

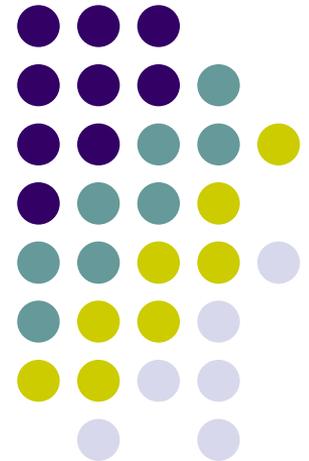
2013年度 実践的並列コンピューティング 第5回

MPIによる
分散メモリ並列プログラミング(1)

遠藤 敏夫

endo@is.titech.ac.jp

2013年5月13日



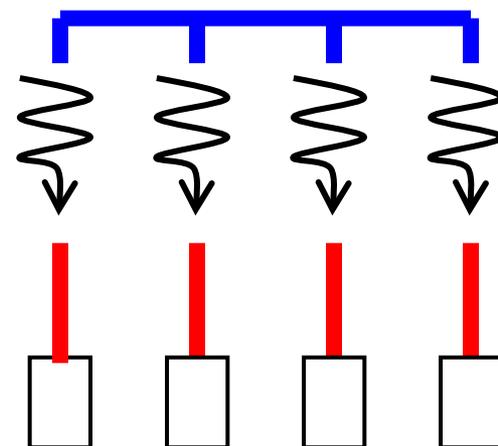
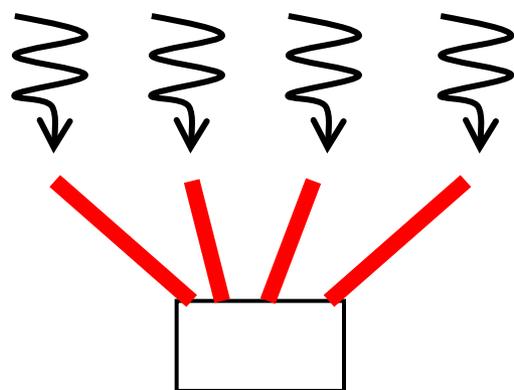
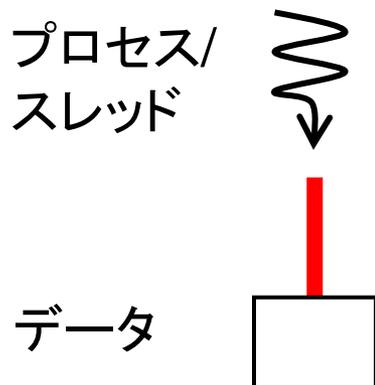
並列プログラミングモデルの、 メモリモデルによる分類



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

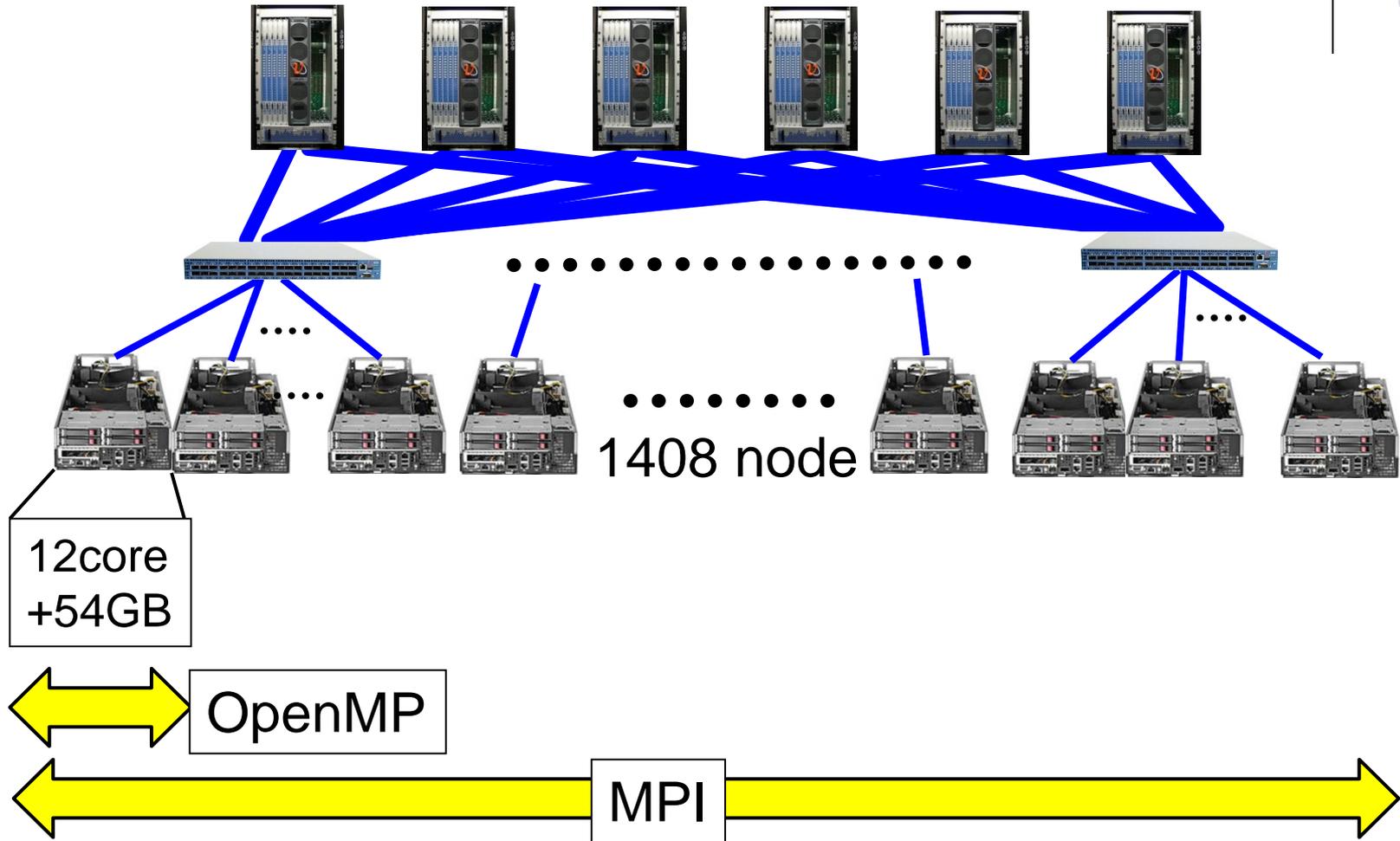
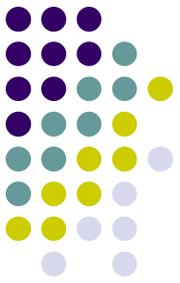
- OpenMP(言語拡張)
- pthread(ライブラリ)
- CUDA (言語拡張)

