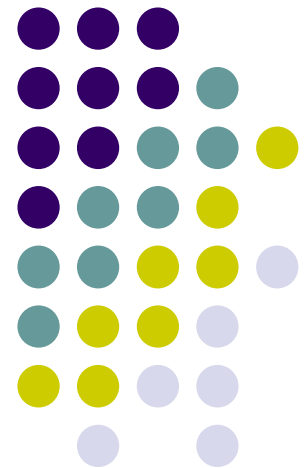


2016年度 実践的並列コンピューティング 第3回

OpenMPによる
共有メモリプログラミング(1)

遠藤 敏夫

endo@is.titech.ac.jp



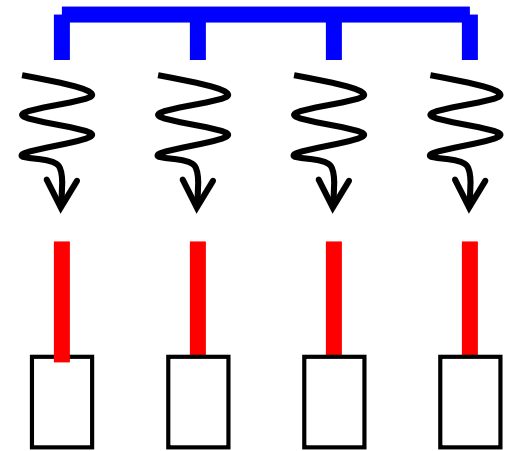
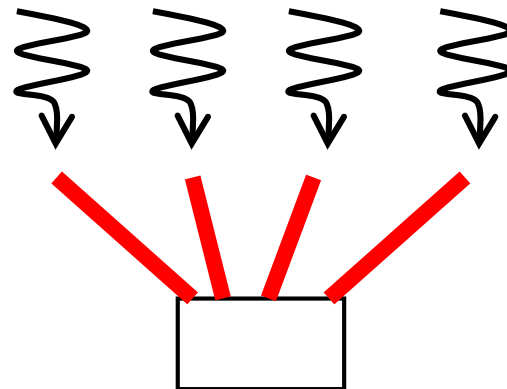
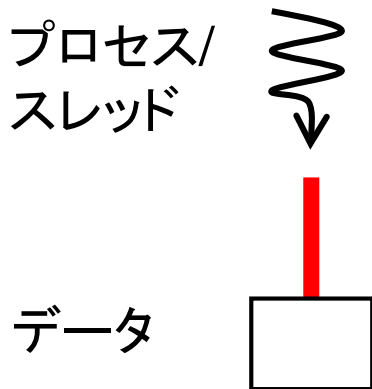
並列プログラミングモデルの メモリモデルによる分類



