TOHOKU UNIVERSITY

Cyberscience Center

Empowered by Innovation NEC

2023年度
はじめての並列化

２０２３年　５月３１日

東北大学サイバーサイエンスセンター
日本電気株式会社

本資料は，東北大学サイバーサイエンスセンターとＮＥＣの共同により作成された.

- 並列化概要編
- OpenMPプログラミング編

演習問題の準備

- 演算問題の環境を自分のホームディレクトリ配下にコピーします。

/mnt/stfs/ap/lecture/parallel/

|-- practice_1　演習問題1
|-- practice_2　演習問題2
|-- practice_3　演習問題3
|-- practice_4　演習問題4

```
$ cd <環境をコピーしたいディレクトリ>
$ cp –r /mnt/stfs/ap/lecture/parallel  .
```

並列化概要

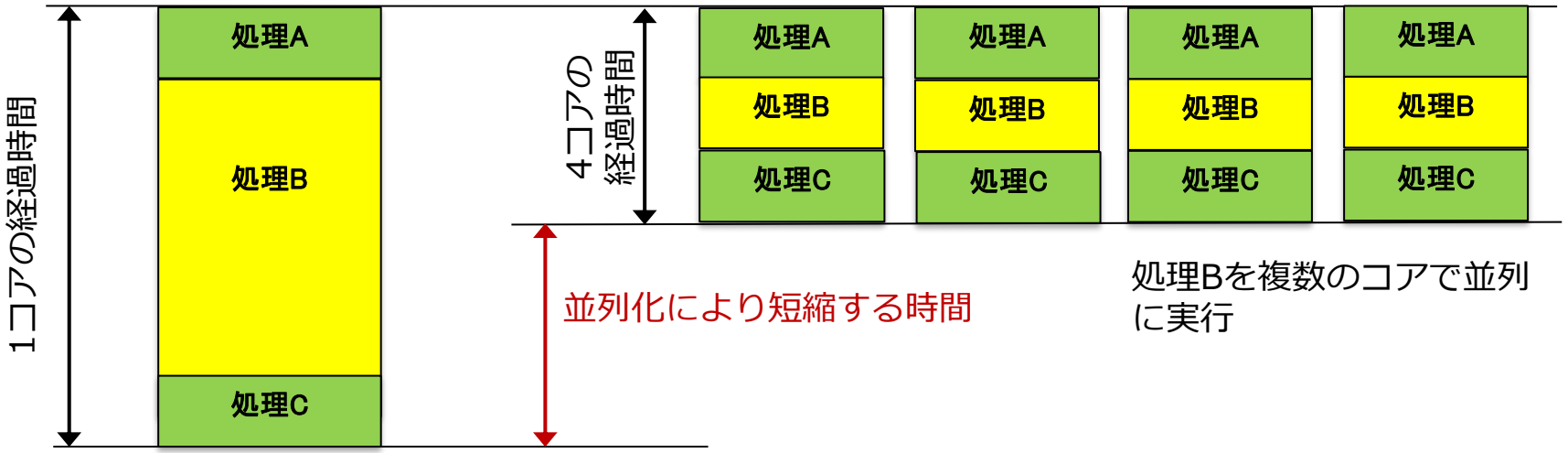
1. 並列化とは

並列処理・並列実行

- 仕事（処理）を複数のコアに分割し，同時に実行すること

並列化

- 並列処理を可能とするために，処理の分割を行うこと



2. 並列化の効果

姫野ベンチマーク（Fortran）

・ポアッソン方程式解法をヤコビの反復法で解く場合に主要なループの処理速度を測るもの

プログラムの一部抜粋

```
      gosa=0.0
      DO K=2,kmax-1
        DO J=2,jmax-1
          DO I=2,imax-1
            S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
     1           +a(I,J,K,3)*p(I,J,K+1)
     2           +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3           -p(I-1,J+1,K)+p(I-1,J-1,K))
     4           +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5           -p(I,J+1,K-1)+p(I,J-1,K-1))
     6           +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7           -p(I+1,J,K-1)+p(I-1,J,K-1))
     8           +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9           +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
            SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
            GOSA=GOSA+SS*SS
            wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
          enddo
        enddo
      enddo
```

- AOBA-A　1コアの実行時間は約 29.2 秒

```
********  Program  Information  ********
 Real Time (sec)                        :           29.197789
 User Time (sec)                        :           29.195756
 Vector Time (sec)                      :           29.140397
 Inst. Count                            :        48890015000
 V. Inst. Count                         :         9526901371
 V. Element Count                       :      2379556386510
 V. Load Element Count                  :      1290401280363
 FLOP Count                             :      1411535186639
 MOPS                                   :           98044.806708
 MOPS (Real)                            :           98038.074875
 MFLOPS                                 :           48347.227656
 MFLOPS (Real)                          :           48343.908098
 A. V. Length                           :             249.772334
 V. Op. Ratio (%)                       :              98.624867
 L1 Cache Miss (sec)                    :               0.043942
 CPU Port Conf. (sec)                   :               0.000000
 V. Arith. Exec. (sec)                  :              14.230997
 V. Load Exec. (sec)                    :              14.907927
 VLD LLC Hit Element Ratio (%)          :              57.370385
 FMA Element Count                      :      443575440000
 Power Throttling (sec)                 :               0.000000
 Thermal Throttling (sec)               :               0.000000
 Memory Size Used (MB)                  :             708.000000
 Non Swappable Memory Size Used (MB)    :              98.000000
```

- 複数のコアを用いることで実行時間の短縮が可能に

2. 並列化の効果

姫野ベンチマーク（C/C++）

・ポアッソン方程式解法をヤコビの反復法で解く場合に主要なループの処理速度を測るもの

プログラムの一部抜粋

```
gosa = 0.0;
for(k=1 ; k<kmax-1 ; ++k)
  for(j=1 ; j<jmax-1 ; ++j)
    for(i=1 ; i<imax-1 ; ++i){
      s0 = a[0][k][j][i] * p[k  ][j  ][i+1]
         + a[1][k][j][i] * p[k  ][j+1][i  ]
         + a[2][k][j][i] * p[k+1][j  ][i  ]
         + b[0][k][j][i] * ( p[k  ][j+1][i+1] - p[k  ][j-1][i+1]
                           - p[k  ][j+1][i-1] + p[k  ][j-1][i-1] )
         + b[1][k][j][i] * ( p[k+1][j+1][i  ] - p[k+1][j-1][i  ]
                           - p[k-1][j+1][i  ] + p[k-1][j-1][i  ] )
         + b[2][k][j][i] * ( p[k+1][j  ][i+1] - p[k+1][j  ][i-1]
                           - p[k-1][j  ][i+1] + p[k-1][j  ][i-1] )
         + c[0][k][j][i] * p[k  ][j  ][i-1]
         + c[1][k][j][i] * p[k  ][j-1][i  ]
         + c[2][k][j][i] * p[k-1][j  ][i  ]
         + wrk1[k][j][i];
      ss = ( s0 * a[3][k][j][i] - p[k][j][i] ) * bnd[k][j][i];
      gosa = gosa + ss*ss;
      wrk2[k][j][i] = p[k][j][i] + omega * ss;
    }
```

- AOBA-A　1コアの実行時間は約 29.0 秒

```
********  Program  Information  ********
 Real Time (sec)                        :           29.017888
 User Time (sec)                        :           29.016413
 Vector Time (sec)                      :           29.016411
 Inst. Count                            :       27317956640
 V. Inst. Count                         :        9367701968
 V. Element Count                       :     2339229193808
 V. Load Element Count                  :     1290401280000
 FLOP Count                             :     1371210136391
 MOPS                                   :        95132.417639
 MOPS (Real)                            :        95128.562776
 MFLOPS                                 :        47255.877717
 MFLOPS (Real)                          :        47253.962860
 A. V. Length                           :          249.712171
 V. Op. Ratio (%)                       :           99.349730
 L1 Cache Miss (sec)                    :            0.000116
 CPU Port Conf. (sec)                   :            0.000000
 V. Arith. Exec. (sec)                  :           13.275749
 V. Load Exec. (sec)                    :           15.740104
 VLD LLC Hit Element Ratio (%)          :           56.720855
 FMA Element Count                      :      403250400000
 Power Throttling (sec)                 :            0.000000
 Thermal Throttling (sec)               :            0.000000
 Memory Size Used (MB)                  :          568.000000
 Non Swappable Memory Size Used (MB)    :           92.000000
```

- 複数のコアを用いることで実行時間の短縮が可能に

3. 演習問題1　（practice_1）

姫野ベンチマークの1コア実行

項目	対象	備考
作業ディレクトリ	practice_1	
使用ソースファイル	sample1.f	編集　不要
ジョブファイル	run.sh	そのまま投入

- 手順①：作業ディレクトリを移動してください.