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)
- **Chapel, X10** (新言語)

青字のものは、PGASと呼ばれ、
一部共有メモリの概念あり
(partitioned global address space)

分散メモリモデルのMPIにより 大規模資源を利用可能に

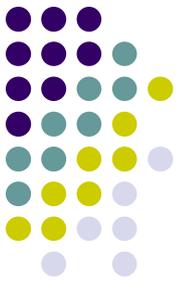


OpenMPとMPI



- OpenMP
 - 共有メモリモデル
 - スレッド間のデータ移動は共有変数で
 - 排他制御によりrace conditionを防ぐ
 - 利用可能な並列度はノード内(TSUBAME2では12CPUコア)
 - #pragmaを無視すると逐次プログラムとして動作する場合が多い
- MPI
 - 分散メモリモデル
 - プロセス間のデータ移動はメッセージで
 - Critical sectionの代わりにメッセージで同期
 - 利用可能な並列度はノードを超える(TSUBAME2では10000CPUコア以上)
 - 逐次プログラムを基にする場合、全体の構造への大幅な変更が必要になりがち

MPI(message-passing interface)とは



- 分散メモリ並列プログラミングの規格
- C, C++, Fortranに対応
- メッセージパッシングのためのライブラリ
- SPMDモデル. プロセス間の相互作用はメッセージで
 - MPI-2規格では, さらにRMA(remote memory access)が追加



スパコン上の科学技術演算の例

京スパコン上の代表的なソフトウェアの、分野と利用並列環境

- 強誘電体MD: OpenMP
- 第一原理MD: MPI+OpenMP
- 差分非圧縮熱流体: MPI+OpenMP
- 5次元プラズマ乱流: MPI+OpenMP
- 有限要素法非圧縮熱流体: MPI+京並列コンパイラ
- 電子状態密度計算: MPI+OpenMP
- 細胞内シグナル伝播計算: MPI+OpenMP
- 神経回路シミュレーション: MPI+OpenMP
- 遺伝子ネットワーク推定: MPI+OpenMP
- 電子相関計算: MPI+OpenMP+並列BLAS
- Linpackベンチマーク(HPL): MPI+並列BLAS



プロセッサコアの利用方法

1プログラムが、12コアx100ノードを利用するには？



(1) MPIを用いてプログラミング

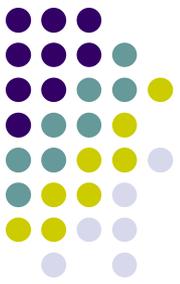
- 1200プロセス起動する

(2) MPIとOpenMPの両方を用いる(ハイブリッド並列)

- 100プロセス起動する
- 1プロセスが12スレッドから成り立つようにする
 - OMP_NUM_THREADS=12
- プログラミングは大変だが、(1)より効率的な場合が多く、前ページのソフトのほとんどがこの手法

