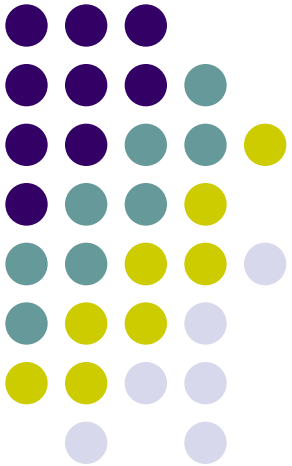
2013年度
実践的並列コンピューティング
第3回



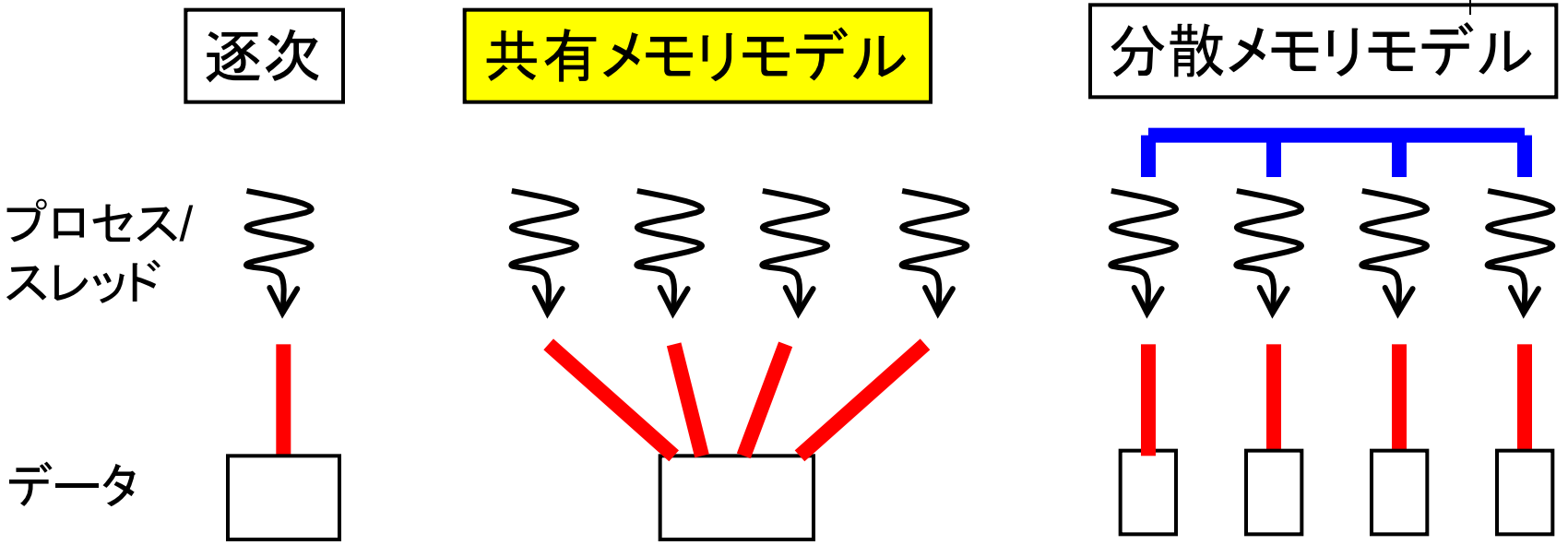
OpenMPによる
共有メモリプログラミング(1)

遠藤 敏夫

endo@is.titech.ac.jp

2013年4月22日

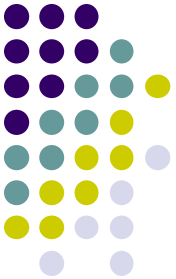
並列プログラミングモデルのメモリモデルによる分類



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)
- CUDA (言語拡張)

プロセス間では通信が必要
- MPI (ライブラリ)
- socket (ライブラリ)
- UPC, HPF(言語拡張)

