最適化2
「並列プログラムの通信最適化」

建部修見
（筑波大学）

講義内容

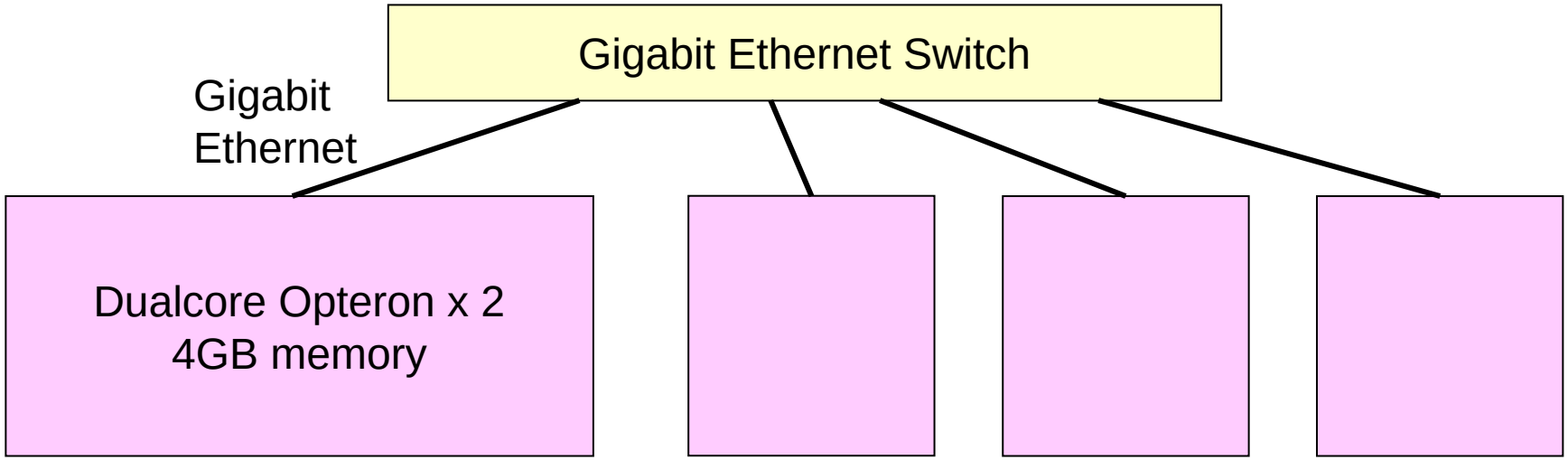
- 基本通信性能
  - 1対1通信
  - 集団通信
- プロファイラ
- 通信最適化
  - 通信の削減
  - 通信遅延隠蔽
  - 通信ブロック
  - 負荷分散

基本通信性能

- 通信最適化のためには基本通信性能を押さえておくことが重要！
  - 各種通信パターンにおける通信性能の把握
  - 通信ブロッキングのブロックサイズの決定
  - ネットワーク性能と比較して、通信ライブラリ自体の性能改善

