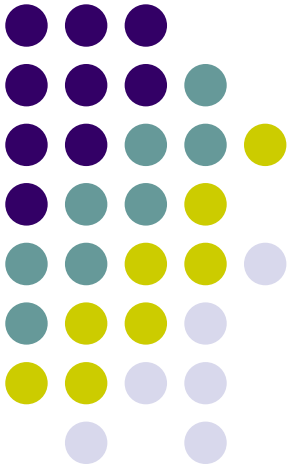
2014年度
実践的並列コンピューティング
第10回



MPIによる
分散メモリ並列プログラミング(3)

遠藤 敏夫

endo@is.titech.ac.jp

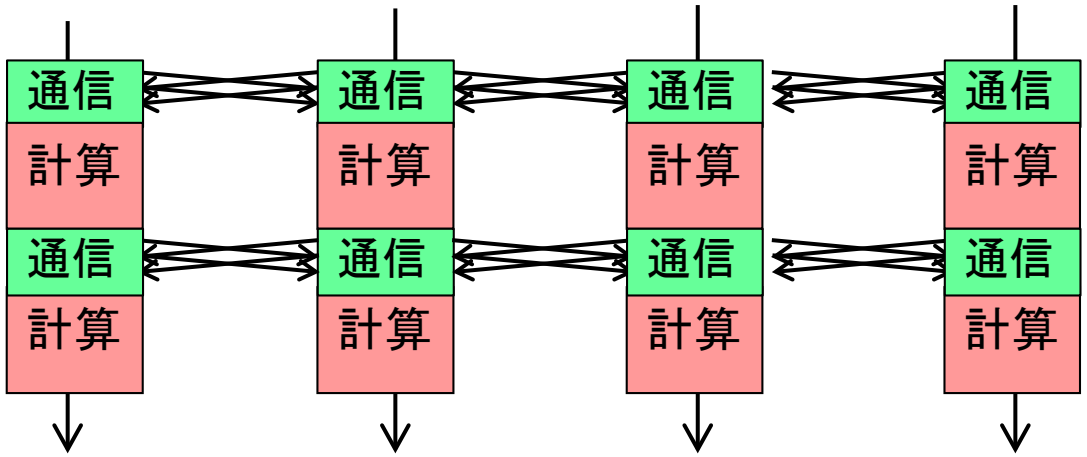
MPIプログラムの性能を考える

- 前回までは、MPIプログラムの挙動の正しさを議論
- 今回は速度性能に注目

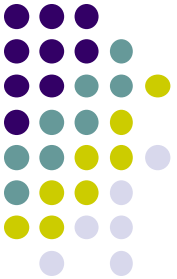
MPIプログラムの実行時間 =
プロセス内計算時間 + プロセス間通信時間

計算量、(プロセス内)ボトルネック有無
メモリアクセス量、計算不均衡…など関係

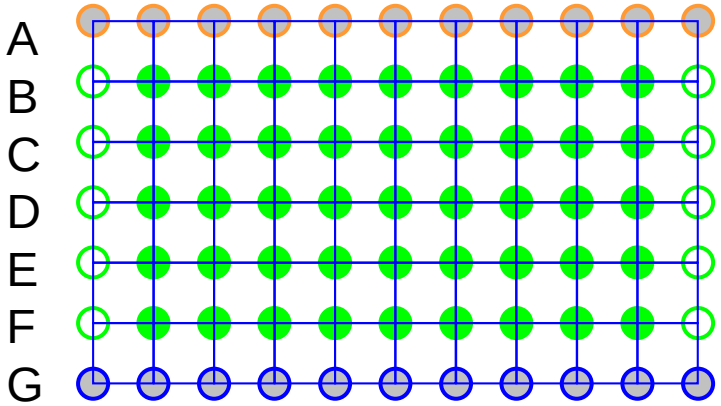
MPI版
ステンシル
計算の挙動



ステンシル計算の工夫:
計算と通信のオーバラップ(1)



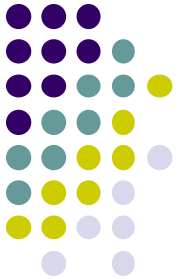
データの依存関係をより詳細に考えると、他プロセスのデータを必要としない計算が存在する！