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

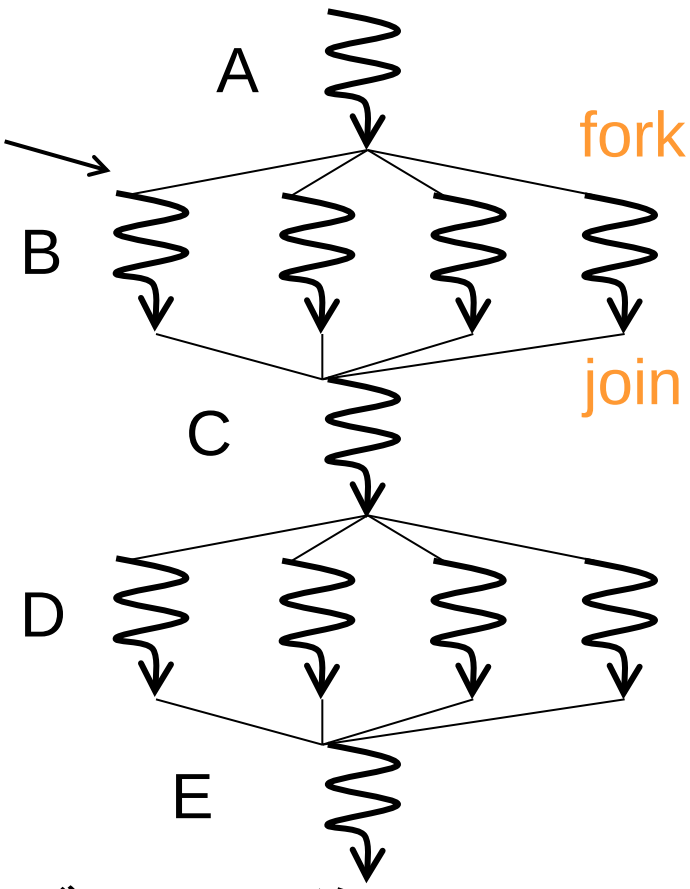
  ⇒以下を明示的に記述
  - タスク分割
  - スレッド間同期
  - 変数の共有・プライベートの区別

並列実行の基本:
OpenMPの並列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

- 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を使う場合にはマニュアル参照

  (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
- 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

変数のデータ共有属性

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　⇒　共有
- 並列Region内で宣言　⇒　プライベート

属性を明示的に指定もできる

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

shared

private

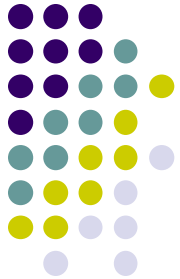
```
int func(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

private

PthreadやJava threadとの違いに注意

- Pthreadでは局所変数はプライベート

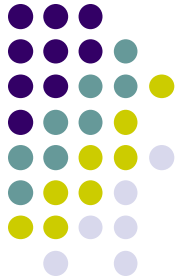
OpenMPの指示文



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)

サンプルプログラムについて

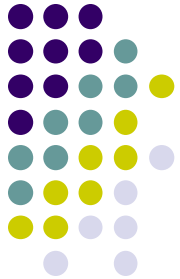


TSUBAME2.0の~endo-t-ac/ppcomp/13/ ディレクトリ以下

ここから、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)
  - 実行例：./pi 1000000
- 行列積 (mm, mm-omp)
  - 実行例: ./mm 500 500 500
- 移流計算 (advection)
  - 実行例: ./main 1
  - GPUチャレンジ2010課題

OpenMPプログラムのコンパイル



OpenMP対応コンパイラは近年増加

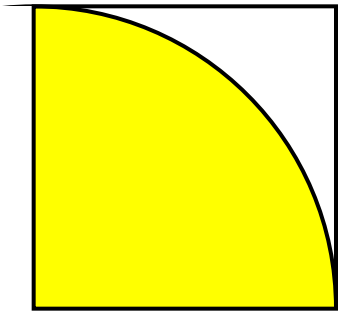
- PGIコンパイラ (pgcc)
  - コンパイル時・リンク時に-mpオプション
- Intelコンパイラ (icc)
  - コンパイル時・リンク時に-openmpオプション
- GCC 4.2以降
  - コンパイル時・リンク時に-fopenmpオプション

サンプルプログラム：pi

モンテカルロ法による円周率計算

- 正方形にランダムにn回矢を射って，四半円に入る確率×4≒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

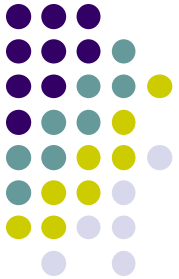


時間計測の手法

- 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



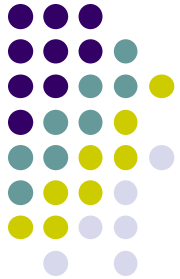
単なる”omp parallel”よりも、気軽に並列化の記述可能

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

- 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);
      }
   }
}
```