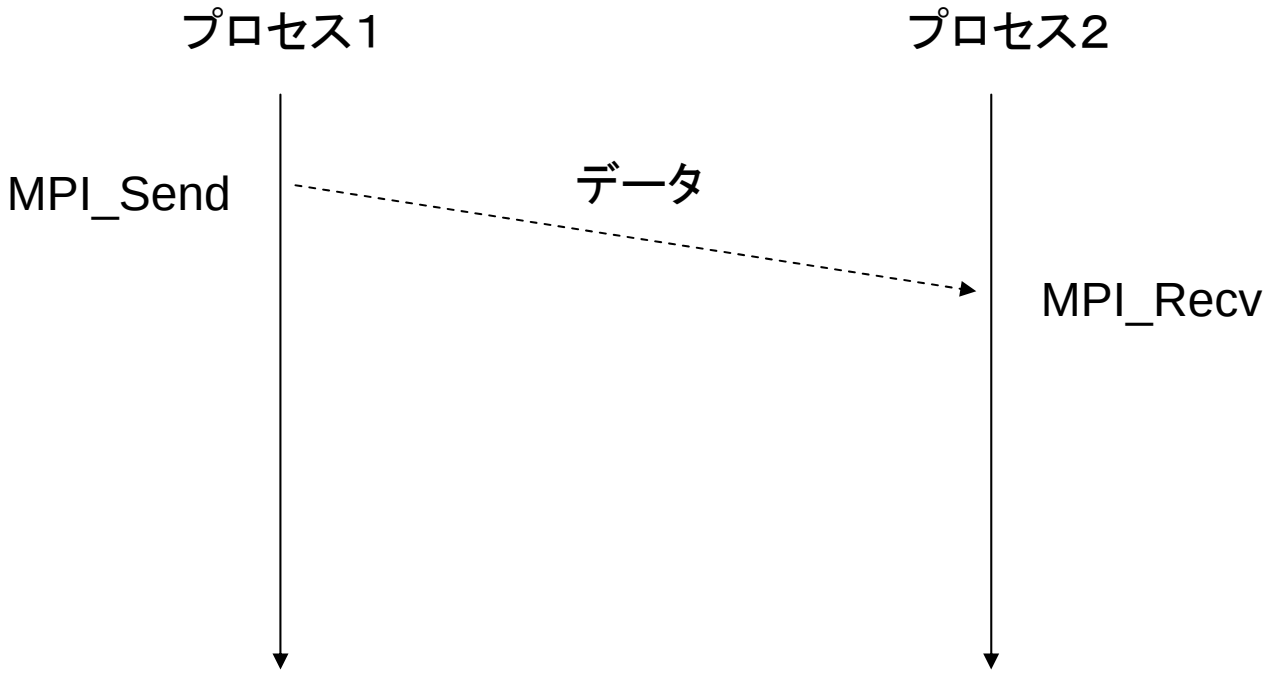
基本通信性能評価環境

- **4クラスタノード**
  - 2.6GHz Dualcore Opteron x 2 sockets (4 cores)
  - 4GB memory
  - Linux 2.6.18-1.2798.fc6
  - OpenMPI 1.1-7.fc6
- **Gigabit Ethernetで接続**
  - TCPでの理論ピーク性能は949Mbps(=113.1MB/sec)

