OpenMP を用いた並列計算（2）

谷口 隆晴

システム情報学研究科　計算科学専攻

2011 年 6 月 2 日

出席の確認のため，端末を立ち上げて
scalar にログインしておいて下さい．

今日の内容

もう少し細かい並列化方法の指定法

- 宿題の解答例
- 演習１：ループでのスレッド割り当て方法の指定 (schedule)
- 演習２：各スレッドに異なる仕事を割り当てる方法 (omp sections)
- 単独のスレッドで実行 (omp single, omp master)
- 演習３・４：スレッドの同期と制御 (barrier, critical, atomic)
- 演習５：ベクトル・行列積のいろいろな並列化方法
- 宿題：２３進数覆面算

前回休んだ人のために：鍵の転送・パスワード変更

ターミナルを新しく起動し，以下のコマンドを実行

1. **公開鍵を転送**

```
% scp .ssh/authorized_keys 133.30.112.246:
```

2. 48 **コアマシンへログイン**

```
% ssh 133.30.112.246    # 今日の授業中のみパスワードログイン可
```

3. **公開鍵を配置**

```
% mkdir .ssh
% mv authorized_keys .ssh
```

4. **アクセス権を設定**

```
% chmod 600 .ssh/authorized_keys
```

5. **パスワードを変更**

```
% passwd
```

今後は scalar **と同様の方法（ただし** scalar.scitec.kobe-u.ac.jp **を** 133.30.112.246 **におきかえる）でログインできます．**

宿題の解答例

```
program pi
implicit none
integer, parameter :: SP = kind(1.0)
integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
integer, parameter :: n = 1000000
integer :: i
real(DP) :: x, dx, p
real(DP) :: time0, time1, omp_get_wtime

dx = 1.0_DP/real(n, DP)

p = 0.0_DP
time0 = omp_get_wtime()
!$omp parallel do private(i,x) shared(dx) reduction(+:p)
do i = 1,n
x = real(i, DP) * dx
p = p + 4.0_DP/(1.0_DP + x**2)*dx
end do
time1 = omp_get_wtime()

print *, p
print *, time1-time0

end program
```

多かった間違い：

i や x を共有変数に設定

➡ コンパイラが安全なコードを出力

➡ 並列化時の実行時間増大

ループでのスレッド割り当て方法の指定

例）三角行列とベクトルの積

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} \\ & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} \\ & & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} \\ & & & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ & & & & a_{55} & a_{56} & a_{57} & a_{58} \\ & & & & & a_{66} & a_{67} & a_{68} \\ & & & & & & a_{77} & a_{78} \\ & & & & & & & a_{88} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix}$$

スレッド０は青の部分を，スレッド１が緑の部分を担当．
素直に２つに分割：青の要素数＝２６個，緑の要素数＝１０個．

➡ 緑の部分よりも青の部分のほうが計算が大変！
スレッド０の計算に時間がかかってしまい，全体としても速くならない． ➡ なるべく負荷を均一にしたい

解決策の例）ブロックサイクリック分割

例）三角行列とベクトルの積

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} \\ & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} \\ & & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} \\ & & & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ & & & & a_{55} & a_{56} & a_{57} & a_{58} \\ & & & & & a_{66} & a_{67} & a_{68} \\ & & & & & & a_{77} & a_{78} \\ & & & & & & & a_{88} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix}$$

