

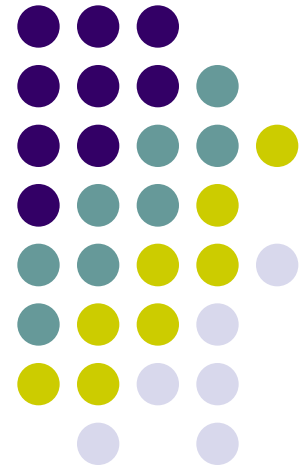
2013年度 実践的並列コンピューティング 第7回-第二版

MPIによる
分散メモリ並列プログラミング(2)

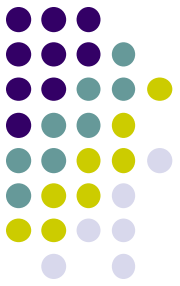
遠藤 敏夫

endo@is.titech.ac.jp

2013年5月27日



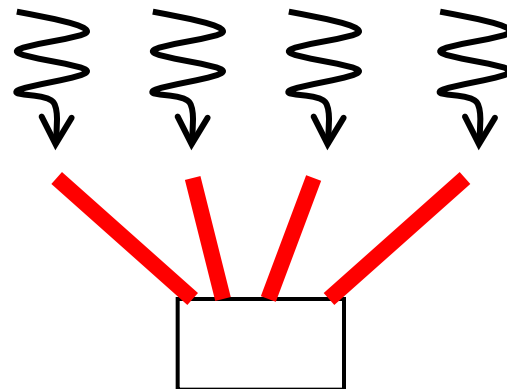
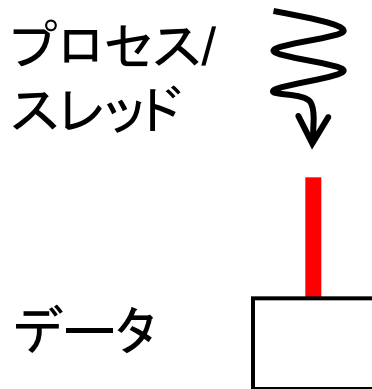
並列プログラミングモデルの、 メモリモデルによる分類



逐次

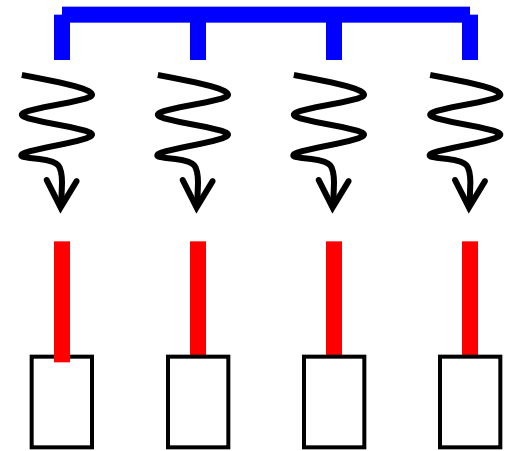
共有メモリモデル

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

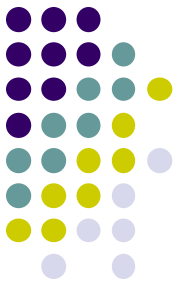
- OpenMP(言語拡張)
- pthread(ライブラリ)
- CUDA (言語拡張)



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)

MPI(message-passing interface)とは



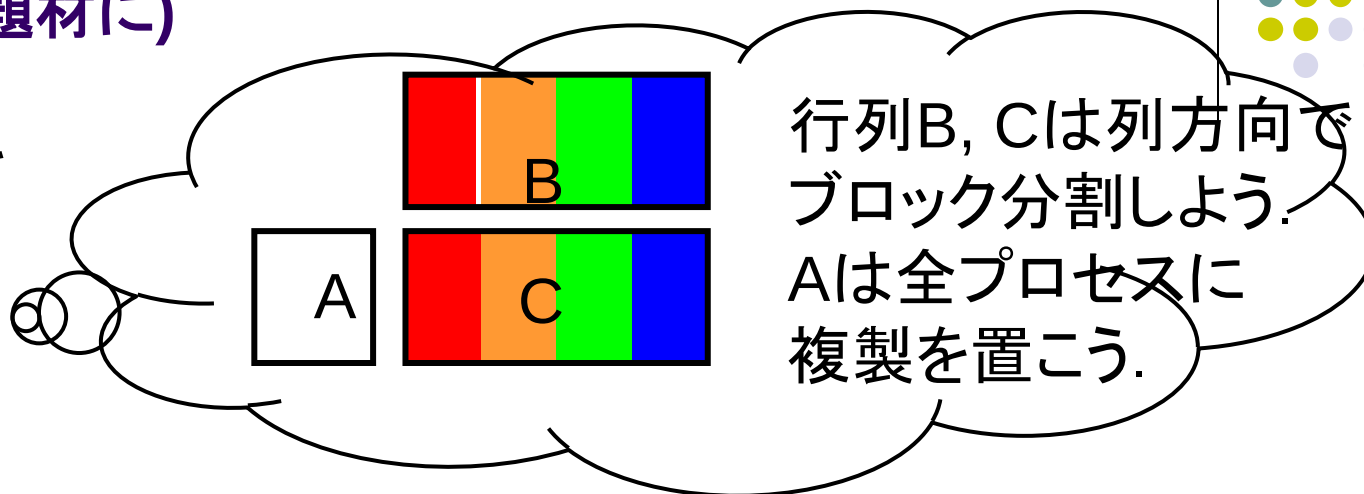
- 分散メモリ並列プログラミングの規格
- C, C++, Fortranに対応
- メッセージパッシングのためのライブラリ
- SPMDモデル. プロセス間の相互作用はメッセージで
 - MPI-2規格では, さらにRMA(remote memory access)が追加

