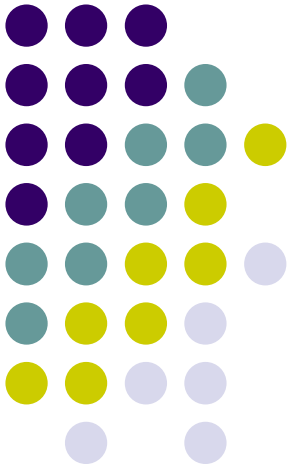
2015年度
実践的並列コンピューティング
第11回



MPIによる
分散メモリ並列プログラミング(4)

遠藤 敏夫

endo@is.titech.ac.jp

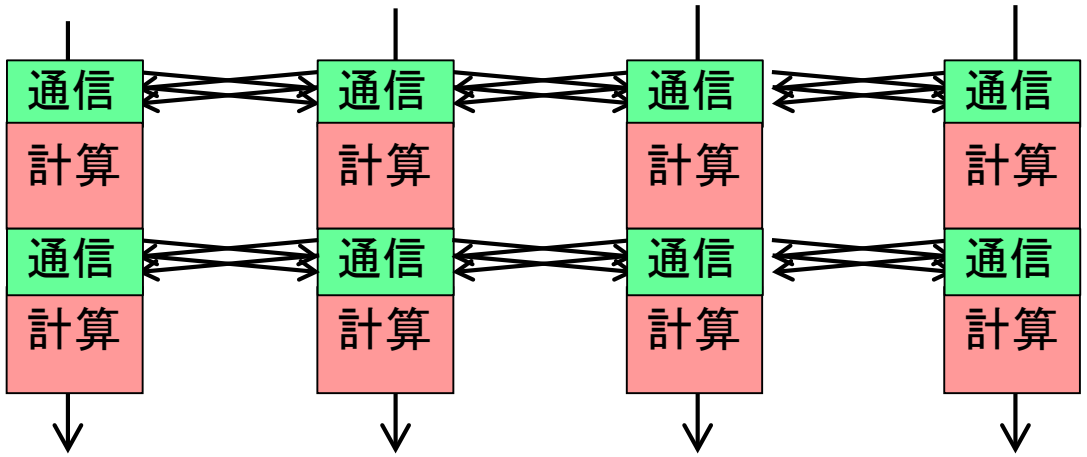
MPIプログラムの性能を考える

- 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

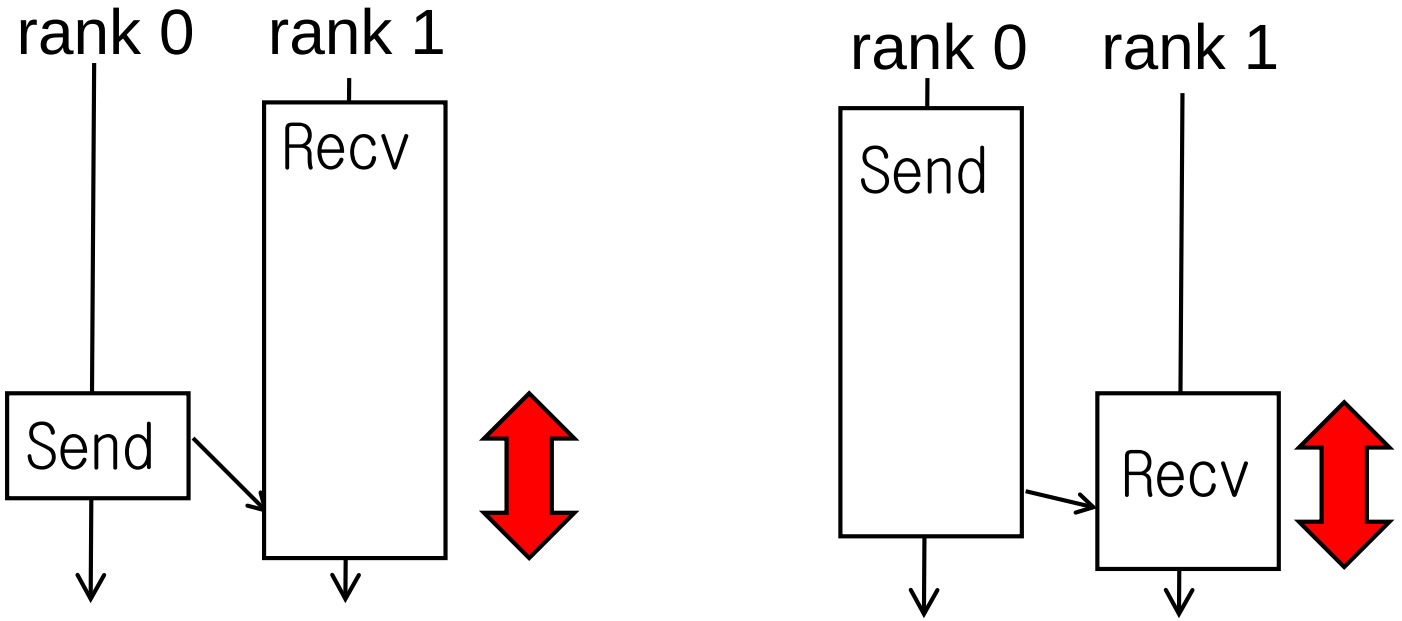
計算量、(プロセス内)ボトルネック有無
メモリアクセス量、計算不均衡…など関係

MPI版
ステンシル
計算の挙動



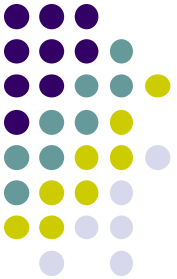
通信時間は何によって決まるか

- MPIプログラムの性能を理解するためには、通信時間の理解が必要



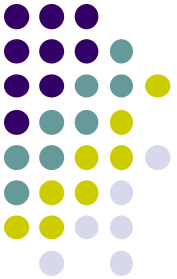
- おおまかには、待ち合わせ時間 + 実際の転送時間

MPI並列プログラムの性能を上げるには?