スレッド０は青の部分を，スレッド１が緑の部分を担当．
２行からなるブロックごとに分割：青の要素数＝２２個，緑の要素数＝１４個．

ちょっと改善．

OpenMP では

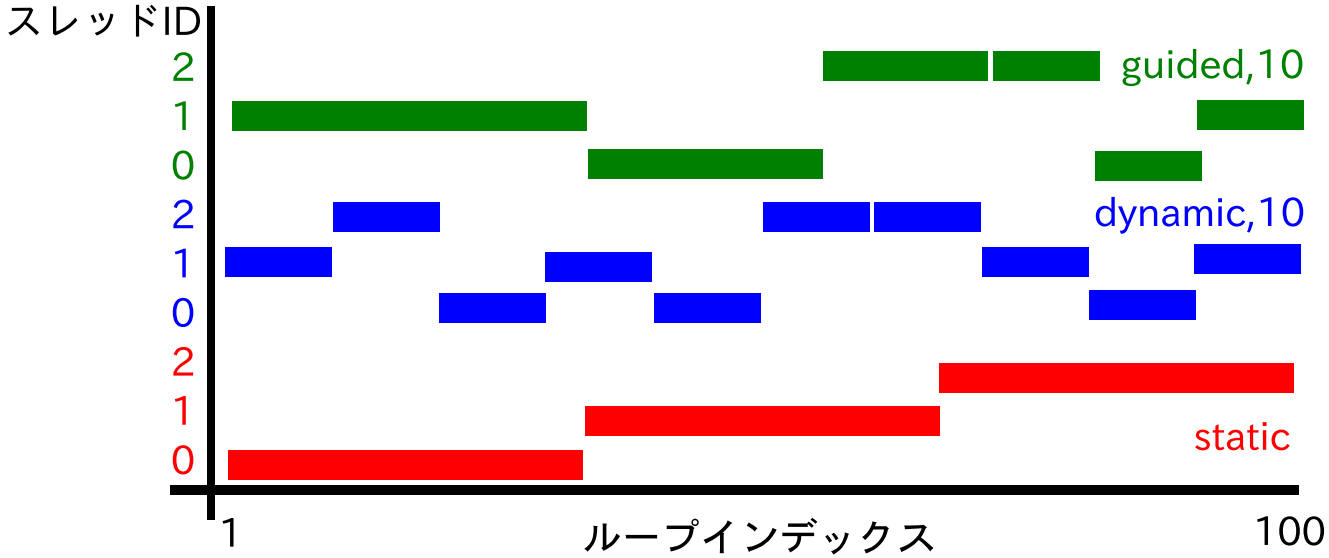
```
!$omp parallel do schedule(static,2)
```

schedule

【書き方】schedule(種類, サイズ)

例）!$omp parallel do schedule(static, 4)

- サイズは指定しなくても良い（指定しない場合，適切な値に自動設定）．
- 種類は次の中から指定．
  - static ：先ほどのブロックサイクリック分割．
  - dynamic ：１ブロックずつから始め，終わったスレッドが順次，次を実行．
  - guided：dynamic と同様だが，ブロックサイズを徐々に細かくしていく（最低でも指定サイズ）．
  - runtime：プログラムの実行時に環境変数 OMP_SCHEDULE で指定．



演習１：いろいろな分割方法を試してみよう！

1. 48 コアマシンへログイン

```
% ssh ＊＊＊.＊＊＊.＊＊＊.＊＊＊
```

2. 48 コアマシンで，今日の演習用のディレクトリ（例えば enshu-openmp2）を作成

```
% mkdir enshu–openmp2
% cd enshu–openmp2
```

3. emacs 等で，次のスライドのプログラムを作成し，schedule.f90 などの名前で保存・コンパイル
4. ４プロセッサを用いて実行
5. 並列化指示行

```
!omp parallel do schedule(static,4)
```

の static を，dynamic, guided に変更し，計算時間を比較

演習１のプログラム

```
program schedule
implicit none
integer, parameter :: SP = kind(1.0)
integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
integer, parameter :: n = 1000
integer :: i,j
real(DP), dimension(n) :: x, y
real(DP), dimension(n,n) :: A
real(DP) :: time0, time1, omp_get_wtime
x(:) = 2.0_DP
A(:,:) = 1.0_DP
time0 = omp_get_wtime()
!$omp parallel do schedule(static,4) private(i,j) shared(A,x,y)
do i=1,n
  y(i) = 0.0_DP
  do j=i,n
    y(i) = y(i) + A(i,j) * x(j)
  end do
end do
!$omp end parallel do
time1 = omp_get_wtime()
print *, time1-time0
end program
```

