2022 年度
並列プログラミング入門 I
（OpenMP）

2022 年　10 月　27 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
- OpenMPプログラミング編
- MPIプログラミング編

OpenMPプログラミング編

OpenMPプログラミング編・目次

演習問題の構成

- 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。

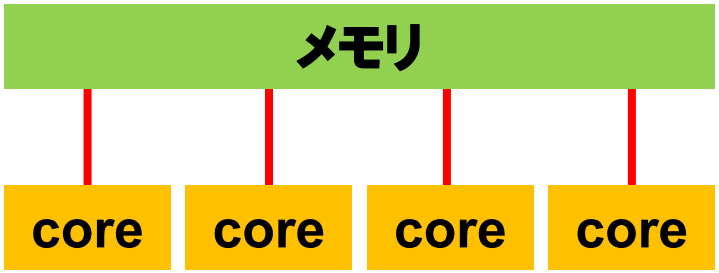
/mnt/stfs/ap/lecture/OpenMP/

|-- 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
|-- practice_5　演習問題5
|-- sample

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

1．OpenMP概要

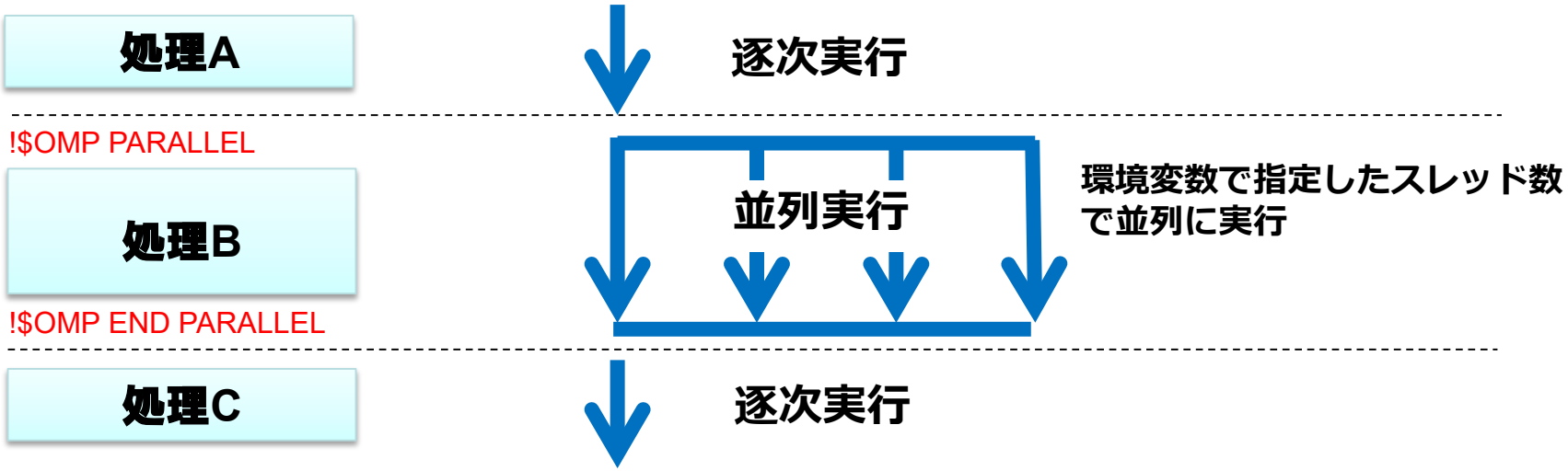
- 複数のコアを利用する並列処理により実行時間(経過時間)の短縮を図る.
- 共有メモリマシン環境においてスレッド並列を行うためのコンパイラに対する指示文(指示行)，関数やサブルーチン，実行時の環境変数等に関する仕様.
- FortranやC言語で使用可能.
- OpenMPをサポートしないコンパイラ，あるいはOpenMPによるスレッド並列を使用しない場合は，指示文はコメント行とみなす.
- OpenMP Architecture Review Board（ARB）によって規定されている業界標準規格であり，ポータビリティに優れている.



SMP(Symmetric Multiprocessing)

OpenMP並列の基本構造

- !$OMP parallel と !$OMP end parallel指示文に囲まれた範囲の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.