行C~Eは通信を待たずに計算できることに注目
⇒B~Dは通信完了前に計算してしまってよい

※ コードがやや複雑になるので、レポート課題[1] (MPIの場合)では必須ではありません

計算と通信のオーバラップ(2)



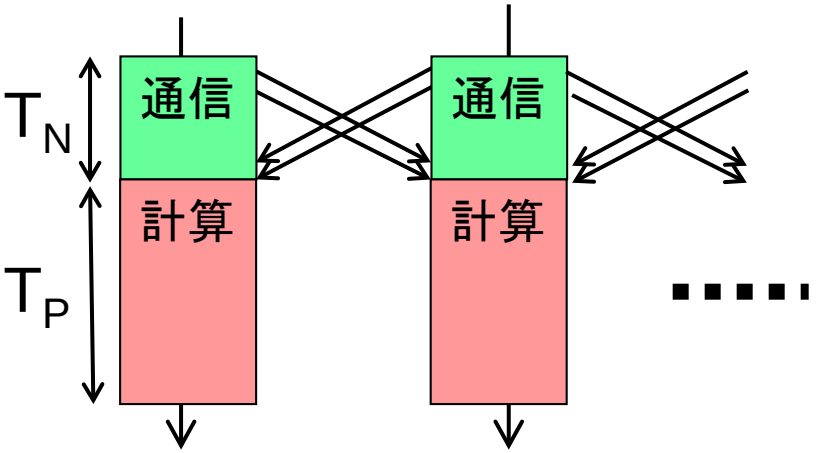
- オーバラップを組み込んだMPIプログラムの流れ

```
for (it…) {
  行Bを前のプロセスへ送信開始，Fを次のプロセスへ送信開始
  行Aを前のプロセスから受信開始，Gを次のプロセスから受信開始
  C~Eの点を計算
  上記の全通信終了を待つ(MPI_WaitかMPI_Waitall)
  行B，Fの点を計算
  二つの配列の切り替え
}
```

計算と通信のオーバラップ(3)