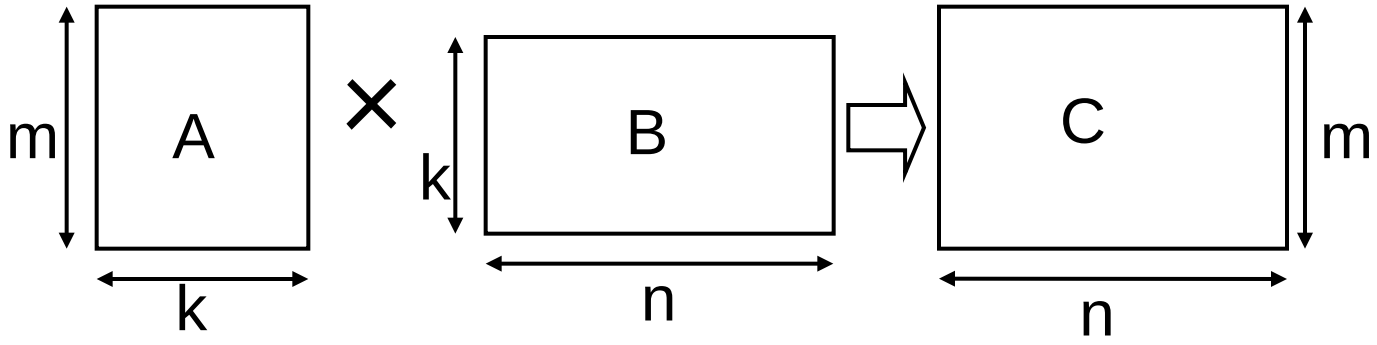
演算子:
+, -, *, &&, || など

- Max, minは無し
- Fortran版には有り

サンプルプログラム：mm

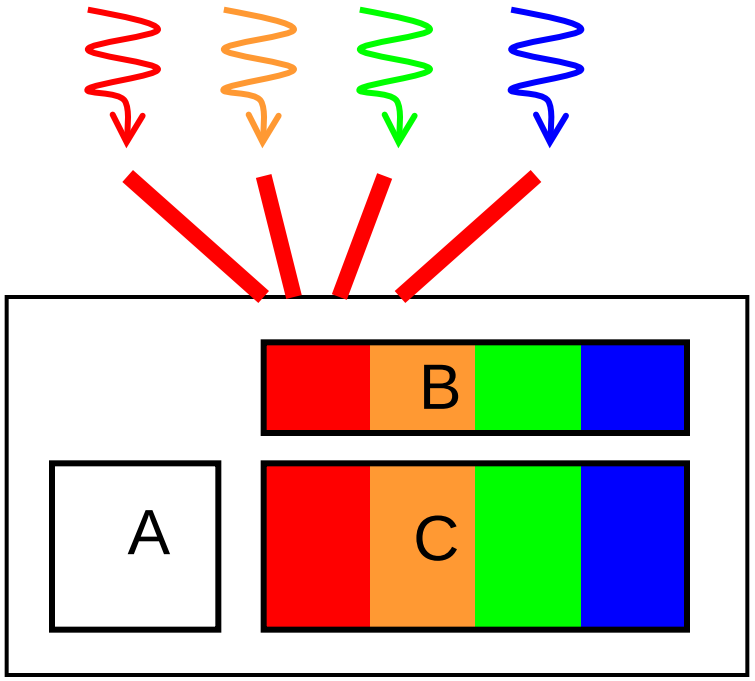
(m×k)行列と(k×n)行列の積

- 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

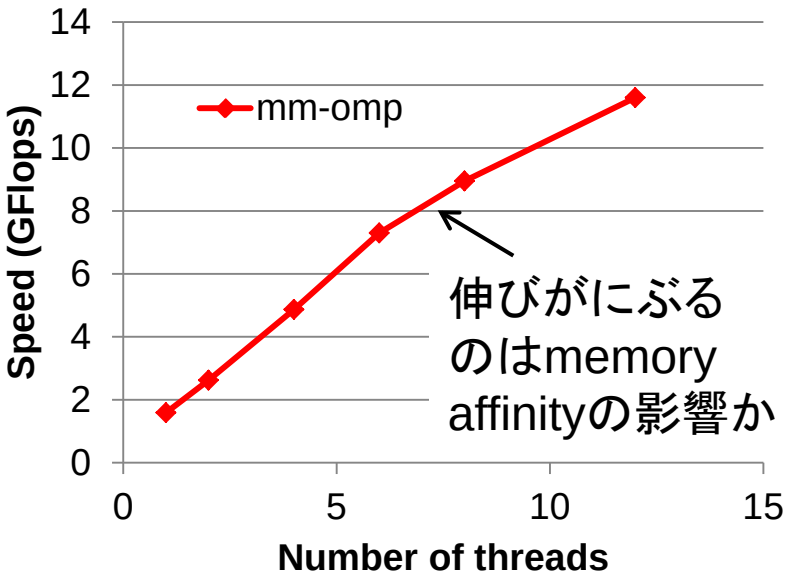


行列Aは全スレッドによってアクセスされる

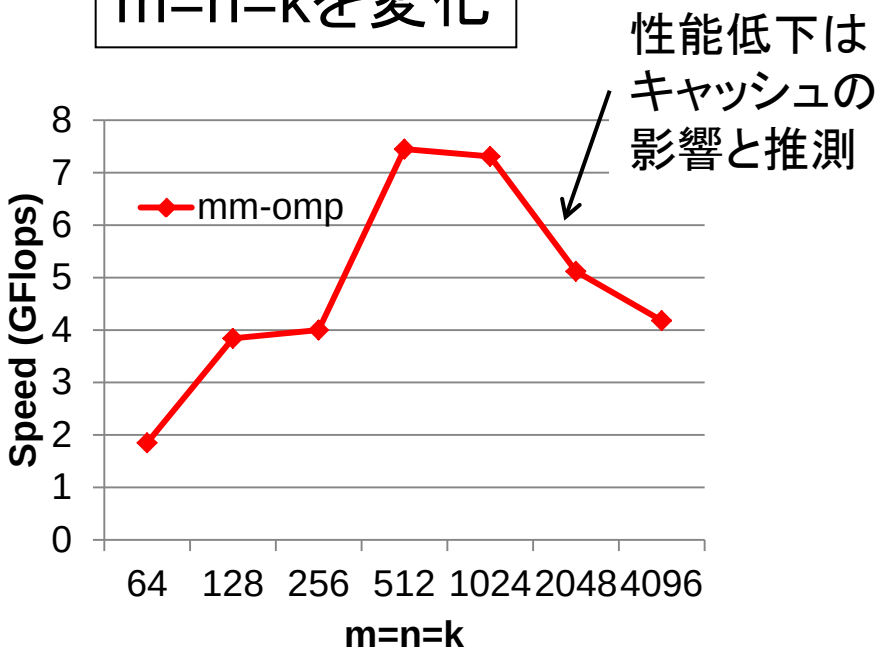
Mm-ompの性能

- 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

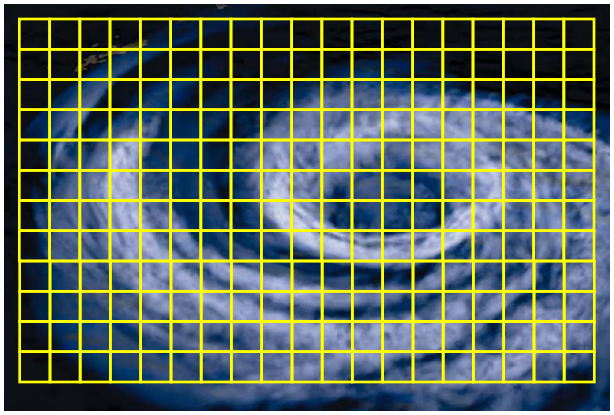


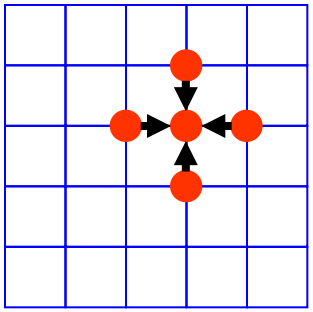
サンプルプログラム: Advection

二次元格子空間の，流体の簡単な流れをシミュレート

詳細は http://sacsis.hpcc.jp/2010/gpu/

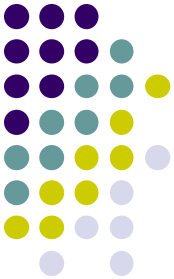
- f(x,y)がその点の「インク濃度」
- 空間の端のfと，流速u, vが与えられたときの，時間経過の様子
- きめられた時間ステップ計算したら終了
- ダブルバッファリング





隣の点の情報を用いて値更新
(正確には最大2つ隣り)

サンプルプログラム: Advection (続き)

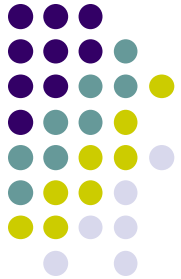


- 実行オプション: ./main [pno]
  - Pno: 問題番号 = 1～3, 11～15
    あとの問題のほうが時間がかかる. gpucapi.c参照
- 計算量: O(nx×ny×nt)
  - Nx, ny: 空間サイズ. nt: 時間ステップ数

これを並列化するには??