OpenMPプログラム例（Hello World)

逐次プログラム sample1.f

```
program sample1
print *,"Hello World"
stop
end
```

OpenMPプログラム sample2.f

```
program sample2
!$OMP parallel
print *,"Hello World“
!$OMP end parallel
stop
end
```

- 以下の処理を複数のスレッドで並列に実行することを指示
- 複数のスレッドで並列実行することを終了することを指示

コンパイル・実行スクリプト (1/2)

- OpenMPプログラムのコンパイル(Fortran)

nfort -fopenmp　[オプション]　ソースファイル名

※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

※オプションは「Fortran Compiler ユーザーズガイド」を参照

https://www.hpc.nec/documentation

- OpenMPプログラムの実行のスクリプト例（SX-Aurora TSUBASA）

```
#!/bin/csh
#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 SX-Aurora TSUBASA用のキューを指定．
通常は「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）

コンパイル・実行スクリプト (2/2)

- SX-Aurora TSUBASAのジョブクラス

利用形態	サブシステム	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	AOBA-A	sxf	1	1VE	1時間/1時間	48GB×VE数
共有	AOBA-A	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)		
	AOBA-C	sxc	1	1VE		
			2～512	8VE単位で確保 (VHを共用しない)		

・OpenMPのみの並列ではSX-Aurora TSUBASAの複数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MPIプログラミング編参照

※実行形態について
8VE単位で確保：8VE単位で計算資源が確保されるため，他のリクエストとVHを共用しないで実行されるため，演算時間のばらつきが少ない.
1VE単位で確保：1VE単位で計算資源が確保されるため，他のリクエストとVHを共用して実行されるため，VEの確保がしやく，待ち時間が短くなる.
1VE：1VE単位で計算資源が確保される.

2．演習問題１　（practice_1）

テキストP9のsample2.fをコンパイルして，実行してください.

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	sample2.f	編集　不要

- 手順①：作業ディレクトリを移動してください.

  % cd OpenMP/practice_1

- 手順②：コンパイルしてください.

  % nfort -fopenmp sample2.f

- 手順③：プログラムを実行してください.

  % qsub run.sh

- 手順④：結果を確認します．結果はp1-practice.oXXXX(XXXXにはリクエストIDが入ります)として格納されます.

  % cat p1-practice.oXXXX

演習問題１　（practice_1)

以下のように結果が表示されます.

Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World

3．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++){
          :
```

総和計算プログラムのOpenMP化 (1/3)

- 1から1000の総和を求める(逐次実行プログラム)

sample3.f

```
program sample3
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
  - ✓ sample3.fでは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.

総和計算プログラムのOpenMP化 (2/3)

- 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，4スレッドで実行する場合，ループは1～250，251～500，501～750，751～1000とスレッドに割り当てられる

スレッド０

```
do i=1,250
  sum=sum+i
enddo
```

スレッド１

```
do i=251,500
  sum=sum+i
enddo
```

スレッド２

```
do i=501,750
  sum=sum+i
enddo
```

スレッド３

```
do i=751,1000
  sum=sum+i
enddo
```

総和計算プログラムのOpenMP化 (3/3)

- 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.
- 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.

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

- 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．演習問題２　（practice_2）

- 行列積を計算するプログラムのOpenMP化

```
      program sample5
      implicit real(8)(a-h,o-z)
      parameter ( n=3840 )
      real(8) a(n,n),b(n,n),c(n,n),t1
      double precision cp1,cp2
      a=0.0d0
      call random_number(b)
      call random_number(c)
      write(6,50) ' Matrix Size = ',n
   50 format(1x,a,i5)
      call ETIME(cp1)

      do j=1,n
        do k=1,n
          do i=1,n
            a(i,j)=a(i,j)+b(i,k)*c(k,j)
          end do
        end do
      end do

      call ETIME(cp2)
      t1=cp2-cp1
      write(6,60) ' Execution Time = ',t1,' sec',' A(n,n) = ',a(n,n)
   60 format(1x,a,f10.3,a,1x,a,d24.15)
      stop
      end
```

- コストが集中しているはこの部分
- OpenMP指示文を挿入し，ループの並列実行を行う．
- 左の　　にOpenMP指示文を入れてください。

４．演習問題２　（practice_2）

行列積を計算するプログラムのOpenMP化

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	sample5.f	編集　必要
コンパイルスクリプト	comp.sh	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 OpenMP/practice_2

- 手順②：エディタでファイルを編集し，並列化を実施してください.

４．演習問題２　（practice_2)

- 手順③：コンパイルを実行します.

  % ./comp.sh

- 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p2-practice.oXXXX(XXXXにはリクエストIDが入ります)として格納されます.

  % cat p2-practice.oXXXX

5．OpenMPの記述法

- PRIVATE指定

  並列化対象のループ中だけで使用される配列や変数はスレッドごとに領域を確保しなければならないため，PRIVATE宣言が必要である．

  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

（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
                :
```

- 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!$OMP parallel do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．

5．OpenMPの記述法

- 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.