分散メモリプログラミングとデータ配置

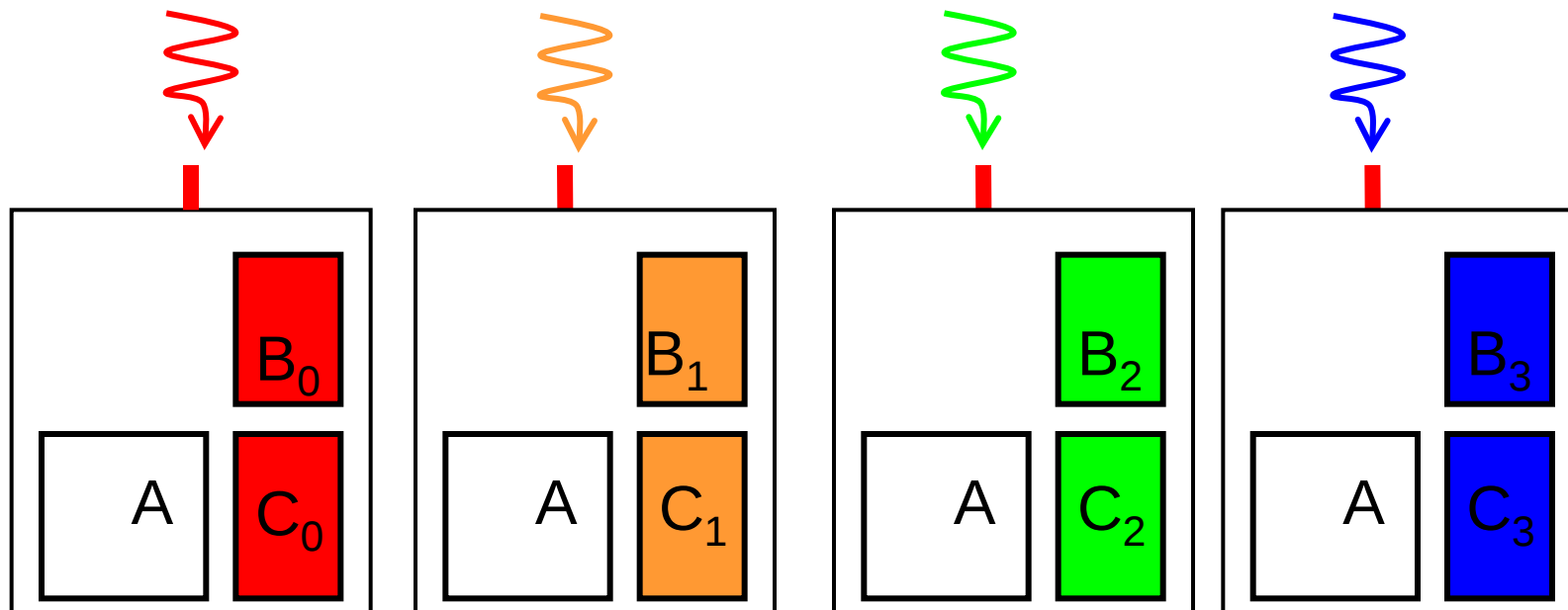
(mm-mpiを題材に)



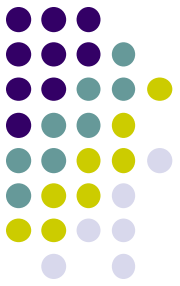
配置方法を
決める:



実際の配置をプログラミング:

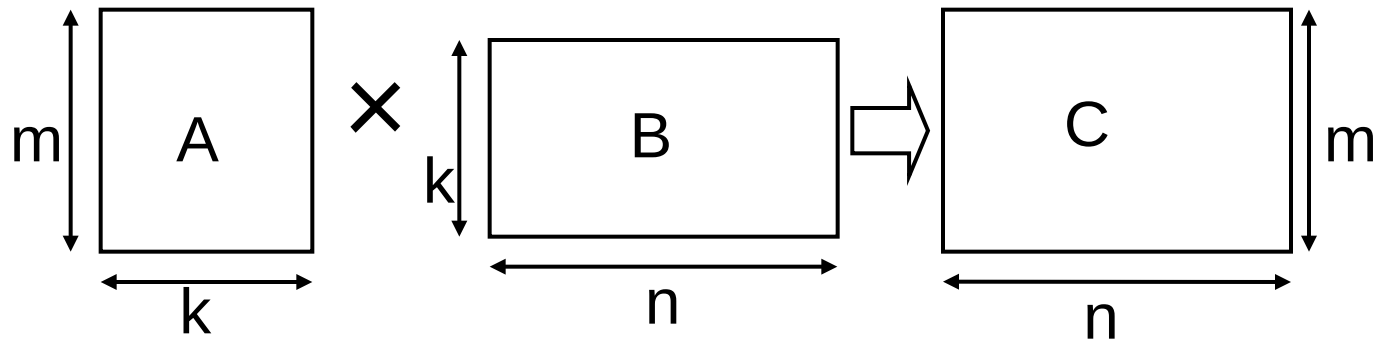


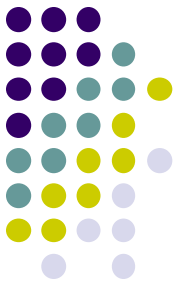
行列積サンプルプログラム (再掲)



$(m \times k)$ 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？

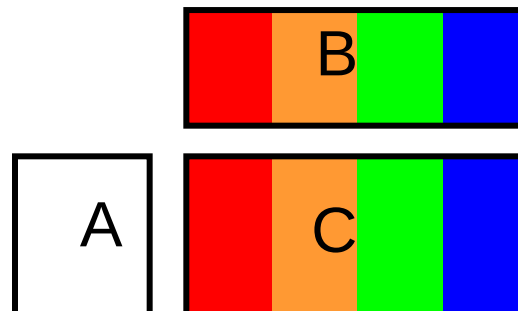
2. 計算中の配置

- 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

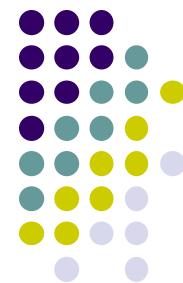
- 結果を(たとえばプロセス0に)集める必要はあるか？

(mm-mpiでは分割したままとした)

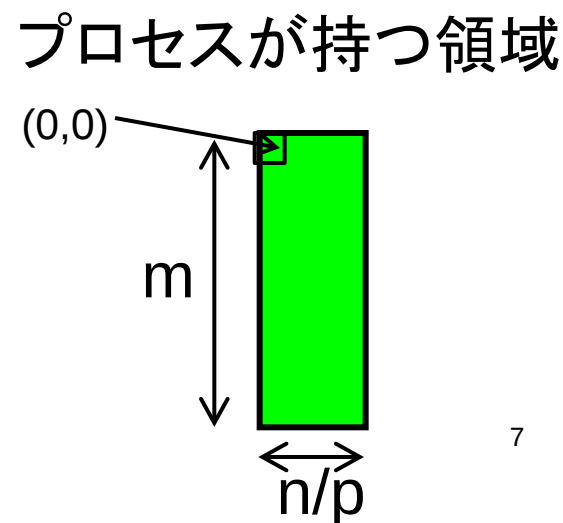
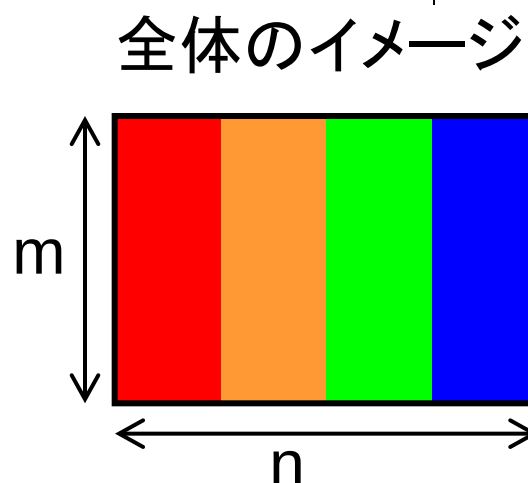