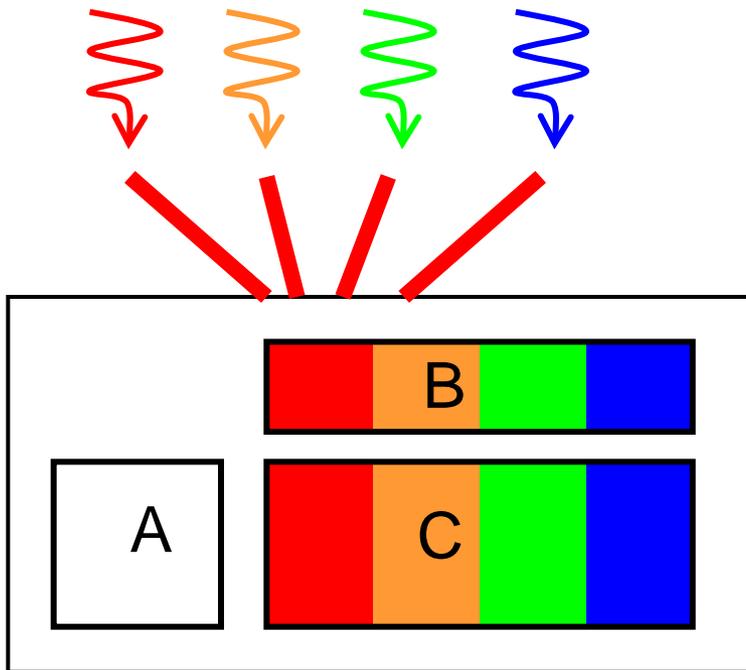
分散メモリと共有メモリの違い

行列積($C=AxB$)の例

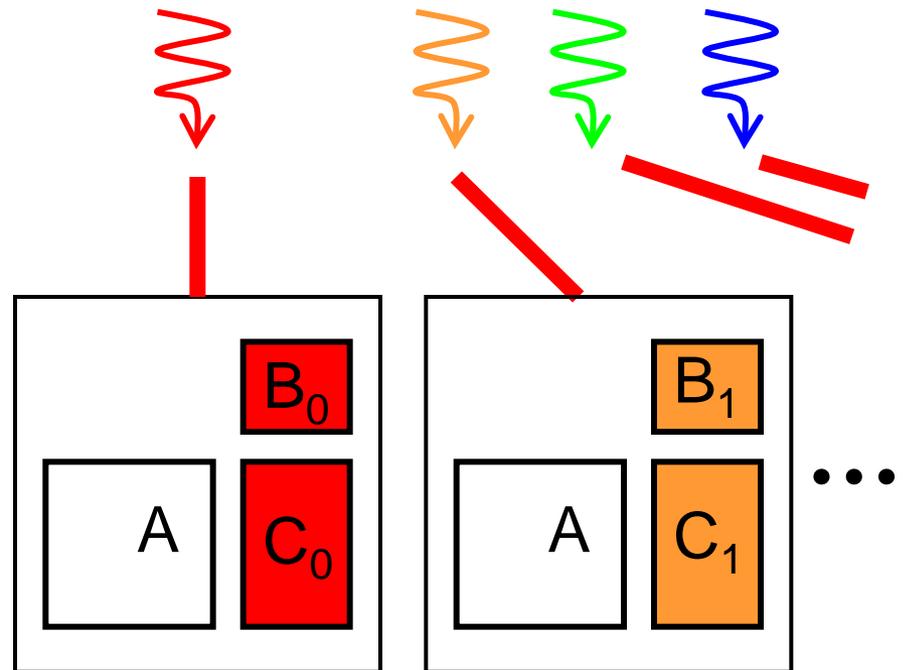


- 共有メモリ: 計算をどうスレッドに分割するか

- 分散メモリ: 計算とデータをどうプロセスに分割するか



行列Aは全スレッドによってアクセスされる



行列Aは全プロセスに置かれる
→ 後に改良案



MPIプロセスとメモリ

- 複数のプロセスが同一プログラムを実行(SPMDモデル)
- プロセスごとに別のメモリ空間 → 全ての変数(大域変数・局所変数)は各プロセスで別々
- プロセスには, 0, 1, 2...という番号(rank)がつく
 - `MPI_Comm_rank(MPI_COMM_WORLD, &rank);` ランク取得
 - `MPI_Comm_size(MPI_COMM_WORLD, &size);` 全プロセス数取得
 - $0 \leq \text{rank} < \text{size}$
 - `MPI_COMM_WORLD`は, 「全プロセスを含むプロセス集団(=コミュニケータ)」
 - メッセージの送信先, 受信元としてrankを利用



MPIプログラムの概要

```
#include <stdio.h>
```

```
#include <mpi.h>
```

```
int main(int argc, char *argv[])
```

```
{
```

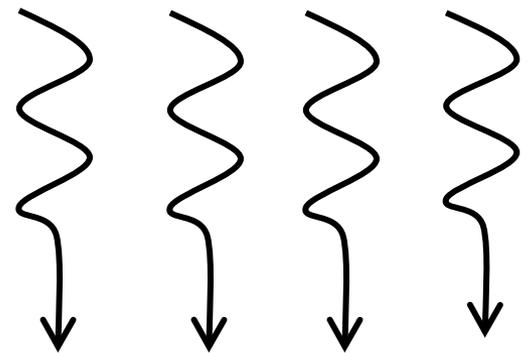
```
    MPI_Init(&argc, &argv); ← MPIの初期化
```

```
        (計算・通信)
```

```
    MPI_Finalize();
```

```
}
```

← MPIの終了

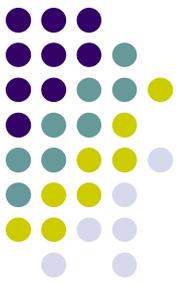


TSUBAME2上でのMPIプログラムのコンパイル, 実行 (1)



- TSUBAME 2にはOpenMPI, MVAPICHがインストールされている
- コンパイル:
 - **mpicc**でコンパイル. オプションは一般のコンパイラと同じ
 - サンプルプログラムは, makeコマンドでok
 - ~endo-t-ac/ppcomp/13/ 以下の
 - mpitest1, pi-mpi, mm-mpi ディレクトリ
- インタラクティブノードでの実行:
 - **mpirun -np [プロセス数] [プログラム名]**
 - プロセス数は4まで. かつ, 実行時間は数分以内にとどめること
 - もっと大きいのはバッチキューで

TSUBAME2上でのMPIプログラムのコンパイル, 実行 (2)



- バッチキューでの実行

- test/myprogというMPIプログラムを, ノードあたり12並列 × 10ノード = **120並列**で実行する場合

(1) スクリプトファイルの作成:

job.shファイル

(2)とつじつまを合わせること

```
#!/bin/sh
cd $PBS_O_WORKDIR
mpirun -np 120 -hostfile $PBS_NODEFILE ./myprog
```

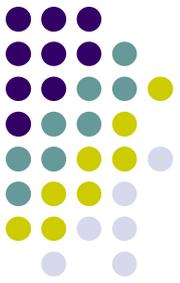
chmod 755 job.sh コマンドで, 「実行可能ファイル」にしておく

(2) t2subコマンドで投入

本授業の場合

```
t2sub -q S -W group_list=t2g-ppcomp
-l select=10:mpiprocs=12 -l place=scatter ./job.sh
```