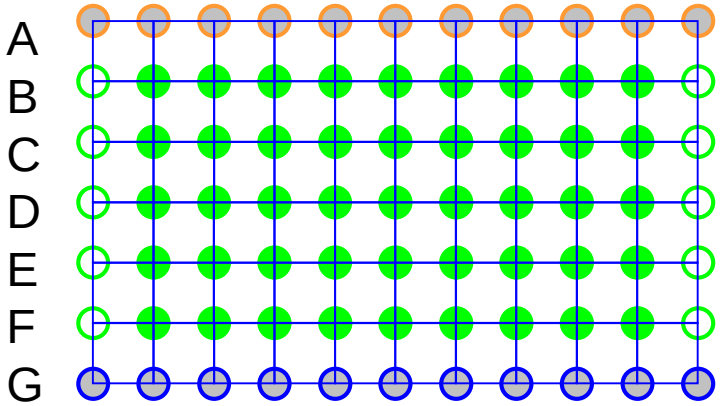


- 計算時間を短くする
  - 計算量の改良
  - ノード内の並列度向上
  - キャッシュをうまく使う
- 通信時間を短くする
  - 通信量の改良
  - 使えるところでは集団通信をうまく使う
- 通信待ち合わせ時間を短くする
  - プロセス間で仕事量を均等化する
- 計算時間と通信時間をオーバーラップする

ステンシル計算の工夫:
計算と通信のオーバラップ(1)



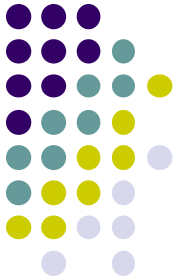
データの依存関係をより詳細に考えると、他プロセスのデータを必要としない計算が存在する！



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)



