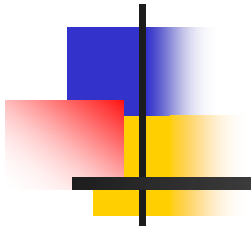


計算科学演習　第7回 講義
「OpenMP並列処理」

2010年6月10日

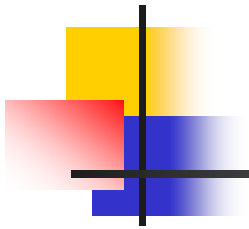
システム情報学研究科　計算科学専攻
臼井　英之

共有変数とプライベート変数

```
!$omp parallel
!$omp do
do i=1,100000
    b(i)=a(i)
enddo
!$omp end do
!$omp end parallel
end
```

Doループの変数 i はプライベート変数

- 共有変数
  - OpenMP のプログラミングモデルでは，**基本的にすべての変数は共有変数**（どのスレッドからも参照・更新が可能）
- プライベート変数
  - ループインデックス変数 i については，スレッド 0 と 1 で共有すると，正しい制御ができない
    - スレッド0では 1≦ i ≦50000，1では 50001≦ i ≦100000 の範囲を動いて欲しい
  - そこで，各スレッドがそれぞれ別の変数を持つ必要がある
  - このような変数を**プライベート変数**と呼ぶ



変数の指定

- デフォルト値
  - 何も指示をしない変数については，**基本的に共有変数**となる
  - しかし，並列化された**ループのインデックス変数**のように，明らかにプライベート変数でなければならない変数については，**特に指示をしなくてもプライベート変数**となる
    - 多重ループの場合は並列化対象ループのインデックス変数のみ
- プライベート変数の指定
  - 並列化指示文の後に，private 節を追加する。
- 共有変数の指定（通常は不要）
  - 並列化指示文の後に，shared 節を追加する。
- 指示を省略せずに書く場合は例えば、
  - `!$omp parallel do shared(a, x, y, z) private(i)`

先週の宿題：　時間計測せよ。

元プログラム

```
program axpy
 implicit none
  integer, parameter ::  SP = kind(1.0)
  integer, parameter ::  DP = selected_real_kind(2*precision(1.0_SP))
  real(DP), dimension(100000) :: x, y, z
   real(DP):: a
  integer :: i
!
! a, x, yの値を各自設定 （なにか定数を設定するとか。）
!
!$omp parallel
!$omp do
do i = 1, 100000
  z(i) = a*x(i) + y(i)
end do
!$omp end do
!$omp end parallel
(結果の確認)
End
```

ベクトルの加算 $z = ax + y$

実行結果：1≦ i ≦50000 がスレッド0,
50001≦ i ≦100000 がスレッド1で計算される。

宿題の解答例

```
program axpy
implicit none
integer, parameter::SP = kind(1.0)
integer, parameter::DP = selected_real_kind(2*precision(1.0_SP))
real(DP), dimension(10000000) :: x, y, z
real(DP):: a, omp_get_wtime, time0, time1
integer :: i
 x=100_DP
 y=100_DP
 a=2_DP
!$omp parallel private(time0, time1)
    time0=omp_get_wtime()
!$omp do
do i = 1, 10000000
  z(i) = a*x(i) + y(i)
end do
!$omp end do
    time1=omp_get_wtime()
   print*, 'Elapsed time = ', time1-time0, '(sec)'
!$omp end parallel
end
```

配列に値を代入（たとえば、x,y に100, aに2をセット）

Time0, time1をスレッド毎の変数に指定

スタート時間をTime0に格納

エンド時間をTime1に格納

スレッド毎の経過時間を表示

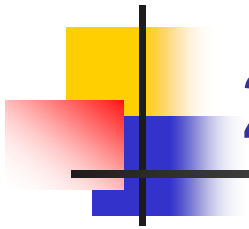
変数の共有指定の例

- 2重ループの並列化（行列ベクトル積）

```
program gemv
implicit none
  integer, parameter::SP = kind(1.0)
  integer, parameter::DP = selected_real_kind(2*precision(1.0_SP))
  real(DP), dimension(100,100) :: a
   real(DP), dimension(100) :: x, y
  integer :: i, j
（a, xの値を設定）
!$omp parallel do private(j)
do i = 1, 100
  y(i) = 0.0_DP
  do j = 1, 100
    y(i) = y(i) + a(i,j) * x(j)
  end do
end do
（yの値を表示）
stop
end
```

jをプライベート変数に指定

a, x, y は自動的に共有変数となる。
i は自動的にプライベート変数となる。
j はプライベート変数とすべきだが，自動的にはそうならない（並列化対象ループのインデックス変数ではない）ので指定が必要



2重ループ並列化の注意

- 何も指示しなければ内側のループインデックスは共有変数
- 共有変数のままだと、各スレッドが互いにそれを上書きし合って内側のループを正しく実行することができない
- これを回避するため、内側ループのインデックス（前の例ではj）をプライベート変数として宣言する必要がある