MPIの基本中の基本： メッセージの送信・受信



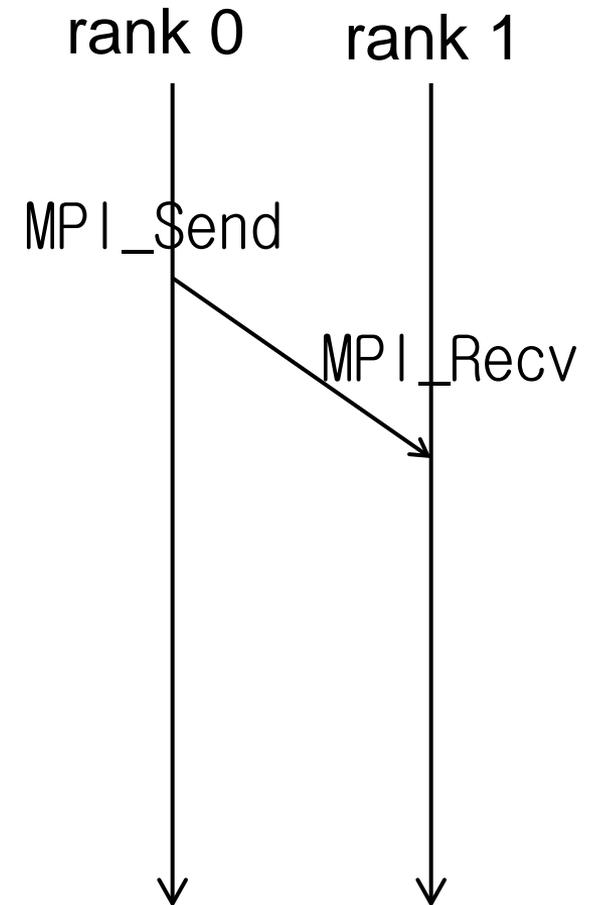
rank 0からrank1へ, int a[16]の中身を送りたい場合

- rank0側で

```
MPI_Send(a, 16, MPI_INT, 1,  
100, MPI_COMM_WORLD);
```

- rank1側で

```
MPI_Recv(b, 16, MPI_INT, 0,  
100, MPI_COMM_WORLD, &stat);
```

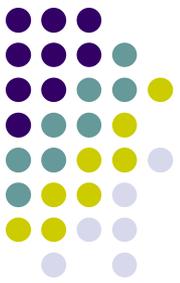




MPI_Send

```
MPI_Send(a, 16, MPI_INT, 1, 100, MPI_COMM_WORLD);
```

- a: メッセージとして送りたいメモリ領域の先頭アドレス
- 16: 送りたいデータ個数
- MPI_INT: 送りたいデータ型
 - 他にはMPI_CHAR, MPI_LONG, MPI_DOUBLE, MPI_BYTE...
- 1: メッセージの宛先プロセスのrank
- 100: メッセージにつけるタグ(整数)
- MPI_COMM_WORLD: コミュニケータ



MPI_Recv

`MPI_Status stat;`

`MPI_Recv(b, 16, MPI_INT, 0, 100, MPI_COMM_WORLD, &stat);`

- `b`: メッセージを受け取るメモリ領域の先頭アドレス
 - 十分な領域を確保しておくこと
- `16`: 受け取るデータ個数
- `MPI_INT`: 受け取るデータ型
- `0`: 受け取りたいメッセージの送信元プロセスのrank
- `100`: 受け取りたいメッセージのタグ. ユーザが決める整数
 - `MPI_Send`で指定したものと同一なら受け取れる
- `MPI_COMM_WORLD`: コミュニケータ
- `&stat`: メッセージに関する補足情報が受け取れる

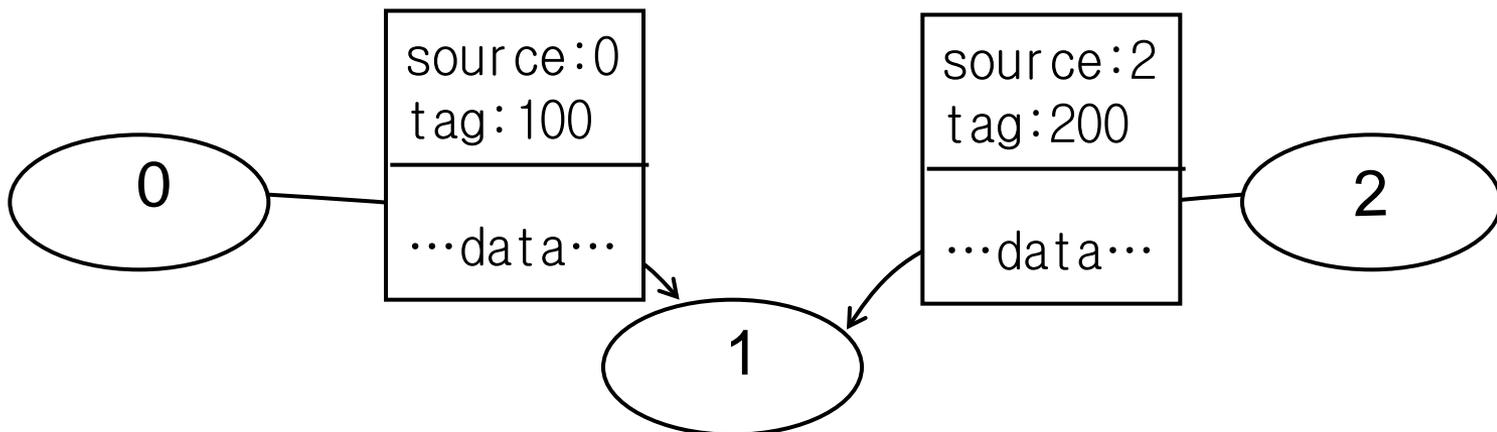
`MPI_Recv`を呼ぶと、メッセージが到着するまで待たされる (ブロッキング)



MPI_Recvのマッチング処理

受信側には複数メッセージがやってくるかも → 受け取りたい条件を指定する

- 受け取りたい送信元を指定するか, MPI_ANY_SOURCE (誰からでもよい)
- 受け取りたいタグを指定するか, MPI_ANY_TAG(どのタグでもよい)





集団通信とは

- 一対一通信: MPI_Send対MPI_Recv
 - これがあれば、原理的にはなんでも書ける
- **集団通信**とは、多数プロセスを巻き込んだ通信
 - Reduce, Bcast, Gather, Scatter, Barrier...
 - 一対一の組み合わせでも実現できるが、専用関数の方が早い・速い
 - プログラムが楽
 - プロセスの木構造・binary exchangeなどの効率的アルゴリズムが使われている(はず)
- 例: MPI_Barrier(MPI_COMM_WORLD);
全プロセスがMPI_Barrier関数を呼び出すまで待つ(バリア同期)

