplace.c
  - 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。
- 2ノード4プロセスで実行し、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[*][2]
#pragma xmp template t[N2][N1]
[XMP distribute指示文]
[XMP align指示文]
[XMP align指示文]
[XMP 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;
        }

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

パターン１

```
[OpenACC data指示文]
    for(k = 0; k < niter; k++){
        /* old <- new */
[OpenACC parallel loop指示文]
[XMP loop指示文]
        for(j = 1; j < N2-1; j++)
          for(i = 1; i < N1-1; i++)
            uu[j][i] = u[j][i];

[XMP reflect指示文 with acc節]
[OpenACC parallel loop指示文]
[XMP 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;
    }

    /* check value */
    value = 0.0;
#pragma xmp loop (j,i) on t[j][i] reduction(+:value)
    for(j = 1; j < N2-1; j++)
        for(i = 1; i < N1-1; i++)
          value += fabs(uu[j][i]-u[j][i]);

#pragma xmp task on p[0][0]
    printf("Verification = %g¥n",value);

    return 0;
}
```

パターン１

パターン２

パターン３

プログラムの並列実行

- コンパイル
  - xmpcc xacc_laplace.c -xacc --device=Pascal
- 実行（Reedbush-Hのバッチジョブ）
  - qsub sample.sh