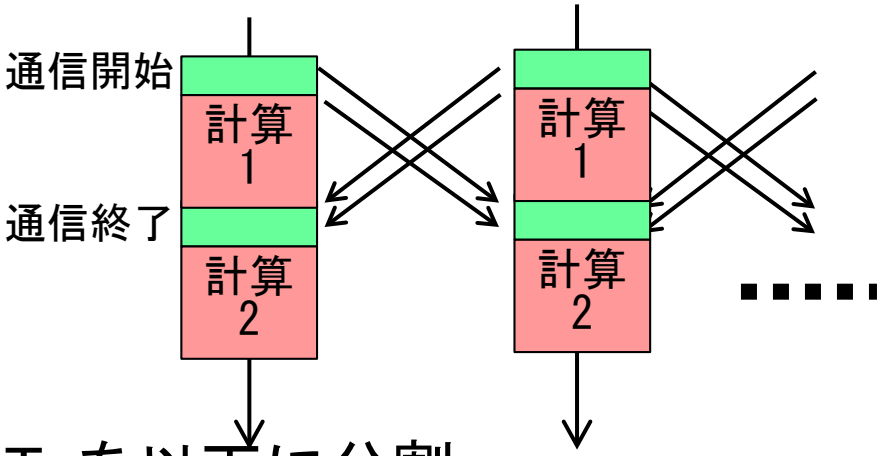
時間ステップあたりの全体時間をT、通信時間T_N, 計算時間T_Pとする

オーバラップなし



$T=T_N+T_P$

オーバラップあり



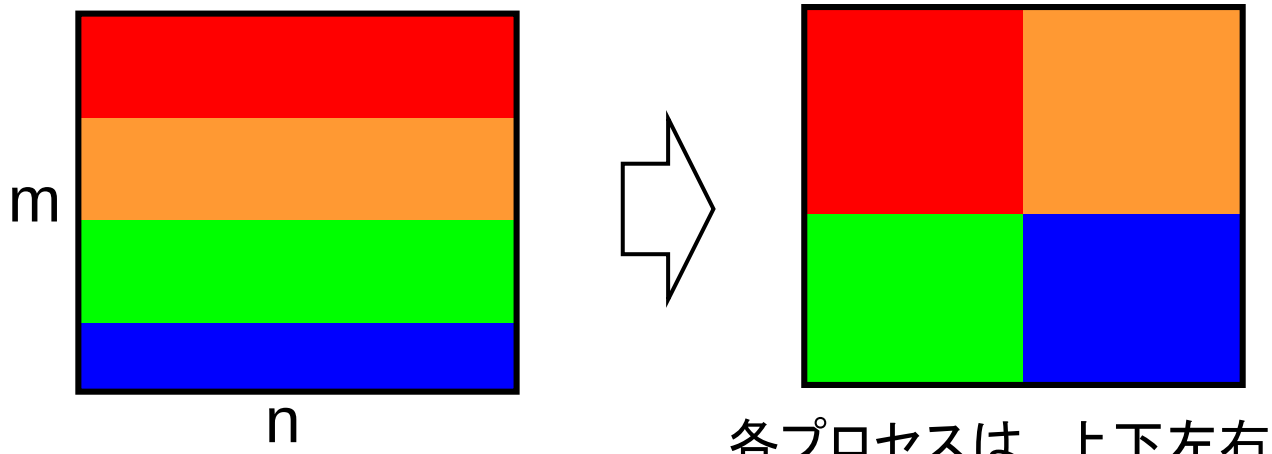
T_Pを以下に分割

- T_{P1}: オーバラップ可能部分
- T_{P2}: オーバラップ不可部分

$T=\max(T_N,T_{P1})+T_{P2}$

ステンシル計算のさらなる工夫: 多次元分割による通信量削減

m×nサイズの二次元領域の場合



各プロセスは、上下左右の隣接プロセスと境界交換

(計算によっては、斜め隣りも)

一プロセスあたりの

- 計算量: O(mn/p)
- 通信量: O(n)

一プロセスあたりの

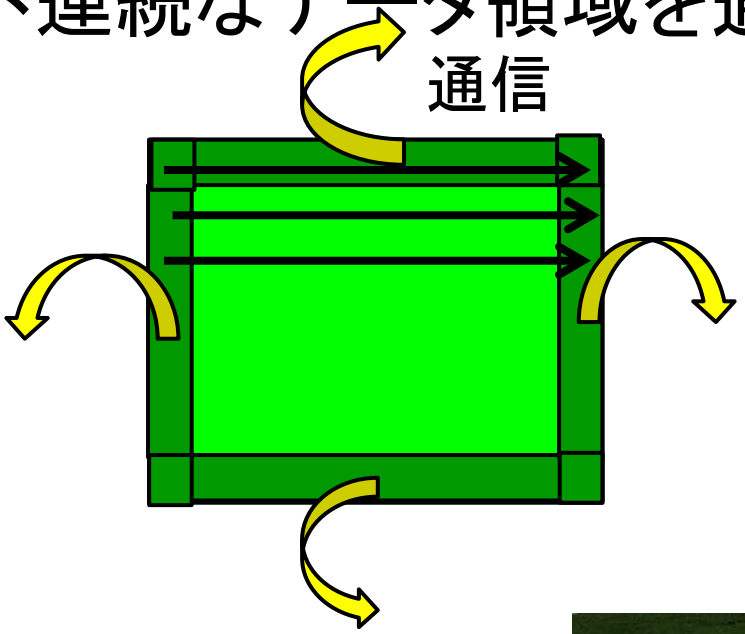
- 計算量: O(mn/p)
- 通信量: $O((m+n)/p^{1/2})$