逐次

共有メモリモデル

分散メモリモデル

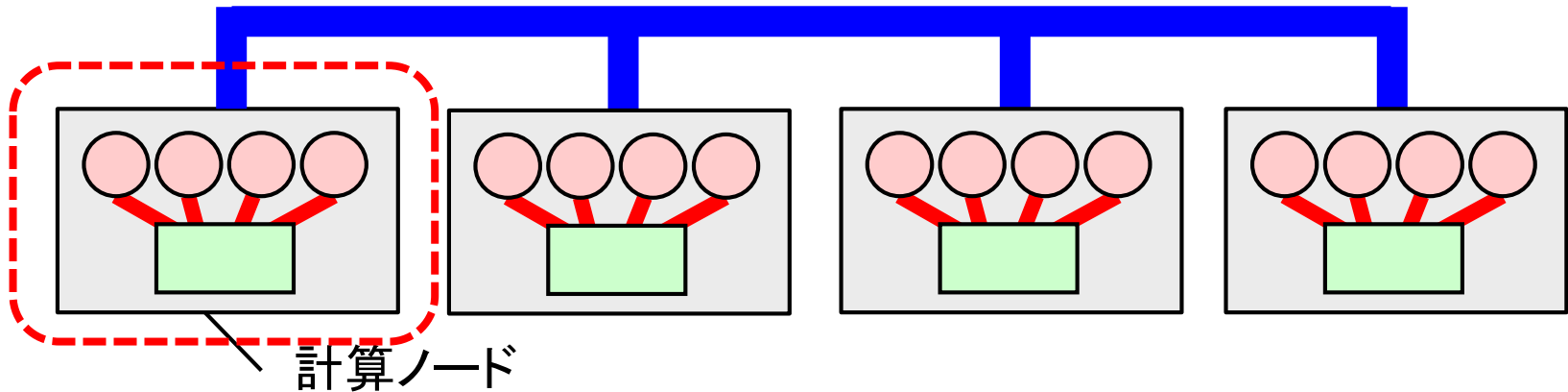


- スレッド達が共通の
データにアクセス可能
- **OpenMP**(言語拡張)
 - **pthread**(ライブラリ)

- プロセス間では通信
が必要
- **MPI** (ライブラリ)
 - **socket** (ライブラリ)



OpenMPの利用可能な計算資源

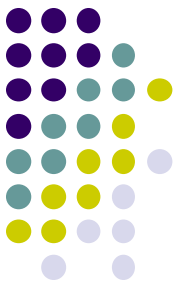


- 一つのOpenMPプログラムが使えるのは一計算ノード中のCPUコアたち
 - TSUBAMEの場合は12コア
- 複数計算ノードを(一プログラムから)用いたい場合は、MPIなどが必要
 - ただしMPIよりOpenMPのほうがとっつきやすい



OpenMPとは

- 共有メモリモデルによる並列プログラミングAPI
- C言語, C++, Fortranに対応
- 並列化のための指示文や, ライブラリ関数
 - 指示文: #pragma omp ~~
- 基本はFork-Joinモデル
- 変数は基本的にスレッド間で共有
 - ⇒ 以下を明示的に記述
 - タスク分割
 - スレッド間同期
 - 変数の共有・プライベートの区別



サンプルプログラムについて

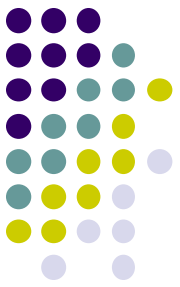
TSUBAME2の~endo-t-ac/ppcomp/16/ ディレクトリ以下

(1) ここから、下記青字のサブディレクトリを、各自のホームディレクトリのどこかにコピーしてください

- 円周率 ([pi](#), [pi-omp](#))
- 行列積 ([mm](#), [mm-omp](#))
- 熱拡散計算 ([diffusion](#))

(2) 各サブディレクトリの中で“make”コマンドで、コンパイルできるようになっている

→ pi, mmなどの実行コマンドができる



サンプルプログラムについて

(3-1) 通常版の実行:

- 円周率 (**pi**)
 - 実行例: `./pi 1000000`
- 行列積 (**mm**)
 - 実行例: `./mm 500 500 500`
- 熱拡散計算 (**diffusion**)
 - 実行例: `./diffusion`

(3-2) OpenMP版の実行 (**pi-omp**, **mm-omp**)

- `export OMP_NUM_THREADS=4` などとしてから上記を実行



OpenMPプログラムのコンパイル

メジャーなコンパイラはすでにOpenMPに対応

- PGIコンパイラ (pgcc)
 - コンパイル時・リンク時に`-mp`オプション
- Intelコンパイラ (icc)
 - コンパイル時・リンク時に`-openmp`オプション
- GCC 4.2以降 (gcc)
 - コンパイル時・リンク時に`-fopenmp`オプション



OpenMP規格のバージョン

- OpenMP規格も年々バージョンアップ → 機能追加
- コンパイラ種類・そのバージョンによって、対応するOpenMPのバージョンが異なる
 - OpenMP 3.0からはタスク並列が追加
→ 再帰関数などの並列化も可能に
 - OpenMP 4.0からはアクセラレータ対応

```
printf("%d¥n", _OPENMP);
```

