

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

2012 年 5 月 31 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。

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 : 応用 (乱数生成の並列化)
- 宿題 : π の数値計算 II

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（Pは大文字）

```
scp -P xx .ssh/authorized_keys xxx.xxx.xxx.xxx:
```

- 2 48 コアマシンへログイン（pは小文字）

```
ssh -p xx xxx.xxx.xxx.xxx
```

今日と次週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 `xxx.xxx.xxx.xxx` 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

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 は青の部分をも、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 = 2 2 個，緑の要素数 = 1 4 個。



ちょっと改善。

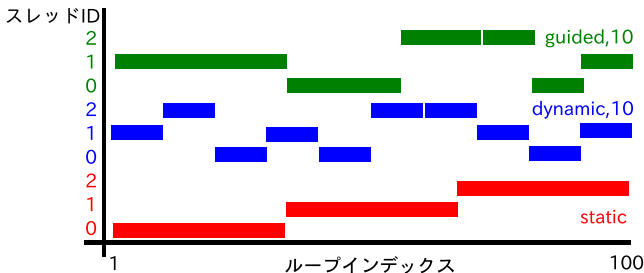
OpenMP では

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

【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小さくしていく（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 -p xx xxx.xxx.xxx.xxx
```

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 次のスライドのプログラム (/tmp/openmp2/schedule.f90) を 4 プロセッサを用いて実行

4 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．

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！

は，本当は，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
```

とすべき。

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 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 .



!\$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

演習 2 : シェルピンスキーギャスケット

【課題】

- 1 プログラム `/tmp/openmp2/sierp.f90` を各自のディレクトリにコピー .
- 2 do ループ中の $x(i)$ の計算と $y(i)$ の計算は並列計算できるので **sections** を利用して並列化 .
(**sections** を使えるように , うまくプログラムを書き換えること .)
- 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 .
(計算時間比較用のスクリプトファイルが `/tmp/openmp2/jscript.sh` においてあります .)

【終わった人は】結果を gnuplot で表示

- プログラム中の計算時間出力部分を消す . 最後 3 行のコメントを外して $x(i)$, $y(i)$ を出力 .
- 1 プロセッサで実行 .
- 一度 , ログアウトして `ssh -p xx -Y xxx.xxx.xxx.xxx` で再ログイン .
- `gnuplot plot "jobname.o???" exit`
(終わったらログアウトして , `-Y` 無しで再ログイン `ssh -p xx xxx.xxx.xxx.xxx`)

演習 2 のプログラム

```
program sierpinski
implicit none
! 変数の定義など
do i=1,n
  call random_number(myrand(i))
end do

time0 = omp_get_wtime()
do i=1,n-1
  if (myrand(i) < 0.33_DP) then
    x(i+1) = x(i) * 0.5_DP + 1.0_DP
    y(i+1) = y(i) * 0.5_DP
  else if (myrand(i) > 0.66_DP) then
    x(i+1) = x(i) * 0.5_DP - 1.0_DP
    y(i+1) = y(i) * 0.5_DP
  else
    x(i+1) = x(i) * 0.5_DP
    y(i+1) = y(i) * 0.5_DP + 1.0_DP
  end if
end do
time1 = omp_get_wtime()

print *, time1-time0

!do i = 1,n
! print *, x(i), y(i)
!end do
end program
```

赤と青の部分を別々のプロセッサで実行。

早めに終わった人は、出力部分を変更して gnuplot で表示。

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)

single を利用

master を利用

```
!$omp parallel  
val = 1.0_DP  
(何か別の処理)  
!$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
```



```
!$omp parallel  
!$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
```

```
!$omp parallel  
!$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 .

```
program sierpinski
implicit none
! 変数の定義など
do i=1,n
  call random_number(myrand(i))
end do

time0 = omp_get_wtime()
do i=1,n-1
  if (myrand(i) < 0.33_DP) then
    x(i+1) = x(i) * 0.5_DP + 1.0_DP
    y(i+1) = y(i) * 0.5_DP
  else if (myrand(i) > 0.66_DP) then
    x(i+1) = x(i) * 0.5_DP - 1.0_DP
    y(i+1) = y(i) * 0.5_DP
  else
    x(i+1) = x(i) * 0.5_DP
    y(i+1) = y(i) * 0.5_DP + 1.0_DP
  end if
end do
time1 = omp_get_wtime()