通信量を削減可能

多次元分割と不連続データ

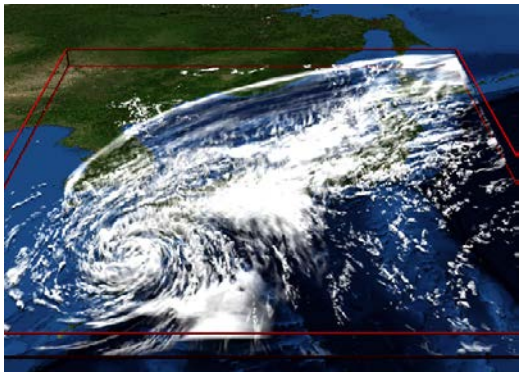
- 多次元分割で通信量合計を削減できるが、

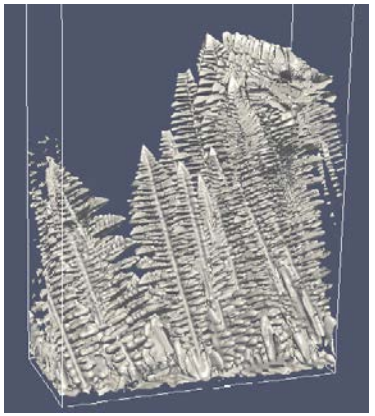
不連続なデータ領域を通信する必要がおこる



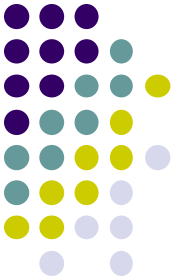
Row-majorならびの場合、
左右プロセスと通信する
領域は不連続となってしまう

※大規模ステンシル計算では、たいてい多次元分割している





不連続データを通信するには



考えられる実現方法

- 一要素ずつ通信を繰り返す

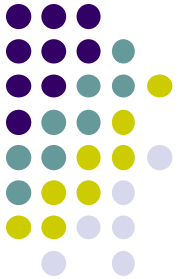
⇒　性能が大きく低下！メッセージ数が莫大になり、レイテンシを何度も食らうので

- 連続領域に一度コピーしてから通信

⇒　これをやっているアプリも多い。やや実装が面倒

- 派生データ型の利用(次ページ以降)

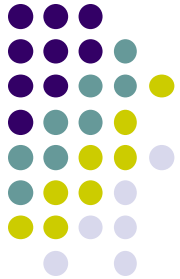
派生データ型(1)



- ユーザが新たにデータ型(MPI_Datatype)を作ることができる
  - MPI_Type_vector関数
  - 他に、MPI_Type_struct, MPI_Type_indexedなど

```
MPI_Datatype mytype;
MPI_Type_vector(lm, 1, ln, MPI_DOUBLE, &mytype);←型作成
MPI_Type_commit(&mytype);  ← 利用前に必要な決まり
                :
(MPI_Send/MPI_Recvなどにmytypeを利用可能)
MPI_Send(mybuf, 1, mytype, dst, 100, MPI_COMM_WORLD);
                :
MPI_Type_free(&mytype);
```

派生データ型(2)



MPI_Type_vector(count, blocklen, stride, oldtype, &newtype)

等間隔で飛び飛びとなるデータ領域を表す派生データ型を作る。

count: ブロックの個数

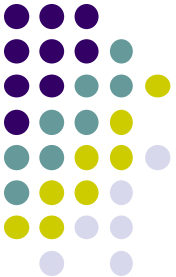
blocklen: 一ブロックの要素数

stride: ブロック間の幅

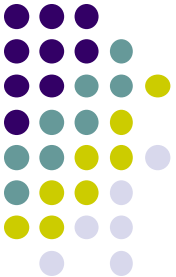
oldtype: 元となるデータ型

newtype: 新たにつくられる派生データ型

多次元分割はいつも有利なわけではない



- 派生データ型を使ってさえ、連続領域よりも不連続領域の通信オーバヘッドは大きくなる
  - 結局MPI関数内部でキャッシュミスやメモリコピーのオーバヘッド
- 一般的には、プロセス数が大きくなるほど有利
- 三次元領域の計算で、あえて二次元分割にした方が有利なケースも



