

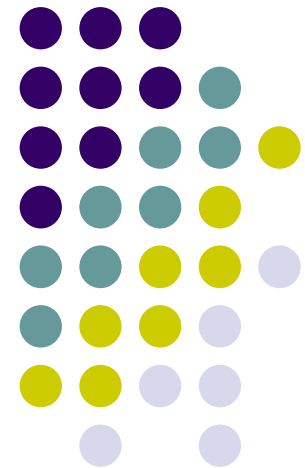
# 2015年度 実践的並列コンピューティング 第9回

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](mailto:endo@is.titech.ac.jp)

2015年6月15日



# TSUBAMEのジョブ投入オプションの 続き(1)

## t2subのオプション



- ジョブが利用メモリ量(ノードあたり)が多そうな場合
  - `-l select=XX:mpiprocs=YY:mem=44gb` → ノードあたり44GBまでメモリを使える
    - 「プロセスあたり」でなく「ノードあたり」
    - 指定しないと1GBまで
    - マシンが持つ54GBぎりぎりには指定できない。OSオーバヘッドなど
- ジョブの実行時間が一時間を超えそうな場合
  - かかりそうな実行時間+ $\alpha$ を予測しておく。+50%くらいに収まると望ましい
  - `-l walltime=6:00:00 -et 1` → 打ち切り時間は6時間
    - etはextend timeの略
  - 指定しないと一時間とみなされ、そこで打ち切り

# TSUBAMEのジョブ投入オプションの 続き(2)



- -l (小文字エル)オプションはくせがつよいので注意
  - select, mpirprocs, memなどはコロンでつないで一つの"-l"に指定の必要
  - walltimeやplaceは独立の"-l"オプションとする必要

## [投入例]

```
t2sub -q S -W group_list=t2g-ppcomp
```

```
-l select=4:mpirprocs=12:mem=44gb -l place=scatter
```

```
-et 1 -l walltime=3:00:00 ./job.sh
```

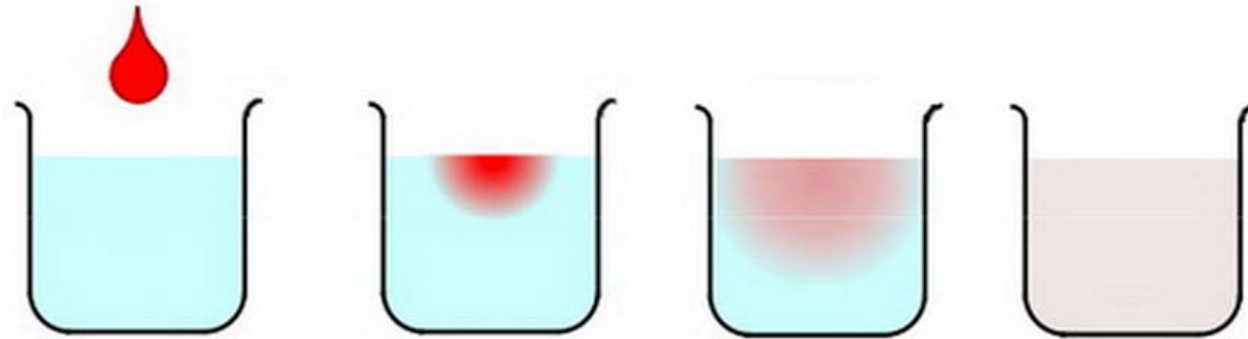
- 1時間以内のときは-etをつけない(短時間割引がきく)
- 詳細はTSUBAME利用の手引き5章参照



# diffusionプログラム(再掲)

拡散現象

コップの中の水に赤インクを落す



次第に拡散して赤インクは拡がって行き、最後は均一な色になる

- 各点のインク濃度は、時間がたつと変わっていく →  
その様子を計算機で計算
  - 天気予報などにも含まれる計算
- 計算量:  $O(n_x \times n_y \times n_t)$ 
  - $n_x, n_y$ : 空間サイズ.  $n_t$ : 時間ステップ数