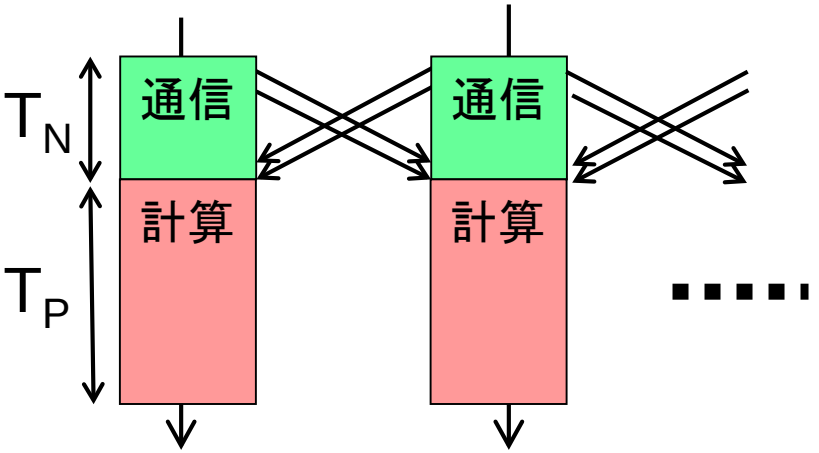
- オーバラップを組み込んだMPIプログラムの流れ

```
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)

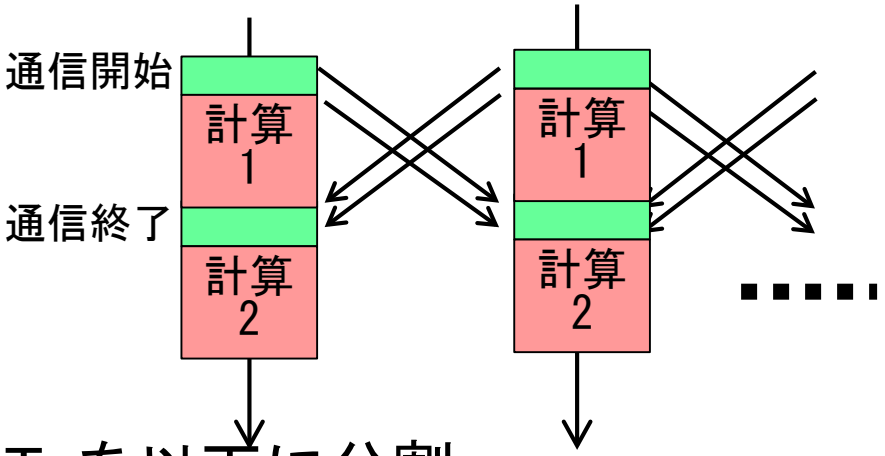
時間ステップあたりの全体時間をT、通信時間T_N, 計算時間T_Pとする

オーバラップなし



$T=T_N+T_P$

オーバラップあり



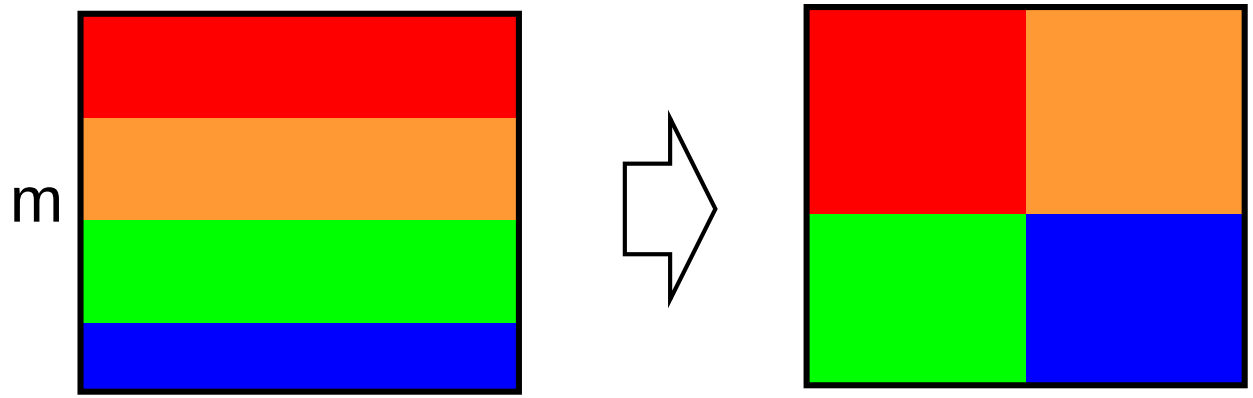
T_Pを以下に分割

- 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

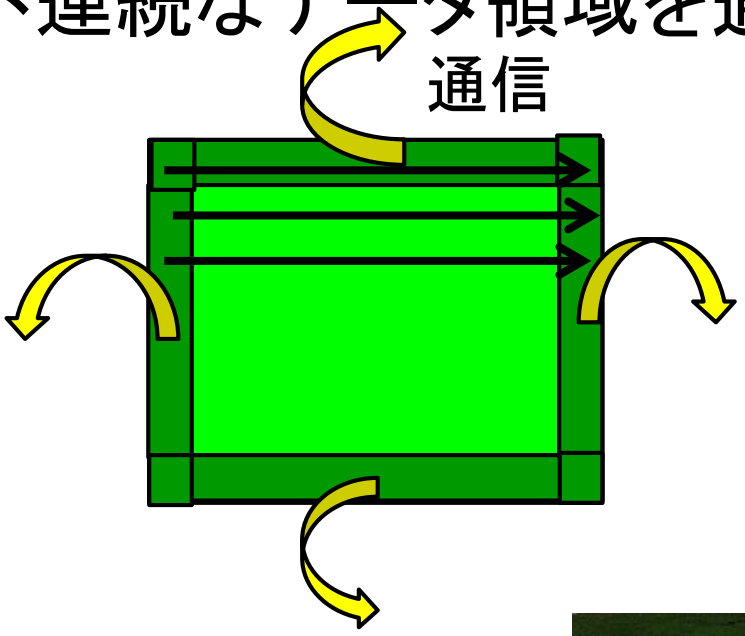
- 計算量: O(mn/p)
- 通信量: $O((m+n)/p^{1/2})$

通信量を削減可能

多次元分割と不連続データ

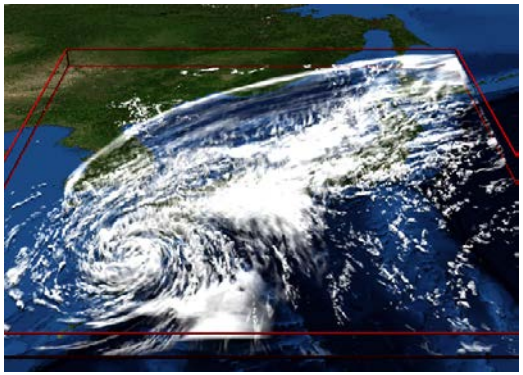
- 多次元分割で通信量合計を削減できるが、

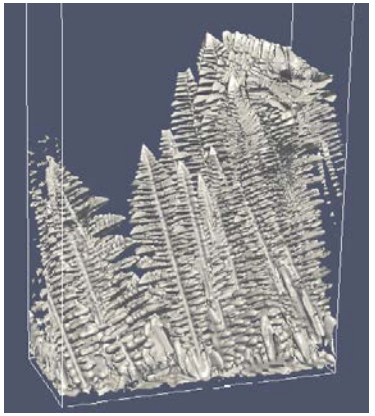
不連続なデータ領域を通信する必要がおこる



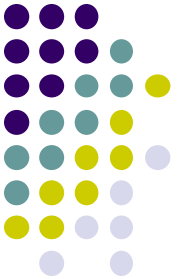
Row-majorならびの場合、
左右プロセスと通信する
領域は不連続となってしまう

※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

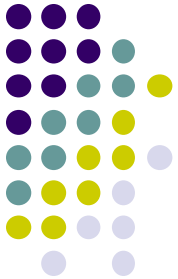
⇒　性能が大きく低下！メッセージ数が莫大になり、レイテンシを何度も食らうので

- 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)

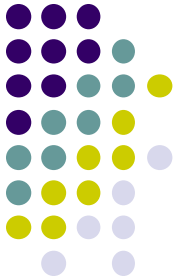
派生データ型(1)



- 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