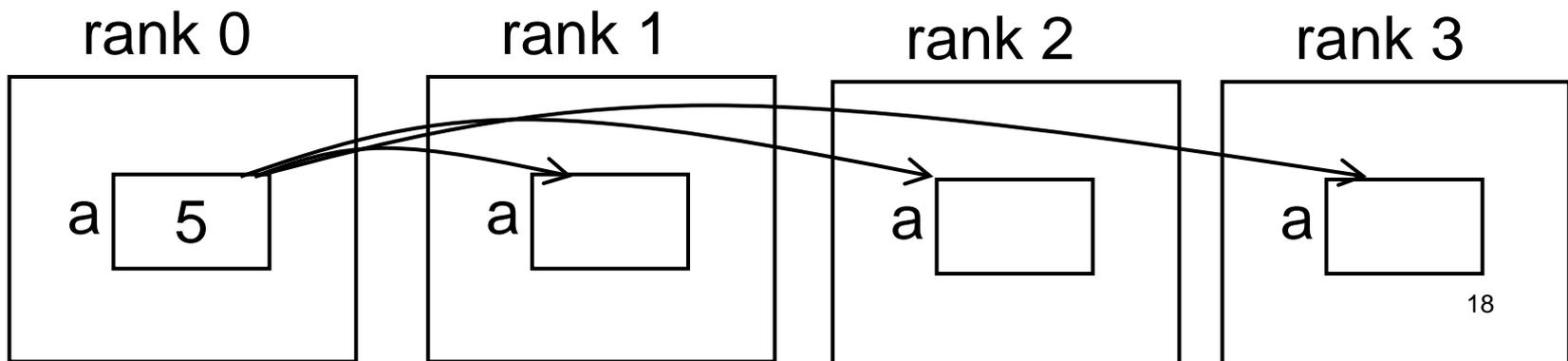


集団通信: MPI_Bcast

- 例: rank 0のプロセスが持っているint aの値を全プロセスに知らせたい(broadcast処理)

```
MPI_Bcast(&a, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

- 全プロセスがMPI_Bcastを呼び出す必要
- この結果, 全プロセスの領域aに結果が格納される
- 第一引数はrootプロセス(ここではrank 0)では入力, それ以外のプロセスでは出力として扱われる



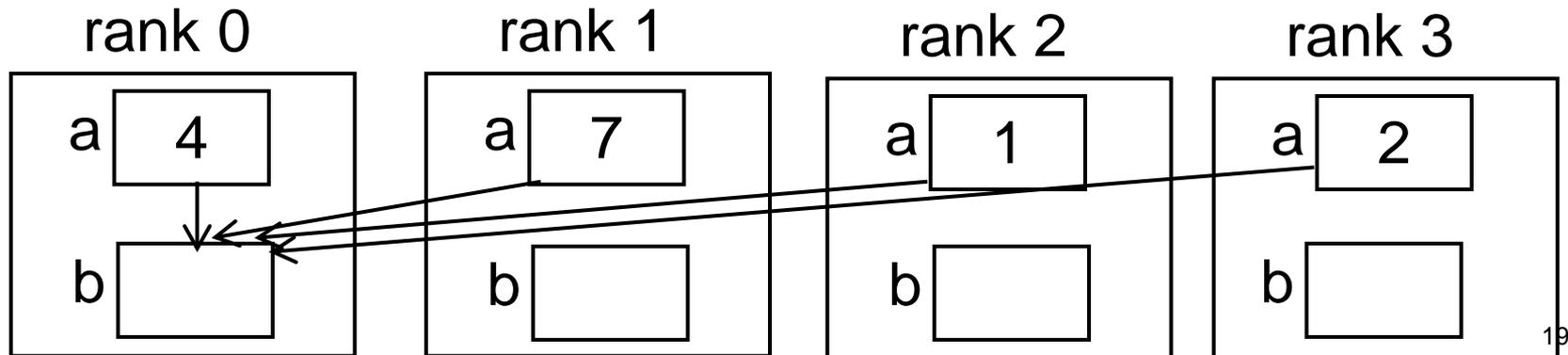


集団通信: MPI_Reduce

- 例: 全プロセスのint aの合計を求めたい (reduction処理)