総和の計算（リダクション変数）1

- 1～10000までの整数の総和

```
program main
integer :: i, sum
sum = 0
  do i=1, 10000
  sum = sum + i
  end do
print *, " sum= ", sum
end
```

- 並列化の際，変数sum は共有変数とすべきか？
  - そうすると，総和が正しく計算できない（書き込みの競合）
- プライベート変数とすべきか？
  - そうすると，並列実行部分終了後に，sumの値が消えてしまう。

総和の計算（リダクション変数）2

- リダクション変数とは
  - 並列実行部分ではプライベート変数で，**並列終了時に各スレッドの値が合計される**ような変数

```
program main
integer :: i, sum
sum = 0
!$OMP parallel do reduction(+:sum)
  do i=1, 100000
  sum = sum + i
  end do
print *, " sum= ", sum
end
```

sumをリダクション変数に指定（総和型）

  - 総和の他に，積や最大値を求めるリダクション変数指定も可能

総和の計算(リダクション変数)3:練習

- 前頁の問題に、Thread毎の部分和、時間計測を追加
  - OMP parallel 環境を定義し、リダクション変数sumを定義
  - 同時に、部分和変数(psum)と時間格納変数をprivateで定義(thread毎の独立変数)
  - OMP parallel環境内において
    - psum初期化後、時間計測スタートし、その後、omp doで各スレッドでのpsumを求める
    - Omp end do の後、sumにpsumを足し込む(reduction)
    - 時間計測エンドで、かかった時間を表示。
  - Omp end parallelでOMP parallel環境から抜ける。

前頁のプログラムを上の指示に従って変更し、
スレッド1とスレッド2の場合で時間計測を行う。

解答例

```
（変数の宣言）
 sum = 0
 !$OMP parallel reduction(+:sum) private(psum,計測時間格納変数)
  psum=0
（スタート時間計測）
 !$OMP do
    do i=1, 10000
    psum =psum + i
    end do
 !$OMP end do
    sum=sum+psum
 （エンド時間計測）
（経過時間の表示）
 !$OMP end parallel
 （総和の表示）
```

- プログラム、
- スレッド1、2の場合について
  psum, sumの値、経過時間

以上の内容をテキストファイルにまとめ、mailでusui に送る。（標題をつける。）

（mail –s “exercise 0610_a” < text.txt）

OMP DO （3）

分割を規定する

```
!$omp parallel
!$omp do schedule(static, 4)
do i=1,100
    b(i)=a(i)
enddo
!$omp end do
!$omp end parallelend
```

1～100を4つずつのchunkにわけて、それをサイクリックに各スレッドに割り当てる

4スレッド実行時
マスタスレッド担当行：
1,2,3,4,17,18,19,20,

ループ分割指定　1

- 素数を求める（1と自分自身でしか割り切れない数字）
  - 1からnregion内の整数iについて、1からiまでの数字で割り算をして余りが0, すなわち約数の数（count)を調べる。
  - Count が2なら素数。

```
 integer:: i, j, count, loop, nregion
 loop  = 0
 nregion = 2**8
 do i=1, nregion
   count = 0
   do j=1, i
     if(mod(i,j)==0)  count = count+1
     loop = loop + 1
   end do
   if(count == 2) print*, 'sosu = ', i
 end do
 print *, " Number of loop = ", loop
end
```

mod(a,b)はaをbで割った余り

ループ分割指定　2: 練習　（宿題）

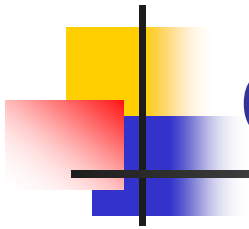
- 前頁の素数プログラムを
  OpenMP並列化＋Thread毎の内部ループ数のカウント表示

- ループ分割指定し、負荷均等化を確認
  – スレッド2の場合の実行時間変化をチェック

•ループ分割したプログラム、
•スレッド2の場合についてループ分割あり、なしで
　　ループ数の違い、経過時間

以上の内容をテキストファイルにまとめ、mailでusui
に送る。（標題をつける。）締め切りは6/16 PM5.