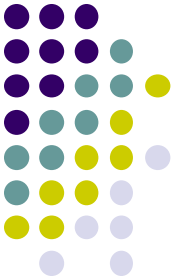
blocklen: 一ブロックの要素数

stride: ブロック間の幅

oldtype: 元となるデータ型

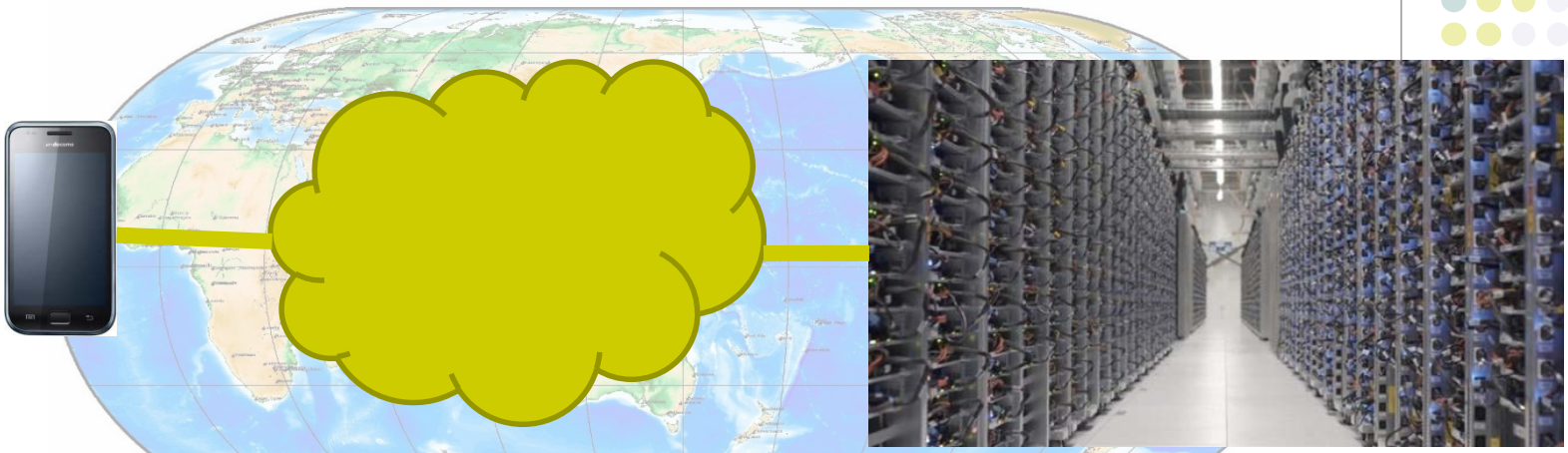
newtype: 新たにつくられる派生データ型

多次元分割はいつも有利なわけではない

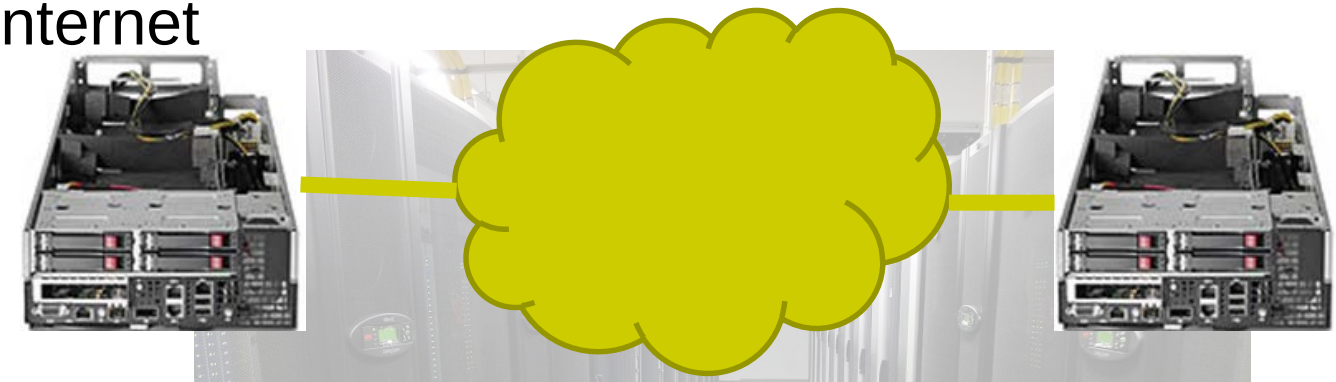


- 派生データ型を使ってさえ、連続領域よりも不連続領域の通信オーバヘッドは大きくなる
  - 結局MPI関数内部でキャッシュミスやメモリコピーのオーバヘッド
- 一般的には、プロセス数が大きくなるほど有利
- 三次元領域の計算で、あえて二次元分割にした方が有利なケースも

Computer Network

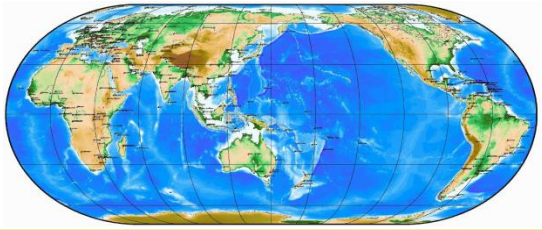


Your smart phone is connected with google.com via Internet



1400 computers in TSUBAME are connected with each other via fast network
(TSUBAME's network is also a part of Internet)

Characteristics of Networks



	World Wide	System Wide (Case of TSUBAME)
Physical Diameter	>10,000 km	~100 m
Peer-to-peer Bandwidth (Larger is better)	<100 Mbps Sometimes <1Mbps	80 Gbps
Latency (Smaller is better)	Sometimes >100ms	<10us

bps = bit per second

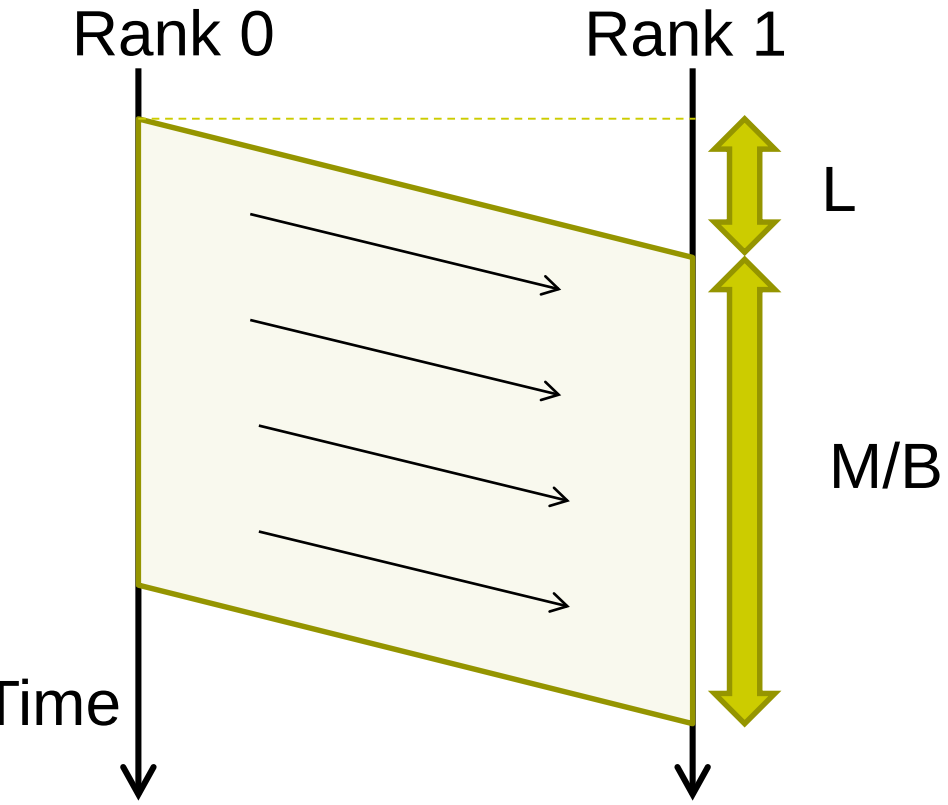
転送時間の性質 (1)

- 転送時間は一般に、
  - メッセージサイズが大きいほど長い
  - メッセージ数が多いほど長い
    - 10KB×1回の転送時間 < 1KB×10回の転送時間