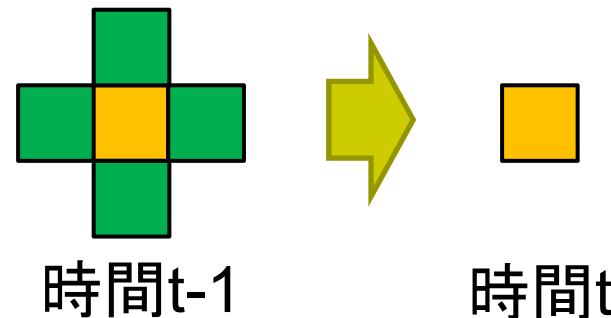
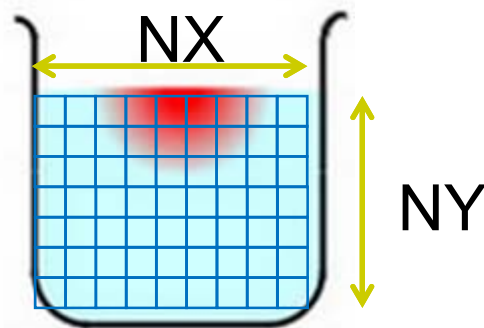
© 青木尊之

# Diffusionのデータアクセス



二次元格子空間を二次元配列で表す

- 空間全体を計算してから、次の時間ステップへ
- 時間0の値(初期条件)と空間の端(境界条件)はgiven
- ダブルバッファリング技術



隣の点の情報を用いて値更新

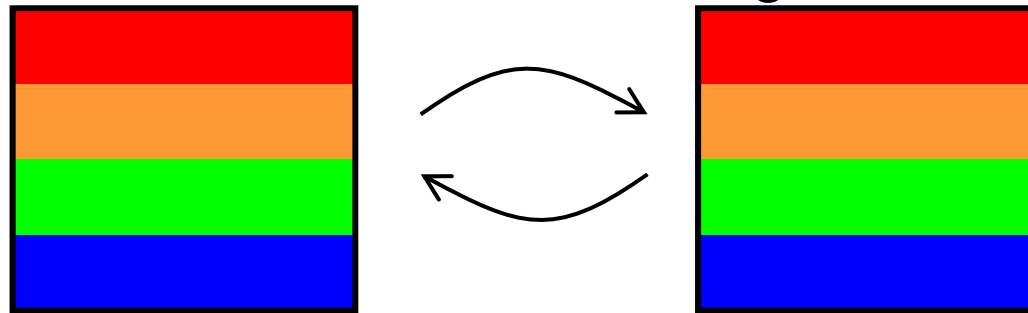
並列化はどうする？ → MPIではプロセスの境界が問題

# Diffusionの並列化方針

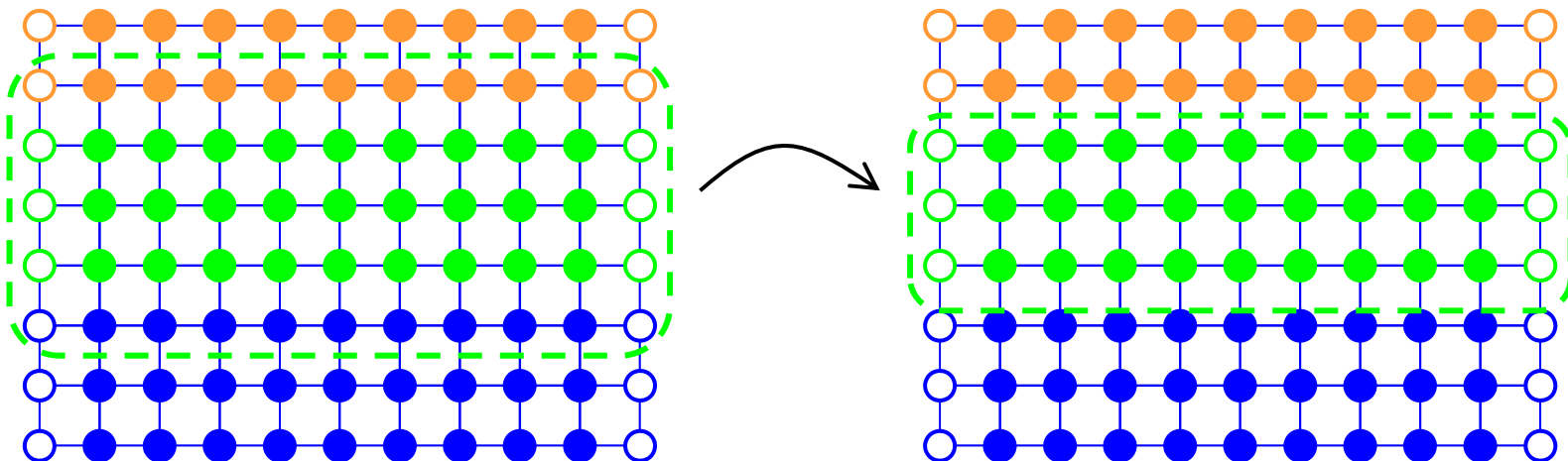


- 二次元配列をそれぞれ行方向分割

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





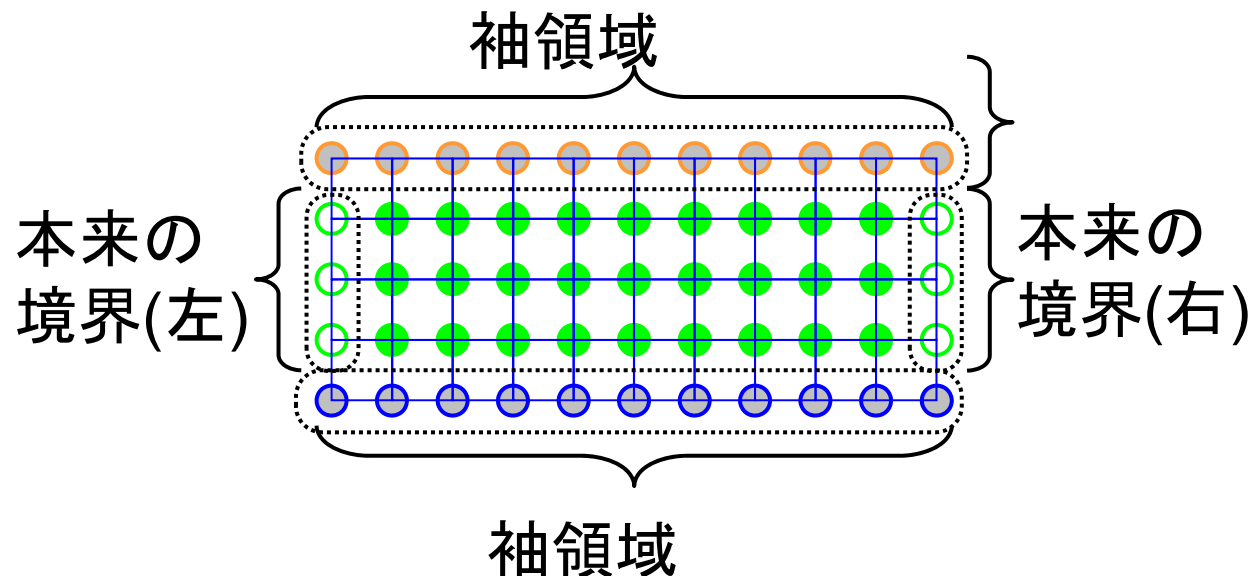