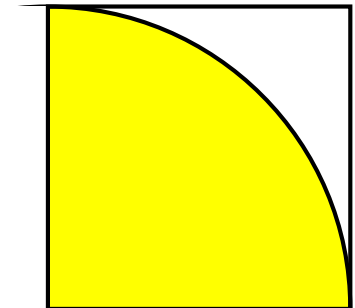
- 200505 が表示 → OpenMP 2.5 (gcc 4.3.4など)
- 200805 が表示 → OpenMP 3.0 (pgcc 15.4など)
- 201307 が表示 → OpenMP 4.0 (icc 15.0など)



サンプルプログラム: pi

モンテカルロ法による擬似(いい加減な)円周率計算

- 正方形にランダムに n 回矢を射って, 四半円に入る確率 $\times 4 \doteq$ 円周率
- 実行オプション: `./pi [n]`
- 計算量: $O(n)$
- pi-ompディレクトリにOpenMP版あり



OpenMP並列実行の基本: 並列Region



```
#include <omp.h>
```

```
int main()  
{
```

```
    A;
```

```
    #pragma omp parallel
```

```
    {
```

```
        B;
```

```
    }
```

```
    C;
```

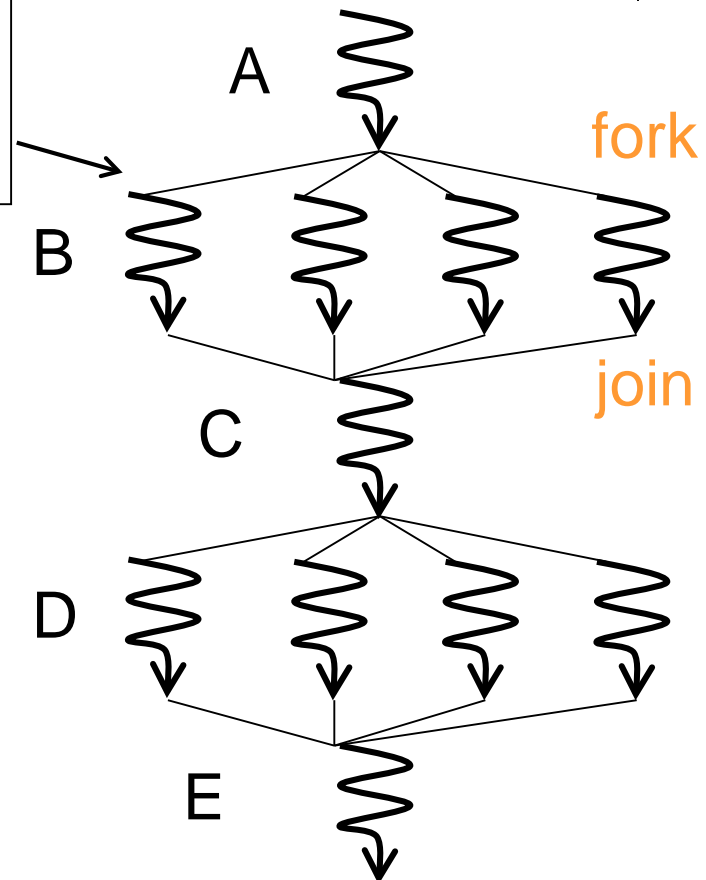
```
    #pragma omp parallel
```

```
    D;
```

```
    E;
```

```
}
```

ここから
4threadsで
並列実行



#pragma omp parallelの直後の文・ブロックは並列Regionとなる
並列Regionから呼ばれる関数も並列実行



スレッド数の指定

- (1) プログラム外で, OMP_NUM_THREADS環境変数
 - インタラクティブノードなら コマンドラインで
export OMP_NUM_THREADS=12など
 - t2subを使う場合には「TSUBAME利用の手引き」など参照
- (2) プログラム内で
 - omp_set_num_threads(n)関数
 - #pragma omp parallel num_threads(n)



スレッド数の取得

- 全スレッド数の取得
 - `omp_get_num_threads()`関数
「全体で何人いるか？」
 - 自スレッドの番号の取得
 - `omp_get_thread_num()`関数
 - 0以上、「全スレッド数」未満
- ⇒番号によって違う処理をさせることができる