- 転送時間の単純なモデル:

$$T = M / B + L$$

- T: 転送時間 (sec)
- M: メッセージサイズ (Byte)
- B: バンド幅 (Byte/sec)
- L: レイテンシ (狭義の) (sec)

転送時間の性質 (2)



T = M / B + L

L: (狭義の)レイテンシ

通信の最小単位が、送信者から受信者に届く時間

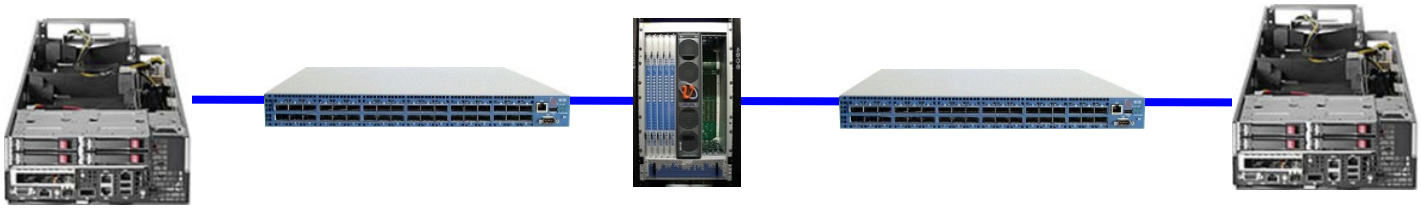
B: バンド幅

通信路が一秒間に通すことのできるデータ量

※Byte, bitの差に注意。1Byte=8bit
「ギガビットイーサネット」は1Gbps = 125MB/s
※Tを通信レイテンシと呼ぶこともある

Why L (Latency) > 0?

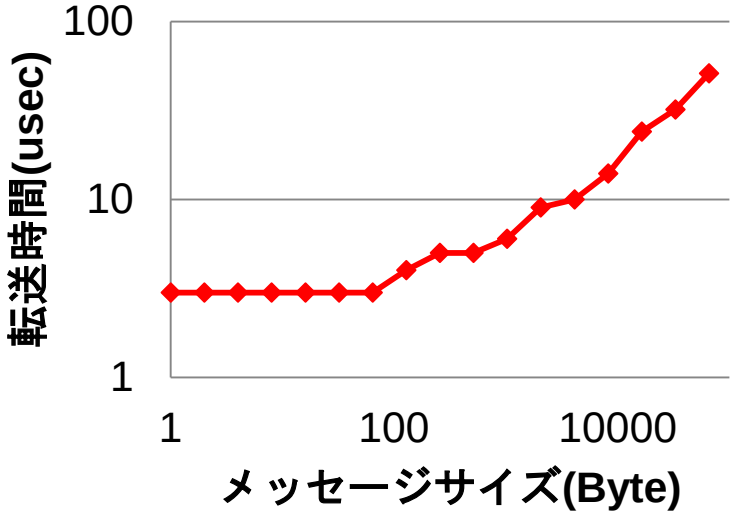
- Transmission speed of electrical signal cannot exceed speed of light = $3x10^8$ m/s
  - Travelling 10,000km takes 33ms at least
- There are costs at network switches

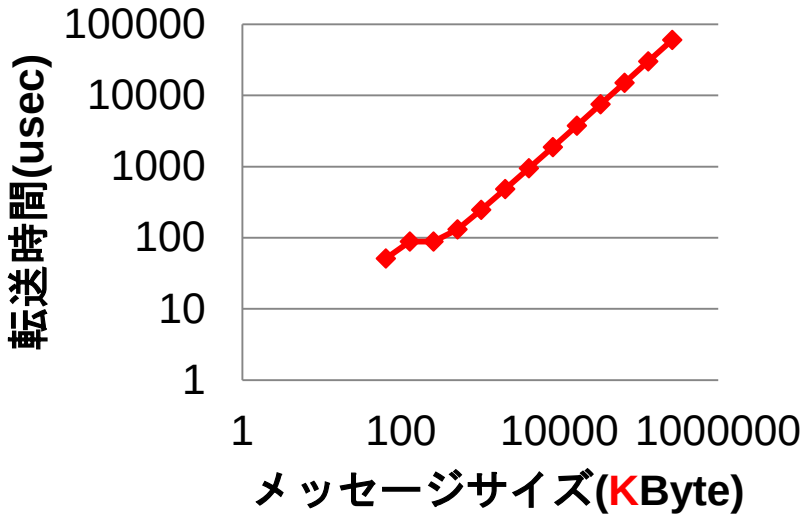


- There are software costs at sender/receiver
  - Cf) Socket library, MPI library performs data copy

TSUBAMEの通信性能(1)

- TSUBAME2の2ノード、OpenMPI 1.6.3で測定





- 切片がL、傾きの逆数がBと考えると

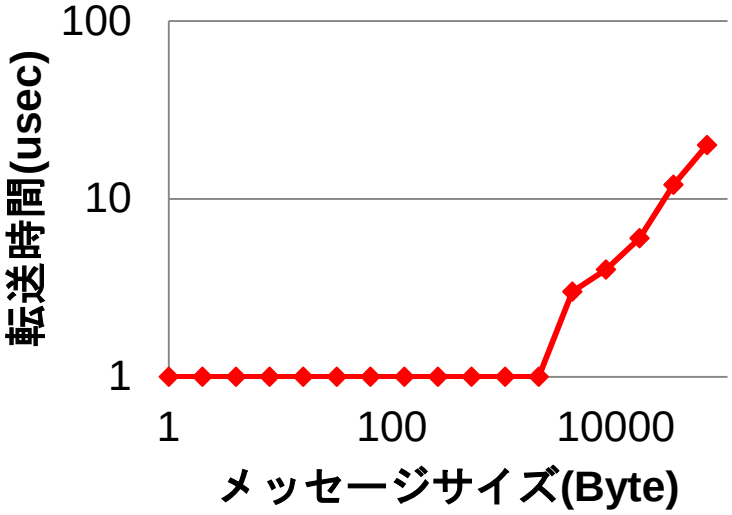
  L ≒ 3(us)、B ≒ 4.4 (GB/s)

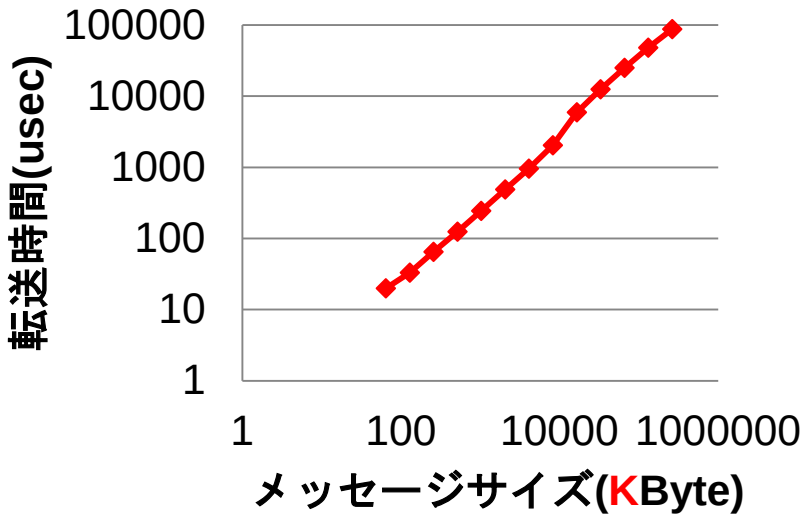
  理論値が4GB/s x 2本 = 8GBなので、55%くらい

- 各ノードに複数プロセスいると、混雑でさらに遅くなることに注意

TSUBAMEの通信性能(2)