  % cd parallel/practice_1

- 手順②：コンパイルします.

  % nfort sample1.f

- 手順③：ジョブを投入します.

  % qsub run.sh

3. 演習問題1　（practice_1）

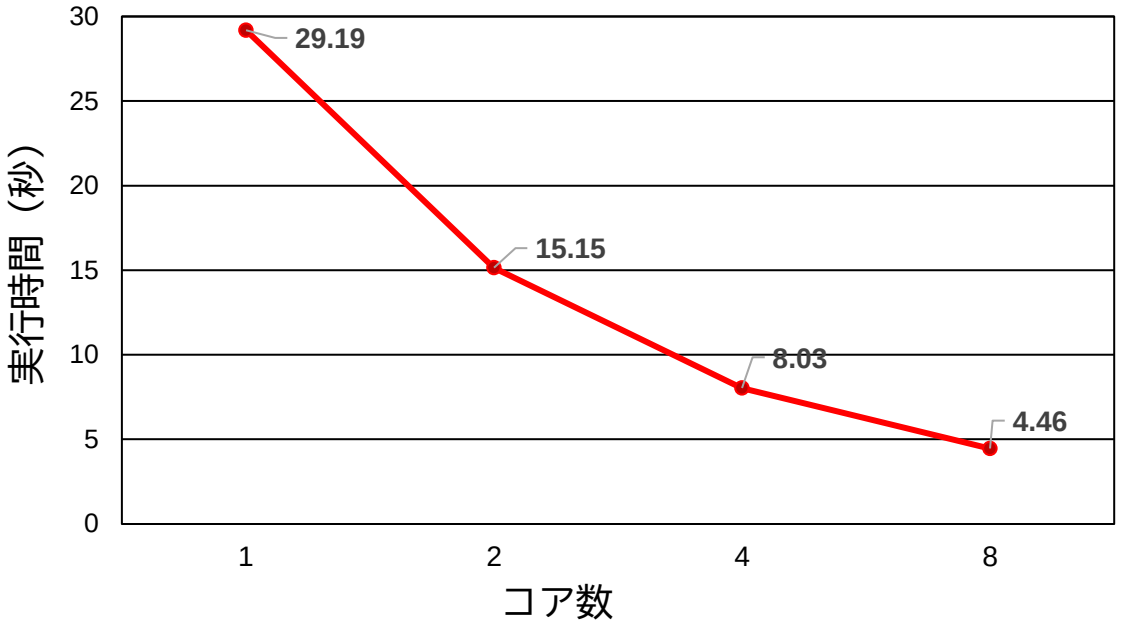
- 手順④：結果を確認します．結果はp1-practice.o.XXXX(XXXXにはジョブIDが入ります)として格納されます．

  % cat p1-practice.o.XXXX

4. 並列化の効果（自動並列）

自動並列化により複数のコアを利用し，実行時間を短縮

AOBA-A 1VE(8コア)



5. 演習問題2　（practice_2）

姫野ベンチマークの自動並列実行

項目	対象	備考
作業ディレクトリ	practice_2	
使用ソースファイル	sample2.f	編集　不要
ジョブファイル	run.sh	そのまま投入

- 手順①：作業ディレクトリを移動してください.

  % cd parallel/practice_2

- 手順②：コンパイルします.

  % nfort -mparallel sample2.f

- 手順③：ジョブを投入します.

  % qsub run.sh

5. 演習問題2　（practice_2）

- 手順④：結果を確認します．結果はp2-practice.o.XXXX(XXXXにはジョブIDが入ります)として格納されます．

  % cat p2-practice.o.XXXX

　ジョブファイル（run.sh）の OMP_NUM_THREADS で設定した数値が，実行時の並列数となります．

　OMP_NUM_THREADSの数値を1～8（AOBA-Aは8コア）に変更することで，実行時間が変化することを確認してください．

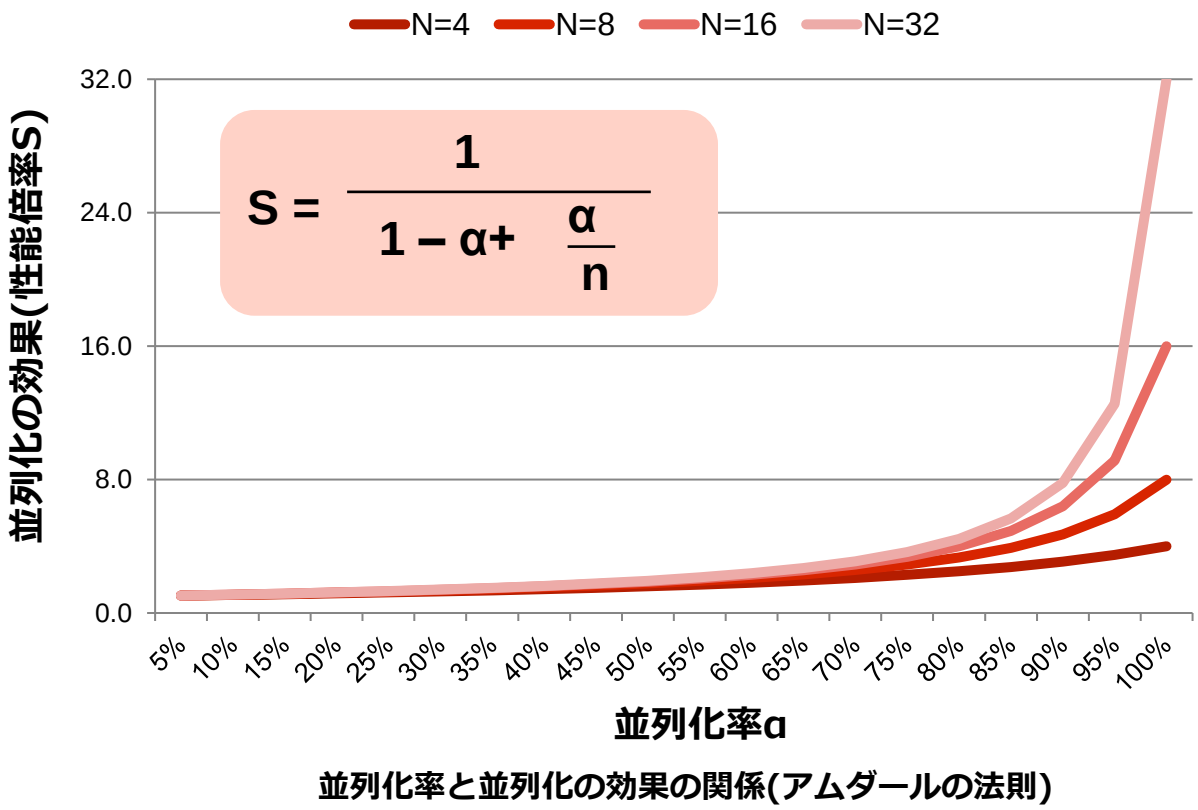
6. 並列化率

- 並列に実行可能(あるいは効果のある)部分と並列に実行不可(あるいは効果のない)部分を見つけ，並列に実行可能部分を複数のコア(CPU)に割り当てる.
- できるだけ多くの部分を並列化の対象としなければ，コア数に応じた効果が得られない.

$$\text{並列化率}\alpha = \frac{\text{並列化対象部分の処理時間}}{\text{全体の処理時間}}$$

(並列化の対象部分と非対象部分の時間の合計)

7. 並列処理の限界（アムダールの法則）



並列化率と並列化の効果の関係(アムダールの法則)

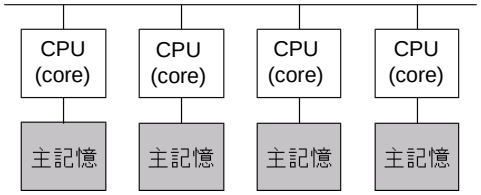
- Nはコア(CPU)数
- 並列化率が100%から下がるにしたがって性能倍率は急速に低下する

並列化率100%はあり得ない(データの入出力で必ず逐次処理発生)が，可能な限り100%に近づかなければ並列化の大きな効果は得られない

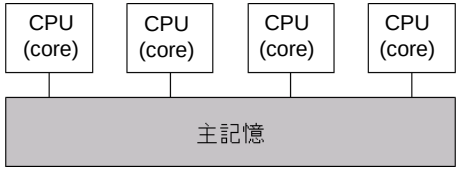
8. 並列処理モデル

コンピュータアーキテクチャに応じた処理の分担(分割)のさせ方によって幾つかの並列処理がある

1. 分散メモリ並列処理
   - AOBA-A(マルチノード)
   - AOBA-B(マルチノード)