# OpenMPではどうだったか

- 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
- parallel regionの終了時に自動でそろえられていた(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++) {
```

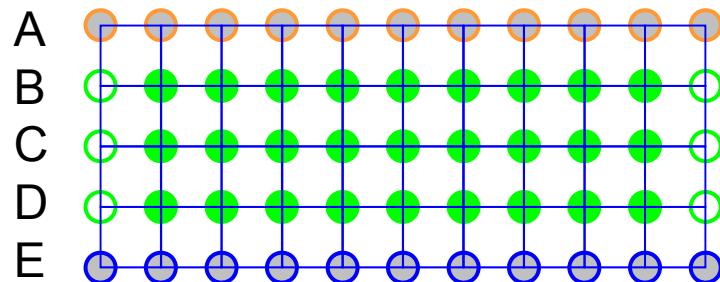
行Bを前のプロセスへ送信, Dを次のプロセスへ送信

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.

その理由と解消が以下のテーマ

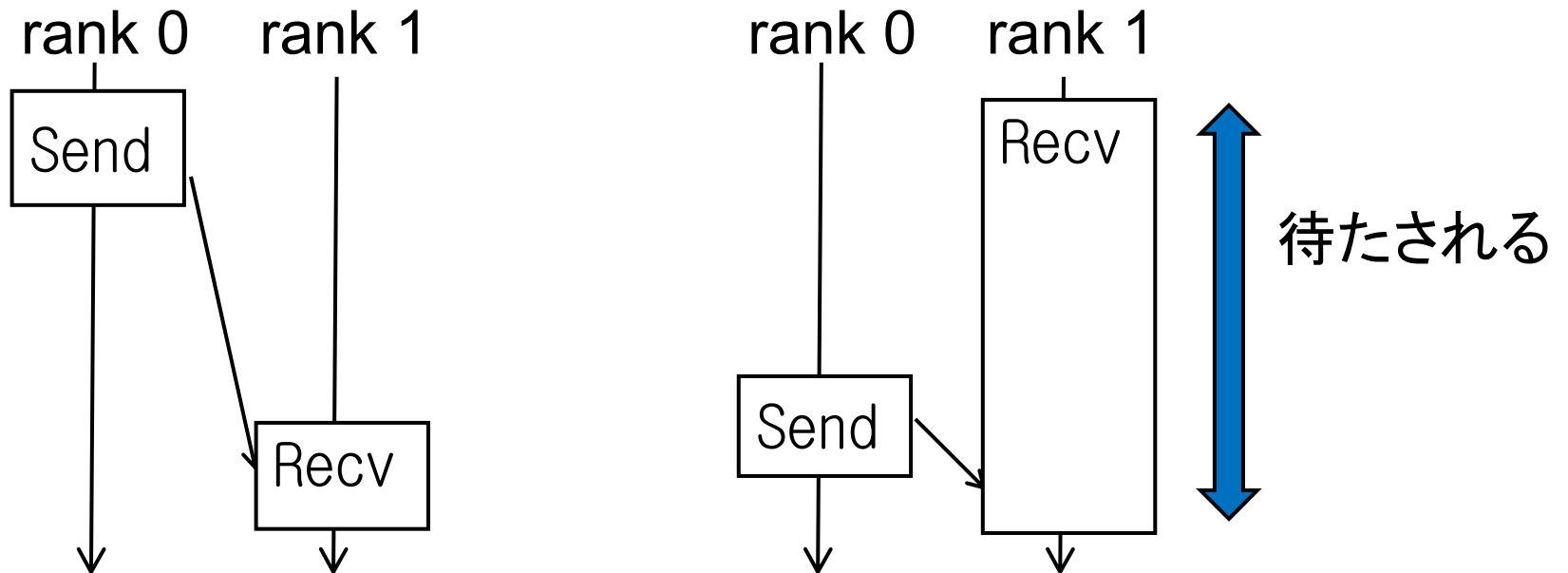
デッドロック(deadlock)とは、互いに「待ちあって」プログラムが進まなくなる事

