

OpenMPによる並列化実装

八木学

(理化学研究所 計算科学研究機構)

謝辞

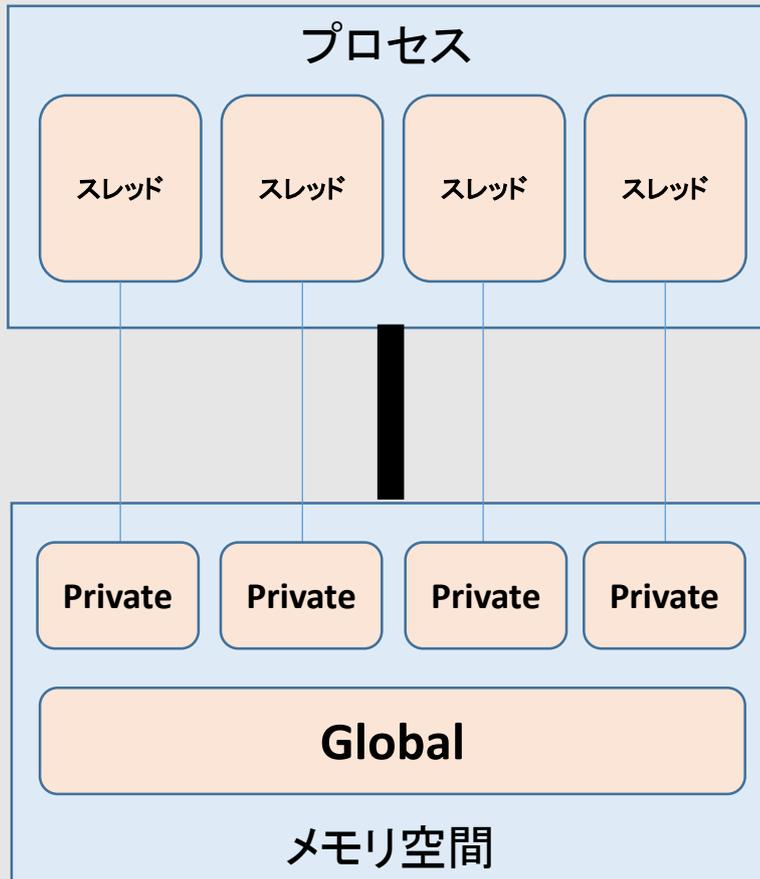
松本洋介氏(千葉大学)