2. 共有メモリ並列処理
   - AOBA-A(シングルノード)
   - AOBA-B(シングルノード)



- MPI(Message Passing Interface)は分散メモリ並列処理のための並列手法
- OpenMPは共有メモリ並列処理のための並列手法

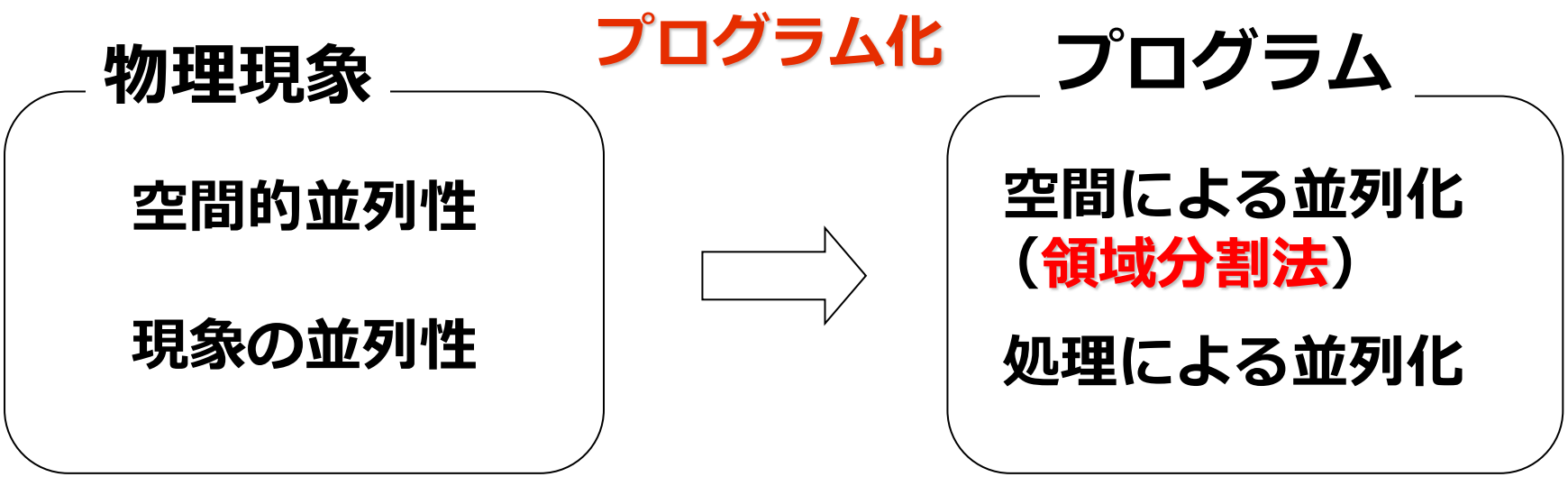
8. 並列処理モデル

並列処理モデルの特徴

	共有メモリ並列処理	分散メモリ並列処理
メモリ空間	すべてのコアが同じメモリ空間をアクセス可能	ネットワーク上のすべてのメモリ空間をデータ通信によりアクセス可能
利用可能な並列化手法	自動並列化 OpenMP並列化 MPI並列化	MPI並列化 (※1)
利用可能なAOBA-Aのコア数	8コア(1VE)	2,048コア(256VE)
利用可能なAOBA-Aのメモリ容量	48GByte	約12TByte