- TSUBAME2の1ノード内2プロセスで測定



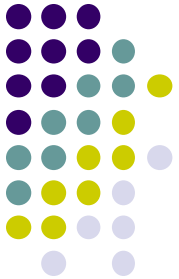


- 切片がL、傾きの逆数がBと考えると

  L ≒ 1(us)、B ≒ 3.2 (GB/s)

  Lは前ページより良好、Bはやや悪化という結果に

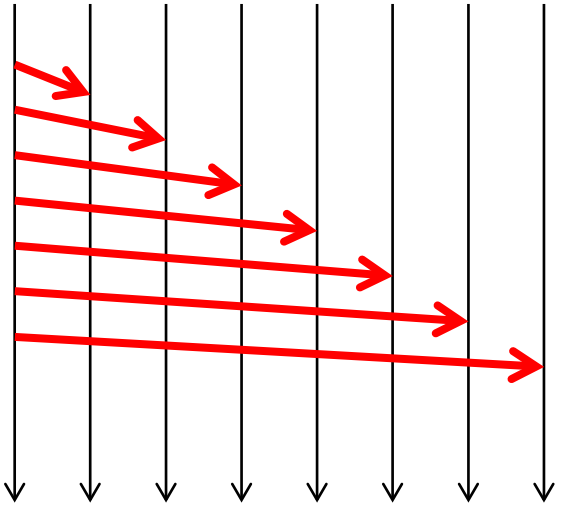
集団通信: 使えるべきところでは使うべき



- 自分でMPI_Send/Recvを使うより速い場合が多い
- Broadcastの様々なアルゴリズム

→ MPI処理系が、ふさわしいアルゴリズムを選んでいる

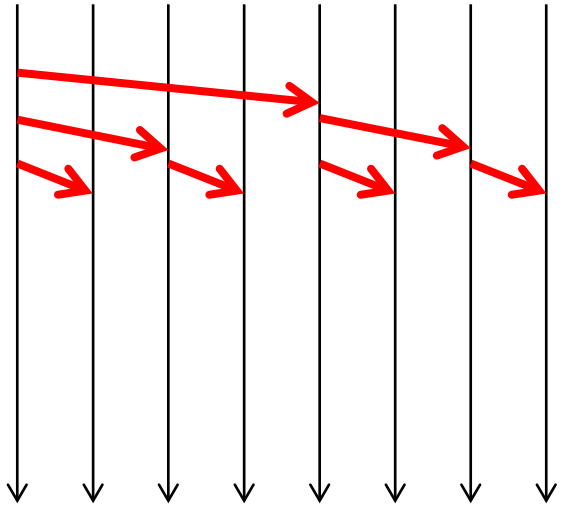
Flat tree algorithm



(p-1)(M/B+L)

→ Slowest

Binomial tree algorithm

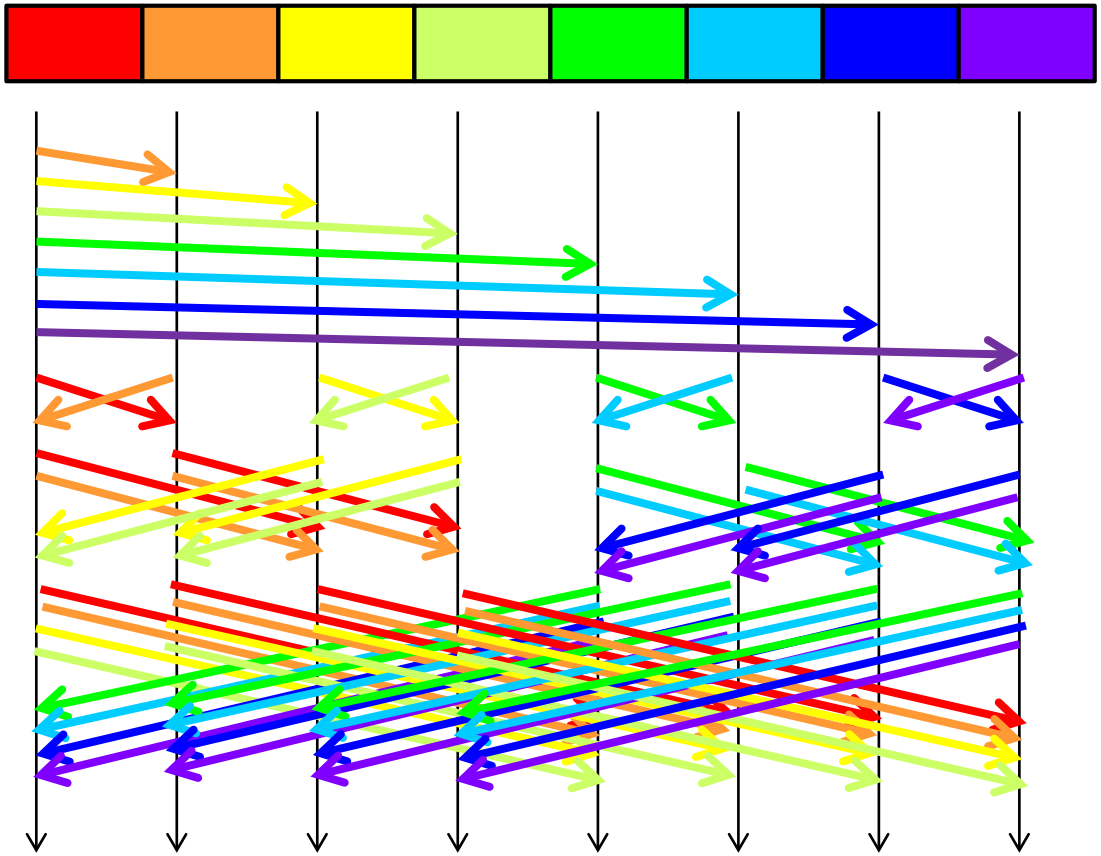


(log p)(M/B+L)

効率的なbcastアルゴリズム

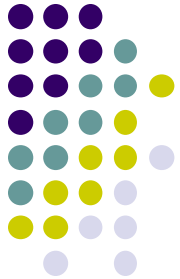
- Scatter&Allgather
  - ある程度の長いメッセージ向け
  - メッセージをプロセス数に分割して考える

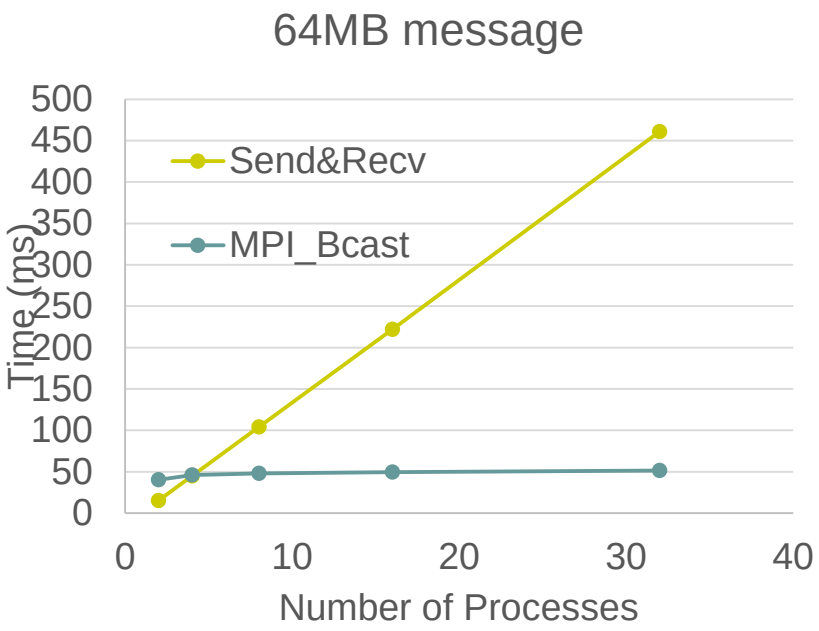
R. Thakur and W. Gropp. Improving the performance of collective operations in mpich. EuroPVM/MPI conference, 2003.

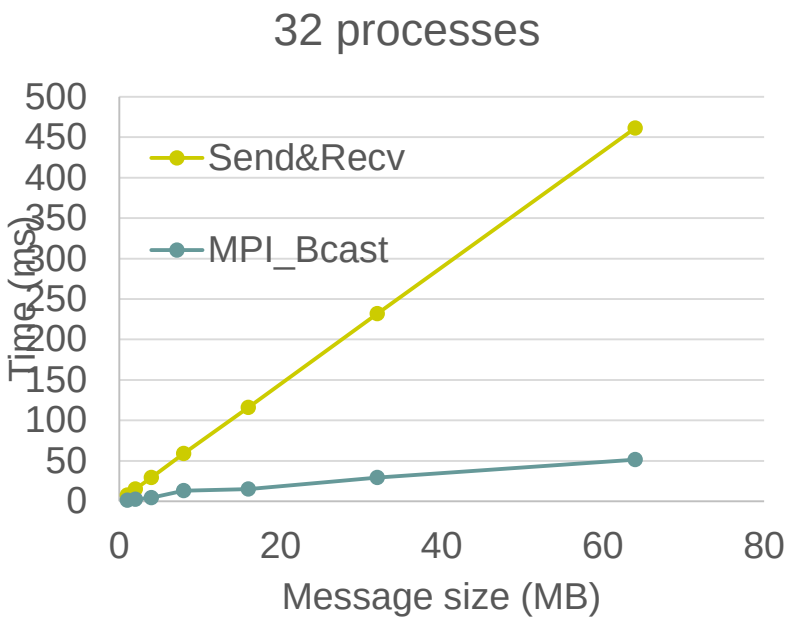


pL + M/B +
(log p)L + M/B

MPI_Bcastの性能



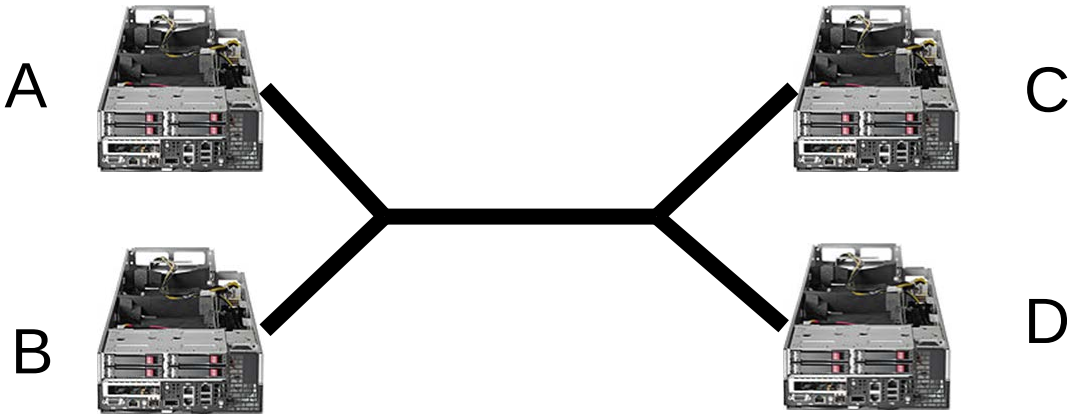




- プロセス数が多いほど、MPI_Bcastが得
- プロセス数にほぼ関わらない性能！

大規模実行で性能を遅くする要素

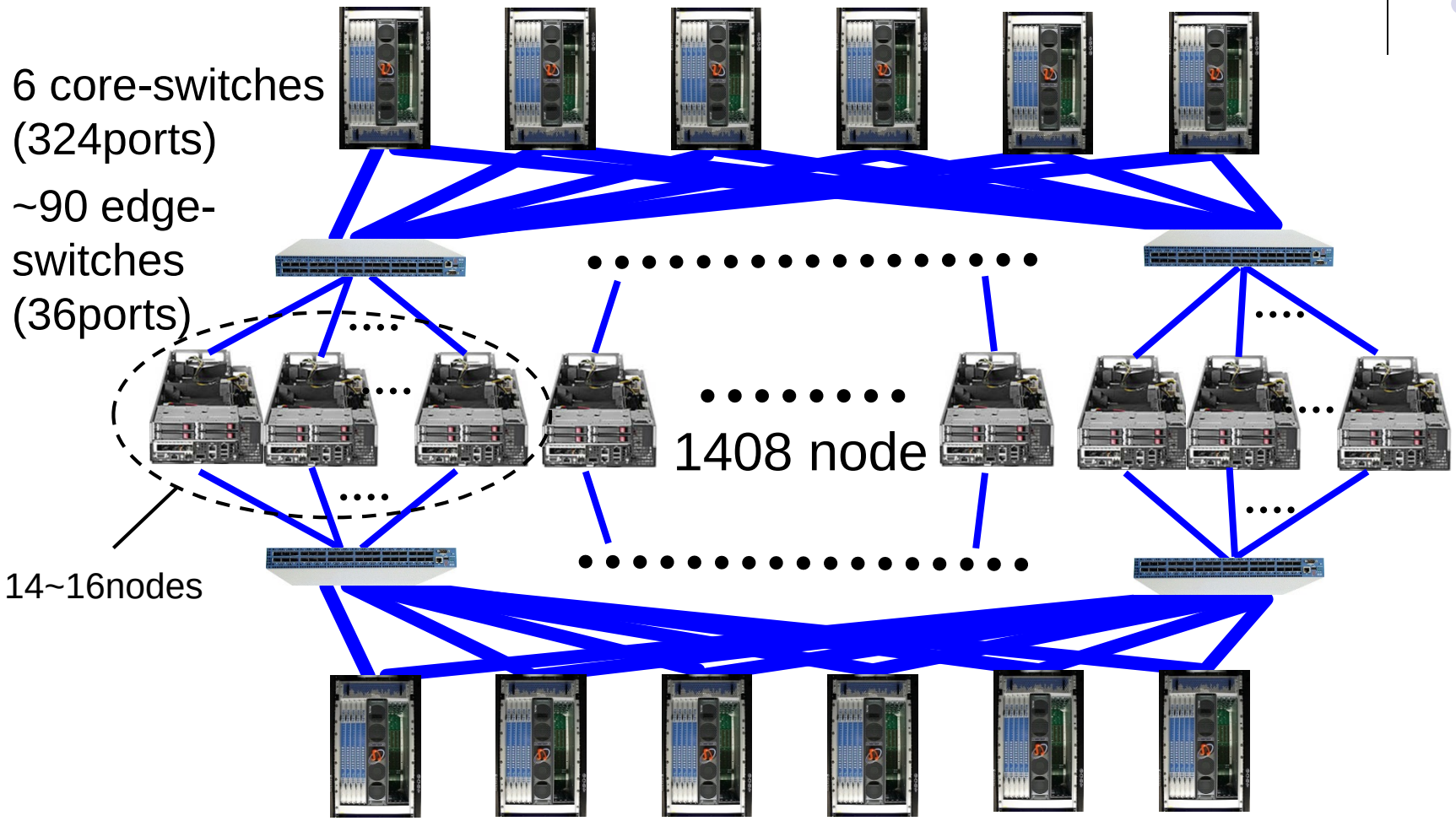
- 通信路の混雑 (congestion)