KOBE HPC Spring School 2017

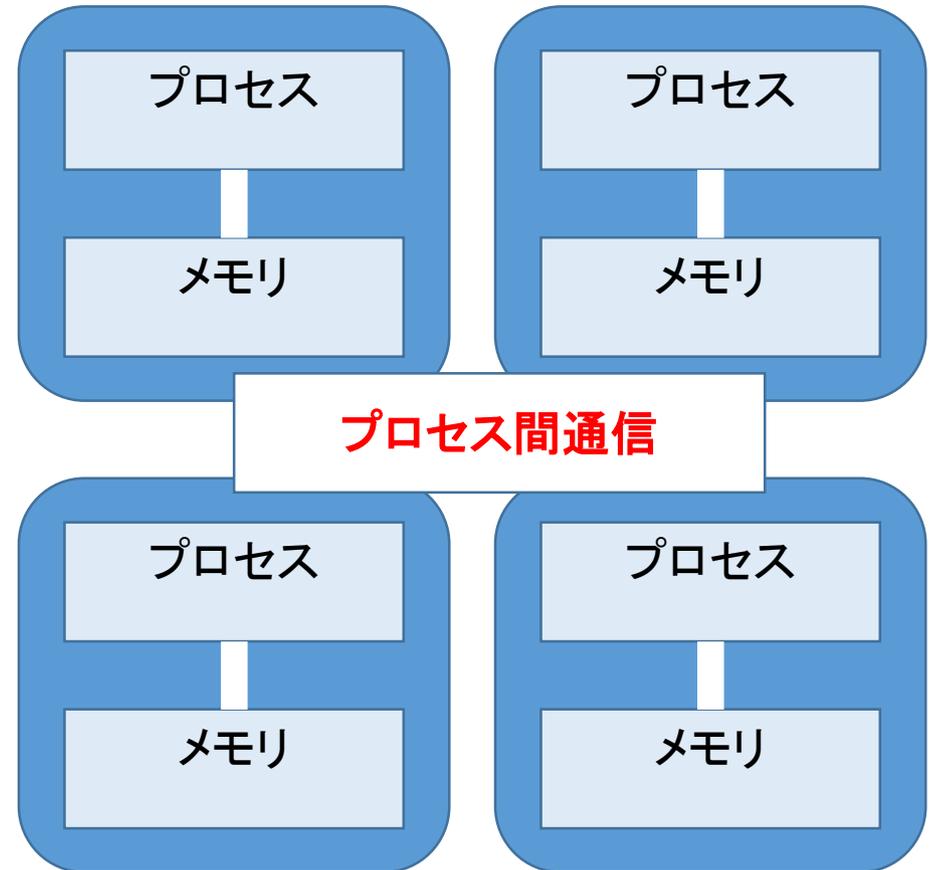
2017年3月14日 神戸大学計算科学教育センター

スレッド並列とプロセス並列

スレッド並列 OpenMP、自動並列化



プロセス並列 MPI



並列計算について

- CPUをN個使って並列計算した時、計算速度がN倍になるのが理想だが・・・
 - 並列化率の問題(アムダールの法則)
 - 通信時間ボトルネック

- 京のような大型計算機を有効利用するためには、如何に計算速度をN倍に近づけられるかが重要

アムダールの法則

プログラムの並列化できる割合を P とし、プロセッサ数を n とすると、並列計算した時の性能向上率は

$$\frac{1}{(1 - P) + \frac{P}{n}}$$

で与えられる。

9割並列化できるが、1割逐次処理が残ってしまうような場合、どれだけプロセッサを投入しても計算速度は10倍以上にはならない。

OpenMPとは

- 共有メモリ型計算機用の並列計算API
 - ノード内の並列（ノード間は不可）
- ユーザーが明示的に並列のための指示を与える
 - コンパイラの自動並列とは異なる
- 標準化された規格であり、広く使われている
- 指示行の挿入で並列化できるため、お手軽

スレッド数の指定

- ・シェルの環境変数で与える(推奨)

`export OMP_NUM_THREADS=16` (bashの場合)

`setenv OMP_NUM_THREADS 16` (tcshの場合)

- ・プログラム内部で設定することも可能

```
!$use omp_lib  
integer, parameter :: nthrd = 16  
  
call omp_set_num_threads(nthrd)
```

コンパイル

- ・コンパイルオプションでOpenMPを有効にする

```
gfortran/gcc -fopenmp test.f90
```

```
frtpx/fccpx -Kopenmp test.f90
```

- ・オプションを指定しない場合はOpenMPの指示行はコメントアウトとして認識される。

```
!$OMP PARALLEL DO F  
do i = 1, 100  
  a(i) = b(i) + c  
enddo  
!$OMP END PARALLEL DO
```

指示行は !\$OMP から始まる。Fortranでは、行頭の ! は通常はコメントを意味する。

Cの場合は #pragma omp という形になるが、Cでは行頭の # はコメントになる。

コンパイル

- ・コンパイルオプションでOpenMPを有効にする

```
gfortran/gcc -fopenmp test.f90
```

```
frtpx/fccpx -Kopenmp test.f90
```

- ・オプションを指定しない場合はOpenMPの指示行はコメントアウトとして認識される。

```
#pragma omp parallel for   
for (i=0; i<100; i++) {  
    a[i] = b[i] + c;  
}
```

指示行は !\$OMP から始まる。Fortranでは、行頭の ! は通常はコメントを意味する。

Cの場合は #pragma omp という形になるが、Cでは行頭の # はコメントになる。

OpenMPの基本関数

[F] use omp_lib **OpenMPモジュールをロード**

[C] #include <omp.h>

最大スレッド数取得 (Integer)

[F][C] nthreads = omp_get_num_threads()

自スレッド番号取得 (Integer)

[F][C] myid = omp_get_thread_num()

OpenMPの基本関数

時間を測る(倍精度型)

[F][C] time = **omp_get_wtime()**

```
real(8) :: dts, dte F  
dts = omp_get_wtime()  
... 処理 ...  
dte = omp_get_wtime()  
print *, dte-dts
```

```
double dts; C  
double dte  
dts = omp_get_wtime();  
... 処理 ...  
dte = omp_get_wtime();
```

OpenMPの基本命令

並列化の開始(スレッド生成)・終了

[F] !\$OMP PARALLEL ~ !\$OMP END PARALLEL

[C] #pragma omp parallel { }

Do/For ループを並列に計算する

[F] !\$OMP DO ~ !\$OMP END DO

[C] #pragma omp for { }

OpenMPの基本命令

[F] !\$OMP PARALLEL

[F] !\$OMP DO

[F] → !\$OMP PARALLEL DO とまとめて書ける

[C] #pragma omp parallel

[C] #pragma omp for

[C] → #pragma omp parallel for でもよい

スレッドの同期待ちをしない

[F] !\$OMP DO ~ !\$OMP END DO **NOWAIT**

[C] #pragma omp for NOWAIT

OpenMPの基本命令

1スレッドのみで処理

[F] !\$OMP SINGLE ~ \$OMP END SINGLE

[C] #pragma omp single { }

プライベート変数を指定

[F] !\$OMP PRIVATE(a)