集団通信

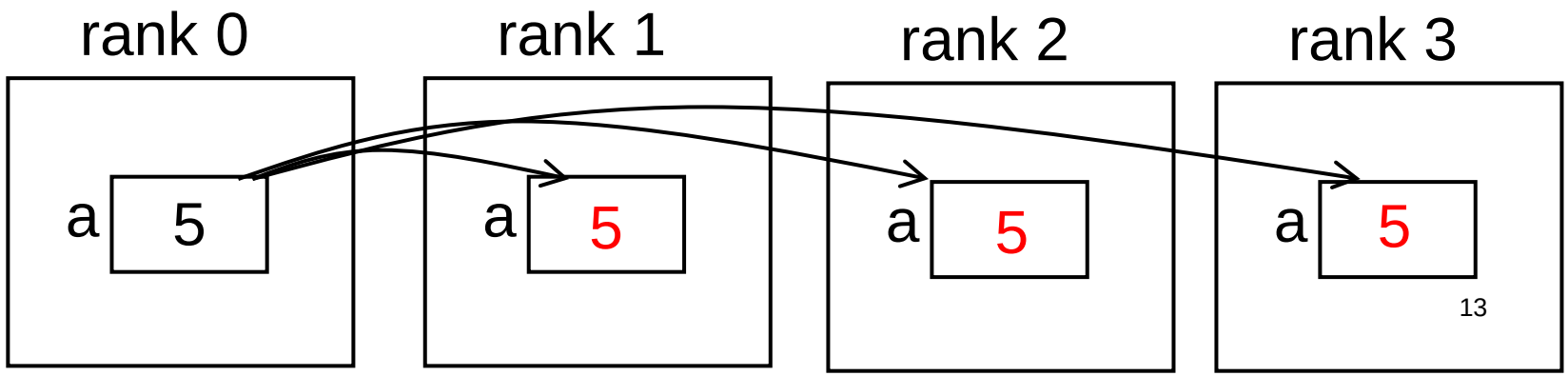
- 一対一通信: MPI_Send対MPI_Recv
  - これがあれば、原理的にはなんでも書ける
- 集団通信とは，多数プロセスを巻き込んだ通信
  - Reduce, Bcast, Gather, Scatter, Barrier・・・
  - 一対一の組み合わせでも実現できるが，専用関数の方が早い・速い
    - プログラムが楽
    - プロセスの木構造・binary exchangeなどの効率的アルゴリズムが使われている(はず)

