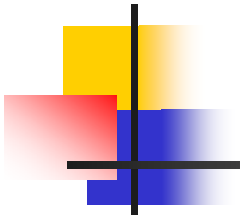


並列アルゴリズム

2005年度後期　火曜　2時限

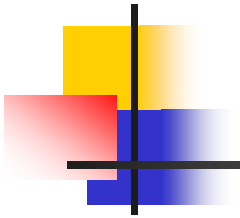
青柳　睦

Aoyagi@cc.kyushu-u.ac.jp

http://server-500.cc.kyushu-u.ac.jp/

1月17日（火）

11. OpenMP 基礎



もくじ

1. 序　並列計算機の現状

2. 計算方式およびアーキテクチュアの分類

3. 並列計算の目的と課題

4. 数値計算における各種の並列化

5. MPIの基礎

6. 並列処理の性能評価

7. 集団通信（Collective Communication）

8. 領域分割（Domain Decomposition）

9. LU分解法とその並列化（講義）
   PCクラスタによる並列プログラミング（演習）

10. OPenMPの概要

11. OpenMP 基礎

11. OpenMP の基礎

- 繰り返しループの並列化

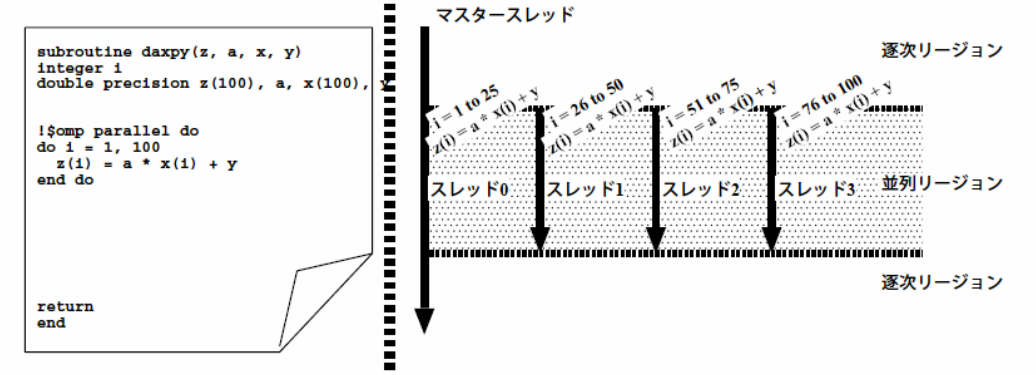
  parallel for 構文

C 言語

```
void daxpy(z, a, x, y)
double z[], a, x[], y;
{
  int i;
#pragma omp parallel for
  for (i = 0; i < 100; i++)
    z[i] = a * x[i] + y;
}
```

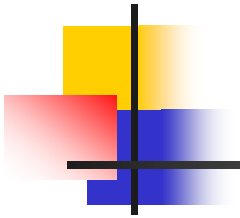
Fortran

```
subroutine daxpy(z, a, x, y)
implicit none
integer :: i
real(8) :: z(0:99), a, x(0:99), y
!$omp parallel do
do i = 0, 99
  z(i) = a * x(i) + y
end do
return
end
```



（注意）各スレッドの実行開始の順序は非制御

変数とスコープ



- 基本的にOpenMPのプログラムで用いられる変数は全スレッドに共有された記憶領域に配置される
  （しかし全ての変数が共有化されていては，
  各スレッドが同じ処理内容を行うことになってしまうので）

- 各スレッドから共有させる変数（共有変数）と，各スレッドにそれぞれ独自の値を持たせることのできる変数（プライベート変数）の区別が重要

  前例DAXPYでは共有変数：a,x,y,z
  　　プライベート変数：ｉ

変数とスコープ（2）

- 共有またはプライベートの別を**スコープ**と呼び，前例DAXPYで明示的にスコープを指定する場合は，

Fortran言語の場合，

#pragma omp parallel for shared(z, a, x, y) private(i)

C言語の場合，

!$omp parallel do shared(z, a, x, y) private(i)

となる.

- ただし，一般にループのインデックス（例ではi）のように明らかに共有変数ではない変数は暗黙の宣言でプライベート変数にアサインされる（OpenMPの仕様）

では多重ループの場合は・・

変数とスコープ（3）

行列・ベクトル積のプログラム例

C 言語

```
void matvec(a, x, y)
double a[][100], x[], y[];
{
  int i, j;

#pragma omp parallel for private(j)
  for (i = 0; i < 100; i++)
    for (j = 0; j < 100; j++)
      y[i] += a[i][j] * x[i];
}
```

Fortran

```
subroutine matvec(a, x, y)
implicit none
integer :: i, j
real(8) :: a(100, 100), x(100), y(100)

!$omp parallel do private(j)
do i = 1, 100
  do j = 1, 100
    y(i) = y(i) + a(j, i) * x(i)
  end do
end do

return
end
```