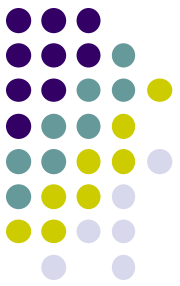
Cを列方向ブロック分割
⇒ Bも列方向の分割
Aは全複製

データ分散とプログラミング



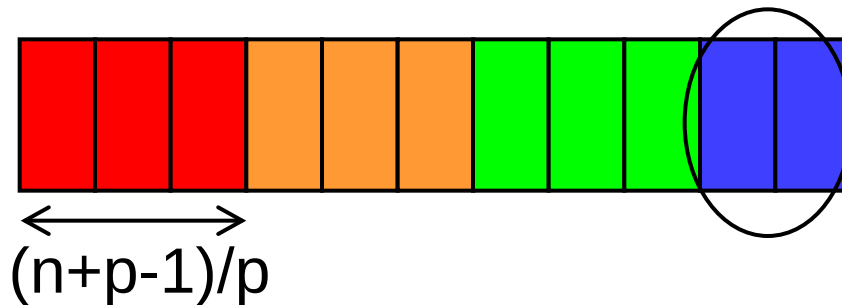
- $m \times n$ 行列を p プロセスで分割する
とはどういうことか？
 - データ並びはcolumn-majorとする
 - ここでは割り切れる場合を仮定
- 各プロセスが持つのは、 $m \times (n/p)$
の部分行列
 - $m \times (n/p) \times \text{sizeof(データ型)}$ のサイズの
領域をmallocすることに
 - 部分行列と全体行列の対応を意識
する必要
 - プロセス r の部分行列の (i, j) 要素 $\leftrightarrow$
全体行列の $(i, n/p \times r + j)$ 要素に対応





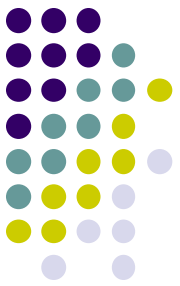
意外とめんどろな端数処理

- 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



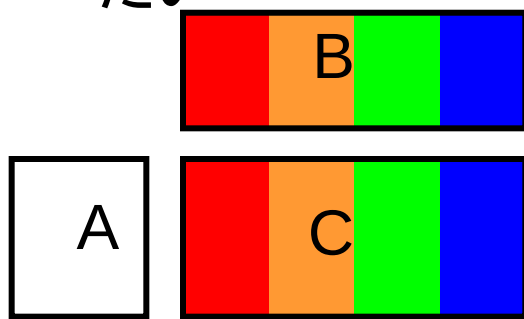
サンプルプログラムのdivide_length()関数は, 自分の担当場所の始点インデックス s と終点インデックス e を返す.
 s 以上 e 未満の, $(e-s)$ 個(列)のデータを担当することを示す.

データ配置の再検討



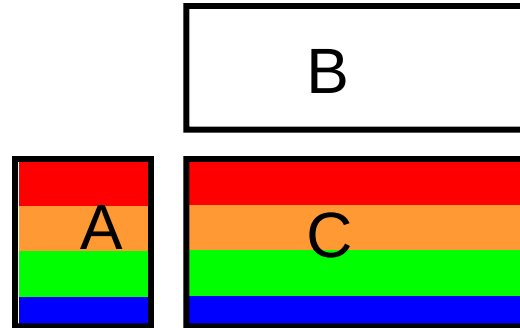
- $C_{i,j}$ の計算には, Aの第i列とBの第j列が必要

⇒ 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



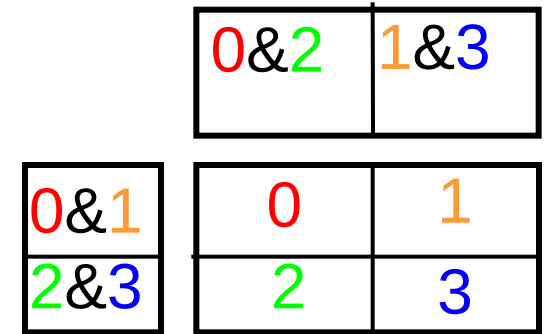
Cを列方向ブロック分割
⇒ Bも列方向の分割
Aは全複製
(mm-mpiの方法)
 $O(mkp + nk + mn)$

メモリ量の全プロセス合計



Cを行方向ブロック分割
⇒ Aも行方向の分割
Bは全複製

$O(mk + nkp + mn)$

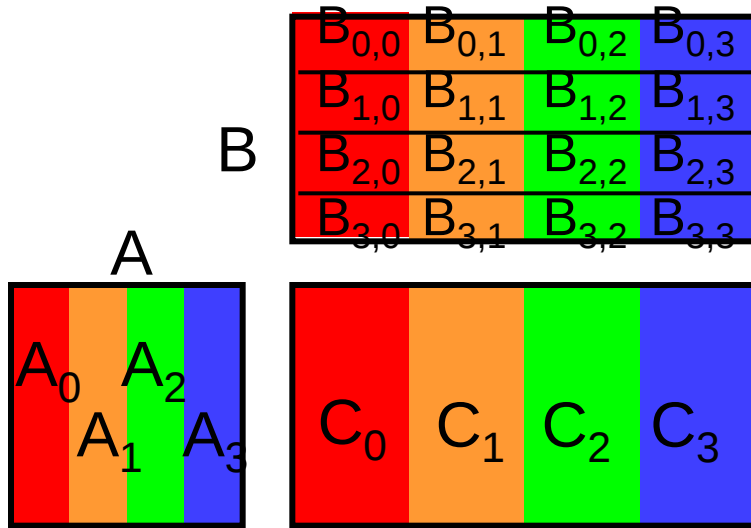
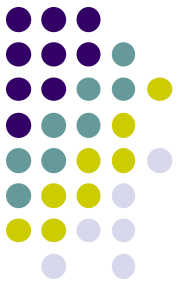


Cを二次元ブロック分割
⇒ A:行方向分割+複製
B:列方向分割+複製

$O(mkp^{1/2} + nkp^{1/2} + mn)$

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非現実的

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,

$C_i += A_0 \times B_{0,i}$ を計算

第1フェーズ:

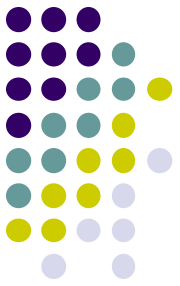
プロセス1が A_1 をBcast

各プロセスiは,

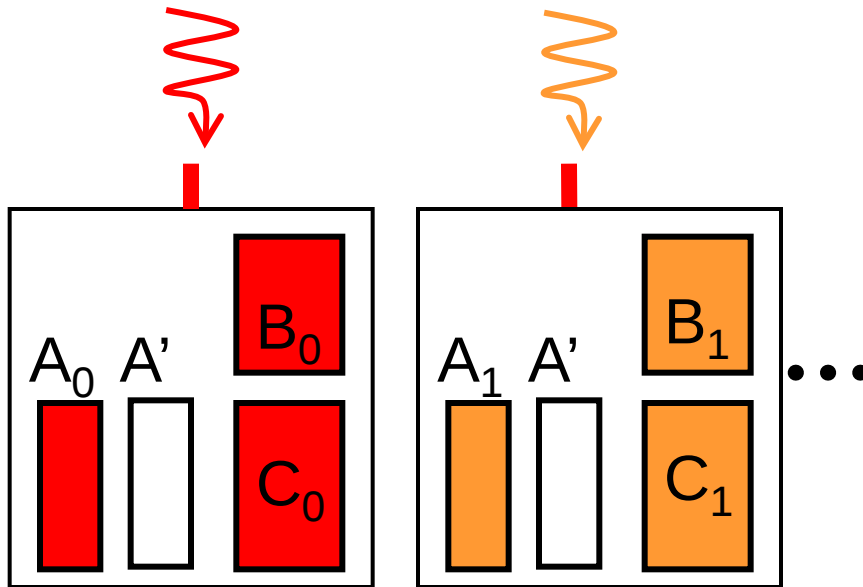
$C_i += A_1 \times B_{1,i}$ を計算

:

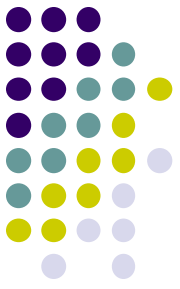
以下同様に、第(p-1)フェーズ
まで行う



実際のメモリ配置はどうなる？



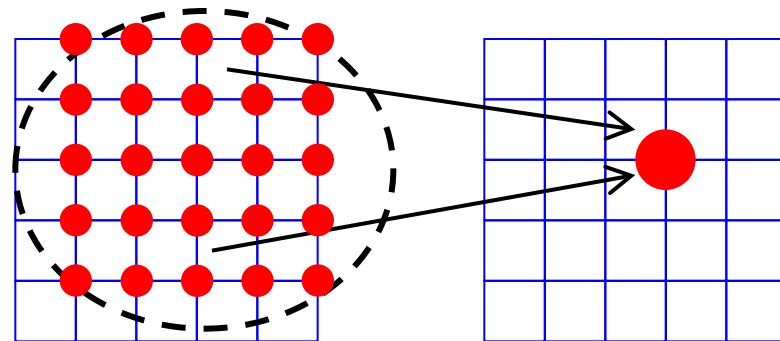
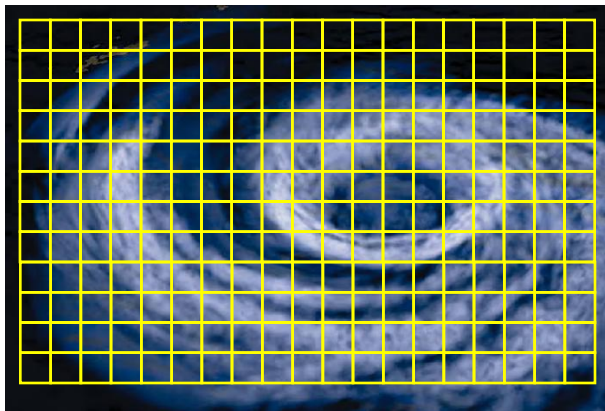
- 各プロセスは、分割されたAに加え、受信バッファ(A')を用意する
- 第rフェーズでは、
 - プロセスrはAからA'へデータコピー（省略する手法もあり）
 - プロセスrをルートとし、領域A'をMPI_Bcast
 - 各プロセスiは、 $A' \times B_{r,i} \rightarrow$ 自分の部分 C_r



Advectionのデータアクセス

二次元格子空間の，ポアソン方程式の反復解法

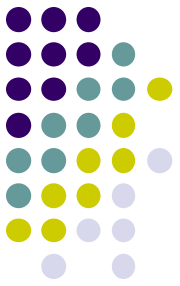
- $f(x,y)$ がその点の圧力もしくはは温度
- 空間の端の f が与えられたときの，内部の定常状態
- 空間全体を計算してから，次の時間ステップへ
- ダブルバッファリング



隣の点の情報を用いて値更新
(Advectionでは二つ隣まで使用するが、以下では一つ隣として説明)

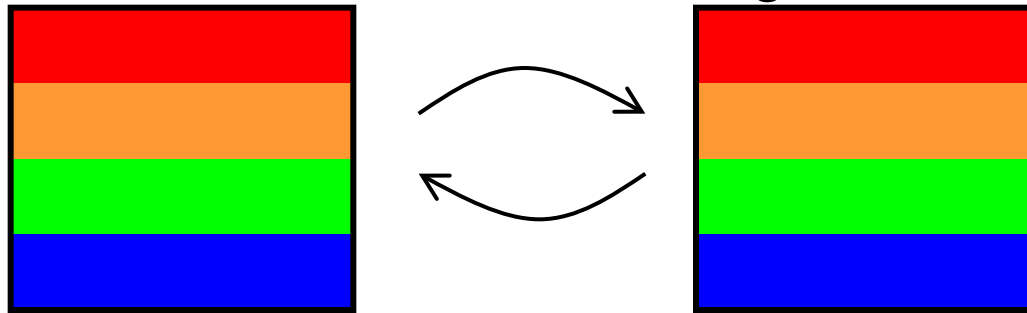
並列化はどうする？ (OpenMP, MPI)