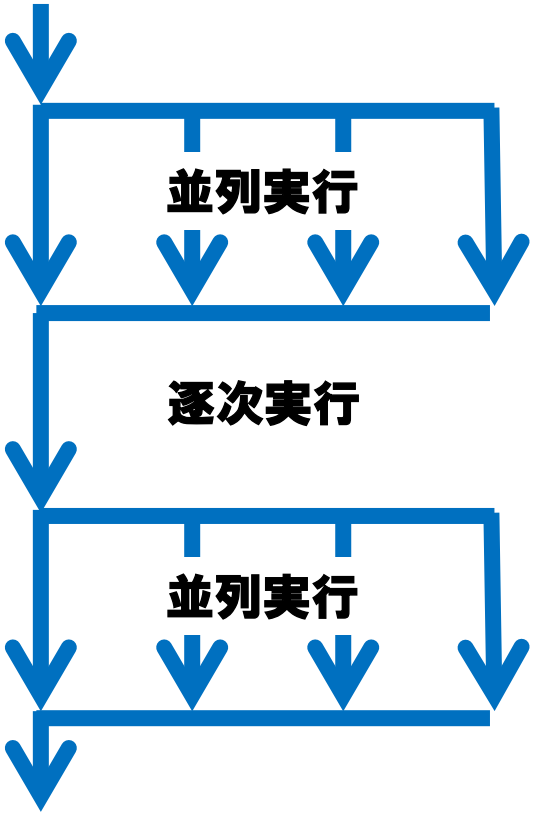
```
!$OMP parallel
!$OMP do
    並列化対象のループ
!$OMP end do
!$OMP do
    並列化対象のループ
!$OMP end do
!$OMP end parallel
```

  - ✓ 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.

5．OpenMPの記述法

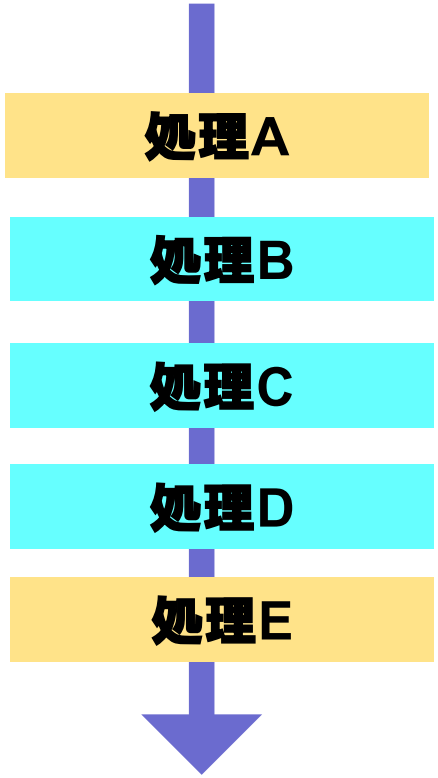
- single～end single指示文(parallel～end parallel)
  - ✓ parallel～end parallel指示文中に非並列で処理(単一のスレッドが実行)する部分がある場合，single～end single 指示文を指定する.

```
!$OMP parallel
!$OMP do
  並列化対象のループ
!$OMP end do
  !$OMP single
  非並列処理の部分
  !$OMP end single
!$OMP do
  並列化対象のループ
!$OMP end do
!$OMP end parallel
```