復習：４スレッドでの実行方法

- 下のようなスクリプトを作成して，適当な名前（例えば enshu.sh など）で保存．
- その後，

```
% qsub enshu.sh
```

とすると jobname.o???? （???? は適当な番号）というファイルの中に結果が書き込まれます．

```
#!/bin/bash
#PBS -N jobname
#PBS -l nodes=1
#PBS -l ncpus=4
#PBS -q default
#PBS -j oe
cd /home/username/enshu-openmp2/
export OMP_NUM_THREADS=4
./a.out
```

最大使用プロセッサ数を指定

作業ディレクトリを指定

ncpus と同じ値を指定

実行ファイル名を指定

同じものが /tmp/openmp2/enshu.sh においてあります．
前回利用したものを使いまわしてもかまいません．

各スレッドに別々の仕事を割り当て（!$omp sections）

例：質点の運動のシミュレーション

```
program
do while (t < 必要な時間)
  （x 軸方向の更新）
  （y 軸方向の更新）
  （z 軸方向の更新）
end do
end program
```



```
!$omp parallel

!$omp sections
!$omp section

!（x 軸方向の更新）

      omp end section は書かない．
!$omp section

!（y 軸方向の更新）

!$omp section

!（z 軸方向の更新）

!$omp end sections

!$omp end parallel
```

!$omp sections の特徴

- それぞれの section を別々のスレッドが実行．
- 他のスレッドは待機．
- 実行される順序は指定できない．

演習２：数値解法の誤差プロット

【課題】

1. プログラム /tmp/openmp2/err.f90 を各自のディレクトリにコピー．

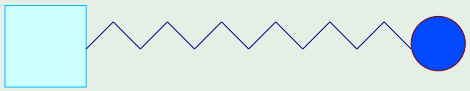
```
% cp /tmp/openmp2/err.f90 ./
```

2. !!!!!! PART1 !!!!!! と !!!!!! PART2 !!!!!! は並行して実行できる計算なので，これらを sections を利用して並列化（赤字部分を追加）．
3. １スレッド，２スレッドで実行した場合について計算時間を比較．

```
program err
! 変数の定義など
!$omp parallel shared(x0,v0,x1,v1) ...
!$omp sections
!!!!!! PART1 !!!!!!
!$omp section
（長い計算）
!!!!!! PART2 !!!!!!
!$omp section
（長い計算）
!$omp end sections
!$omp end parallel
! print 文など
end program
```

プログラムについて：

- ばねにつながった質点の運動を計算．



- ある計算法について，パラメータ (dt) を変えて計算（PART2，計算量小×複数回）し，それぞれの場合の誤差を推定．
- 誤差は高精度数値解法 (PART1，計算量大×１回) で求めたものと比較して推定．

時間計測用ジョブスクリプトの例

```
#!/bin/bash
#PBS -N jobname
#PBS -l nodes=1
#PBS -l ncpus=2
#PBS -q default
#PBS -j oe
cd /home/username/enshu-openmp2/
for opn in 1 2
do
export OMP_NUM_THREADS=$opn
./a.out
done
```

最大使用プロセッサ数を指定

作業ディレクトリを指定

opn を変えながら do 内を実行

スレッド数を opn に設定

実行プログラム名を指定

/tmp/openmp2/jscript.sh に置いてあります．

一つのスレッドだけで実行（!$omp single）

```
 !$omp parallel
val = 1.0_DP
!$omp do private(i,j)
do  j=1,n
   do  i=1,n
      A(i,j) = val
   end do
end do
!$omp end do
!$omp end parallel
```



```
!$omp parallel
!$omp single
val = 1.0_DP    ←（１スレッドだけで実行，他は待機）
!$omp end single
!$omp do private(i,j)
do  j=1,n
   do  i=1,n
      A(i,j) = val
   end do
end do
!$omp end do
!$omp end parallel
```