※1：共有メモリ内は共有メモリ並列処理，分散メモリ間は分散メモリ並列処理のハイブリッド並列処理が可能

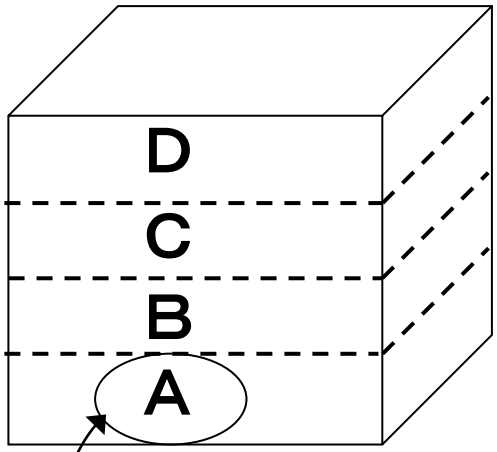
9. 並列化の対象



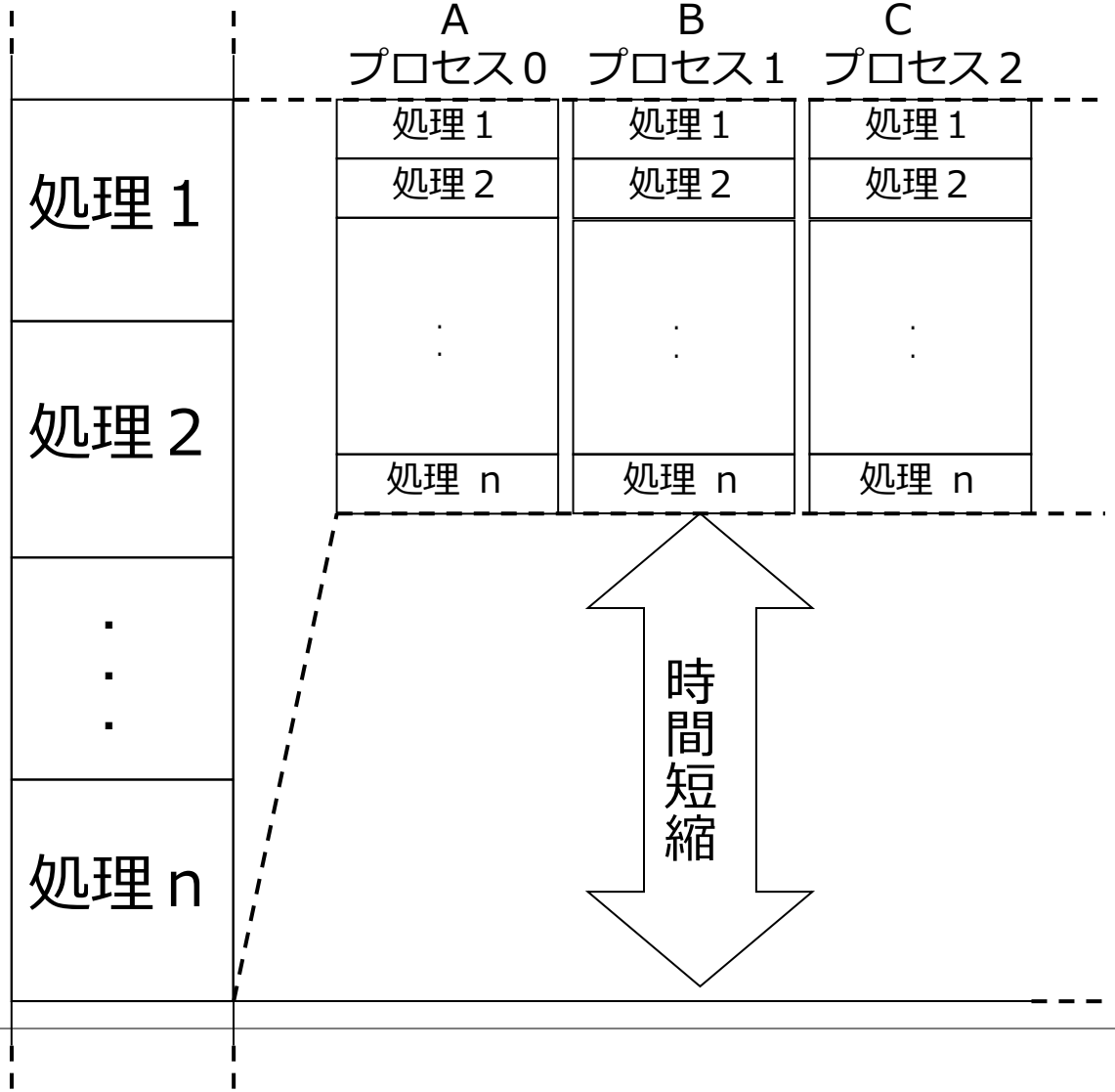
9. 並列化の対象

空間による並列化（イメージ）

領域分割法



各々，コア(CPU)に
割り当てる



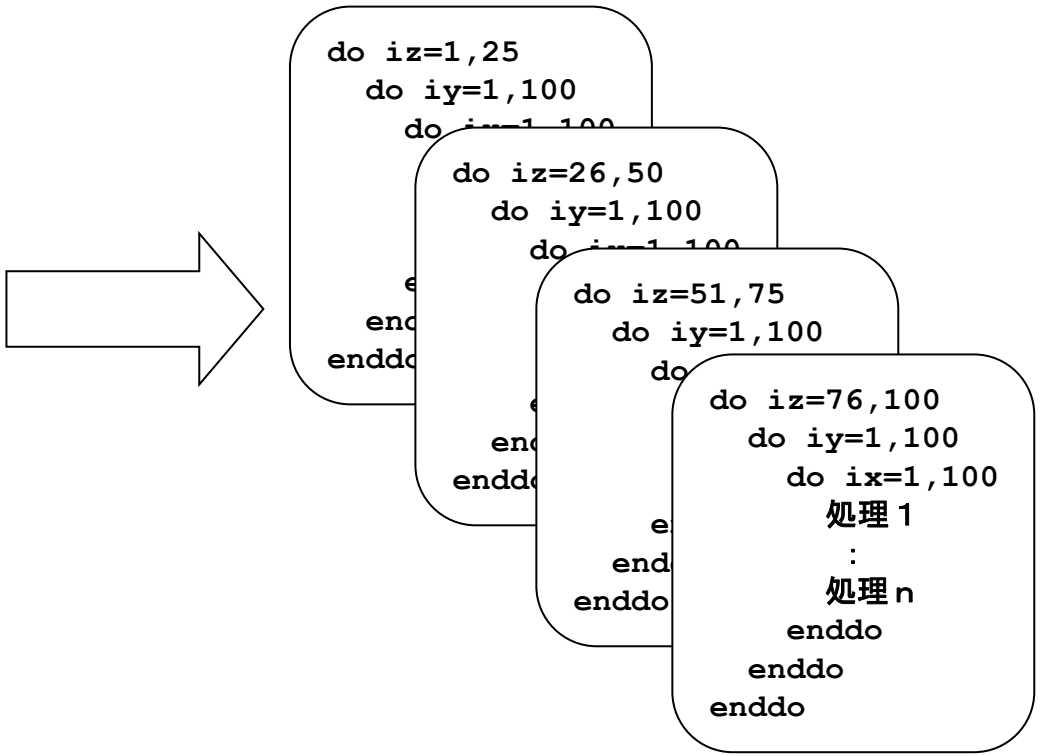
9. 並列化の対象

空間による並列化の例：Fortran

DOループ単位での並列処理

例）領域分割法

```
do iz=1,100
  do iy=1,100
    do ix=1,100
      処理１
        :
      処理ｎ
    enddo
  enddo
enddo
```



より外側のループで並列化することが重要

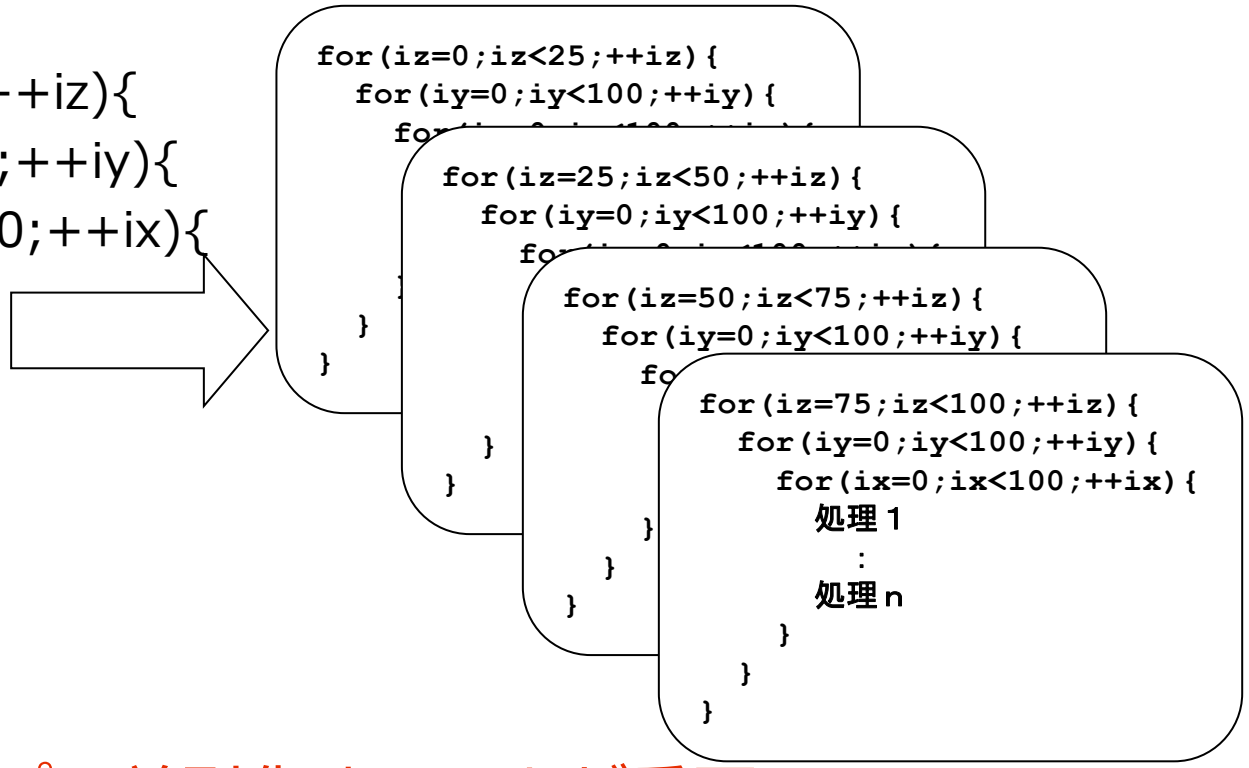
9. 並列化の対象

空間による並列化の例：C/C++

forループ単位での並列処理

例）領域分割法

```
for(iz=0;iz<100;++iz){
  for(iy=0;iy<100;++iy){
    for(ix=0;ix<100;++ix){
      処理１
        :
      処理ｎ
    }
  }
}
```