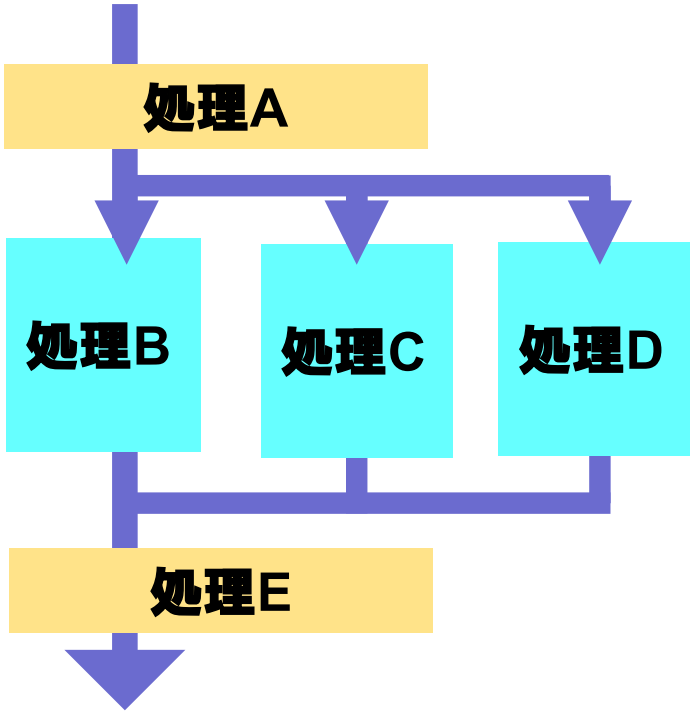
5．OpenMPの記述法

- sections～section～end sections
  - ✓ サブルーチン単位に並列性がある場合，sections～section～end sectionsを指定して，スレッドに分散して実行する．



- ➢ サブルーチンB,C,Dの実行順序に並列性(順序が入れ替わっても結果に差異が生じない)場合，以下のように指定することでスレッドに分散して実行する

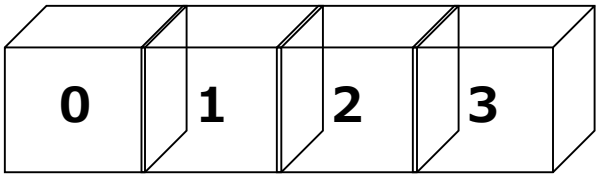
```
処理A
!$OMP parallel
!$OMP sections
処理B
!$OMP section
処理C
!$OMP section
処理D
!$OMP end sections
!$OMP end parallel
処理E
```



5．OpenMPの記述法

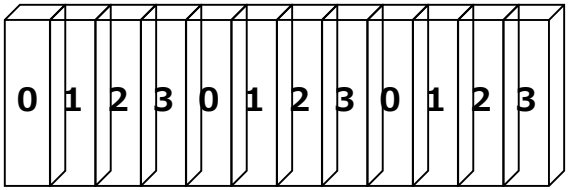
- DOループはなるべく均等にスレッドに割当てられる.
- しかしながらDOループ中の処理量に偏りが生じる場合，スレッドに割当てられる処理が均等でなくなるため，高い並列効果が得られない.
- scheduleオプションでスレッドへの割当て方法を変更する.

※何も指定しない，あるいはschedule(static)の場合



- 領域(並列化対象のループ長)をスレッド数で分割する
- ブロック分割と呼ばれる手法

※schedule(dynamic,1)の場合



- 領域(並列化対象のループ長)を帯状に細分し，巡回的にスレッドに動的に割当てる(終了したスレッド毎に割当てるので上記のように順番に割当てられるとは限らない)
- サイクリック分割と呼ばれる手法
- (dynamic,N)のNは帯幅(チャンクサイズ)を指定する

- チャンクサイズを小さくサイクリックに分割するとオーバーヘッドが大きくなるので，処理量の不均衡な状態を見ながら適切なチャンクサイズを指定する.

5．OpenMPの記述法

- scheduleオプションで指定可能な主な分割方法は以下の通り．

指定方法	動作
schedule(static)	領域(ループ長)をスレッド数で分割する．scheduleを指定しない場合もこの分割方法が採用される(規定値)．
schedule(dynamic,N)	最初にN分の反復回数が各スレッドに割当てられ，ラウンドロビン方式で各スレッドに割付ける．Nを省略した場合N=1となる．

- 指定方法は以下の通り．

```
!$OMP parallel do schedule(dynamic,1)
      :
!$OMP end parallel do
```