print *, time1-time0

!do i = 1,n
! print *, x(i), y(i)
!end do
end program
```

青い部分の do ループも、
一見、独立に計算できそうなので、
並列化できそう。

➡ 本当に独立に計算できる？

乱数は「乱数っぽい数列」

- 通常，計算機上の乱数は疑似乱数 = 乱数っぽいけど規則的な数列．
- 例) 乗算合同法

$$r^{(n+1)} = ar^{(n)} + c \pmod{m} \quad (a, c, m \text{ は適当な整数})$$

- ➡ 乱数同士には依存関係があり，すぐには並列化できない．
(サブルーチン `random_number()` はスレッドセーフとは限らない.)

(復習) 並列化法 : recursive doubling

$$\begin{aligned} r^{(n+1)} &= a(ar^{(n-1)} + c) + c \pmod{m} \\ &= a^2r^{(n-1)} + (a+1)c \pmod{m} \quad =: a'r^{(n-1)} + c' \pmod{m} \end{aligned}$$

こうすると偶数項と奇数項を並列に計算可能！

演習 5 : シェルピンスキーギャスケット II

【課題】

- 1 演習 2 のプログラム冒頭の乱数生成部分を, 下のように変更 .
- 2 スレッドを用い, 乱数生成部分を偶数・奇数に分けることで並列化 .
 - 偶数・奇数に分けるために, 2ステップ後の値を計算する漸化式を作成 (前のスライドの a' , c' を求める) .
 - **!\$omp sections** で並列化 .
- 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 .

```
do i=1,n
  call random_number(myrand(i))
end do
```



```
integer, parameter :: a = 109
integer, parameter :: c = 1021
integer, parameter :: m = 2**15
integer :: tmp

tmp = 15
do i=1,n
  tmp = mod(tmp*a+c,m)
  myrand(i) = real(tmp,DP) / (m-1)
end do
```

(これを, さらに偶奇に分けたプログラムに変更.)

【課題】

有限体 F_{13} フィボナッチ数列

$$a^{(n+2)} = a^{(n+1)} + a^{(n)} \pmod{13}$$

について

- 1 これを $n = 100000$ まで求めるプログラムを書き，
- 2 並列化せよ．

演習 5 まで終わってしまった人用，宿題の発展課題の一部
なので，やらなくてもいいです．

宿題： π の数値計算 II

【課題】 次のスライドのプログラムは

$$200 \int_0^1 \int_0^1 \exp(-200((x - 0.5)^2 + (y - 0.5)^2)) dx dy = \pi$$

を用いて円周率を計算するプログラムである（/tmp/openmp2/pi2.f90 にあります）。このプログラムについて、

- 1 **【必須課題】** do ループの分割の仕方を適切に指定することにより pgf95 -mp pi2.f90 でコンパイルしたときの **4 スレッドでの実行時間が 0.002 秒を下回るようにせよ**。
- 2 **【発展課題：途中まででもいいので、何か書いたら加点】** フィボナッチ数列の並列化をやってみた人は、その並列プログラムと同様に、最初の数列計算を並列化せよ（2 スレッドで良い）。
- 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】 4 8 コアマシン上で mail yaguchi < result.txt

【締切】 6月6日（水）、午後5時。

分からない場合は yaguchi@pearl.kobe-u.ac.jp までメールを下さい。

宿題のプログラム (/tmp/openmp2/pi2.f90)

```
program pi2
! 変数の定義
! フィボナッチ数列を求める
a(1) = 1.0d0
a(2) = 1.0d0
do i=1,n-2
a(i+2) = a(i+1) + a(i)
end do

allocate(b(a(n)))
time0 = omp_get_wtime()
!$omp parallel do default(none) shared(a,b) private(m,res,x,y,i)
do m=2,n
res = 0.0d0
do i=0,a(m)-1
x = dble(i) / dble(a(m)) * 1.0d0
y = dble(mod(i*a(m-1),a(m)))/ dble(a(m))
res = res + 200.0d0/dble(a(m))*dexp(-200.0d0*((x-0.5)**2 + (y-0.5)**2))
end do
b(m) = res
end do
!$omp end parallel do
time1 = omp_get_wtime()

print *, time1-time0  この部分が 0.002 以下になるように！

do m=2,n
print *, a(m),b(m)
end do

deallocate(b)
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 pi2.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。

- 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 .