左側のコードでは val の値に全てのスレッドが同時に書き込む

- 理論的には大丈夫だが，
- 同じアドレスに同時にアクセス ➡ パフォーマンスの低下

マスタースレッドだけで実行（!$omp master）

```
!$omp parallel
val = 1.0_DP
（何か別の処理）
!$omp do private(i,j)
do  j=1,n
  do  i=1,n
    A(i,j) = val
  end do
end do
!$omp end do
!$omp end parallel
```


single を利用

```
!$omp parallel
!$omp single
val = 1.0_DP
!$omp end single
（何か別の処理）
!$omp do private(i,j)
do  j=1,n
  do  i=1,n
    A(i,j) = val
  end do
end do
!$omp end do
!$omp end parallel
```

master を利用

```
!$omp parallel
!$omp master
val = 1.0_DP
!$omp end master
（何か別の処理）
!$omp do private(i,j)
do  j=1,n
  do  i=1,n
    A(i,j) = val
  end do
end do
!$omp end do
!$omp end parallel
```

- !$omp master: マスタースレッドだけで実行．他は待たない．
- 次の処理を進められる場合に有効．
  （何か別の処理）の部分では val を使ってはいけない（更新が終わっていないため）．val にアクセスする際は，次に説明する barrier が必要．

スレッドの同期と制御

- !$omp barrier ：全てのスレッドがここに来るまで待機．
  - !$omp end do，!$omp end single などの後には，自動的に barrier が設置される．
  - !$omp end do nowait などとすることで，設置しないようにもできる．
- !$omp critical ：同時に２つ以上のスレッドが実行しないようにする．
- !$omp atomic ：同時書き込みの禁止（スカラー値の更新のみ）．

```
real(DP) :: sval, pval
real(DP), dimension(n) :: svec
!$omp parallel shared(sval, svec) private(pval)
（pval の値を各スレッドで計算）
!$omp critical
svec(:) = pval * svec(:)
!$omp end critical
!$omp atomic
sval = sval + pval
              omp end atomic は書かない
!$omp end parallel
```

演習３：reduction を使わない総和計算

【課題】 下記のプログラムを次の３通りに修正し，６スレッドで実行．

1. そのまま実行．
2. omp atomic の部分を削除して実行．
3. omp atomic の代わりに omp critical, omp end critical を用いたプログラムを作成し，実行．

```
program summation
integer, parameter :: SP=kind(1.0)
integer, parameter :: DP=selected_real_kind(2*precision(1.0_SP))
integer, parameter :: n=1000
real(DP) :: sval, pval
real(DP), dimension(n) :: svec
svec(:) = 1.0_DP
sval = 0.0_DP
!$omp parallel shared(sval, svec) private(pval)
pval=0.0_DP
!$omp do
do i=1,n
pval = pval + svec(i)
end do
!$omp end do
!$omp atomic
sval = sval + pval
!$omp end parallel
print *, sval
end program
```

ソースファイルは /tmp/openmp2/sum.f90 に置いてあります．

演習４：演習３のプログラムの高速化

omp end do の後には barrier が置かれるが，ここで全員が揃うまで待っている必要はない ➡ nowait を挿入することで barrier を除去．

```
!$omp parallel shared(sval, svec) private(pval)
pval=0.0_DP
!$omp do
do  i=1,n
pval = pval + svec(i)
end  do
!$omp end do nowait
!$omp atomic
sval = sval + pval
!$omp end parallel
```

【課題】

- 演習２のプログラムについて nowait を入れた場合，入れない場合の実行速度（６スレッド）を比較．
- 余裕のある人は atomic を利用した場合と critical を利用した場合の実行時間も比較．