Advectionの並列化方針

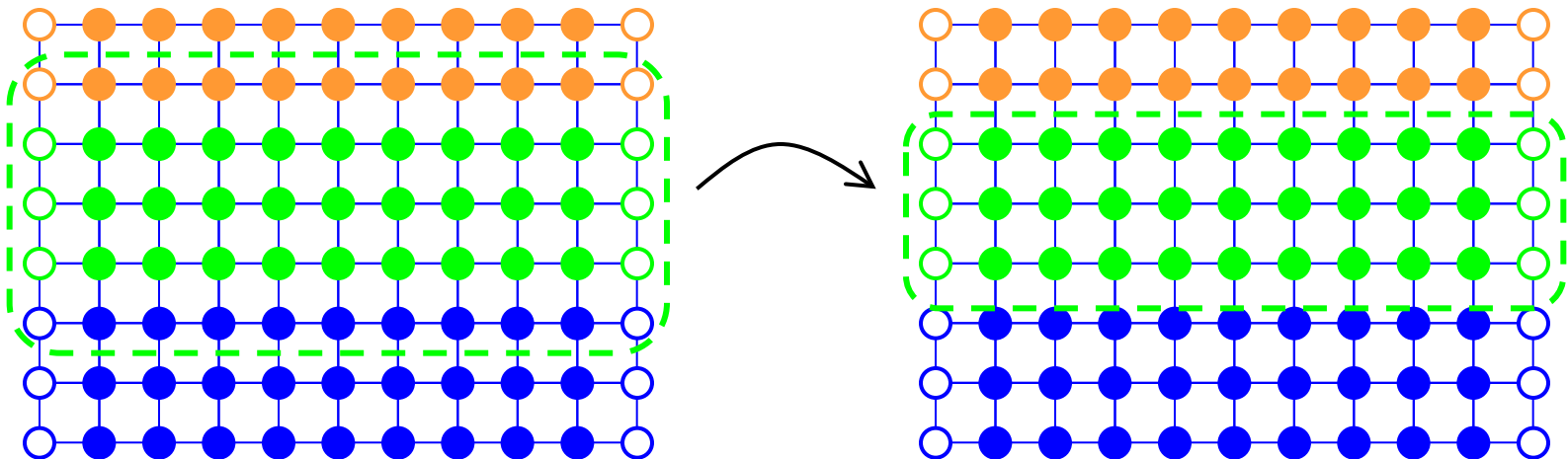


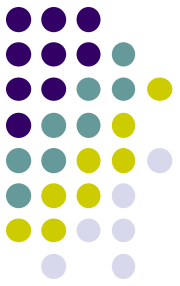
- 二次元配列をそれぞれ行方向分割

Double buffering



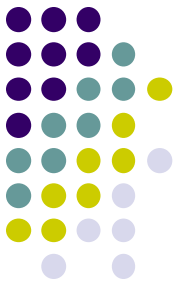
- 各プロセスがwriteする領域よりもreadする領域が大きい → プロセス間に依存関係





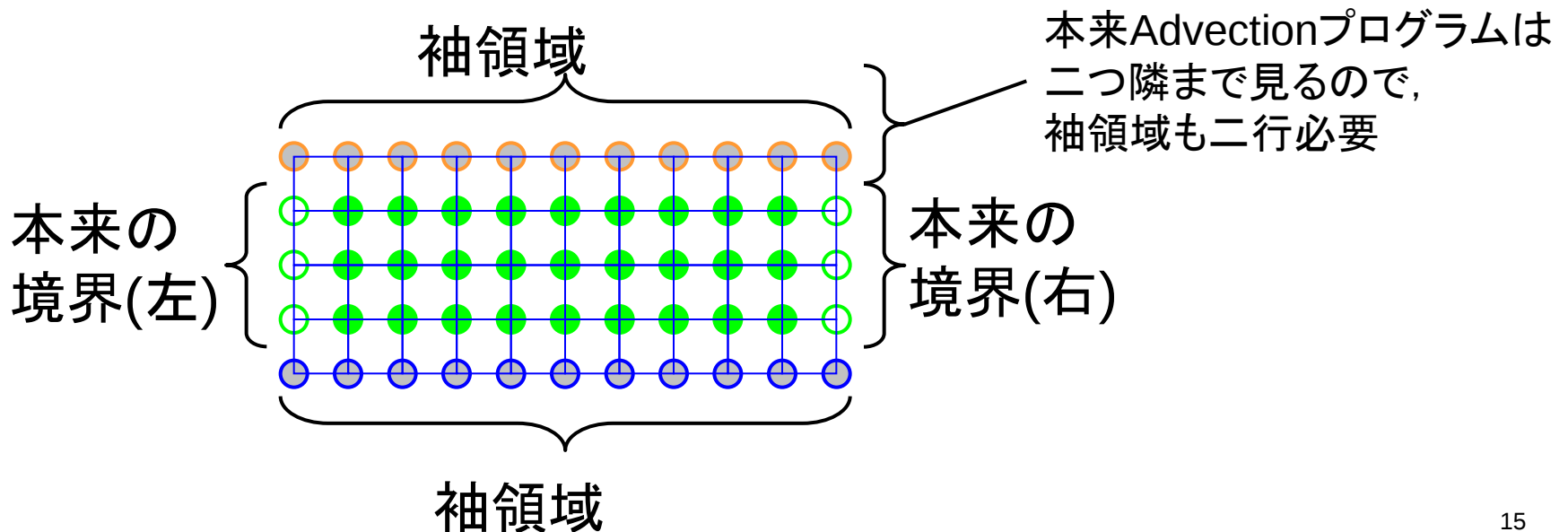
OpenMPでは

- データ構造は逐次と同じまま, forループを並列化すればよい
- 隣のスレッドのデータもそのまま読める, が...
- スレッド間で足並みをそろえる必要 → バリア同期

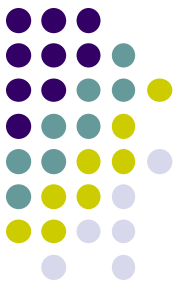


MPIによる並列化 (1)

- 各プロセスは、自分の担当領域配列を持つ
 - 最初と最後のプロセスは、上下境界部分に注意
 - 端数処理
- 隣プロセスのデータを読むためには、send/recvが必要
- 「袖領域(のりしろ領域)」つきの配列を持つのがよい



MPIによる並列化 (2)



簡単化のため、以下ではのりしろ領域一行として説明

```
for (it = 0; it < nt; it++) {
```

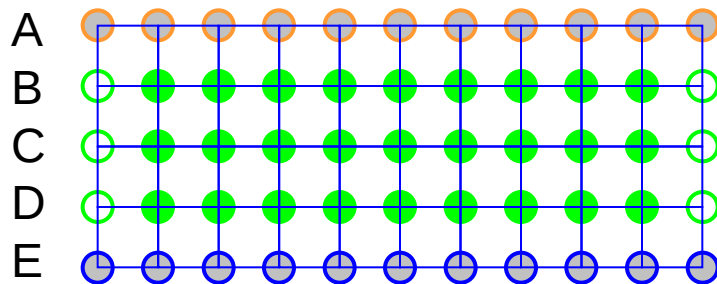
行Bを前のプロセスへ送信, Dを次のプロセスへ送信

行Aを前のプロセスから受信, Eを次のプロセスから受信 (注)

B-Dの全点を計算

二つの配列の切り替え

}

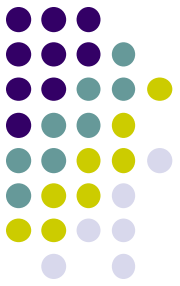


(注)これはデッドロックするダメなプログラム.