集団通信:MPI_Bcast

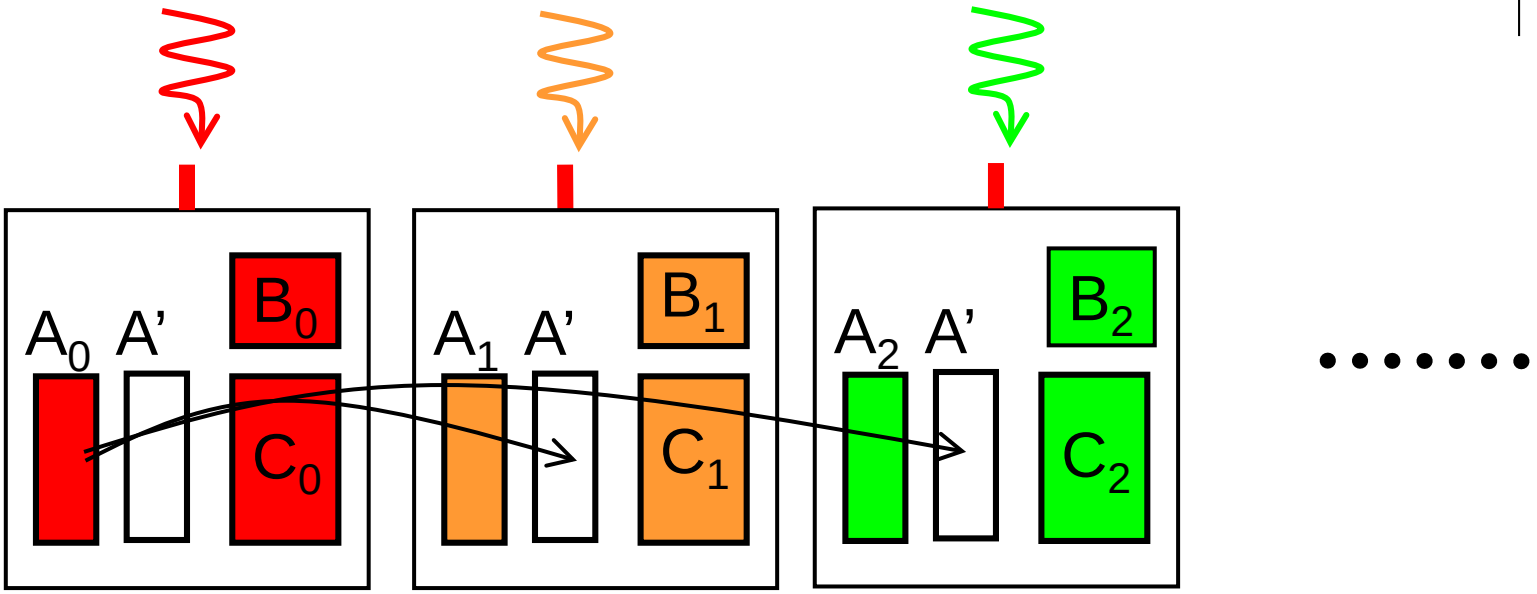
- 例:rank 0のプロセスが持っているint aの値を全プロセスに知らせたい(broadcast処理)

```
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



MPI_Bcastの使い道の一つ: メモリ削減型 行列積から再掲



- 第rフェーズでは，プロセスrが起点(root)となって自分の部分Aを全員に知らせてあげる

→ MPI_Bcastが使える

- 受取側ではもらった内容をA'に格納する必要

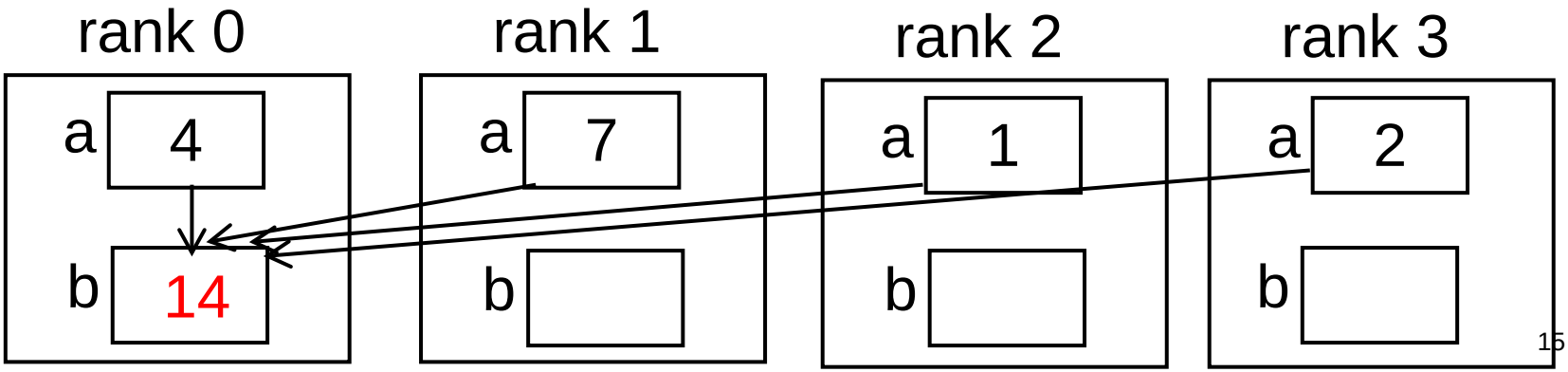
→ RootプロセスではA，他プロセスではA'を第一引数に指定

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

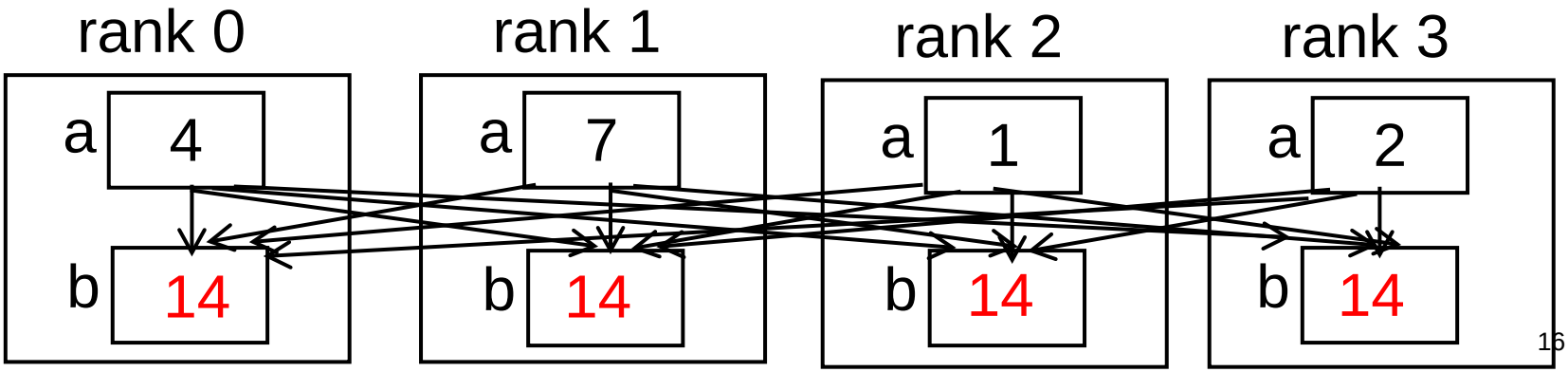


集団通信:MPI_Allreduce

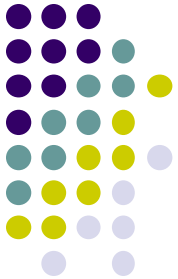
- Reduction処理を行い、その結果を全プロセスが知りたい

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

  - この結果，全員の領域bに合計(SUM)が格納される



集団通信: MPI_Barrier

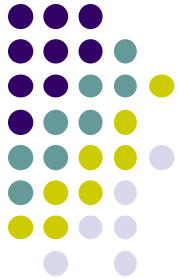


- 全プロセスで足並みをそろえる。バリア同期と呼ぶ

  MPI_Barrier(MPI_COMM_WORLD);

  - サンプルでは、時間計測をより精確にするために利用

本授業のレポートについて

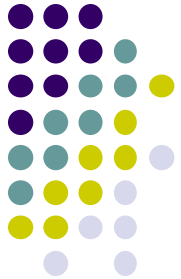


- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

予定パート:

- OpenMPパート
  - ノード内のCPUコアを使う並列プログラミング
- MPIパート
  - 複数ノードを使う並列プログラミング
- GPU(CUDA)パート
  - 1GPU内の数百コアを使う並列プログラミング

MPIパート課題説明 (1)



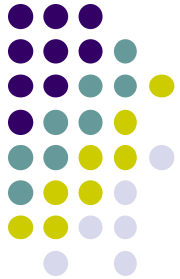
以下の[M1]—[M3]のどれか一つについてレポートを提出してください

[M1] diffusionサンプルプログラムを，MPIで並列化してください.