- どのリンクも1GB/sと仮定する。
- A-C間、B-D間も混雑がなければ1GB/sで通信可能
- しかし、A-C, B-D間の同時通信が起こると理論的には半分の0.5GB/sずつとなってしまう

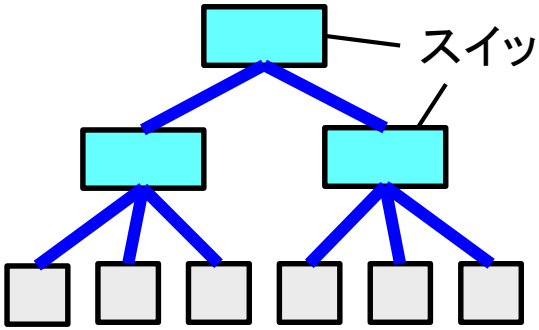
TSUBAME2のネットワーク構成

Dual-rail Full-bisection Fat-tree topology

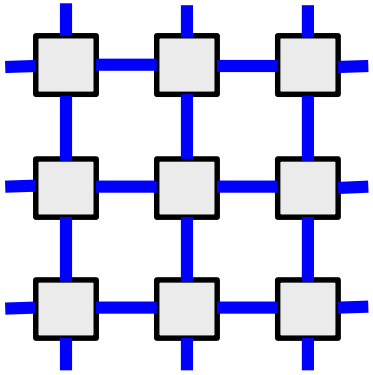


様々なネットワークトポロジー

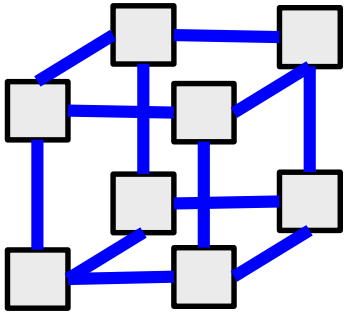
ツリー構造



メッシュ/トーラス構造

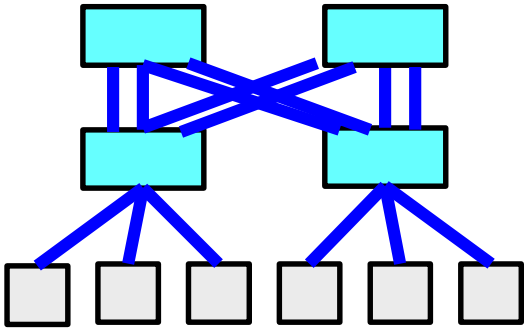


ハイパーキューブ構造



京、BlueGeneなど

ファットツリー構造

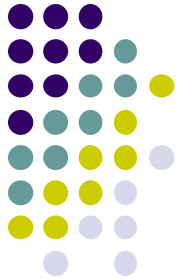


TSUBAME2など

- トーラス：ノード数増加時のパーツ数が少ない
- TSUBAME2ではそれほど混雑が顕在化することはない

⇒それでも約500ノード以上になると問題が見えてくる

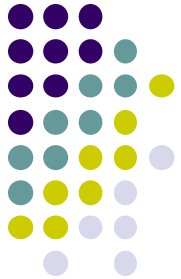
OpenMP/MPI編を終えて：なぜ並列プログラムは遅くなるのか？(1)



下記のように様々な性能低下原因が考えられる!!

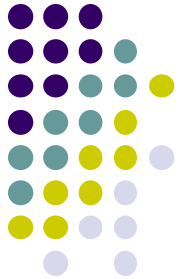
- アルゴリズム上の問題
  - 排他制御などによるボトルネック
  - スレッド間・プロセス間で計算量が不均衡
  - 通信量のオーダーを忘れがち
    - 計算量オーダーで通信量オーダーが同じなら通信量が多すぎて遅い傾向
- システム内での混雑
  - CPUコアにスレッドが集中
  - メモリへのアクセスの集中
  - ノード間ネットワークの混雑
  - ストレージへのアクセスの集中・経路でのネットワーク混雑…
    - TSUBAMEではMPIもストレージ(/home, /work)も同じInfiniBandを利用
    - 他プログラムの影響もありうる