その理由と解消がテーマ

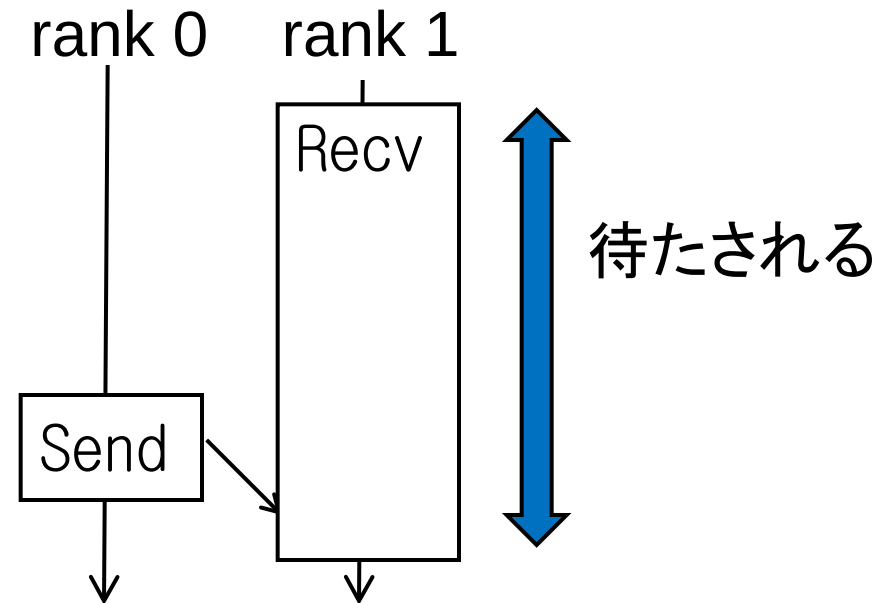
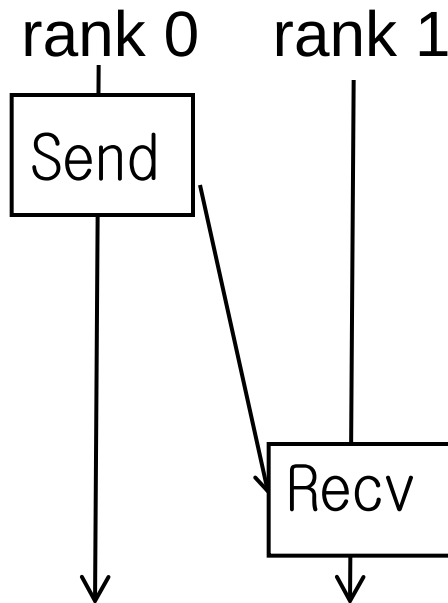
デッドロック(deadlock)とは、互いに「待ちあって」プログラムが進まなくなること

ブロッキング(blocking)通信とは: Recvの場合

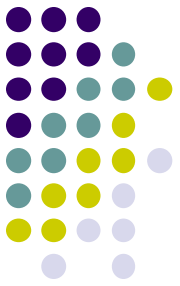


- あるプロセスがMPI_Send, もうひとつのプロセスがMPI_Recvを呼ぶとき、どちらが先かは分からない
- MPI_Recvは, メッセージが届くまで待たされる → いつもblocking通信

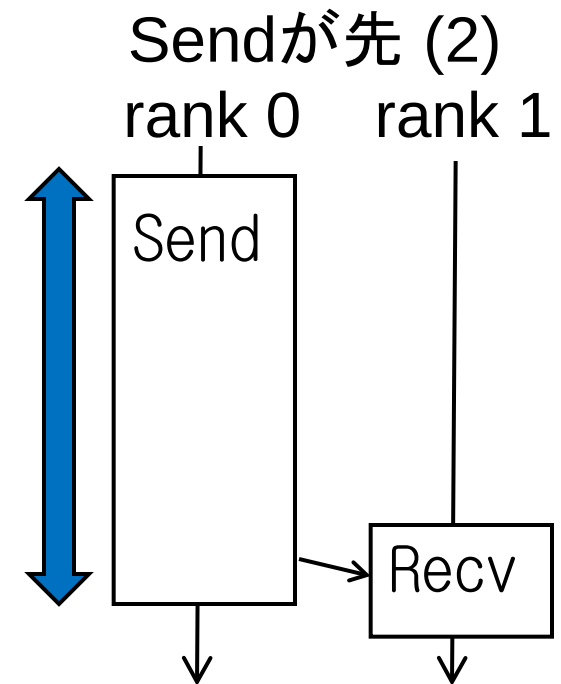
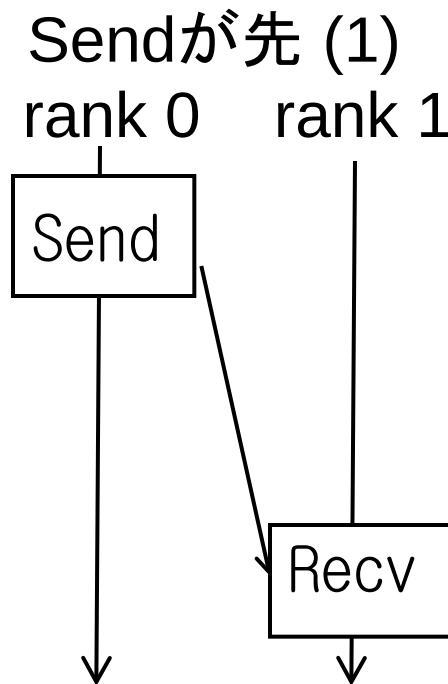
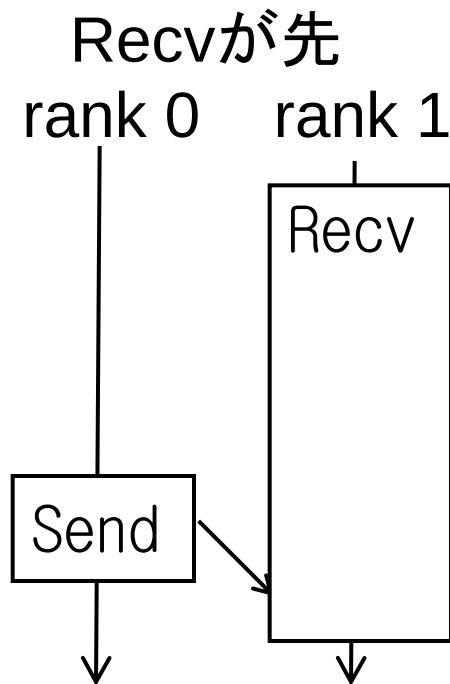
Sendが先に呼ばれた場合 Recvが先に呼ばれた場合

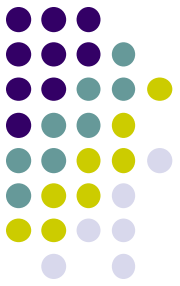


ブロッキング(blocking)通信とは: Sendの場合



- MPI_Sendは、場合によって挙動が違う
 - (1) すぐ終了する場合 (non-blocking)
 - (2) 対応するRecvが呼ばれるまで待たされる場合(blocking)