# 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

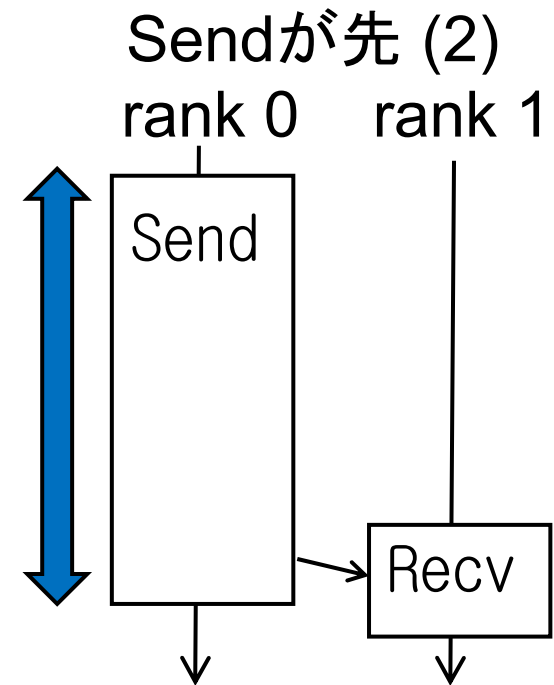
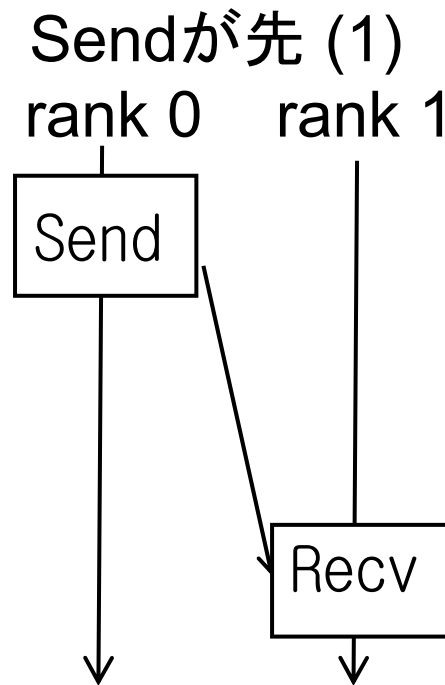
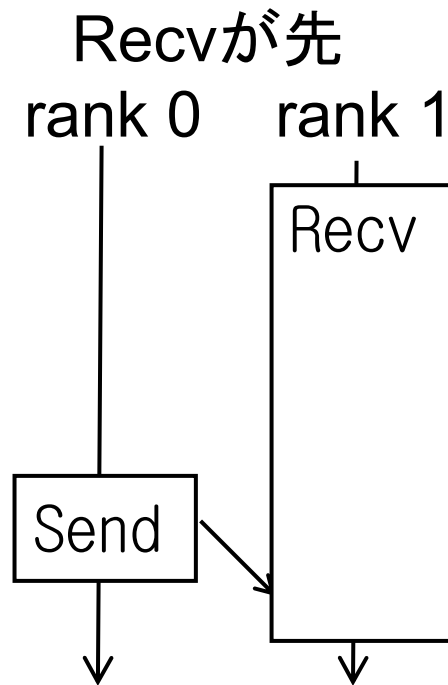
Sendが先に呼ばれた場合      Recvが先に呼ばれた場合



# 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

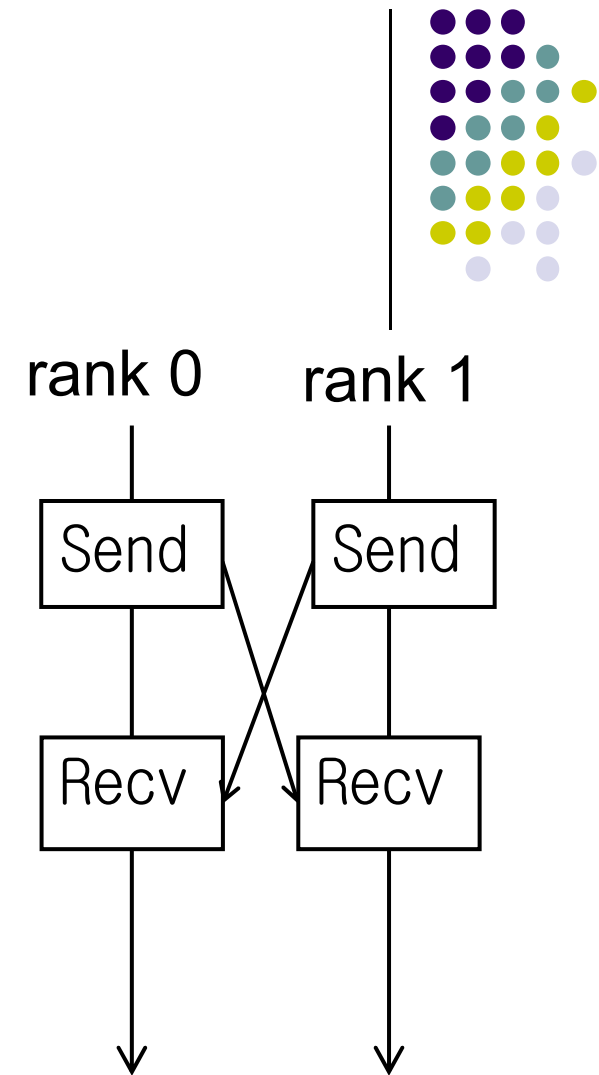
MPI処理系の内部にメッセージの置き場が無いため

→ 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

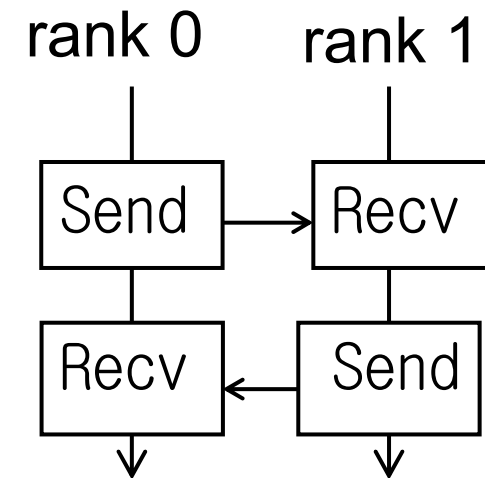
解決2: MPI\_Sendrecv関数