OpenMP/MPI編を終えて:なぜ並列プログラムは遅くなるのか？(2)



- 性能向上のためには、今、なぜ性能が低下しているかの原因を特定する必要
  - 原因特定だけでも決して容易ではない
  - 別のスパコンに移ると性能が全く変わる可能性も…
- 問題を切り分けるにはどのような実験をすればよいか考える必要がある
  - 通信時間のみの測定により、原因がノード内なのかノード間通信なのか…
    - 1ノードと複数ノードの比較も使える場合も
  - ノード内であれば、(明示的な)ボトルネックがあるか否か、メモリアクセス量はどうか…
- その議論の過程で、スパコンの構造の知識も必要に

本授業のレポートについて

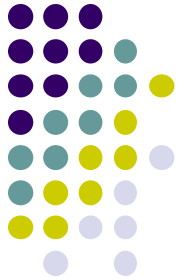


- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

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)



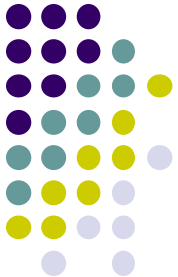
以下の[M1]—[M3]のどれか一つについてレポートを提出してください

[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を減らせないか？

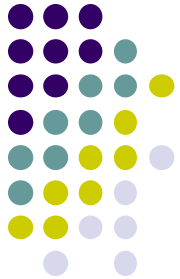
MPIパート課題説明/Report (2)



[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

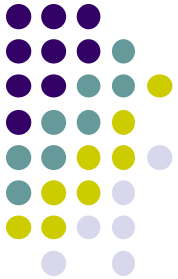
MPIパート課題説明/Report (3)



[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
  - 7/13（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

- 7/6(月)
  - GPUプログラミング (1)

- 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