MPI_Sendがblockする場合

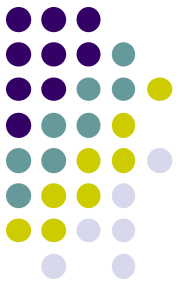
- MPI_Sendの挙動
 - メッセージがある閾値より大きいときはblocking
 - 小さいメッセージでも、Sendを繰り返して、合計のサイズが閾値を超えるとblocking
 - それ以外(小さいメッセージを単発など)はnon-blocking

閾値はMPI処理系依存

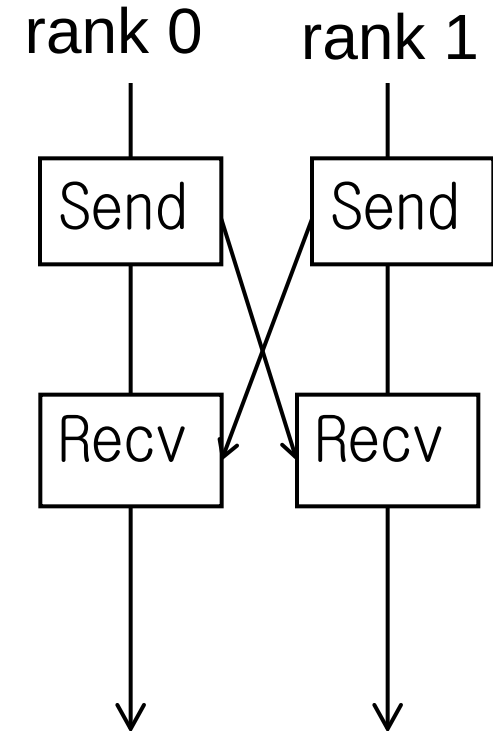
MPI処理系の内部にメッセージの置き場が無いため

→ プログラマは、どちらの挙動だとしても、動くプログラムを作る必要

ブロッキング通信の問題



- 待っている間の時間がもったいない
 - 計算と通信のオーバーラップをしたい
- 一見自然なプログラムがデッドロックする場合も
 - 右の、メッセージを交換するプログラムは動くか??
- メッセージが大きいときにはデッドロック
小規模では動き、大規模では動かない厄介なバグに.



Advectionはこれより少しややこしい。Rank 1は、0とも2とも通信する必要があるの
で。

交換プログラムのデッドロック解決



解決1:

rank0 ではSend→Recv

rank1ではRecv→Send

解決2: MPI_Sendrecv関数

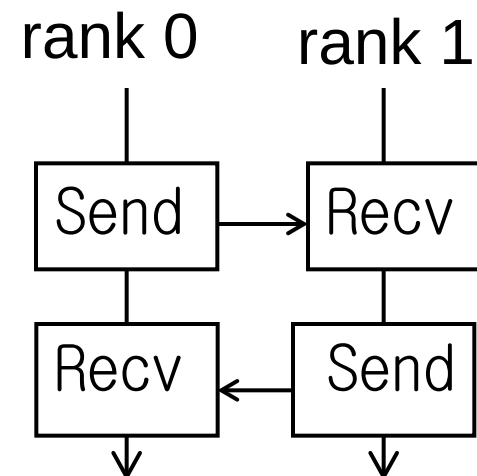
MPI_Sendrecv(

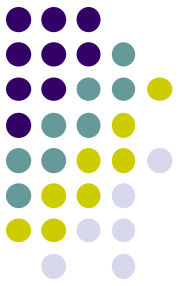
sbuf, sn, stype, dest, stag,

rbuf, rn, ttype, src, rtag,

comm, stat)

解決3: ノンブロッキング通信

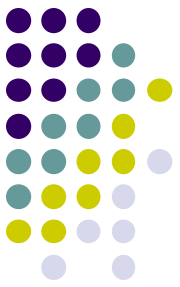




ノンブロッキング通信とは

- データの送受信の指示だけまず行い、その**完了を待たない**こと
 - 送信、受信とも、ノンブロッキング用MPI関数が存在
 - その後、他の処理を行うことができる
- 後で別途、完了の確認を行う必要がある

ノンブロッキング(non-blocking)通信: Recvの場合



```
MPI_Status stat;  
MPI_Recv(buf, n, type, src, tag, comm, &stat);
```

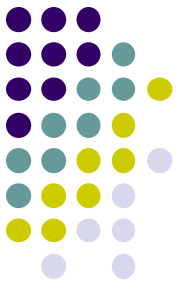
```
MPI_Status stat;  
MPI_Request req;  
MPI_Irecv(buf, n, type, src, tag, comm, &req); ←受信開始  
MPI_Wait(&req, &stat); ←完了待ち
```

MPI_Irecv: 受信を開始するが, すぐ終了

MPI_Waitを行うと受信を待つ → その終了後, メッセージを利用可能

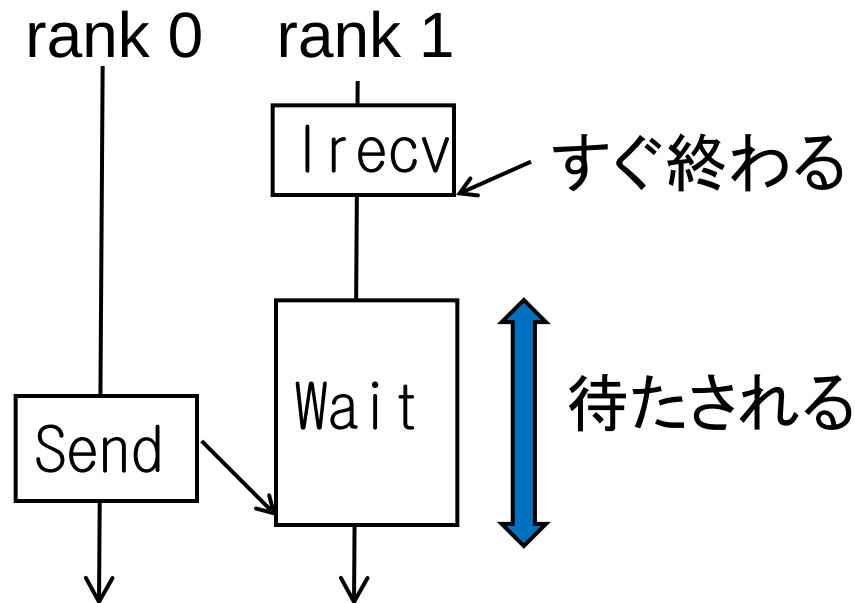
IrecvとWaitの間に別の仕事をすることができる

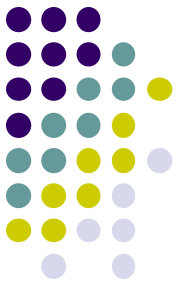
MPI_Requestは、通信を表す「チケット」のようなもの



MPI_Irecvの動き

- Irecv自身はすぐ終わる(non-blocking)
- が、データを使えるのはWait以降





ノンブロッキング通信: Sendの場合

```
MPI_Send(buf, n, type, dest, tag, comm);
```

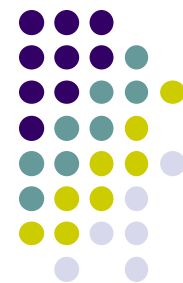
```
MPI_Status stat;  
MPI_Request req;  
MPI_Isend(buf, n, type, dest, tag, comm, &req);  
MPI_Wait(&req, &stat);
```