```
MPI_Reduce(&a, &b, 1, MPI_INT, MPI_SUM, 0,  
          MPI_COMM_WORLD);
```

- 全プロセスがMPI_Reduce()を呼び出す必要
- この結果, rank 0の領域bに合計(SUM)が格納される
- 演算は他に, MPI_PROD(積), MPI_MAX, MPI_MIN, MPI_LAND(論理積)など

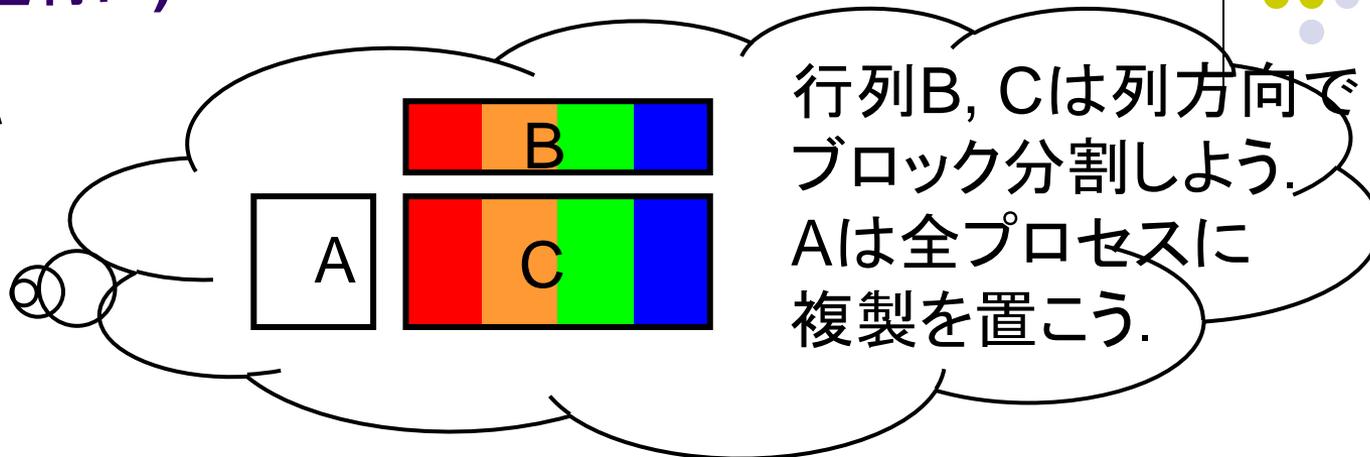


分散メモリプログラミングとデータ配置

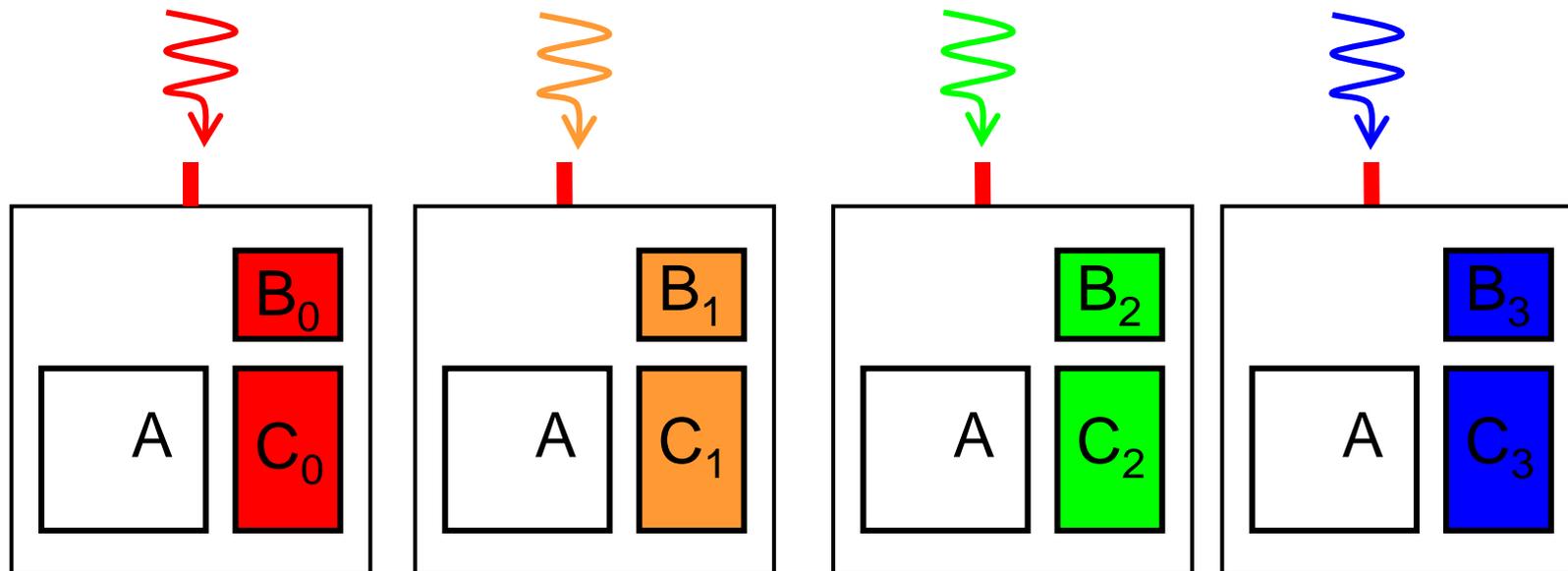
(mm-mpiを題材に)



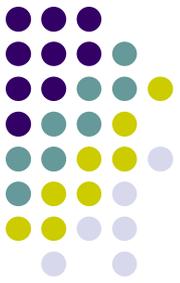
配置方法を
決める:



実際の配置をプログラミング:

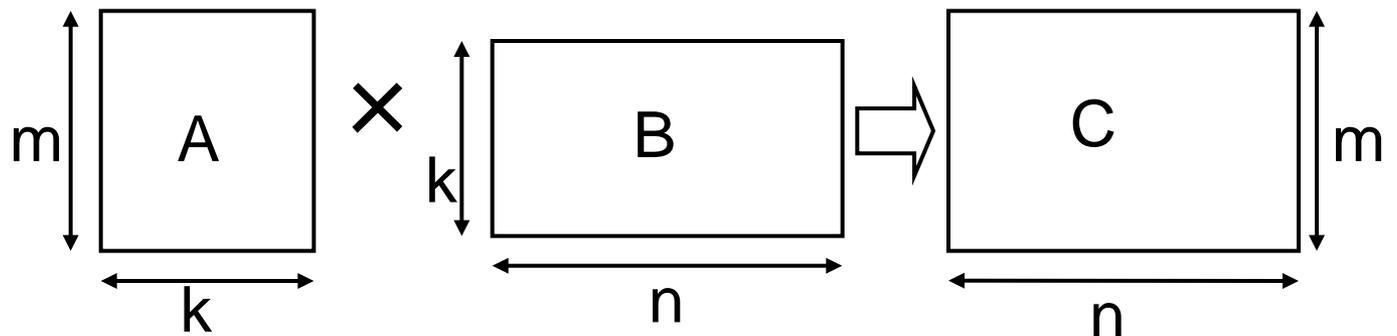


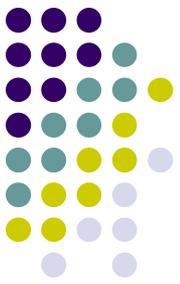
行列積サンプルプログラム (再掲)



($m \times k$)行列と($k \times n$)行列の積

- 三重のforループで記述
- 動的長さ配列. 二次元を一次元で表現 (column-major)
- 実行オプション: `./mm [m] [n] [k]`
- 計算量: $O(m \times n \times k)$





データ配置を考える

1. 初期配置

- 各データは最初から分散しているとしてよいか？
- 初期値はプロセス0が持っているとするか？

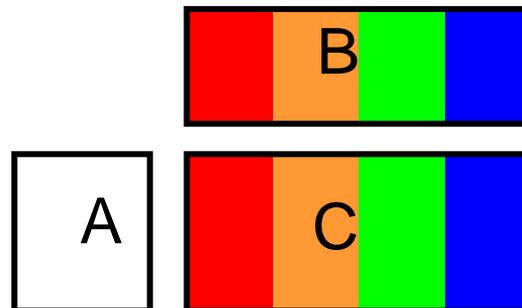
2. 計算中の配置

- プロセス間の通信量が少なくなる配置が望ましい
- メモリ消費とトレードオフの場合も

3. 結果の配置

- 結果を(たとえばプロセス0に)集める必要はあるか？

(mm-mpiでは分割したままとした)



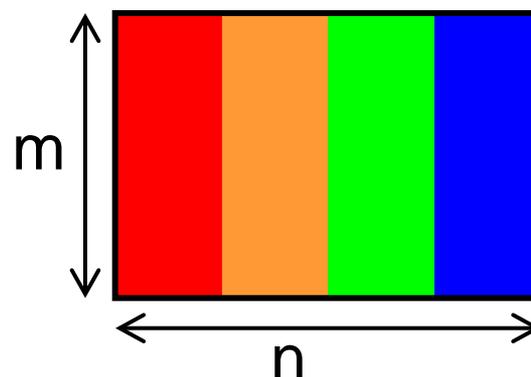
Cを列方向ブロック分割
⇒ Bも列方向の分割
Aは全複製

データ分散とプログラミング

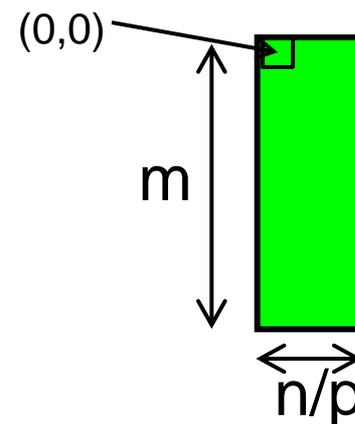
- $m \times n$ 行列を p プロセスで分割するとはどういうことか？
 - データ並びはcolumn-majorとする
 - ここでは割り切れる場合を仮定
- 各プロセスが持つのは、 $m \times (n/p)$ の部分行列
 - $m * (n/p) * \text{sizeof}(\text{データ型})$ のサイズの領域をmallocすることに
 - 部分行列と全体行列の対応を意識する必要