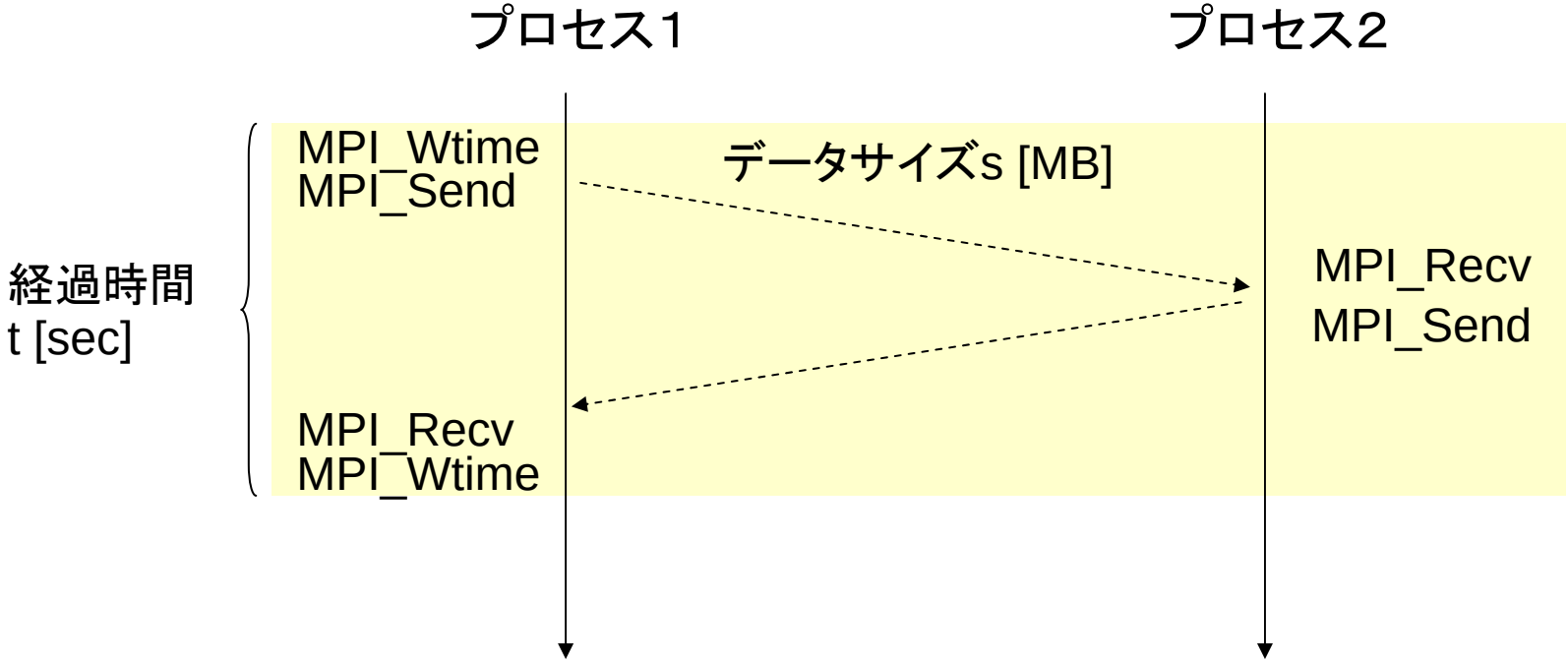


1対1通信の性能

- **1対1通信はMPIにおける基本通信**



PingPongベンチマーク

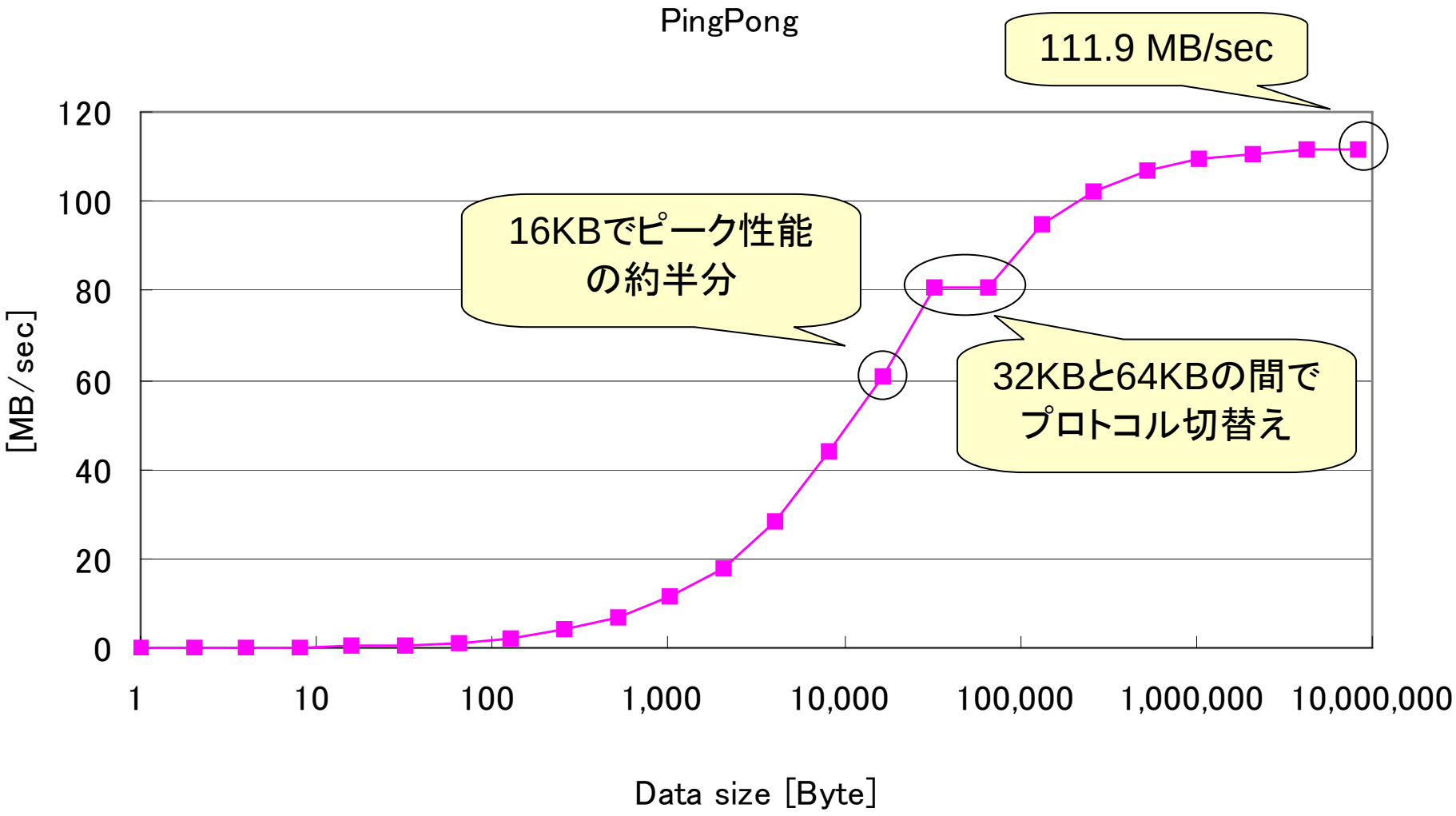


通信バンド幅 $s/t/2$ [MByte/sec]