MPI_Isend: 送信を開始するが, すぐ終了

MPI_Waitを行うと送信完了を待つ

注: MPI_Waitが終了するまで, buf領域を他目的に使ってはいけない

ノンブロッキング通信による 交換プログラムのデッドロック解決



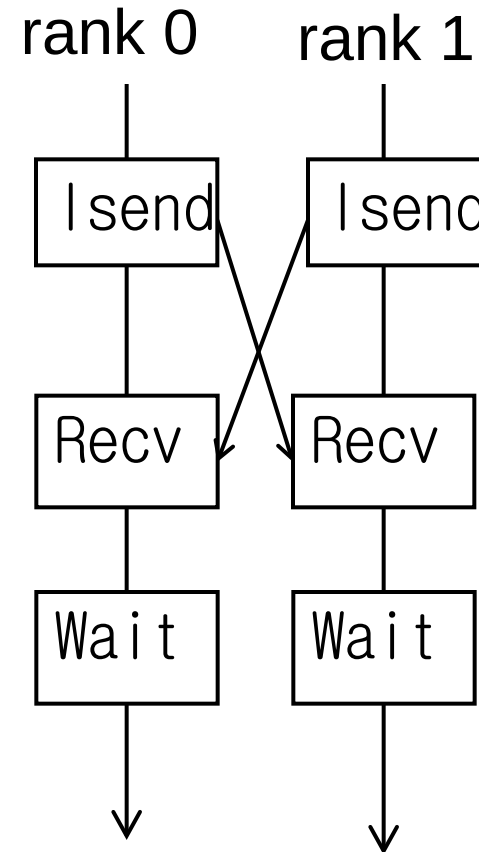
- Isend, Recv, Waitの順で呼び出すと、デッドロックしない

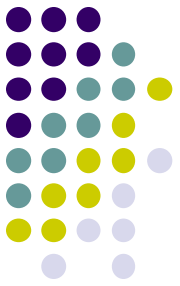
以下でも同じ効果

- Irecv, Send, Wait
- Isend, Irecv, Wait, Wait
 - MPI_Requestが2つ必要
- Advectionでは、両隣と交換する必要があるので...

Isend, Isend, Irecv, Irecv, Wait,
Wait, Wait, Waitなど

この場合はMPI_Request4つ





MPI_Waitの仲間

`MPI_Wait(&req, &stat);` → reqに対応する通信を待つ

`MPI_Status stats[...]; MPI_Request reqs[...];`

`MPI_Waitall(n, reqs, stats);` → reqs全部を待つ

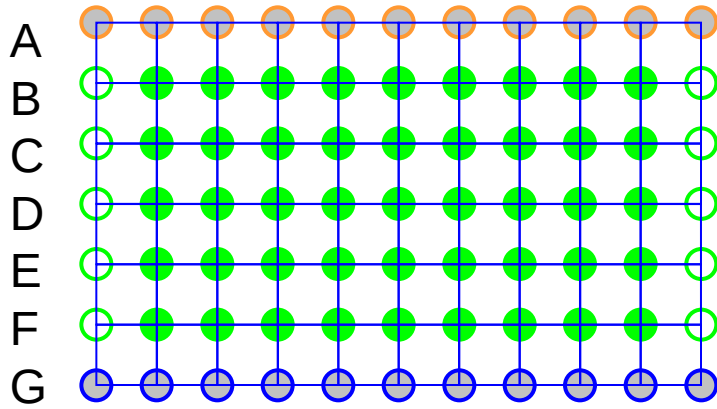
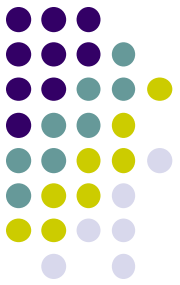
`MPI_Waitany(n, reqs, &idx, &stat);` → reqsのどれかを待つ

`MPI_Test(&req, &flag, &stat);`

→ `MPI_Wait`に似るが、すぐ終了。通信完了していればflagに0以外が帰る。

`MPI_Testall`, `MPI_Testany`あり

ステンシル計算の、デッドロックしないMPI並列化



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

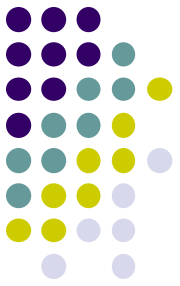
行Aを前のプロセスから受信開始, Gを次のプロセスから受信開始

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

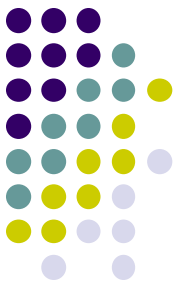
}



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)

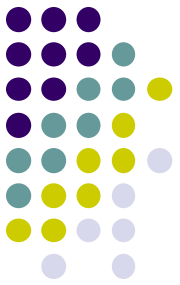
以下の[1]—[3]のどれか一つについてレポートを提出してください。二つ以上でも良い。

[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

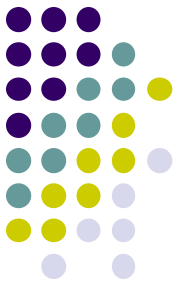
- MPIで一般のサイズに対応するには端数処理が必要である。本レポートではそれは必須ではない
- より良いアルゴリズムにしてもよい。ブロック化・計算順序変更でキャッシュミスが減らせないか？



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

- 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.

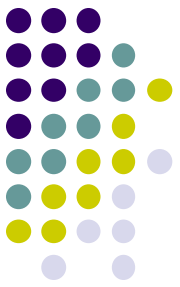
- 単純な並列化で済む問題ではないことが望ましい
 - スレッド・プロセス間に依存関係がある
 - 均等分割ではうまくいかない、など

- 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)...

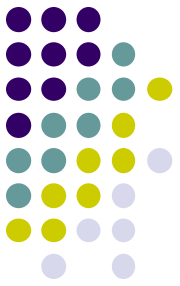
入力データは自分で作る必要あり

- たとえば, 自分が研究している問題



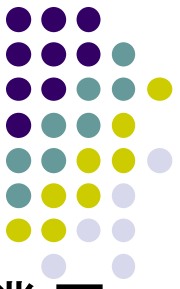
課題の注意

- 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



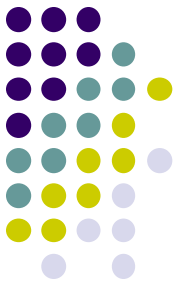
About Account

- 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

- 6/3(月)
 - MPI (3)
- 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