- 空間ループをomp forで並列化が良い. 結果的に空間を分割して, スレッドたちで分担することになる.
- Barrier同期も必要(次回)
- 時間ループにomp forをつけてはいけない！なぜか？

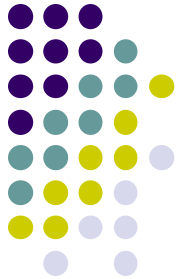
本授業のレポートについて



- 基礎編から一問＋応用編から一問、計二問のレポート提出を必須とします
- 基礎編
  - OpenMP+MPIの、選択問題の中から一問以上
- 応用編
  - CUDA
  - PGAS (予定)

  } この中から一問以上

基礎編課題説明/Report (1)



以下の[1]—[3]のどれか一つについてレポートを提出してください．二つ以上でも良い．

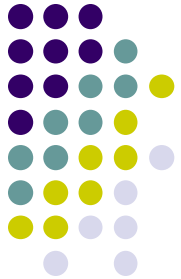
[1] Advectionサンプルプログラムを，以下のいずれかの方法で並列化してください．

a) OpenMP

b) MPI

- MPIで一般のサイズに対応するには端数処理が必要である。本レポートではそれは必須ではない
- より良いアルゴリズムにしてもよい．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？

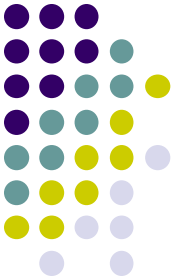
基礎編課題説明/Report (2)



[2] MPIで並列化され，メモリ利用量を抑えた行列積プログラムを実装してください (予定)

- mm-mpiサンプルの改造でよい
- データ分割は本授業の通りでもそれ以外でもよい
- 端数処理はあった方が望ましいが，必須ではない

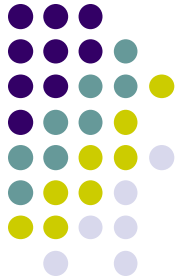
基礎編課題説明/Report (3)



[3] 自由課題：任意のプログラムを，OpenMPまたはMPI(MPI-2も可)を用いて並列化してください．

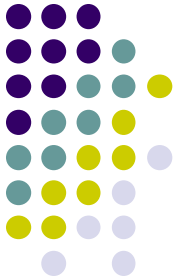
- 単純な並列化で済む問題ではないことが望ましい
  - スレッド・プロセス間に依存関係がある
  - 均等分割ではうまくいかない、など
- たとえば，過去のSuperConの本選問題
  http://www.gsic.titech.ac.jp/supercon/
  たんぱく質類似度(2003)，N体問題(2001)・・・
  入力データは自分で作る必要あり
- たとえば，自分が研究している問題

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．大規模のほうがよい．XXコア以上で発生する問題に触れているとなお良い
    - 問題サイズを様々に変化させたとき(可能な問題なら)
  - 高性能化のための工夫が含まれているとなお良い
    - 「XXXのためにXXXをしてみたが高速にならなかった」のような失敗でも可
  - プログラムについては，zipなどで圧縮して添付
    - 困難な場合，TSUBAME2の自分のホームディレクトリに置き，置き場所を連絡．分かりにくいディレクトリ名推奨

基礎編の課題の提出について



- 提出期限
  - 7/1 (月)
- レポート形式
  - PDF, Word, テキストファイルのいずれか (その他は相談)
- 送り先 : endo@is.titech.ac.jp
- メール題名 : ppcomp report

About Account

- TSUBAME2のアカウントができたら，連絡してください．授業用のTSUBAMEグループへ登録します．

Subject: TSUBAME2 ppcomp account

To: endo@is.titech.ac.jp

- 専攻・研究室
- 学年
- 氏名
- アカウント名

次回/Next Lecture

- 5/9(Thu)　(木曜に月曜授業日)
- OpenMP(2)

  - スケジュールについてはOCW pageも参照
    - http://www.el.gsic.titech.ac.jp/~endo/

    → 2013年度前期情報(OCW) → 講義ノート