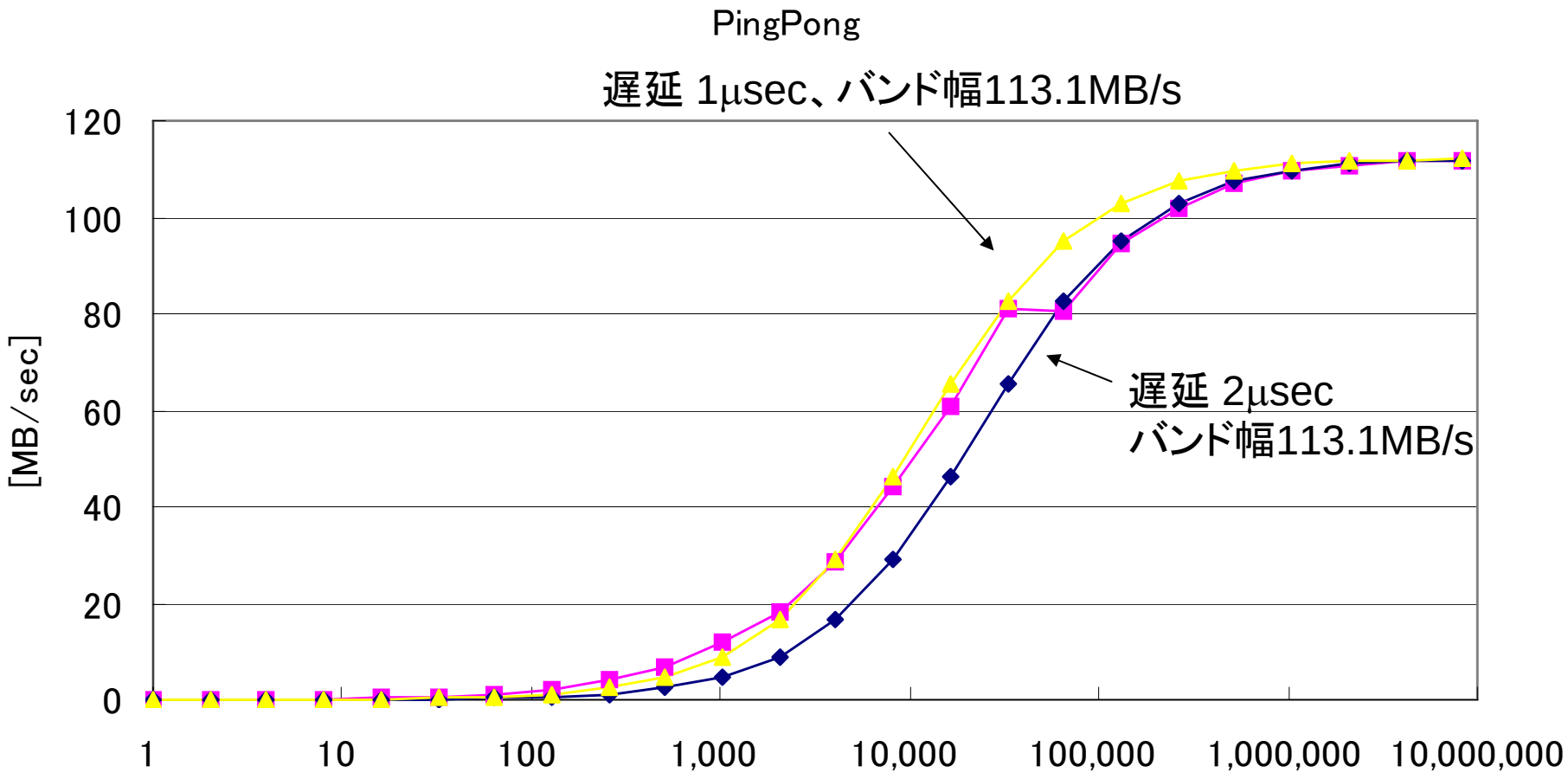
PingPongベンチマークの例

```
for (s = 1; s < MAX_MSGSIZE; s <<= 1) {
    t = MPI_Wtime();
    for (i = 0; i < ITER; ++i)
        if (rank == 0) {
            MPI_Send(BUF, s, MPI_BYTE, 1, TAG1, COMM);
            MPI_Recv(BUF, s, MPI_BYTE, 1, TAG2, COMM, &status);
        } else if (rank == 1) {
            MPI_Recv(BUF, s, MPI_BYTE, 0, TAG1, COMM, &status);
            MPI_Send(BUF, s, MPI_BYTE, 0, TAG2, COMM);
        }
    t = (MPI_Wtime() – t) / 2 / ITER;
    if (rank == 0)
        printf("%d %g %g¥n", s, t, s / t); // サイズ、時間、バンド幅
}
```

PingPongベンチマークの結果



遅延、バンド幅での曲線との比較



理論曲線 $s/(L+s/B)$ Data size [Byte]

L 遅延時間 B バンド幅

PingPongベンチマークの考察

- データサイズは大きい方が高性能
- 参考：理論ピーク性能は113.1MB/sec
- ピークの半分以上→データサイズ16KB以上
- ピークの9割以上→データサイズ512KB以上