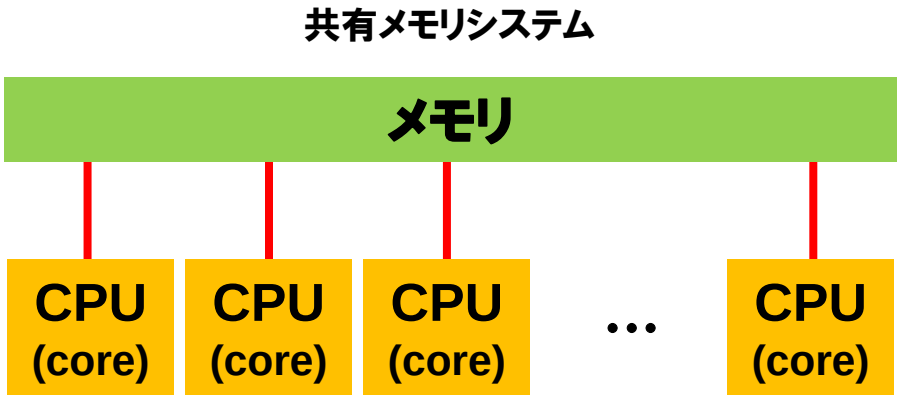
より外側のループで並列化することが重要

OpenMPプログラミング編

1．OpenMP概要

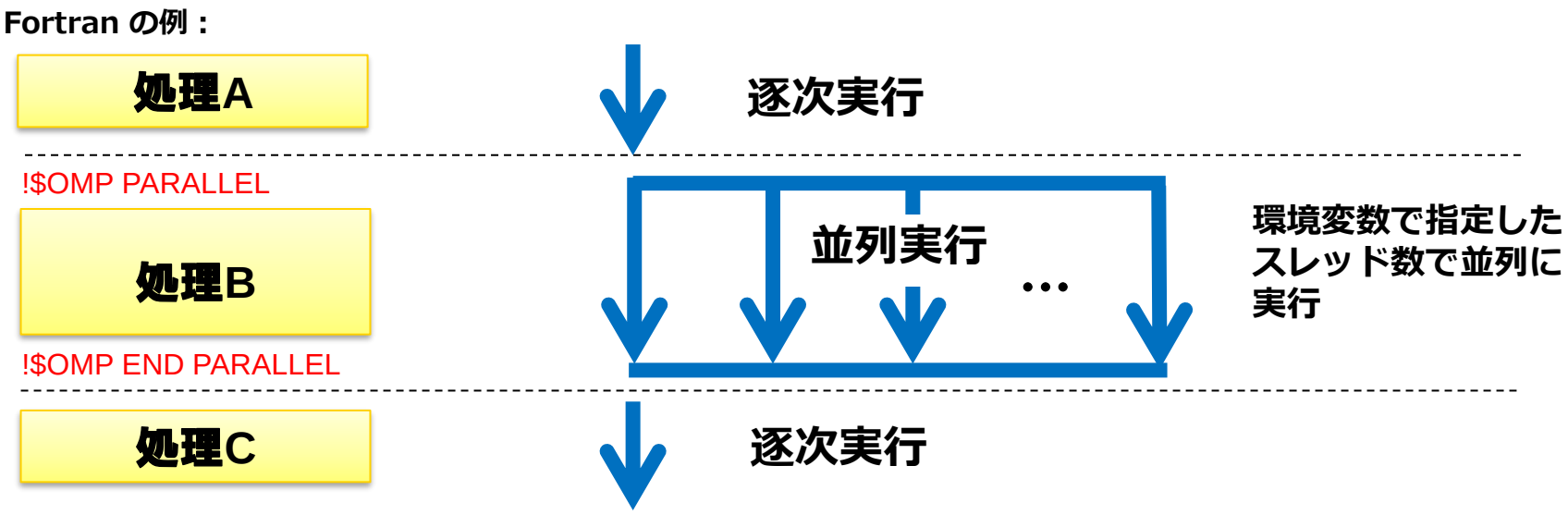
OpenMPとは

- 共有メモリマシン環境における並列化方法の１つ.
- 指示行によって並列化可能な場所を指示.
- プログラム実行時に，並列数を指定して実行.
- Fortranおよび，C/C++言語で使用可能.



OpenMP並列の基本構造

- OpenMP並列指示文で指定された処理を複数のスレッドで実行する.
- スレッド数は実行時の環境変数で指定する.



2. OpenMP指示文(指示行)の書き方

- Fortranプログラムの場合

「!$OMP [指示文]」を1カラム目から指定する．DO文を並列化する場合はDO文の前に「parallel do」を指示し，ENDDO文の後ろに「end parallel do」を指示する.

（例）
```
!$OMP parallel do
      do k=1,nz
        do j=1,ny
          :
        enddo
      enddo
!$OMP end parallel do
```

- Cプログラムの場合

「#pragma omp [指示文]」で指定する．for文を並列化する場合は「parallel for」と指示する.

（例）
```
#pragma omp parallel for
for(i=0;i<100;i++){
      :
}
```

3. OpenMP化の例（総和計算プログラム：Fortran）

- 1から1000の総和を求める(逐次実行プログラム)

```
program sample1
parameter(n=1000)
integer sum
sum=0
do i=1,n
  sum=sum+i
enddo
print *,"Total = ",sum
stop
end
```

- 並列化の対象はDOループ

✓ 1から1000の総和を計算するループが対象.

✓ ループ内の変数sumは総和演算を行っており，**各スレッドが個々に変数sumの内容を更新すると正しい結果が得られない**.

3. OpenMP化の例（総和計算プログラム：C/C++）

- 1から1000の総和を求める(逐次実行プログラム)

```
#include <stdio.h>
int main()
{
  int n=1000;
  int sum;
  sum=0;
  for(int i=1 ; i<=n ; ++i){
    sum+=i;
  }
  printf("Total = %d¥n",sum);
  return 0;
}
```

- 並列化の対象はforループ

✓ 1から1000の総和を計算するループが対象.

✓ ループ内の変数sumは総和演算を行っており，**各スレッドが個々に変数sumの内容を更新すると正しい結果が得られない**.

3. OpenMP化の例（総和計算プログラム：Fortran）

- DOループの並列化は parallel do ～ end parallel do指示文でスレッド並列化を指示

```
!$OMP parallel do
      do i=1,n
        sum=sum+i
      enddo
!$OMP end parallel do
```

- !$OMP parallel do～!$OMP end parallel doで囲んだDOループはスレッド数で分割され並列に実行される
- この例では，8スレッドで実行する場合，ループは1～125，126～250，251～375，376～500，501～625，626～750，751～875，876～1000とスレッドに割り当てられる

スレッド0

```
do i=1,125
  sum=sum+i
enddo
```

スレッド1

```
do i=126,250
  sum=sum+i
enddo
```

スレッド2

```
do i=251,625
  sum=sum+i
enddo
```

…

スレッド7

```
do i=876,1000
  sum=sum+i
enddo
```

3. OpenMP化の例（総和計算プログラム：C/C++）

- forループの並列化は parallel for指示文でスレッド並列化を指示

```
#pragma omp parallel for
  for(int i=1 ; i<=n ; ++i){
    sum+=i;
  }
```

- #pragma omp parallel forで指示したforループはスレッド数で分割され並列に実行される
- この例では，8スレッドで実行する場合，ループは1～125，126～250，251～375，376～500，501～625，626～750，751～875，876～1000とスレッドに割り当てられる

スレッド０　　スレッド１　　スレッド２　　…　　スレッド7

```
for(int i=1 ; i<=125 ; ++i){
  sum+=i;
}
```

```
for(int i=126 ; i<=250 ; ++i){
  sum+=i;
}
```

```
for(int i=251 ; i<=625 ; ++i){
  sum+=i;
  }
```

```
for(int i=876 ; i<=1000 ; ++i){
  sum+=i;
}
```

3. OpenMP化の例（総和計算プログラム）

複数のスレッドが同時に同じ変数に書き込む可能性がある場合には注意が必要

- 各スレッドの計算結果は変数sumに格納するが，**全スレッドが同じ領域の値を更新すると正しい結果は得られないため，特別な指定が必要**.
- スレッドごとに部分和を格納する領域を用意し，各スレッドの計算結果の部分和を変数sumに足し込む必要がある.
- このような演算を「リダクション演算」と呼ぶ.
- 変数sumに対する演算はリダクション演算であることを明示する.

Fortranの例：

```
!$OMP parallel do reduction(+:sum)
      do i=1,n
        sum=sum+i
      enddo
!$OMP end parallel do
```

C/C++の例：

```
#pragma omp parallel for reduction(+:sum)
 for(int i=1 ; i<=n ; ++i){
   sum+=i;
 }
```

- reduction(オペレータ:変数名リスト) オプションを記述する．オペレータの種類として和(+)，差(-)，積(＊)のほかに最大・最小値を求めるMAX，MINやビット操作を行うIAND，IOR，IEOR，論理演算のAND.，.OR.，.EQV.，.NEQV.を使用することができる.
- 複数の変数を指定する場合は，変数名リストをカンマ(,)で区切って記述する.

4. OpenMPの記述法

スレッドごとに独立した変数が必要になる場合には注意が必要

- PRIVATE指定

スレッドごとに独立した変数が必要な場合には，**PRIVATE句**が必要である．

①ループ中で作業配列・変数として使用されている

②並列化の対象となるループ変数の次元を持たない配列

Fortranの例：

```
do k=1,nz
  do j=1,ny
    do i=1,nx
      a(i,j) = b(i,j,k)*c(i,k)
      d(i,j,k) = d(i,j,k) + a(i,j)*e(j,k)
                :
```

```
!$OMP parallel do private(a)
      do k=1,nz
        do j=1,ny
          do i=1,nx
                :
```

C/C++の例：

```
for(int k=0 ; k<nz ; ++k){
  for(int j=0 ; j<ny ; ++j){
    for(int i=0 ; i<nx ; ++i){
      a[j][i] = b[k][j][i]*c[k][i];
      d[k][j][i] = d[k][j][i] + a[j][i]*e[k][j];
                :
```

```
#pragma omp parallel for private(a)
 for(int k=0 ; k<nz ; ++k){
   for(int j=0 ; j<ny ; ++j){
     for(int i=0 ; i<nx ; ++i){
                :
```

- 配列aはi,jの次元しか持たず，ループ中で定義・参照されている．
- このような配列はスレッドごとに領域を確保しなければ，正しい結果を得ることができない．
- 配列aに対して指示文でprivate指定を行うことにより，配列aはスレッドごとに独立した領域に確保される．
- private(a,b,c)のように複数の配列，変数を指定することが可能．

4-1. OpenMPの記述法（Fortran）

- do～end do指示文(parallel～end parallel)

✓ 連続する複数のループが並列化の対象となる場合，parallel do～end parallel do指示文を複数書くのではなく，do～end do指示文を組合せる.

```
!$OMP parallel
!$OMP do
    並列化対象のループ
!$OMP end do
!$OMP do
    並列化対象のループ
!$OMP end do
!$OMP end parallel
```

✓ parallel～end parallelの範囲を広げることにより，OpenMPによるオーバーヘッドを軽減することが可能になる.