いろいろな並列化法（例：行列ベクトル積の並列化）

並列化の方法はいろいろなやり方が有り得る ➡ 性能も変わる！

例）演習１のプログラム

```
!$omp parallel do
do  i=1,n
   y(i) = 0.0_DP
   do  j=1,n
      y(i) = y(i) + A(i,j) * x(j)
   end  do
end  do
!$omp end parallel do
```

fortran では左側の添字を先に動かしたほうがキャッシュミスが少ない
➡ このプログラムは遅い！

そこで，ループを入れ替えて

```
do  j=1,n
   y(i) = 0.0_DP
   do  i=1,n
      y(i) = y(i) + A(i,j) * x(j)
   end  do
end  do
```

➡

```
y(:) = 0.0_DP
do  j=1,n
   do  i=1,n
      y(i) = y(i) + A(i,j) * x(j)
   end  do
end  do
```

を並列化してみる．

並列化例１：内側のループを並列化

```
do  j=1,n
!$omp parallel do
   do  i=1,n
      y(i) = y(i) + A(i,j) * x(j)
   end  do
!$omp end parallel do
end  do
```

- 正しい答えは出るので間違いではないが，
- 外側のループが回るたびにスレッドの生成・消滅を行うので
  オーバーヘッドが大きい．

並列化例２：外側のループの並列化

```
!$omp parallel do private(i,j)
do  j=1,n
   do  i=1,n
      y(i) = y(i) + A(i,j) * x(j)
   end do
end do
!$omp end parallel do
```

y(i) に複数のスレッドから同時書き込みが有り得る ➡ atomic を挿入

```
!$omp parallel do private(i,j)
do  j=1,n
   do  i=1,n
!$omp atomic
      y(i) = y(i) + A(i,j) * x(j)
   end do
end do
!$omp end parallel do
```

並列化例３：reduction を利用する方法

並列化例２のプログラム

```
!$omp parallel do private(i,j)
do  j=1,n
   do  i=1,n
!$omp atomic
      y(i) = y(i) + A(i,j) * x(j)
   end do
end do
!$omp end parallel do
```

は，実質，reduction 演算 ➡ reduction を用いて書き換え可能．

```
!$omp parallel do private(i,j) reduction(+:y)
do  j=1,n
      y(:) = y(:) + A(:,j) * x(j)
end do
!$omp end parallel do
```

並列化例４：より複雑な並列化

```
real(DP), allocatable :: localsum(:)
!$omp parallel private(j,localsum)
allocate(localsum(n))
localsum(:) = 0.0_DP
!$omp do
do j=1,n
   localsum(:) = localsum(:)+A(:,j)*x(j)
end do
!$omp end do nowait
!$omp critical
y(:) = y(:) + localsum(:)
!$omp end critical
deallocate(localsum)
!$omp end parallel
```

- 考え方は reduction と同じ．
- 各スレッドごとに private の計算用配列 (localsum) を用意．

演習５：行列・ベクトル積

課題

- これまでに説明した並列化例１～４のうち，２つ以上を実装．
- ４スレッドを用いた場合について計算時間を比較．

```
program main
implicit none
integer, parameter :: SP=kind(1.0)
integer, parameter :: DP=selected_real_kind(2*precision(1.0_SP))
integer, parameter :: n = 1000
integer :: i,j
real(DP),dimension(n,n) :: A
real(DP),dimension(n) :: x,y
real(DP) :: time0,time1,omp_get_wtime
A(:,:) = 1.0_DP
x(:) = 2.0_DP
y(:) = 0.0_DP
time0 = omp_get_wtime()
（選んだ行列ベクトル積のプログラム）
time1 = omp_get_wtime()
print *, time1-time0
end program
```

例４を実装する場合
real(DP), allocatable :: localsum(:)
はプログラム冒頭に記述

宿題：２３進数覆面算　【締切】６月７日，午後５時．

【課題】 次のスライドのプログラムは，下の覆面算の答えの数を数えるための，素朴な並列プログラムである：

$$\begin{array}{cccccc} & C & O & M & P \\ + & S & Y & S & M \\ \hline & Z & Z & Z & Z \end{array}$$