- 0バイトのPingPongベンチマークの遅延時間は52μsecであったが、ショートメッセージは1μsec、ロングメッセージは2μsecの遅延時間の曲線に従う

Intel® MPI Benchmark

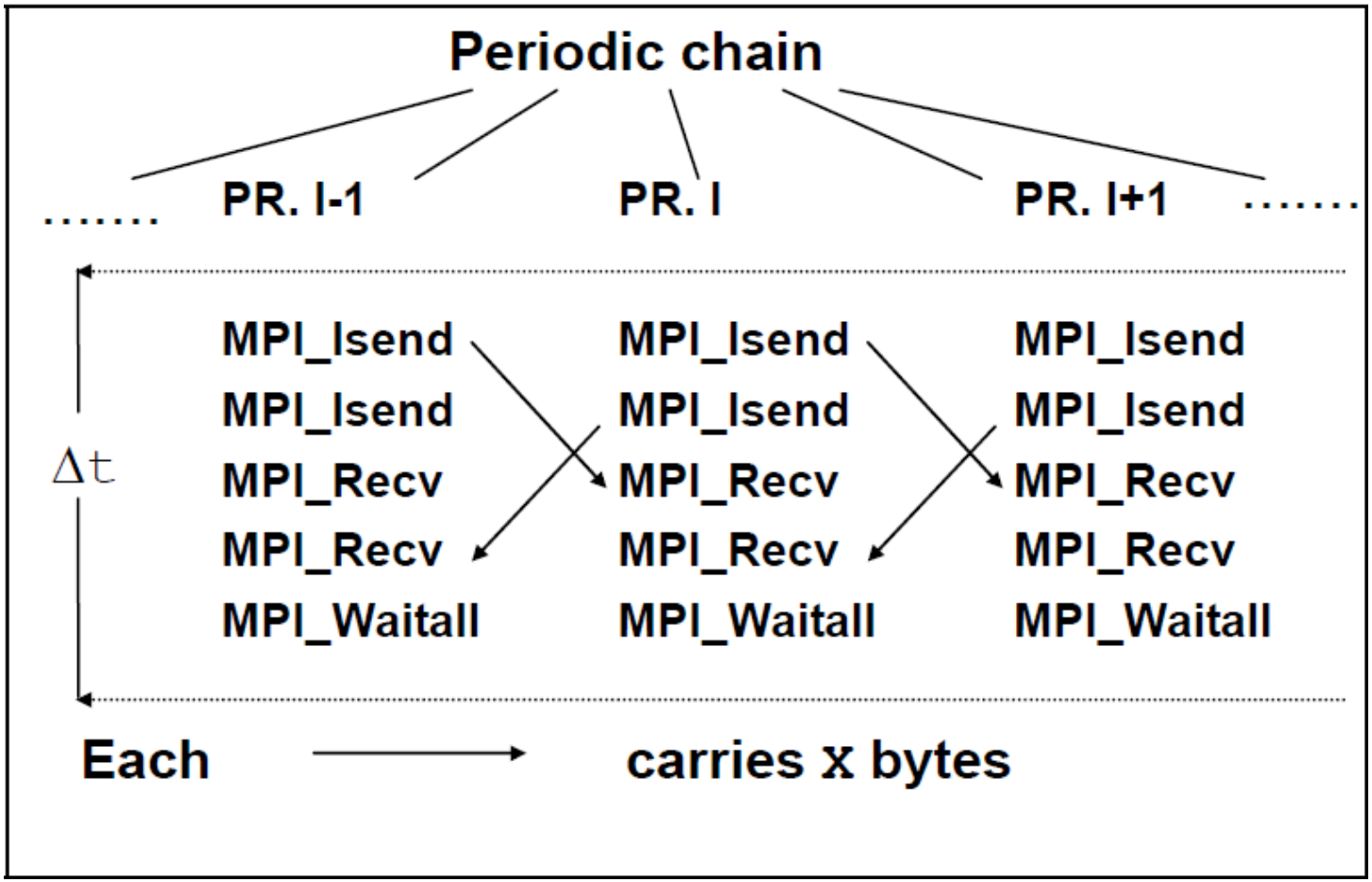
- 基本MPIベンチマークカーネル
- MPI1

PingPong PingPing	**Single Transfer**
Sendrecv Exchange*	**Parallel Transfer**
Bcast Allgather Allgatherv Alltoall* Alltoallv* Reduce Reduce_scatter Allreduce* Barrier	**Collective**

  - 上記を複数一斉に行うMulti版
- EXT
  - Window
  - Unidir_Put
  - Unidir_Get
  - Bidir_Get
  - Bidir_Put
  - Accumulate
- IO
  - S_{Write,Read}_{indv,expl}
  - P_{Write,Read}_{indv,expl,shared,priv}
  - C_{Write,Read}_{indv,expl,shared}