 - プロセス r の部分行列の (i, j) 要素 \leftrightarrow 全体行列の $(i, n/p * r + j)$ 要素に対応



全体のイメージ



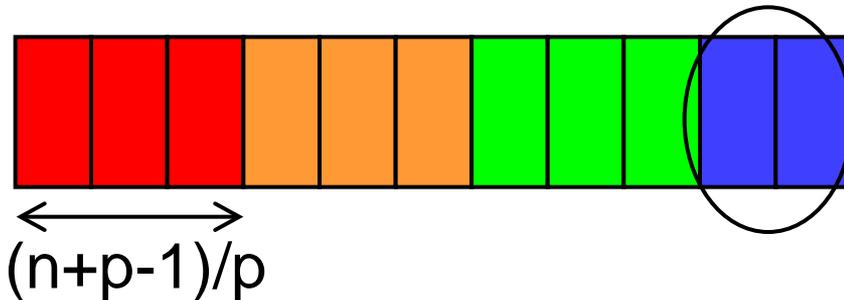
プロセスが持つ領域





意外とめんどうな端数処理

- データサイズ n が、プロセス数 p で割り切れるとは限らない
- 11個(11列)のデータを4プロセスで分割するには？
 - C言語の整数割り算は切り捨て
 - $n/p = 2$ 個ずつ担当していくとデータが余る → 切り上げの必要
→ $(n+p-1)/p = 3$ 個ずつ担当する。
最後のプロセスは他より仕事が少なくなる

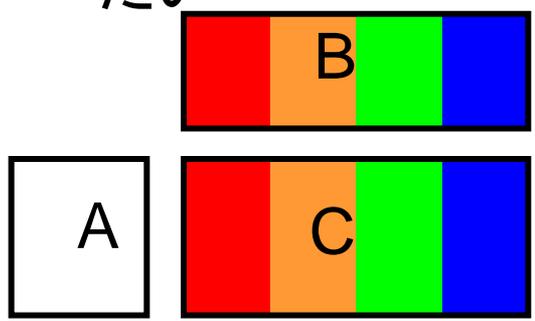


サンプルプログラムの`divide_length()`関数は、自分の担当場所の始点インデックス s と終点インデックス e を返す。 s 以上 e 未満の、 $(e-s)$ 個(列)のデータを担当することを示す。

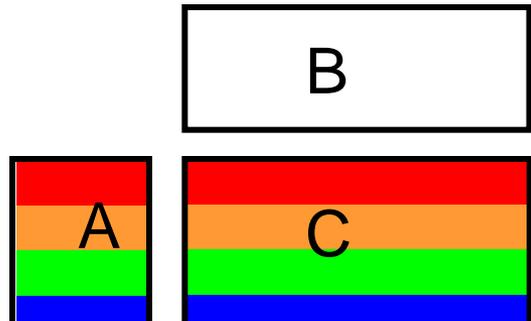
データ配置の再検討



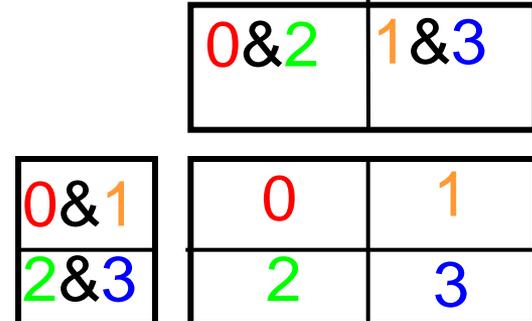
- $C_{i,j}$ の計算には、Aの第*i*列とBの第*j*列が必要
⇒ 単純には、依存するデータをできるだけ同じプロセスに置きたい



Cを列方向ブロック分割
⇒ Bも列方向の分割
Aは全複製
(mm-mpiの方法)
 $O(mkp+nk+mn)$



Cを行方向ブロック分割
⇒ Aも行方向の分割
Bは全複製
 $O(mk+nkp+mn)$

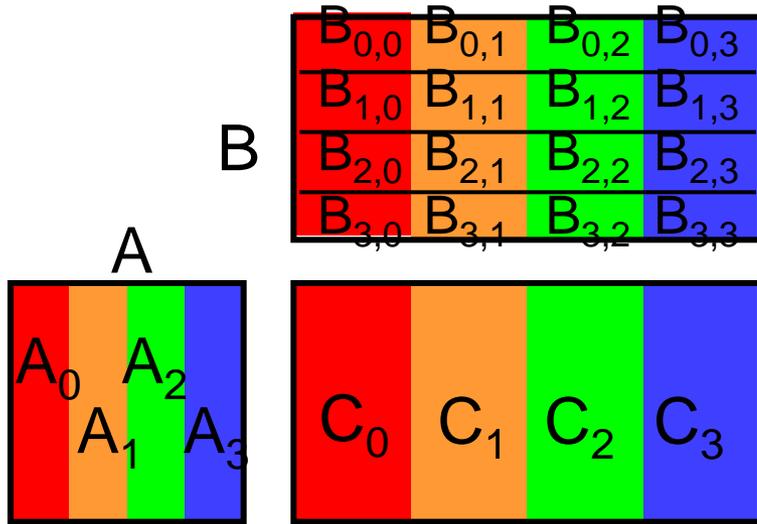
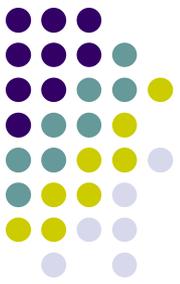


Cを二次元ブロック分割
⇒ A:行方向分割+複製
B:列方向分割+複製
 $O(mkp^{1/2}+nkp^{1/2}+mn)$

↑ メモリ量の全プロセス合計

以上は全て、計算最中の通信は不要だがメモリ利用量が多い
プロセス数・問題サイズが大規模になるとダメ

メモリ利用量を抑えるデータ配置の例



Aもブロック分割
(列方向でなくても可)
⇒ ローカルデータだけでは
 C_i の計算できないので、
通信が必要

第0フェーズ:

プロセス0が A_0 をBcast
各プロセスiは,

$$C_i += A_0 \times B_{0,i} \text{ を計算}$$

第1フェーズ:

プロセス1が A_1 をBcast
各プロセスiは,

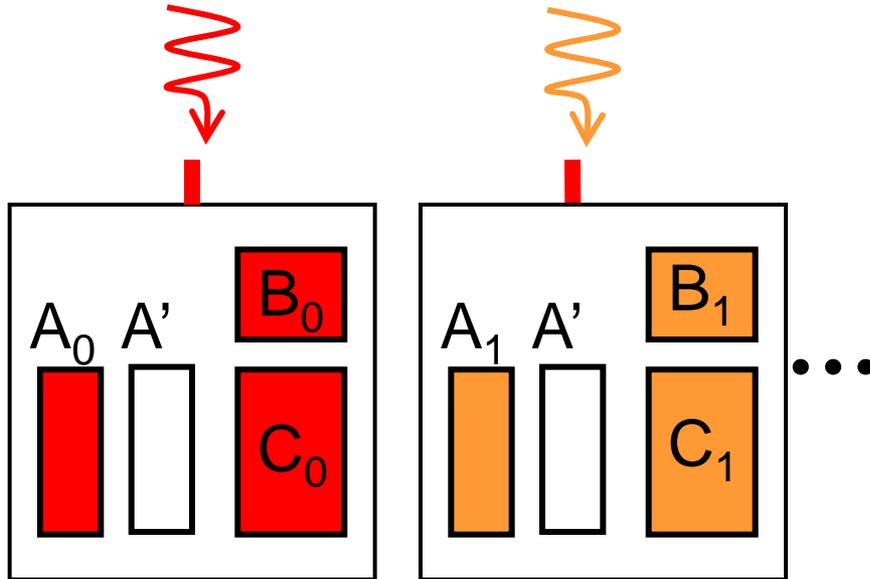
$$C_i += A_1 \times B_{1,i} \text{ を計算}$$