4-1. OpenMPの記述法（Fortran）

- 実行時の総スレッド数や自スレッド番号は関数呼出しで取得可能.
  - **omp_get_num_threads( ):**総スレッド数の取得
  - **omp_get_thread_num( ) :**自スレッド番号の取得
- スレッド番号で処理を制御する際に使用.
- OpenMP実行時ルーチンを使用する場合は，moduleのomp_libを使用することを宣言する.

```
      program sample2
      use omp_lib
!$OMP parallel
      print *,"Hello World",omp_get_num_threads(),omp_get_thread_num()
!$OMP end parallel
      stop
      end
```

➢総スレッド数と自スレッド番号を取得しprintする

```
# setenv OMP_NUM_THREADS 4
# ./a.out
 Hello World        4        0
 Hello World        4        1
 Hello World        4        2
 Hello World        4        3
```

4-2. OpenMPの記述法（C/C++）

- for指示文(parallel)

✓ 連続する複数のループが並列化の対象となる場合，parallel for指示文を複数書くのではなく，for指示文を組合せる.

```
#pragma omp parallel
#pragma omp for
    並列化対象のループ

#pragma omp for
    並列化対象のループ
```

✓ parallelの範囲を広げることにより，OpenMPによるオーバーヘッドを軽減することが可能になる.

4-2. OpenMPの記述法（C/C++）

- 実行時の総スレッド数や自スレッド番号は関数呼出しで取得可能.
  - **omp_get_num_threads( ):**総スレッド数の取得
  - **omp_get_thread_num( ) :**自スレッド番号の取得
- スレッド番号で処理を制御する際に使用.
- OpenMP実行時関数を使用する場合は，ヘッダのomp.hを使用することを宣言する.

```
#include <stdio.h>
#include <omp.h>
int main()
{
#pragma omp parallel
  printf("Hello World %d %d¥n",omp_get_num_threads(),omp_get_thread_num());
  return 0;
}
```

➢総スレッド数と自スレッド番号を取得しprintする

```
# setenv OMP_NUM_THREADS 4
# ./a.out
Hello World 4 0
Hello World 4 3
Hello World 4 2
Hello World 4 1
```

5. コンパイル・実行スクリプト

- OpenMPプログラムをコンパイルするには，OpenMP専用のコンパイルオプションを指定してコンパイルする必要がある.
- 並列数は，実行時環境変数（OMP_NUM_THREADS）で指定する.
  - AOBA-Aの場合
    - コンパイル

Fortran :

```
nfort -fopenmp　[オプション] ソースファイル名
```

C/C++ :

```
ncc -fopenmp　[オプション] ソースファイル名　・・・Cプログラムのコンパイル
nc++ -fopenmp　[オプション] ソースファイル名　・・・C++プログラムのコンパイル
```

※OpenMPを利用する場合は-fopenmpオプションが必須
※自動並列化-mparallelとの併用が可能

5. コンパイル・実行スクリプト

- 実行スクリプト（AOBA-Aの場合）

実行スクリプトの例

```
#!/bin/sh
#PBS  -q ①
#PBS  --venode ②
#PBS  -l  elapstim_req=③
#PBS  -v  OMP_NUM_THREADS=④

cd  $PBS_O_WORKDIR
./a.out (実行文)
```

① AOBA-Aのキューを指定.
通常利用の場合は「sx」と指定．※占有利用の方は別途お知らせします．(必須)
② 使用VE数を指定．(必須)
③ 使用計算時間（経過時間）を設定.
「hh:mm:ss」のように指定.
例えば1時間半の場合には
#PBS -l elapstim_req=01:30:00
と指定．(設定を強く推奨)
④ スレッド数を指定（最大は8）．
(共有並列の場合必須)

5. コンパイル・実行スクリプト

AOBA-Bの場合

AOBA-BでもOpenMPの使用は可能．使い方はAOBA-Aと同じ．
フロントエンド(front.cc.tohoku.ac.jp)上で，以下のようにコンパイルする．

Fortran :

- AOCC**コンパイラ**
  flang –fopenmp [オプション] ソースファイル名
- GCC**コンパイラ**
  gfortran –fopenmp [オプション] ソースファイル
- Intel**コンパイラ**
  ifort -qopenmp [オプション] ソースファイル名

C/C++ :

- AOCC**コンパイラ**
  clang –fopenmp [オプション] ソースファイル名 ・・・Cプログラムのコンパイル
  clang++ –fopenmp [オプション] ソースファイル名 ・・・C++プログラムのコンパイル
- GCC**コンパイラ**
  gcc –fopenmp [オプション] ソースファイル ・・・Cプログラムのコンパイル
  g++ –fopenmp [オプション] ソースファイル ・・・C++プログラムのコンパイル
- Intel**コンパイラ**
  icc -qopenmp [オプション] ソースファイル名 ・・・Cプログラムのコンパイル
  icpc -qopenmp [オプション] ソースファイル名 ・・・C++プログラムのコンパイル

5. コンパイル・実行スクリプト

コンパイルオプションについては，以下を参照

AOBA-A 「AOBA-A 利用方法」
https://www.ss.cc.tohoku.ac.jp/sx-aurora/#toc4

「Fortran Compiler ユーザーズガイド」
https://www.hpc.nec/documentation

AOBA-B 「AOBA-B 利用方法」
https://www.ss.cc.tohoku.ac.jp/lx406rz-2/#toc4
「AMD Optimizing C/C++ Compiler」
https://developer.amd.com/amd-aocc/

5. コンパイル・実行スクリプト

- AOBA-Aのジョブクラス

利用形態	キュー名	VE数	実行形態(※)	最大経過時間 既定値/最大値	メモリサイズ
無料	sxf	1	1VE	1時間/1時間	48GB×VE数
共有	sx	1	1VE	72時間/720時間	
		2～256	8VE単位で確保(VHを共用しない)		
	sxmix	2～8	1VE単位で確保(VHを共用する)		

※OpenMPのみの並列では複数VEは利用できません.
※MPIとのハイブリッド実行については後期の講習会にて詳しく説明します.

- AOBA-Bのジョブクラス

利用形態	キュー名	ノード数 (-bオプション)	最大経過時間 既定値/最大値	メモリサイズ
共有	lx	1～16	72時間/720時間	256GB×ノード数

※OpenMPのみの並列では複数ノードは利用できません.
※MPIとのハイブリッド実行については後期の講習会にて詳しく説明します.

6. 演習問題３　（practice_3)

姫野ベンチマークコードをOpenMPで並列化し，逐次実行（1core実行），自動並列実行と実行性能を比較する.

項目	対象	備考
作業ディレクトリ	practice_3	
使用ソースファイル	sample3.f	編集　必要
ジョブファイル	run.sh	そのまま投入

- 手順①：作業ディレクトリを移動してください.

  % cd parallel/practice_3

- 手順②：エディタでソースファイルを編集し，OpenMP並列化を実施してください.

6. 演習問題３　（practice_3）

- ヒント：サブルーチンjacobiの以下のループで202行目のループは並列化不可
  (ただし、204行目、224行目からの3重ループは並列化可能)

```
202: +------>          DO loop=1,nn
203: |                    gosa=0.0
204: |+----->             DO K=2,kmax-1
205: ||+---->                DO J=2,jmax-1
206: |||+--->                   DO I=2,imax-1
207: ||||                         S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
208: ||||               1                +a(I,J,K,3)*p(I,J,K+1)
209: ||||               2                +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
210: ||||               3                -p(I-1,J+1,K)+p(I-1,J-1,K))
211: ||||               4                +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
212: ||||               5                -p(I,J+1,K-1)+p(I,J-1,K-1))
213: ||||               6                +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
214: ||||               7                -p(I+1,J,K-1)+p(I-1,J,K-1))
215: ||||               8                +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
216: ||||               9                +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
217: ||||                         SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
218: ||||                         GOSA=GOSA+SS*SS
219: ||||                         wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
220: |||+---                   enddo
221: ||+----                 enddo
222: |+-----              enddo
223: |           C
224: |+----->             DO K=2,kmax-1
225: ||+---->                DO J=2,jmax-1
226: |||+--->                   DO I=2,imax-1
227: ||||                         p(I,J,K)=wrk2(I,J,K)
228: |||+---                   enddo
229: ||+----                 enddo
230: |+-----              enddo
231: |           C
232: +------          enddo
```

S0
（エスゼロ）

6. 演習問題３　（practice_3）

- 手順③：コンパイルします.