[F] !\$OMP DO PRIVATE(a) ~ !\$OMP END DO

[C] #pragma omp private(a) { }

[C] #pragma omp parallel for private(a) { }

OpenMPの基本命令

Section構文

スレッドごとに処理を分岐させる

```
!$OMP PARALLEL SECTIONS  
!$OMP SECTION  
    処理1  
!$OMP SECTION  
    処理2  
!$OMP SECTION  
    ...  
!$OMP END PARALLEL  
SECTIONS
```

F

```
#pragma omp parallel sections  
{  
#pragma omp section  
    処理1;  
#pragma omp section  
    処理2;  
#pragma omp section  
    ...;  
}
```

C

OpenMPの基本命令

!\$OMP PARALLEL

F

!\$OMP DO

do i = 1, 100

 a(i) = i

enddo

!\$OMP END DO

!\$OMP SINGLE

call output(a)

!\$OMP END SINGLE

!\$OMP DO

do i = 1, 100

.....

!\$OMP END DO

!\$OMP END PARALLEL

!\$OMP PARALLELによるスレッドの立ち上げ回数は極力減らすほうがオーバーヘッドが少ない。

逐次処理やデータのファイル出力のような処理が入る場合は、!\$OMP SINGLEで対処

OpenMPの基本命令

```
#pragma omp parallel {  
  #pragma omp for {  
    for (i=0; i<100; i++) {  
      a[i] = i  
    }  
  }  
}
```

#pragma omp single{

```
  ...  
}  
  
#pragma omp for {  
  for (...)  
  .....  
}  
}
```



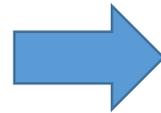
#pragma omp parallel によるスレッドの立ち上げ回数は極力減らすほうがオーバーヘッドが少ない。

逐次処理やデータのファイル出力のような処理が入る場合は、#pragma omp singleで対処

OpenMPの基本命令

起こりがちなミス

```
!$OMP PARALLEL DO
do i = 1, 100
  tmp = myfunc(i)
  a(i) = tmp
enddo
!$OMP END PARALLEL DO
```



```
!$OMP PARALLEL DO
!$OMP PRIVATE(tmp)
do i = 1, 100
  tmp = myfunc(i)
  a(i) = tmp
enddo
!$OMP END PARALLEL DO
```

tmpを各スレッドで上書きしてしまう
ため、正しい結果にならない

private宣言を入れる

並列化したループ内で値を設定・更新する場合は要注意
→privateにすべきではないか確認する必要あり

注意点等

- ・自動並列化と違い、並列化できるかどうかの判断はプログラマが行う
- ・依存関係などにより並列化できないループであっても、明示してしまえば並列化されてしまう。
- ・スレッド内でのグローバル変数、プライベート変数を間違えるとRunごとに結果が変わってしまう。
 - 数回実行し、結果が変わらないことを確認
- ・OpenMPは手軽だが、デバッグには注意が必要

演習問題2-1

次のサンプルプログラムをOpenMPで並列化せよ

1d_v102.f90 : 1次元移流(3次風上)

余力があれば

1d_v204.f90 : 1次元流体(3次風上+4次中心差分)

2d_v101.f90 : 2次元流体

にも挑戦！

補足：πコンピュータ上の操作

ジョブスクリプトの投入

```
[xxxx@pi]$ frtpx -Kopenmp 1d_v102omp.f90
```

```
[xxxx@pi]$ pjsub run.sh
```

```
run.sh
```

```
#!/bin/sh
```

```
#PJM -L "node=1"
```

ノード数

```
#PJM -L "elapse=00:01:00"
```

実行時間

```
#PJM =L "rscgrp=small"
```

実行キュー

```
#PJM -j
```

```
#PJM -o "output.lst"
```

標準出力ファイル名

```
export OMP_NUM_THREADS=16
```

スレッド数を指定

```
./a.out
```

実行ファイル

演習問題2-2

OpenMPを用いて並列化したプログラムを、スレッド数を変えて実行し処理時間を計測せよ

スレッド数の指定方法: スクリプト run.sh の
OMP_NUM_THREADS=xx の数値を変更

処理時間の計測

```
dts = omp_get_wtime()  
... 処理 ...  
dte = omp_get_wtime()  
print *, dte-dts
```