:

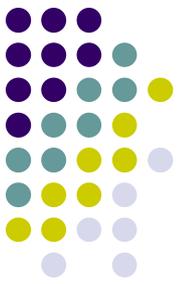
以下同様に、第(p-1)フェーズ
まで行う



実際のメモリ配置はどうなる？



- 各プロセスは，分割されたAに加え，受信バッファ(A')を用意する
- 第rフェーズでは，
 - プロセスrはAからA'へデータコピー（省略する手法もあり）
 - プロセスrをルートとし，領域A'をMPI_Bcast



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

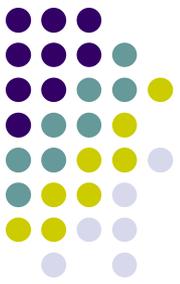
- MPIで一般のサイズに対応するには端数処理が必要である。本レポートではそれは必須ではない
- より良いアルゴリズムにしてもよい。ブロック化・計算順序変更でキャッシュミスが減らせないか？



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
- 今回のスライドのアルゴリズムよりも進化した、SUMMA (Scalable Universal Matrix Multiplication Algorithm)[Van de Geijn 1997] もok
- 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)

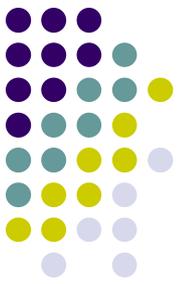
[3] 自由課題: 任意のプログラムを, OpenMPまたはMPI(MPI-2も可)を用いて並列化してください.

- 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



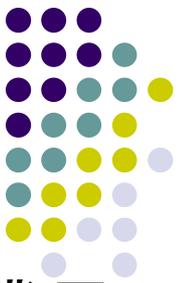
課題の注意

- 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を試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)
 - プログラムも添付 (前ページ参照)
- 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なら、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

- 5/20(月) 特別講義予定
 - 集合場所は普段通りW832
 - GSIC下川辺先生による超並列アプリに関する講演
 - TSUBAME2.0スパコン見学
- 5/27(月)
 - MPI (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