（注意） C言語では，多重ループの外側ループを並列化した場合には，暗黙宣言でプライベート変数となるのは該当する外側ループのインデックス(i)のみで， 内側ループのインデックスは共有変数となってしまいます．しかし並列実行時に内側ループのインデックス変数が共有変数のままだと，各スレッドが互いに内側ループのインデックス変数(j)を上書きし合うために，内側ループを正しく実行することができません．そこで，内側ループのインデックス変数をプライベート変数とするよう，明示的に指示を追加する必要があります． 一方，Fortran言語では内側ループについても暗黙のプライベート宣言がなされる(Fortran OpenMP仕様に準拠した）処理系がほとんどです．

リダクション変数

ベクトルの総和を求める例

C 言語

```
double total(x)
double x[];
{
  int i;
  double t;
  t = 0.0;
  for (i = 0; i < 100; i++)
    t += x[i];
  return t;
}
```

Fortran

```
function total(x)
integer i
double precision t, total, x(100)
t = 0.0
do i = 1, 100
  t = t + x(i)
end do
total = t
end
```

単純にparallel do 指示文(parallel for 指示文) をループi の前に挿入して並列化した場合には，変数tの値は各スレッドで「共有」されているので，あるスレッドの演算が，他のスレッドの演算を上書きしてしまう危険が，スレッド実行開始のタイミングに依存してしまいます．

一方，変数t をプライベート変数として指示する場合，各スレッドが独自のt を持つので，上記の問題は解決しそうに見えますが しかし，他のスレッドのプライベート変数を直接参照する手段がないため，このままでは部分和は求められても，全スレッドの合計を計算することができません．

リダクション変数（2）

OpenMPでは「リダクション変数」と呼ぶ，プライベート変数と共有変数の両方の性質を持った変数を扱うことが出来ます．まず，並列ループで各スレッドがループを実行している間，リダクション変数はプライベート変数として扱われ，各スレッドはそれぞれのリダクション変数に対して独自の値を持ちます．その後全スレッドが並列ループの処理を終了すると，各スレッドのリダクション変数の値は一つにまとめられ，共有変数のように（マスター）スレッドの対応する変数に格納されます．リダクション変数を最後にまとめる際の計算方法には，和，積，最大値等が用意されています．

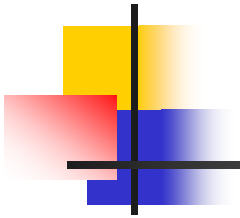
リダクション変数としたい変数がある場合，parallel for 指示文にreduction 指示節を用い，前頁のベクトルの総和を求める例では以下のようになります．

C 言語

```
double total(x)
double x[];
{
  int i;
  double t;
  t = 0.0;
#pragma omp parallel for reduction(+:t)
  for (i = 0; i < 100; i++)
    t += x[i];
  return t;
}
```

Fortran

```
function total(x)
implicit none
integer :: i
real(8) :: t, total, x(100)
t = 0.0
!$omp parallel do reduction(+:t)
do i = 1, 100
  t = t + x(i)
end do
total = t
return
end
```



（プログラム例）実数非対称疎行列と実数ベクトルの積

ハーウェルボーイング(Compressed Column Storage とも呼ばれる) 形式で格納された実数非対称疎行列

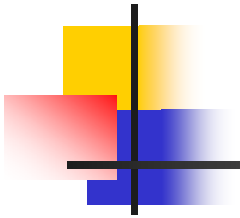
row_start, col_idx, a の3 つの1 次元配列を使用

col_idx は, 疎行列の非ゼロ要素の列番号

a にはその位置の非ゼロ要素の値

row_start には, 疎行列の各行について最初の非ゼロ要素の場所が格納されている.

よって，、i 行目の非ゼロ要素に対する「列番号」は,
col_idx(row_start(i)) から
col_idx(row_start(i+1)-1)) に格納されており，それに対応する疎行列の非ゼロ要素の値は,
a(row_start(i)) からa(row_start(i+1)-1) に格納されています.



簡単なOpenMP 並列プログラミングの例

例 疎行列ベクトル積

```
Matvec(double a[],int row_start,int col_idx[],
 double x[],double y[],int n)
{
   int i,j,start,end; double t;
#pragma omp parallel for private(j,t,start,end)
   for(i=0; i<n;i++){
      start=row_start[i];
      end=row_start[i+1];
      t = 0.0;
      for(j=start;j<end;j++)
         t += a[j]*x[col_idx[j]];
      y[i]=t;
  }
}
```

レポート課題（2006年1月10日出題）

【課題】

FFT(高速フーリェ変換)の数値計算に関して，以下の各項目について調査しレポートにまとめて提出してください．

(1)　FFTの数値計算アルゴリズム

(2)　FFTの並列計算の方法

(3) 理工学分野でFFTを利用している具体的な応用分野の説明

「提出締め切り日を変更」

提出先：aoyagi@cc.kyushu-u.ac.jp
Subject: 並列アルゴリズム課題
締め切り：平成18年1月31日（火）

形式：
text,pdf,ps,Word,Excel
ファイルの添付可