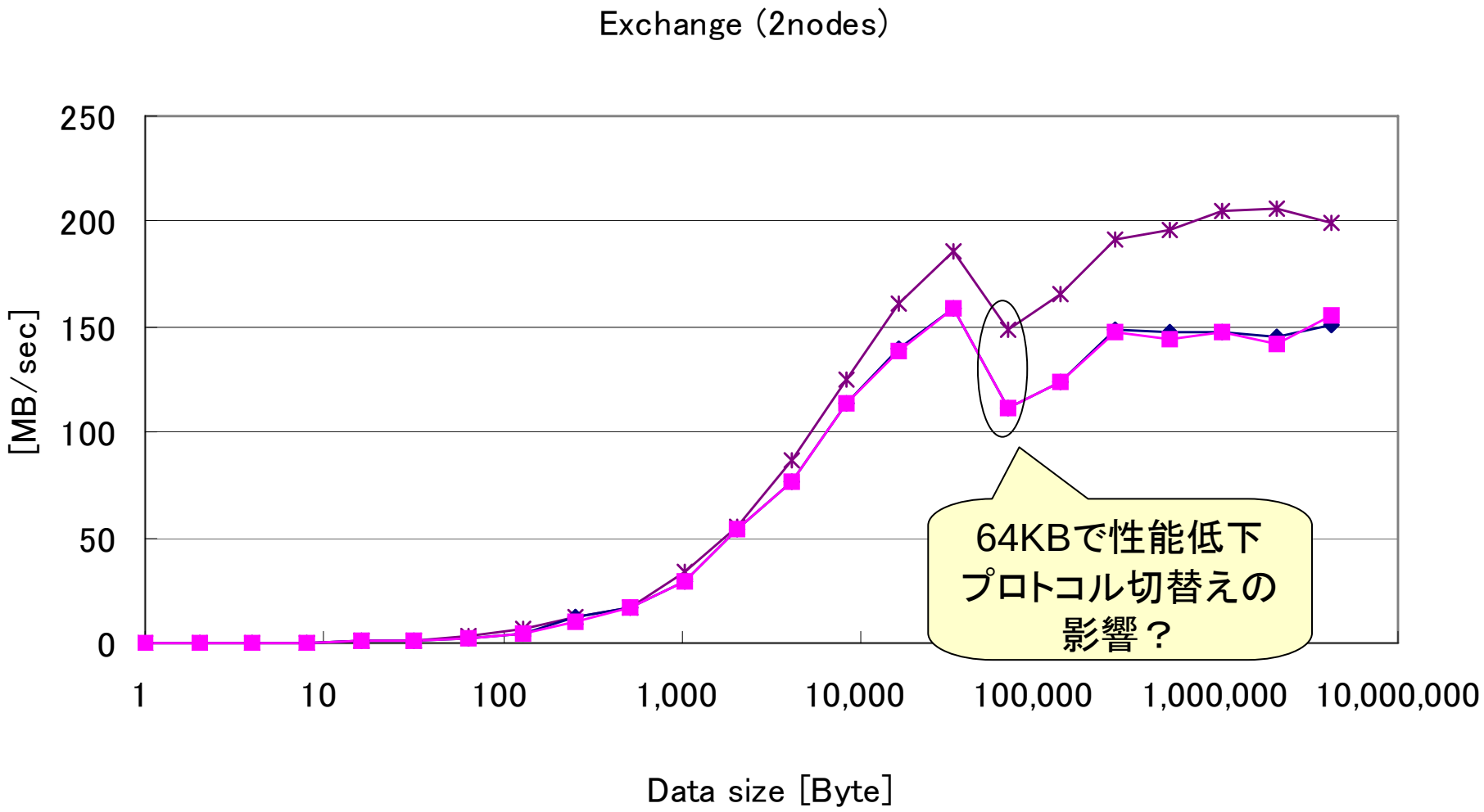
Exchangeパターン

- 境界の要素を交換する通信パターン



*Intel MPI Benchmarks Users Guide and Methodology Descriptionより

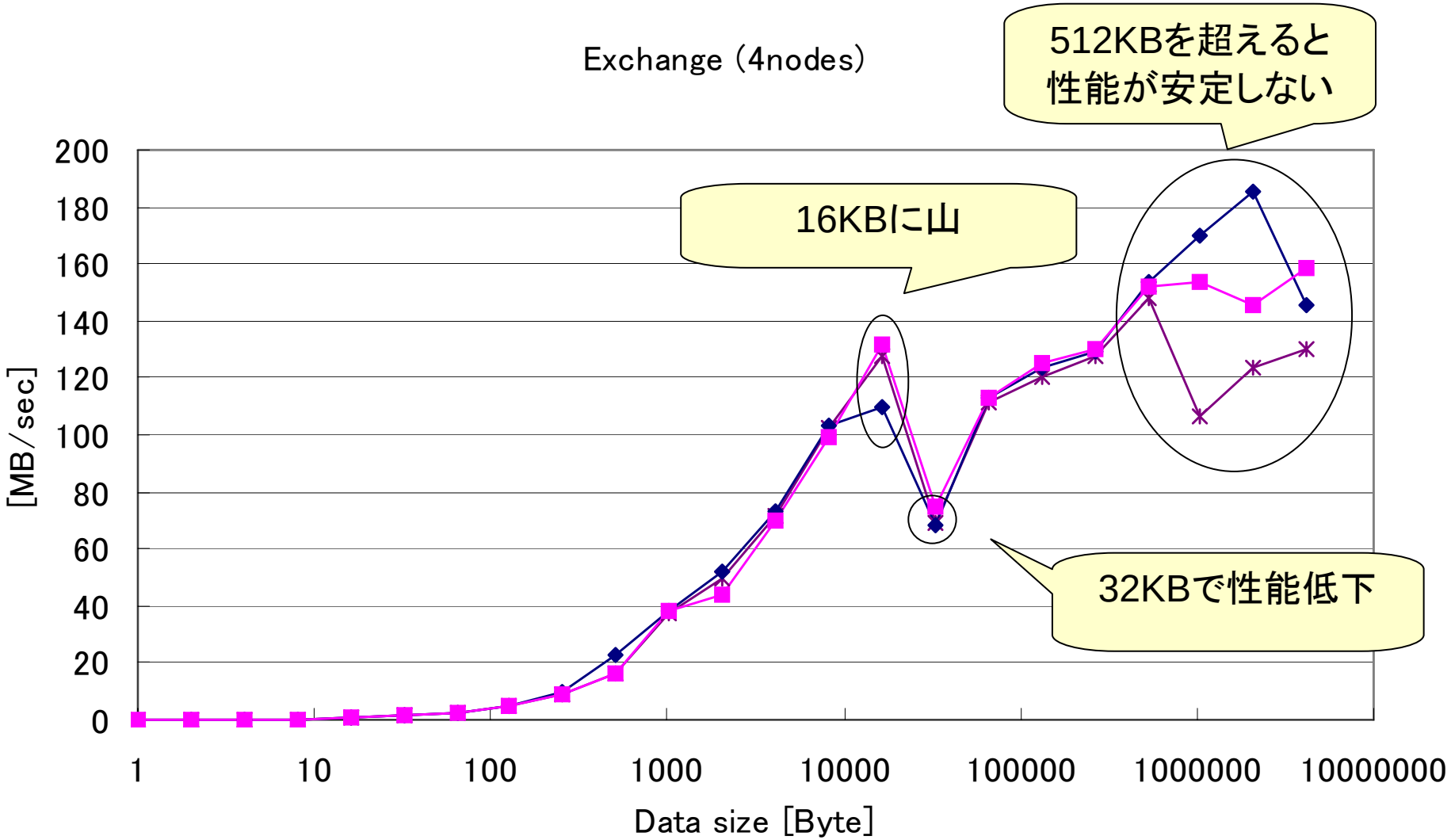
Exchange（2ノード）の結果 [試行3回]



Exchange（2ノード）の考察

- データサイズは基本的には大きい方が高性能だが、64KB付近で落ちる
- 参考：理論ピーク性能は2*113.1=226.2MB/sec