5．OpenMPの記述法

- 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
```

6．演習問題３

- 姫野ベンチマークコード(himenoBMTsp_s.f)をOpenMPで並列化する.
  - ✓ 姫野ベンチマーク：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計るもの(http://accc.riken.jp/supercom/himenobmt/).
  - ✓ 本演習では上記Webページからダウンロードしたソースコードの一部修正したものを用いる.

- 演習問題は穴埋め形式になっており，コードの一部（タイマー処理部分）についてはOpenMP用に修正済み※

※逐次版のタイマーはCPU時間を計測するもので，並列プログラムではCPU時間は短縮されないので，並列化の効果を検証できない.
（P36参照）

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．(テキストのP36参照)

```
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)

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

行追加

修正(一つ上の行と置換)

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さい。

```
143 [　　]
144 [　　]
145       do k=1,mkmax
146         do j=1,mjmax
147           do i=1,mimax
148             a(i,j,k,1)=0.0
149             a(i,j,k,2)=0.0
150             a(i,j,k,3)=0.0
151             a(i,j,k,4)=0.0
152             b(i,j,k,1)=0.0
153             b(i,j,k,2)=0.0
154             b(i,j,k,3)=0.0
155             c(i,j,k,1)=0.0
156             c(i,j,k,2)=0.0
157             c(i,j,k,3)=0.0
158             p(i,j,k) =0.0
159             wrk1(i,j,k)=0.0
160             bnd(i,j,k)=0.0
161           enddo
162         enddo
163       enddo
164 [　　]
```

145行目から163行目のループが並列化可能か検討します.

```
166 [　　]
167       do k=1,kmax
168         do j=1,jmax
169           do i=1,imax
170             a(i,j,k,1)=1.0
171             a(i,j,k,2)=1.0
172             a(i,j,k,3)=1.0
173             a(i,j,k,4)=1.0/6.0
174             b(i,j,k,1)=0.0
175             b(i,j,k,2)=0.0
176             b(i,j,k,3)=0.0
177             c(i,j,k,1)=1.0
178             c(i,j,k,2)=1.0
179             c(i,j,k,3)=1.0
180             p(i,j,k) =float((k-1)*(k-1))/float((kmax-1)*(kmax-1))
181             wrk1(i,j,k)=0.0
182             bnd(i,j,k)=1.0
183           enddo
184         enddo
185       enddo
186 [　　]
187 [　　]
```

167行目から185行目までのループも並列化可能であれば、2つのループ全体（143行目～187行目）でOpenMP指示行による並列化を行います.

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．

```
213      DO loop=1,nn
214        gosa=0.0
215
216
217        DO K=2,kmax-1
218          DO J=2,jmax-1
219            DO I=2,imax-1
220              S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
221     1            +a(I,J,K,3)*p(I,J,K+1)
222     2            +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
223     3            -p(I-1,J+1,K)+p(I-1,J-1,K))
224     4            +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
225     5            -p(I,J+1,K-1)+p(I,J-1,K-1))
226     6            +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
227     7            -p(I+1,J,K-1)+p(I-1,J,K-1))
228     8            +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
229     9            +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
230              SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
231              GOSA=GOSA+SS*SS
232              wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
233            enddo
234          enddo
235        enddo
236
```

- 217行目から235行目のDOループを並列化するためには、プライベート指定とリダクション演算が必要です．

```
238
239        DO K=2,kmax-1
240          DO J=2,jmax-1
241            DO I=2,imax-1
242              p(I,J,K)=wrk2(I,J,K)
243            enddo
244          enddo
245        enddo
246
247
248 C
249      enddo
```

- 217行目から245行目までのループ全体が並列化可能であれば、ループ全体でOpenMP指示行による並列化を行います．

6．演習問題３　（practice_3)

姫野ベンチマークコード(himenoBMTsp_s.f)をOpenMPで並列化.

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	sample6.f	編集　必要
オリジナルソースファイル	sample6.f.org	参考情報
コンパイルスクリプト	comp.sh	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 OpenMP/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） つづき

- 手順③：コンパイルを実行します.

  % ./comp.sh

- 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.oXXXX(XXXXにはジョブIDが入ります)として格納されます.

  % cat p3-practice.oXXXX

7. 演習問題4　（practice_4)

姫野ベンチマークコード(himenoBMTsp_s.f)をOpenMPで並列化.
(LXで実行)

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	sample6.f	編集　不要
コンパイルスクリプト	comp.sh	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 OpenMP/practice_4

- 手順②：演習問題3で使用したソースコードをコピーしてください.
  % cp ../practice_3/sample6.f .