  % nfort -fopenmp sample3.f

- 手順④：ジョブを投入します.

  % qsub run.sh

- 手順⑤：結果を確認します．結果はp3-practice.o.XXXX(XXXXにはジョブIDが入ります)として格納されます.

  % cat p3-practice.o.XXXX

　ジョブファイル（run.sh）の OMP_NUM_THREADS で設定した数値が，実行時の並列数となります.
　OMP_NUM_THREADSの数値を1～8（AOBA-Aは8コア）に変更することで，実行時間が変化することを確認してください.
　また，OpenMPの実行結果と，演習問題1，演習問題2の実行結果を比較してください.

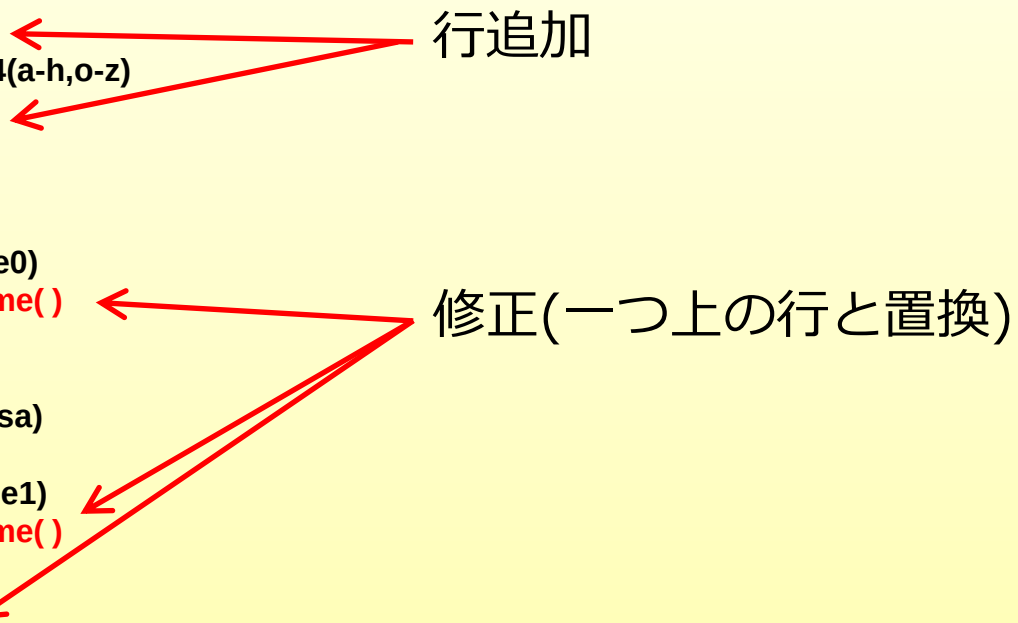
6．演習問題３（practice_3） つづき

- 時間計測は以下のように修正ずみ.

```
38  C -------------------
39  C "use portlib" statement on the next line is for Visual fortran
40  C to use UNIX libraries. Please remove it if your system is UNIX.
41  C -------------------
42  !     use portlib
43        use omp_lib
44        IMPLICIT REAL*4(a-h,o-z)
45        real*8 t1,t2

          (省略)

81  !      cpu0=dtime(time0)
82        t1=omp_get_wtime( )
83  C
84  C Jacobi iteration
85        call jacobi(nn,gosa)
86  C
87  !      cpu1= dtime(time1)
88        t2=omp_get_wtime( )
89  !      cpu = cpu1
90        cpu = t2-t1
91        flop=real(kmax-2)*real(jmax-2)*real(imax-2)*34.0*real(nn)
```



6．演習問題３（practice_3） つづき

◆ サブルーチンinitmtの以下の [　　] の部分を埋めてくだい。

```
148 [　　]
149 [　　]
150       do k=1,mkmax
151         do j=1,mjmax
152           do i=1,mimax
153             a(i,j,k,1)=0.0
154             a(i,j,k,2)=0.0
155             a(i,j,k,3)=0.0
156             a(i,j,k,4)=0.0
157             b(i,j,k,1)=0.0
158             b(i,j,k,2)=0.0
159             b(i,j,k,3)=0.0
160             c(i,j,k,1)=0.0
161             c(i,j,k,2)=0.0
162             c(i,j,k,3)=0.0
163             p(i,j,k) =0.0
164             wrk1(i,j,k)=0.0
165             bnd(i,j,k)=0.0
166           enddo
167         enddo
168       enddo
169 [　　]
```

①　③

```
171 [　　]
172       do k=1,kmax
173         do j=1,jmax
174           do i=1,imax
175             a(i,j,k,1)=1.0
176             a(i,j,k,2)=1.0
177             a(i,j,k,3)=1.0
178             a(i,j,k,4)=1.0/6.0
179             b(i,j,k,1)=0.0
180             b(i,j,k,2)=0.0
181             b(i,j,k,3)=0.0
182             c(i,j,k,1)=1.0
183             c(i,j,k,2)=1.0
184             c(i,j,k,3)=1.0
185             p(i,j,k) =float((k-1)*(k-1))/float((kmax-1)*(kmax-1))
186             wrk1(i,j,k)=0.0
187             bnd(i,j,k)=1.0
188           enddo
189         enddo
190       enddo
191 [　　]
192 [　　]
```

②　③

① 150行目から168行目までのループが並列化可能か検討します．
② 172行目から190行目までのループが並列化可能か検討します．
③ ①および②ともに並列化可能であれば，2つの連続するループが並列化の対象となるため，OpenMP並列化のオーバーヘッドを軽減するよう， parallel～end parallel 指示文およびdo～end do指示文を組み合わせて記述します．

6．演習問題３（practice_3）　つづき

◆ サブルーチン jacobi の以下の ［　　　］ の部分を埋めてください．

```
218       DO loop=1,nn
219         gosa=0.0
220  [          ]
221  [      ] [   ] [    ]
222         DO K=2,kmax-1
223           DO J=2,jmax-1
224             DO I=2,imax-1
225               S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
226      1             +a(I,J,K,3)*p(I,J,K+1)
227      2             +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
228      3             -p(I-1,J+1,K)+p(I-1,J-1,K))
229      4             +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
230      5             -p(I,J+1,K-1)+p(I,J-1,K-1))
231      6             +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
232      7             -p(I+1,J,K-1)+p(I-1,J,K-1))
233      8             +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
234      9             +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
235               SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
236               GOSA=GOSA+SS*SS
237               wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
238             enddo
239           enddo
240         enddo
241  [          ]
```

Bracket ① spans lines 221–240; bracket ③ spans lines 220–241.

```
243  [          ]
244         DO K=2,kmax-1
245           DO J=2,jmax-1
246             DO I=2,imax-1
247               p(I,J,K)=wrk2(I,J,K)
248             enddo
249           enddo
250         enddo
251  [          ]
252  [          ]
253  C
254       enddo
```

Bracket ② spans lines 243–251; bracket ③ spans lines 243–252.

① 222行目から240行目までのループが並列化可能か検討します．並列化に必要なプライベート指定とリダクション演算指定を検討します．

② 244行目から250行目までのループが並列化可能か検討します．

③ ①および②ともに並列化可能であれば，2つの連続するループが並列化の対象となるため，OpenMP並列化のオーバーヘッドを軽減するよう， parallel～end parallel 指示文およびdo～end do指示文を組み合わせて記述します．

7. 演習問題４　（practice_4)

姫野ベンチマークコードをOpenMPで並列化し，AOBA-Bで実行し，AOBA-Aと比較する.

項目	対象	備考
作業ディレクトリ	practice_4	
使用ソースファイル	sample3.f	演習問題3の結果を使用
ジョブファイル	run.sh	そのまま投入

- 手順①：作業ディレクトリを移動してください.

  % cd parallel/practice_4

- 手順②：ソースファイルをコピーします.

  % cp ../practice_3/sample3.f .

- 手順③：コンパイルします.

  % flang –fopenmp sample3.f

7. 演習問題４　（practice_4）

- 手順③：ジョブを投入します.

  % qsub run.sh

- 手順④：結果を確認します．結果はp4-practice.o.XXXX(XXXXにはジョブIDが入ります)として格納されます.