- ピークの半分以上→データサイズ8KB以上
- ただし8KBを超えると性能は安定しない
- データサイズ256KB以上ではピークの8割以上でることもある

Exchange（4ノード）の結果 [試行3回]



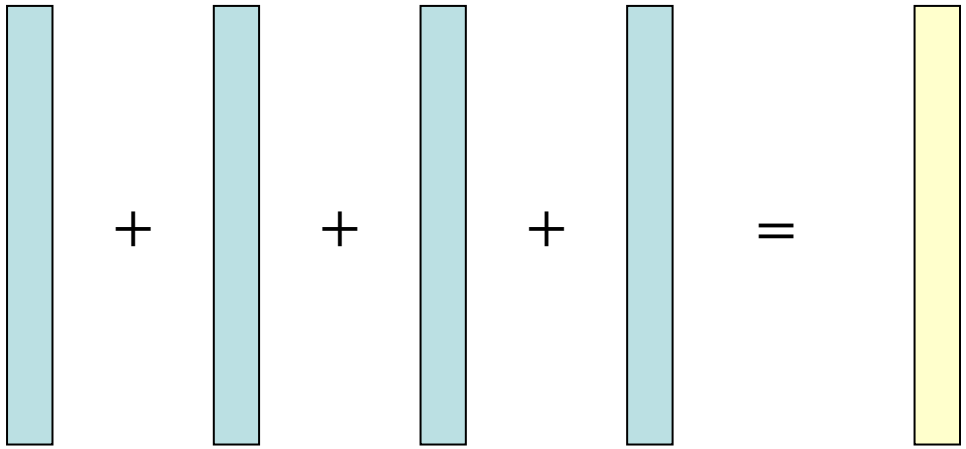
Exchange（4ノード）の考察

- データサイズは基本的には大きい方が高性能だが、32KB付近で落ちる
- 参考：理論ピーク性能は2*113.1=226.2MB/sec
- ピークの半分以上→データサイズ16KBと128KB以上
  - 32KB、64KBはピークの半分以下
- 512KB超では性能が安定しない