（mail –s “exercise 0610_b” < text.txt）



OMP Sections

Section毎にスレッドに仕事が
割り当てられる

Sectionの数よりもスレッド数が
多い場合には仕事をしないスレッド
が発生する

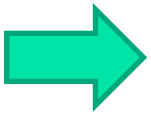
```
!$OMP Sections
!$OMP Section
      （ここにスレッド0が行う計算内容を記述）
!$OMP Section
      （ここにスレッド1が行う計算内容を記述）
!$OMP Section
      （ここにスレッド2が行う計算内容を記述）
!$OMP END Sections
```

OMP Single

```
!$OMP Parallel
        並列処理

!$OMP Single
        逐次処理
!$OMP END Single

        並列処理
!$OMP END Parallel
```



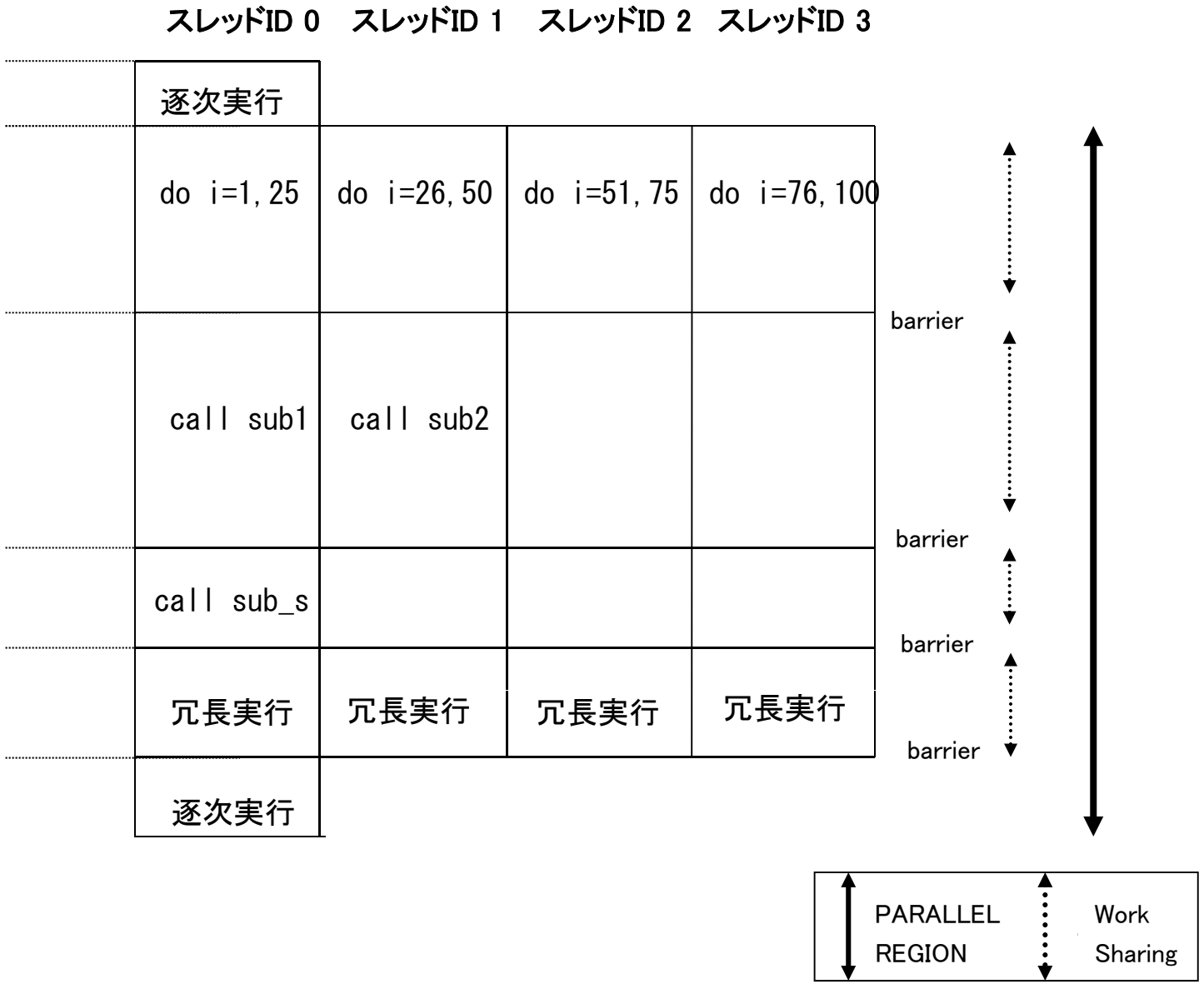
一つのスレッドのみが処理を行う
（冗長実行を防ぐ）

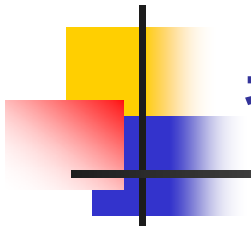
OpenMP 実行イメージ

```
program  main
         :
!$OMP PARALLEL
!$OMP DO
   do i=1,100
     a(i)=i
   end do
!$OMP END  DO
!$OMP  SECTIONS
!$OMP SECTION
   call sub1
!$OMP SECTION
   call sub2
!$OMP END  SECTIONS
!$OMP SINGLE
    call sub_s
!$OMP END SINGLE

    b(1)=a(1)

!$OMP END PARALLEL
         :
end  program main
```





参考文献例

- 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.