オプション:

- MPIで一般のサイズ(プロセス数で割り切れないかもしれない)に対応するには端数処理が必要である。本レポートではその対応はオプションとする
- 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？
- 二次元分割の効果はあるか？

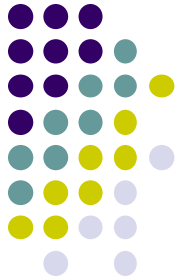
MPIパート課題説明/Report (2)



[M2] 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

- 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

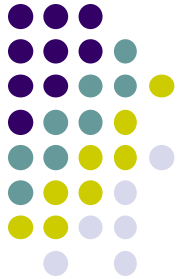
MPIパート課題説明/Report (3)



[3] 自由課題：任意のプログラムを，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

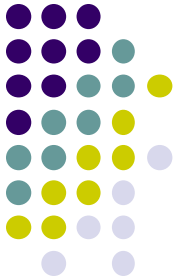


- いずれの課題の場合も，レポートに以下を含むこと
  - 計算・データの割り当て手法の説明
  - TSUBAME2などで実行したときの性能
    - プロセッサ(コア)数を様々に変化させたとき．大規模のほうがよい．XXコア以上で発生する問題に触れているとなお良い
    - 問題サイズを様々に変化させたとき(可能な問題なら)
  - 高性能化のための工夫が含まれているとなお良い
    - 「XXXのためにXXXをしてみたが高速にならなかった」のような失敗でも可
  - 作成したプログラムについても、zipなどで圧縮して添付
    - 困難な場合，TSUBAME2の自分のホームディレクトリに置き，置き場所を連絡

課題の提出について

- MPIパート提出期限
  - ~~6/30（月）23:50~~ ⇒ 7/14（月) 23:50
- 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
  - 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

- 6/16(月)
  - MPI (4)
  - 6/23(月)は休講
  - スケジュールについてはOCW pageも参照
    - http://www.el.gsic.titech.ac.jp/~endo/

    → 2014年度前期情報(OCW) → 講義ノート