  % cat p4-practice.o.XXXX

　ジョブファイル（run.sh）の OMP_NUM_THREADS で設定した数値が，実行時の並列数となります.
　OMP_NUM_THREADSの数値を8以上（AOBA-Bは最大128コア）に変更することで，実行時間が変化することを確認してください.
　また，OpenMPの実行結果と，演習問題3の実行結果を比較してください.

8. 参考(参考文献)

- OpenMPの仕様はhttp://www.openmp.org/で公開されている.
- 現在日本語で出版されている参考書籍は「OpenMPによる並列プログラミングと数値計算法」(牛島省著,丸善株式会社)や「OpenMP並列プログラミング」(菅原清文著，株式会社カットシステム)などがある.

演習問題解答例

演習問題3　回答例

```
C***********************************************************************
C
 （省略）
C --------------------
!     use portlib
      use omp_lib
      IMPLICIT REAL(4)(a-h,o-z)
      real(8) :: t1,t2
C
C      PARAMETER (mimax=513,mjmax=257,mkmax=257)
      PARAMETER (mimax=257,mjmax=129,mkmax=129)
C      PARAMETER (mimax=129,mjmax=65,mkmax=65)
C      PARAMETER (mimax=65,mjmax=33,mkmax=33)
C
C     ttarget specifys the measuring period in sec
        PARAMETER (ttarget=60.0)
CC Arrey
      common /pres/ p(mimax,mjmax,mkmax)
      common /mtrx/ a(mimax,mjmax,mkmax,4),
     +     b(mimax,mjmax,mkmax,3),c(mimax,mjmax,mkmax,3)
      common /bound/ bnd(mimax,mjmax,mkmax)
      common /work/ wrk1(mimax,mjmax,mkmax),wrk2(mimax,mjmax,mkmax)
CC Other constants
      common /others/ imax,jmax,kmax,omega
C
!     dimension time0(2),time1(2)

!     integer ts, te, tr
```

演習問題3　回答例

```
C
      omega=0.8
      imax=mimax-1
      jmax=mjmax-1
      kmax=mkmax-1
CC Initializing matrixes
      call initmt
      write(*,*) ' mimax=',mimax,' mjmax=',mjmax,' mkmax=',mkmax
      write(*,*) ' imax=',imax,' jmax=',jmax,' kmax=',kmax
CC Start measuring
C
      nn=10000
        write(*,*) ' Start rehearsal measurement process.'
        write(*,*) ' Measure the performance in 10000 times.'
C
!     cpu0=dtime(time0)
      t1=omp_get_wtime()
C
C Jacobi iteration
      call jacobi(nn,gosa)
C
!     cpu1= dtime(time1)
      t2=omp_get_wtime()
!     cpu = cpu1
      cpu=t2-t1
      flop=real(kmax-2)*real(jmax-2)*real(imax-2)*34.0*real(nn)
      xmflops2=flop/cpu*1.0e-6
        write(*,*) '  MFLOPS:',xmflops2
!       write(*,'(a10,f10.3)') '  time(s):',(te-ts)/dble(tr)
        write(*,'(a10,f10.3)') '  time(s):',cpu
        write(*,*) '    gosa:',gosa
```

演習問題3　回答例

```
C
C     end the test loop
!     nn=ifix(ttarget/(cpu/3.0))
!     write(*,*) 'Now, start the actual measurement process.'
!       write(*,*) 'The loop will be excuted in',nn,' times.'
!       write(*,*) 'This will take about one minute.'
!       write(*,*) 'Wait for a while.'
C
C Jacobi iteration
!     cpu0=dtime(time0)
!     call jacobi(nn,gosa)
C
!     cpu1= dtime(time1)
!     cpu = cpu1
!     flop=real(kmax-2)*real(jmax-2)*real(imax-2)*34.0*real(nn)
!     xmflops2=flop*1.0e-6/cpu
C
CCC      xmflops2=nflop/cpu*1.0e-6*float(nn)
C
!     write(*,*) ' Loop executed for ',nn,' times'
!     write(*,*) ' Gosa :',gosa
!     write(*,*) ' MFLOPS:',xmflops2, '   time(s):',cpu
!     score=xmflops2/82.84
!     write(*,*) ' Score based on Pentium III 600MHz :',score
C
!     pause
      stop
      END
```

演習問題3　回答例

```
C
C
C*********************************************************************
      subroutine initmt
C*********************************************************************
      IMPLICIT REAL(4) (a-h,o-z)
C
C       PARAMETER (mimax=513,mjmax=257,mkmax=257)
       PARAMETER (mimax=257,mjmax=129,mkmax=129)
C       PARAMETER (mimax=129,mjmax=65,mkmax=65)
C       PARAMETER (mimax=65,mjmax=33,mkmax=33)
C


CC Arrey
      common /pres/ p(mimax,mjmax,mkmax)
      common /mtrx/ a(mimax,mjmax,mkmax,4),
     +      b(mimax,mjmax,mkmax,3),c(mimax,mjmax,mkmax,3)
      common /bound/ bnd(mimax,mjmax,mkmax)
      common /work/ wrk1(mimax,mjmax,mkmax),wrk2(mimax,mjmax,mkmax)
CC other constants
      common /others/ imax,jmax,kmax,omega
C
!$OMP parallel
!$OMP do
      do k=1,mkmax
         do j=1,mjmax
            do i=1,mimax
               a(i,j,k,1)=0.0
               a(i,j,k,2)=0.0
```

演習問題3　回答例

```
                a(i,j,k,3)=0.0
                a(i,j,k,4)=0.0
                b(i,j,k,1)=0.0
                b(i,j,k,2)=0.0
                b(i,j,k,3)=0.0
                c(i,j,k,1)=0.0
                c(i,j,k,2)=0.0
                c(i,j,k,3)=0.0
                p(i,j,k) =0.0
                wrk1(i,j,k)=0.0
                bnd(i,j,k)=0.0
              enddo
            enddo
          enddo
!$OMP end do
C
!$OMP do
          do k=1,kmax
            do j=1,jmax
              do i=1,imax
                a(i,j,k,1)=1.0
                a(i,j,k,2)=1.0
                a(i,j,k,3)=1.0
                a(i,j,k,4)=1.0/6.0
                b(i,j,k,1)=0.0
                b(i,j,k,2)=0.0
                b(i,j,k,3)=0.0
                c(i,j,k,1)=1.0
                c(i,j,k,2)=1.0
```

演習問題3　回答例

```
                c(i,j,k,3)=1.0
                p(i,j,k) =float((k-1)*(k-1))/float((kmax-1)*(kmax-1))
                wrk1(i,j,k)=0.0
                bnd(i,j,k)=1.0
              enddo
            enddo
        enddo
!$OMP end do
!$OMP end parallel
C
        return
        end
C
C**********************************************************************
        subroutine jacobi(nn,gosa)
C**********************************************************************
        IMPLICIT REAL(4)(a-h,o-z)
C
C       PARAMETER (mimax=513,mjmax=257,mkmax=257)
        PARAMETER (mimax=257,mjmax=129,mkmax=129)
C       PARAMETER (mimax=129,mjmax=65,mkmax=65)
C       PARAMETER (mimax=65,mjmax=33,mkmax=33)
C

CC Arrey
        common /pres/ p(mimax,mjmax,mkmax)
        common /mtrx/ a(mimax,mjmax,mkmax,4),
       +      b(mimax,mjmax,mkmax,3),c(mimax,mjmax,mkmax,3)
        common /bound/ bnd(mimax,mjmax,mkmax)
        common /work/ wrk1(mimax,mjmax,mkmax),wrk2(mimax,mjmax,mkmax)
```

演習問題3　回答例

```
CC other constants
      common /others/ imax,jmax,kmax,omega
C
C
      DO loop=1,nn
         gosa=0.0
!$OMP parallel
!$OMP do private(SO,SS) reduction(+:GOSA)
         DO K=2,kmax-1
            DO J=2,jmax-1
               DO I=2,imax-1
                  SO=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
                  SS=(SO*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
                  GOSA=GOSA+SS*SS
                  wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
               enddo
            enddo
         enddo
!$OMP end do
```

演習問題3　回答例

```
C
!$OMP do
          DO K=2,kmax-1
             DO J=2,jmax-1
                DO I=2,imax-1
                   p(I,J,K)=wrk2(I,J,K)
                enddo
             enddo
          enddo
!$OMP end do
!$OMP end parallel
C
       enddo
CC End of iteration
       return
       end
```