OpenMPを使ってみる

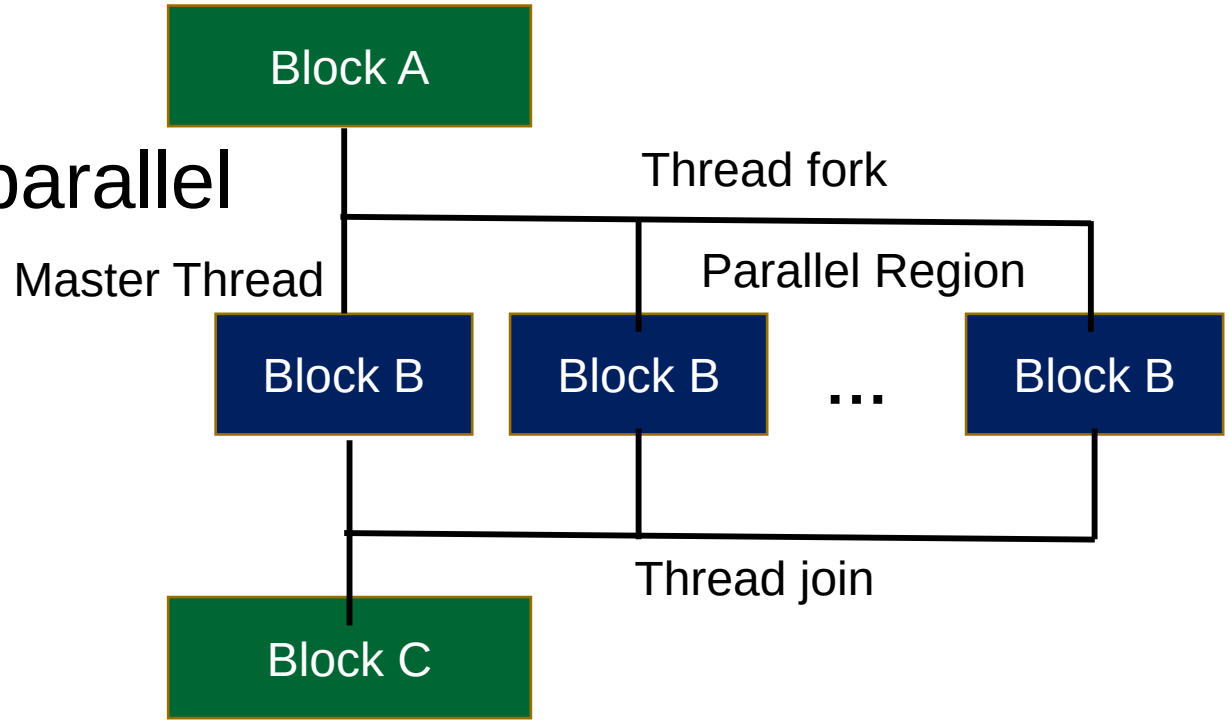
天野

OpenMP

- プログラムを並列化するためのプラグマ(プログラムに対する指示文:directive)、ライブラリ、環境変数からできている並列プログラム環境
  - #pragma…をdirectiveと呼び、この中で用いる特殊な構文をsub-directiveと呼ぶ
- 共有メモリを使うため、データを分散しなくて良い ↔ MPI
- 比較的小規模のマルチコアシステム向き
- 大規模なシステムでは高度の最適化が必要

OpenMPの実行モデル

```
Block A
#pragma omp parallel
{
        Block B
{
Block C
```



環境変数: OMP_NUM_THREADS で実行スレッド数を設定

並列化の単位となる構文

- #pragma omp parallel内で並列処理を記述
  - for (do)
  - sections
  - single (master)
- 一文で実行と制御を兼ねる
  - parallel for
  - parallel section