OpenMPの指示文

以下は並列region内で使われる

- `#pragma omp critical`
 - 次のブロック・文が「critical section」となる
 - 同時にcritical sectionを実行できるのみのみは1スレッドのみ、となる
- `#pragma omp barrier`
 - スレッド間でバリア同期をとる: 全スレッドの進行がそろうまで待つ
 - ただし並列regionの終わりでは, 自動的に全スレッドを待つ(暗黙のbarrier)
- `#pragma omp single`
 - 次のブロック・文を1スレッドのみで実行する
- `#pragma omp for` (後述)



変数のデータ共有属性(1)

OpenMPでは、変数はスレッドによって共有されるか否か、注意
「基本的には」

- 並列Region外で宣言された変数 ⇒ 共有変数
- 並列Region内で宣言された変数 ⇒ プライベート変数

明示的に指定もできる

```
{  
    int s = 1000; shared  
    #pragma omp parallel  
    {  
        int i; private  
        i = func(s, omp_get_thread_num());  
        printf("%d¥n", i);  
    }  
}
```

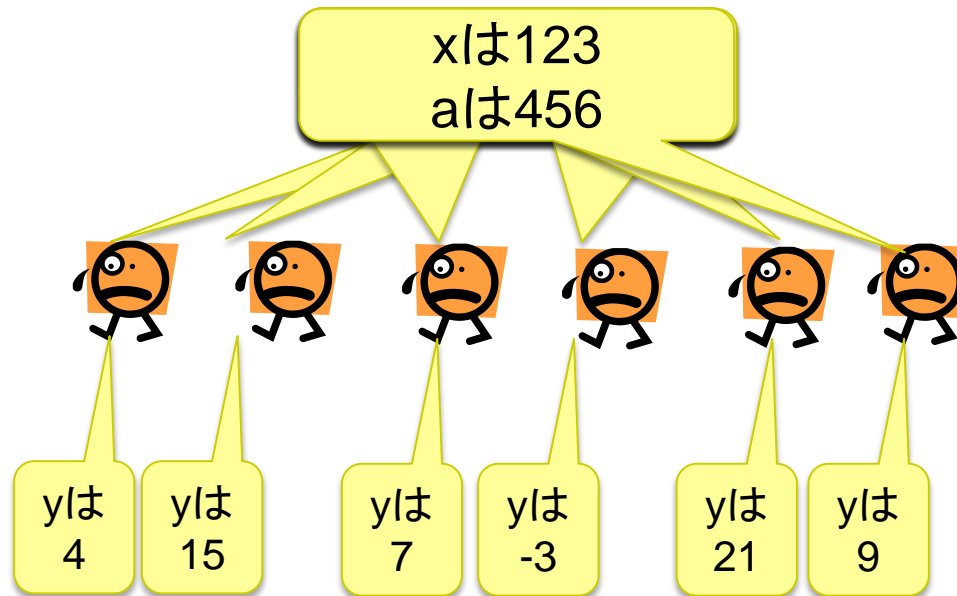
```
int func(int a, int b)  
{  
    int rc = a+b;  
    return rc;  
}
```

private



変数のデータ共有属性(2)

- 変数xとaが共有
- 変数yがプライベートのとき



- 共有変数を誰かが書き換えると、ほかのスレッドにも伝わる



共有/プライベートの落とし穴

- 二重以上のループがあり、並列性を持つプログラム
- 外側ループを並列化することにした
- 以下の「いかにもokそうなプログラム」はバグあり
 - コンパイルは通るのに、期待通りに動かないやっかいなケース

```
int i, j;  
#pragma omp parallel for  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        ...  
    }  
}
```

i,jの宣言がparallelの外側にあるので共有変数。jがまずい



共有/プライベートの落とし穴(2)

- 修正方法

```
int i;  
#pragma parallel for  
for (i = 0; i < m; i++) {  
    int j; // j はプライベート  
    for (j = 0; j < n; j++) {  
        ...  
    } }  
}
```

```
int i, j;  
#pragma parallel for private(j)  
// jは外側で宣言されているがプライベートに  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        ...  
    } }  
}
```