ただし，演算は２３進数で行い，各アルファベットには０～２２までの数字が（重複を許して）入れるとする．例えば，$Z = 14, C = 1, O = 8, M = 1, P = 13, S = 13, Y = 6$ は

$$\begin{aligned} & 23^3C + 23^2O + 23M + P + 23^3S + 23^2Y + 23S + M \\ & = 1 \times 23^3 + 8 \times 23^2 + 1 \times 23 + 13 + 13 \times 23^3 + 6 \times 23^2 + 13 \times 23 + 1 \\ & = 14 \times 23^3 + 14 \times 23^2 + 14 \times 23 + 14 = 23^3Z + 23^2Z + 23Z + Z \end{aligned}$$

となるので，この覆面算の答えの一つである．このプログラムについて，

1. 並列プログラムとして誤りがあるので，それを訂正．
2. ４８コアマシン上，PGI コンパイラ，最適化オプション無し (pgf95 -mp fukumenzan.f90)，４スレッドでの計算時間が，４秒以下になるように工夫．
3. プログラムと実行結果を，１つのテキストファイル（例えば result.txt）に入れて，その内容を yaguchi までメール．

【メールの送り方】４８コアマシン上で mail yaguchi < result.txt

宿題のプログラム（/tmp/openmp2/fukumenzan.f90）

```
program fukumenzan
implicit none
integer, parameter :: SP=kind(1.0)
integer, parameter :: DP=selected_real_kind(2*precision(1.0_SP))
double precision :: time0, time1, omp_get_wtime
integer :: c,o,m,p,s,y,z
integer :: cnt
cnt=0
time0=omp_get_wtime()
!$omp parallel do private(c,o,m,p,s,y,z) shared(cnt)
do z=1,22
do c=1,z      (高速化のヒント：c は最大でも z までしか動かない ➡ z によって c の動く範囲が変化 ➡ 計算量の偏りが発生)
do o=0,22
do m=0,22
do p=0,22
do s=1,22
do y=0,22
if(c*(23**3)+o*(23**2)+m*23+p+s*(23**3)+y*(23**2)+s*23+m-z*(23**3)-z*(23**2)-z*23-z==0)then
cnt = cnt+1
end if
end do
end do
end do
end do
end do
end do
end do
end do
!$omp end parallel do
time1=omp_get_wtime()
print *, 'computation time:', time1-time0        （目標：ここの出力が４秒以下！）
print *, cnt, 'solutions'
end program
```

宿題のプログラム実行用スクリプト

プログラムと実行結果をまとめるのが大変な人は，以下のスクリプトで実行して下さい．実行結果のファイル（jobname.o????, ???? は実行時に決まる数字）の中にプログラムも含まれるようになりますので

mail yaguchi < jobname.o????

で宿題が提出できるようになります．

```
#!/bin/bash
#PBS -N jobname
#PBS -l nodes=1
#PBS -l ncpus=4
#PBS -q default
#PBS -j oe
cd  /home/username/enshu-openmp2/
export OMP_NUM_THREADS=4
./a.out
cat fukumenzan.f90
```

作業ディレクトリを指定
４スレッドで実行
実行ファイル名を指定
プログラムファイル名を指定

同じものが /tmp/openmp2/shukudai.sh においてあります．

参考文献

- **南里豪志，天野浩文．** OpenMP **入門** (1), (2), (3)**，** http://www.cc.kyushu-u.ac.jp/scp/system/library/OpenMP/OpenMP.html.
- **黒田久泰．** C **言語による** OpenMP **入門，** http://www.cc.u-tokyo.ac.jp/publication/kosyu/03/kosyu-openmp_c.pdf.
- **北山洋幸．**OpenMP **入門- マルチコア** CPU **時代の並列プログラミング，秀和システム，**2009.
- Barbara Chapman, Gabriele Jost and Ruud van der Pas (Foreword by David J. Kuck). Using OpenMP –Portable Shared Memory Parallel Programming–, The MIT Press, 2007.

質問は ｙａｇｕｃｈｉ@ｐｅａｒｌ．ｋｏｂｅ−ｕ．ａｃ．ｊｐ まで．