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

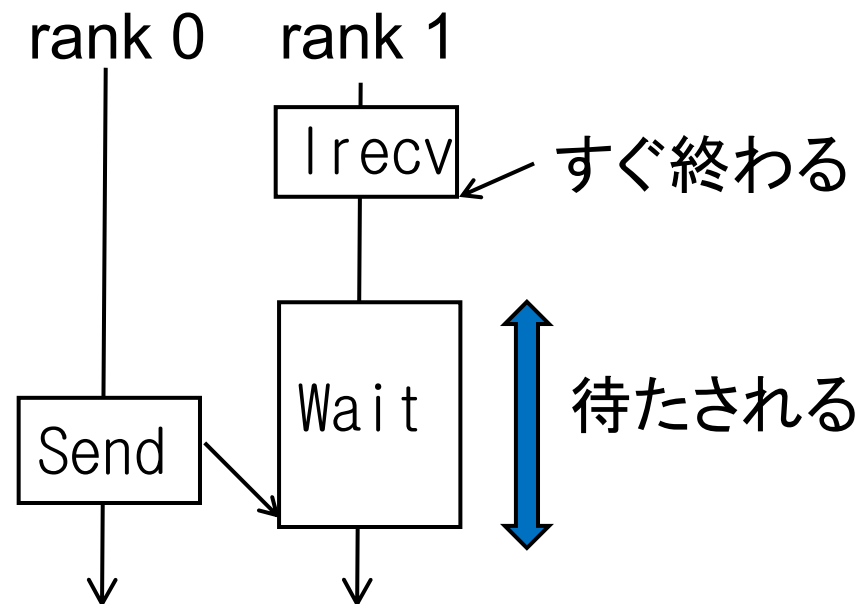
IrecvとWaitの間に別の仕事をすることができる

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

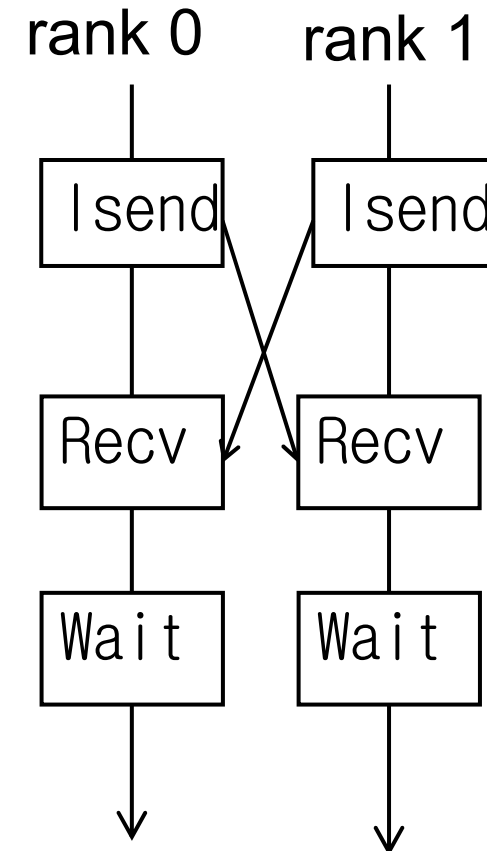
MPI\_Waitを行うと送信完了を待つ

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

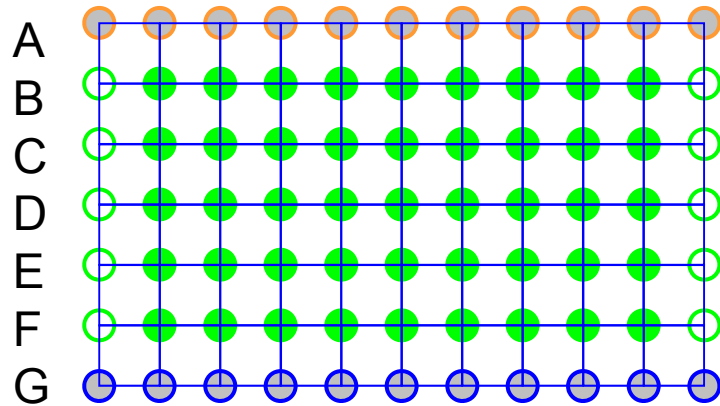
`MPI_Waitany(n, reqs, &idx, &stat);` → reqsのどれかを待つ

`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

- 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が減らせないか？

# 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
- 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を試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
  - 7/13(月) 23:50 (変更しました)
- 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](mailto:ppcomp@el.gsic.titech.ac.jp)
  - メール題名: ppcomp report



# 次回

- 6/22(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/>  
→ 2015年度前期情報(OCW) → 講義ノート