時間計測の手法

- gettimeofday関数
 - 実時間(CPU時間でなく)が計れ, かつ精度がマイクロ秒
 - clock関数は精度が低い

```
#include <stdio.h>
#include <sys/time.h>
:
{
    struct timeval st, et;
    long us;
    gettimeofday(&st, NULL); /* 開始時刻を記録 */
    ...計測したい部分...
    gettimeofday(&et, NULL); /* 終了時刻を記録 */
    us = (et.tv_sec-st.tv_sec)*1000000+
        (et.tv_usec-st.tv_usec); /* 時刻の差分 */
}
```

OpenMPのワークシェアリング

構文: for



単なる“omp parallel”よりも、気軽に並列化の記述可能！

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

- “omp for”の直後のfor文は、複数スレッドにより並列実行される
- 左のプログラムが、もし4スレッドで実行されるならスレッドあたり25ずつ仕事
 - ・ ループ回数÷スレッド数が割り切れなくてもok

- omp parallelとomp forをまとめてomp parallel forとも書ける
- 残念ながら、どんなforでも対応できるわけではない。詳細は次回以降

For指示文のオプション: reduction



- For文にありがちなこと: 各反復の結果を一変数に集計する (例: piプログラムのカウンタ)
- Criticalセクションで更新することもできるが非効率 ⇒ reductionオプションが用意されている

```
{
    int count = 0;
    #pragma omp parallel
    {
        #pragma omp for reduction (+:count)
        for (i = 0; i < 100; i++) {
            count += f(i);
        }
    }
}
```

演算子:

+, -, *, &&, || など

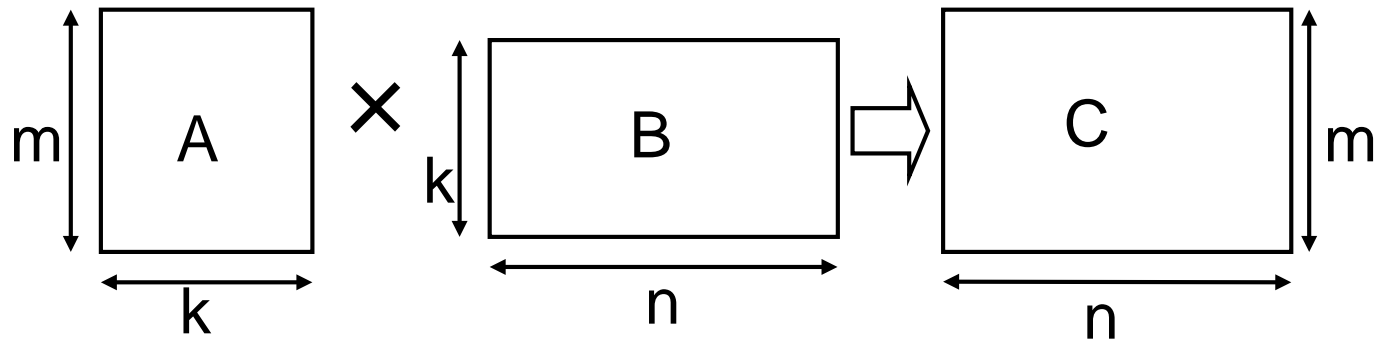
- Max, minは無し
- Fortran版には有り



サンプルプログラム:mm

$(m \times k)$ 行列と $(k \times n)$ 行列の積

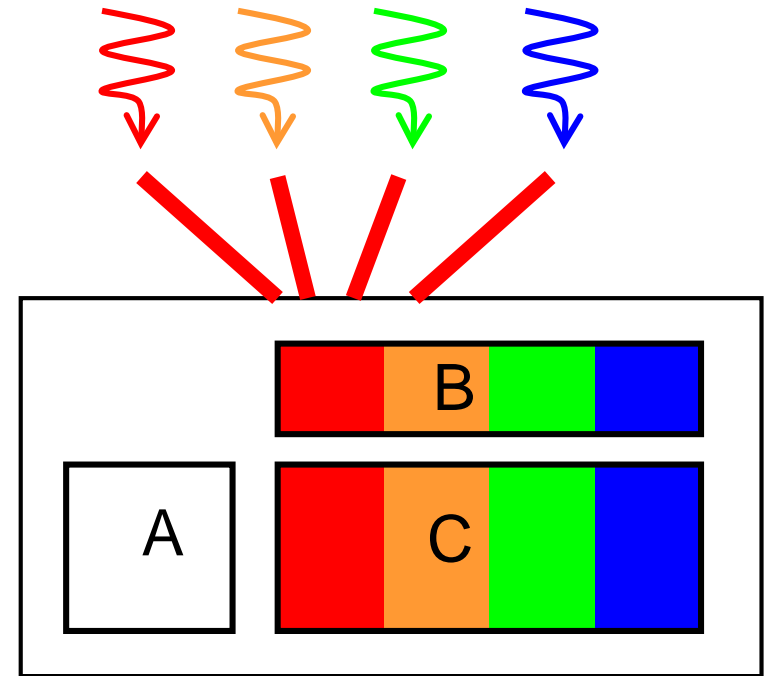
- 三重のforループで記述
- 動的な長さの配列. 二次元を一次元で表現
 - Column major format
- 実行オプション: `./mm [m] [n] [k]`
- 計算量: $O(mnk)$