for 文

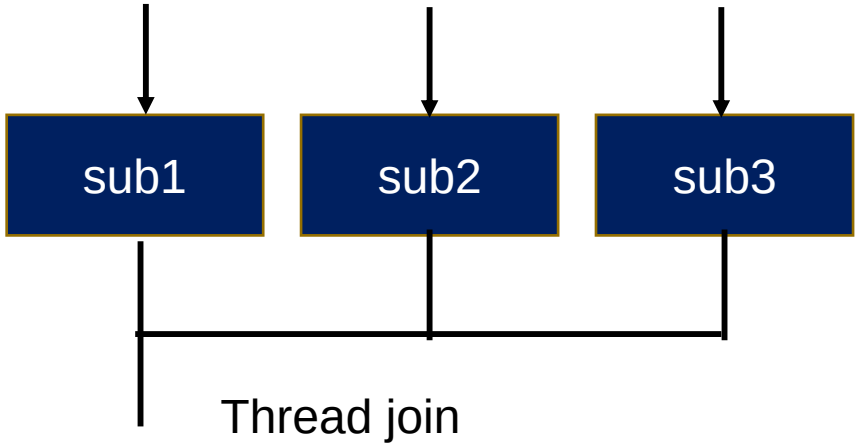
反復は各スレッドに等分に割り当てられる

```
# pragma omp parallel
{
#pragma omp for
        for(i=0; i<1000; i++) {
                c[i]=a[i]+b[i];
        }
}
```

```
# pragma omp parallel for
        for(i=0; i<1000; i++) {
                c[i]=a[i]+b[i];
        }
```

sections 文

```
#pragma omp parallel sections
{
#pragma omp section
    sub1();
#pragma omp section
    sub2();
#pragma omp section
    sub3();
}
```



三つの違った処理が並列に実行され、
終了時に同期される

private sub-directive

```
# pragma omp parallel for private(c)
        for(i=0; i<1000; i++) {
                d[i]=a[i]+c*b[i];
        }
```

c は各スレッドにコピーされる → 高速実行が可能

private sub-directiveの利用

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

この文をprivate(j)なしに実行したらどうなるだろう？→
全てのスレッドでjが更新される→エラー！

reduction sub-directive

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

リダクション演算は、データを足し込んでいく演算。
良く用いられるが、並列実行は、このsub-directiveを使わないと難しい

組み込み関数

- omp_get_num_threads();
  - 並列実行されるスレッド数を返す。
- omp_get_thread_num();
  - 自分のスレッド番号を返す。
- omp_get_max_threads();
  - 並列実行可能な最大スレッド数を返す
- 使い方

```
#include <omp.h>
int nth, myid;
nth = omp_get_num_threads();
myid = omp_get_thread_num();
```

時間を計る: omp_get_wtime();

```
#include <omp.h>
double ts, te;
ts = omp_get_wtime();

   実行

te = omp_get_wtime();
printf(“time[sec]:%lf¥n”,te-ts);
```

他のpragma

- single:

  #pragma omp single

  { blocks….. }

  指定されたブロック内の文を単一スレッドに割り当てる
- master:

  #pragma omp master

  { blocks..... }

  指定されたブロック内の文をマスタースレッドに割り当てる。

テスト環境

- http://www.am.ics.keio.ac.jp/arc からOpenMPの演習資料ex18.tarをダウンロード
- tar xvf ex18.tarで解凍

コンパイルと実行

gcc –fopenmp hello.c –o hello

./hello

Hello OpenMP world from 2 of 8

….

ここではスレッド数は8に設定されている。

これは、環境変数OMP_NUM_THREADS をコマンドラインで設定することで変更できる。

例

export OMP_NUM_THREADS=6

./reduct

じっさいのコア数を超える設定も可能だが速くならない

例題プログラムreduct4k.c

- 乱数で作った配列aとbの積を計算してcに入れる
- このcの要素の全てを足してsumに入れる（リダクション演算）

- extport OMP_SUM_THREADS=xにより、スレッド数を1，2，3，4，6，8に設定して実行し、実行時間を計測してみよ。

演習ex1s.c

- 下のプログラムex1sをpragmaを入れて高速化せよ。スレッド数を1，2，4と変えて実行時間を測定して高速化されていることを確認せよ。
- 提出物はpragmaの入ったプログラム

```
 sum = 0.0;
for (i=0; i<N; i++) {
        c[i]=0.0;
      for(j=0; j<N; j++)
       c[i] += (a[i]-b[j])*(a[i]-b[j]); }
for (i=0; i<N; i++) sum+=c[i];
```