7. 演習問題4（practice_4） つづき

- 手順③：コンパイルを実行します.

  % ./comp.sh

- 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p4-practice.oXXXX(XXXXにはジョブIDが入ります)として格納されます.

  % cat p4-practice.oXXXX

8. 演習問題5　（practice_5）

姫野ベンチマークコード(himenoBMTsp_s.f)をOpenMPで並列化.
(並列数を変えて実行)

項目	対象	備考
作業ディレクトリ	practice_5	
使用ソースファイル	演習3,4の環境を使用	編集　不要
コンパイルスクリプト	演習3,4の環境を使用	編集　不要
ジョブファイル	run.sh	修正　必要

- 手順①：作業ディレクトリを移動してください.

```
% cd OpenMP/practice_5
```

- 手順②：演習問題3, 演習問題4で使用した環境をコピーしてください.

```
% cp -r ../practice_3  .
% cp -r ../practice_4  .
```

8. 演習問題5（practice_5） つづき

- 手順③：実行コア数を修正します.

  % cd ./practice_3　もしくは　% cd ./practice_4

  % vi run.sh

  環境変数 OMP_NUM_THREADS の数値を変更します.

- 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.oXXXXもしくはp4-practice.oXXXX(XXXXにはジョブIDが入ります)として格納されます.

  % cat p3-practice.oXXXX もしくは p4-practice.oXXXX

9. 参考(時間計測)

- OpenMPでは時間計測の関数omp_get_wtimeを提供している.
- 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.
- 時間を格納する変数は倍精度整数で宣言する.

```
        use omp_lib
        real*8 t1,t2
        num=0
        t1=omp_get_wtime( )
!$OMP parallel do reduction(+:num)
        do i=1,1000
          num=num+i
        enddo
!$OMP end parallel do
        t2=omp_get_wtime( )
        print *,num,t2-t1
        stop
        end
```

```
$ setenv OMP_NUM_THREADS 8
$ ./a.out
  500500   2.7736998163163662E-05
```

9. 参考(LX 406Rz-2の利用方法)

- サイバーサイエンスセンターのLX 406Re-2でもOpenMPの使用は可能.
- OpenMPの使い方はSX-Aurora TSUBASAと同じ.
- フロントエンドサーバ上で，以下のようにコンパイルする.

•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

※オプションは「AOBA-B プログラム開発・実行環境　利用手順書」を参照
https://www.ss.cc.tohoku.ac.jp/sscc/wp-content/uploads/pdf/AOBA-B**プログラム開発・実行環境利用手順書**.pdf

- 並列コンピュータ LX 406Rz-2のジョブクラス

利用形態	キュー名	ノード数	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のノードを利用した実行はできません.

9. 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２　回答例

```
      program sample5
      implicit real(8)(a-h,o-z)
      parameter ( n=4096 )
      real(8) a(n,n),b(n,n),c(n,n)
      real(8) t1,t2
      double precision cp1,cp2
      a=0.0d0
      call random_number(b)
      call random_number(c)
      write(6,50) ' Matrix Size = ',n
 50 format(1x,a,i5)
      call ETIME(cp1)
!$OMP parallel do
      do j=1,n
        do k=1,n
          do i=1,n
            a(i,j)=a(i,j)+b(i,k)*c(k,j)
          end do
        end do
      end do
!$OMP end parallel do
      call ETIME(cp2)
      t1=cp2-cp1
      write(6,60) ' Execution Time = ',t1,' sec',' A(n,n) = ',a(n,n)
 60 format(1x,a,f10.3,a,1x,a,d24.15)
      stop
      end
```

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)
C       PARAMETER (mimax=257,mjmax=129,mkmax=129)
      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)
```

演習問題3　回答例　つづき

```
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,'  time(s):',cpu,gosa
```

演習問題3　回答例　つづき

```
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　つづき

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

```
                  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　つづき

```
                  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)
C           PARAMETER (mimax=257,mjmax=129,mkmax=129)
          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　つづき

```
CC other constants
      common /others/ imax,jmax,kmax,omega
C
C
      DO loop=1,nn
        gosa=0.0
!$OMP parallel
!$OMP do private(S0,SS) reduction(+:GOSA)
        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
!$OMP end do
```

演習問題3　回答例　つづき

```
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
```