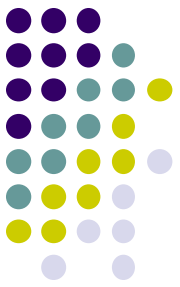


MmのOpenMPによる並列化

- 三重ループの最外ループを並列化
 - #pragma omp parallel for
 - nをスレッド間で分割することになる



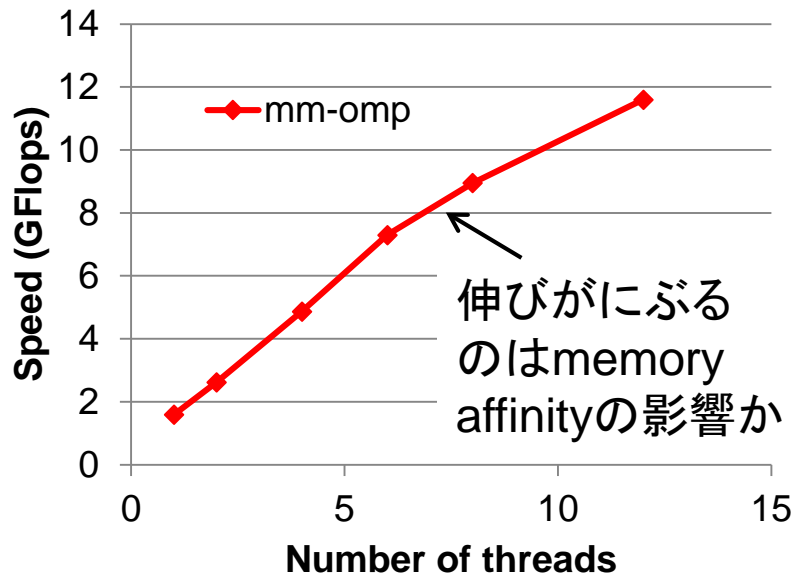
行列Aは全スレッドによってアクセスされる



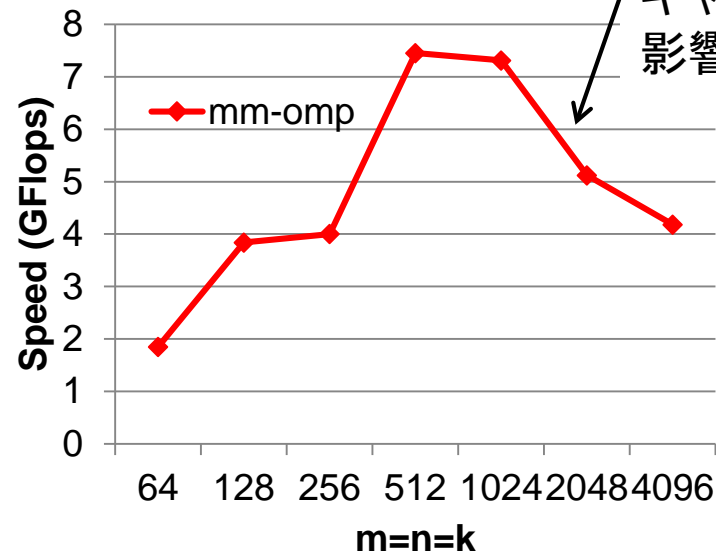
Mm-ompの性能

- TSUBAME2ノード上(Xeon X5670 2.93GHz 12core)
- OMP_NUM_THREADS環境変数によりスレッド数指定
- (2mnk/経過時間)にてFlops単位の世界速度を取得

m=n=k=2000固定、
スレッド数を変化



4スレッド、
m=n=kを変化



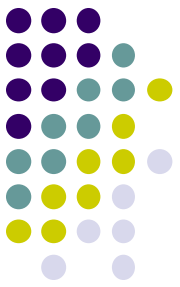


本授業のレポートについて

- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

予定パート:

- OpenMPパート
- MPIパート
- GPUパート



OpenMPパート課題説明 (1)

以下の[O1]—[O3]のどれか一つについてレポートを提出してください

[O1] diffusionサンプルプログラム(次回以降説明)を、OpenMPで並列化してください。

オプション:

- 配列サイズや時間ステップ数を可変パラメータにしてみる。引数で受け取って、配列をmallocで確保するようにする、など。
- より良いアルゴリズムにしてみる。ブロック化・計算順序変更でキャッシュミスを減らせないか？



OpenMPパート課題説明 (2)

[O2] sortサンプルプログラム(次回以降説明)を、OpenMPで並列化してください.

- 注意: OpenMP3.0以上のtask対応コンパイラである必要
 - TSUBAMEではpgcc (-mpつき)や icc (-openmpつき)

オプション:

- クイックソート以外のアルゴリズムではどうか?
 - ヒープソート? マージソート?

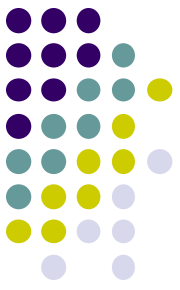


OpenMPパート課題説明 (3)

[O3] 自由課題: 任意のプログラムを, OpenMPを用いて並列化してください.

- 単純な並列化で済む問題ではないことが望ましい
 - スレッド・プロセス間に依存関係がある
 - 均等分割ではうまくいかない、など
- たとえば, 過去のSuperConの本選問題
<http://www.gsic.titech.ac.jp/supercon/>
たんぱく質類似度(2003), N体問題(2001)・・・
入力データは自分で作る必要あり
- たとえば, 自分が研究している問題

課題の注意



- いずれの課題の場合も、レポートに以下を含むこと
 - 計算・データの割り当て手法の説明
 - TSUBAME2などで実行したときの性能
 - プロセッサ(コア)数を様々に変化させたとき
 - 問題サイズを様々に変化させたとき(可能な問題なら)
 - 「XXコア以上で」「問題サイズXXX以上で」発生する問題に触れているとなお良い
 - 高性能化・機能追加などのための工夫が含まれているとなお良い
 - 「XXXのためにXXXを試みたが高速にならなかった」のような失敗でもgood
 - 作成したプログラムも提出
 - zipなどで圧縮してOCW-ilに提出
 - 困難な場合は、TSUBAME2の自分のホームディレクトリに置き、置き場所を連絡(パーミッションに注意)



課題の提出について

- OpenMPパート提出期限
 - 5/19 (木) (予定)
- OCW-i ウェブページから下記ファイルを提出のこと
- レポート形式
 - レポート本文: PDF, Word, テキストファイルのいずれか
 - プログラム: zip形式に圧縮するのがのぞましい
- OCW-iからの提出が困難な場合、メールでもok
 - 送り先: ppcomp@el.gsic.titech.ac.jp
 - メール題名: ppcomp report



次回:

- OpenMP(2)
 - Diffusion: ステンシル計算のサンプルプログラム
 - 排他制御