Allreduce

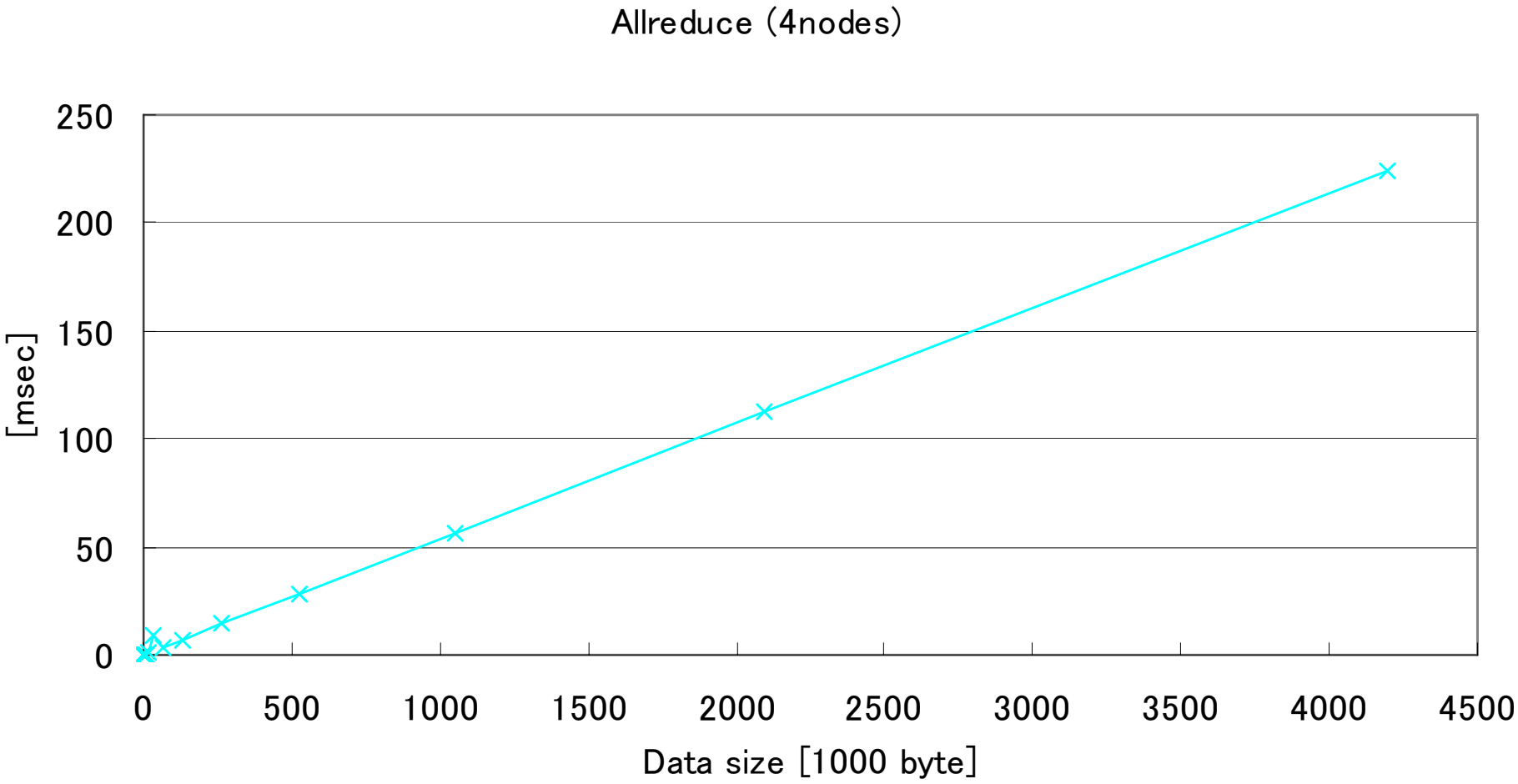
- 集合通信
- 各プロセスの配列間で指定された演算（加算、AND/OR演算など）を施し、結果は全プロセスで保持
- MPI_SUMの例

$$x_1 + x_2 + x_3 + x_4 = \sum_{i=1}^{4} x_i$$



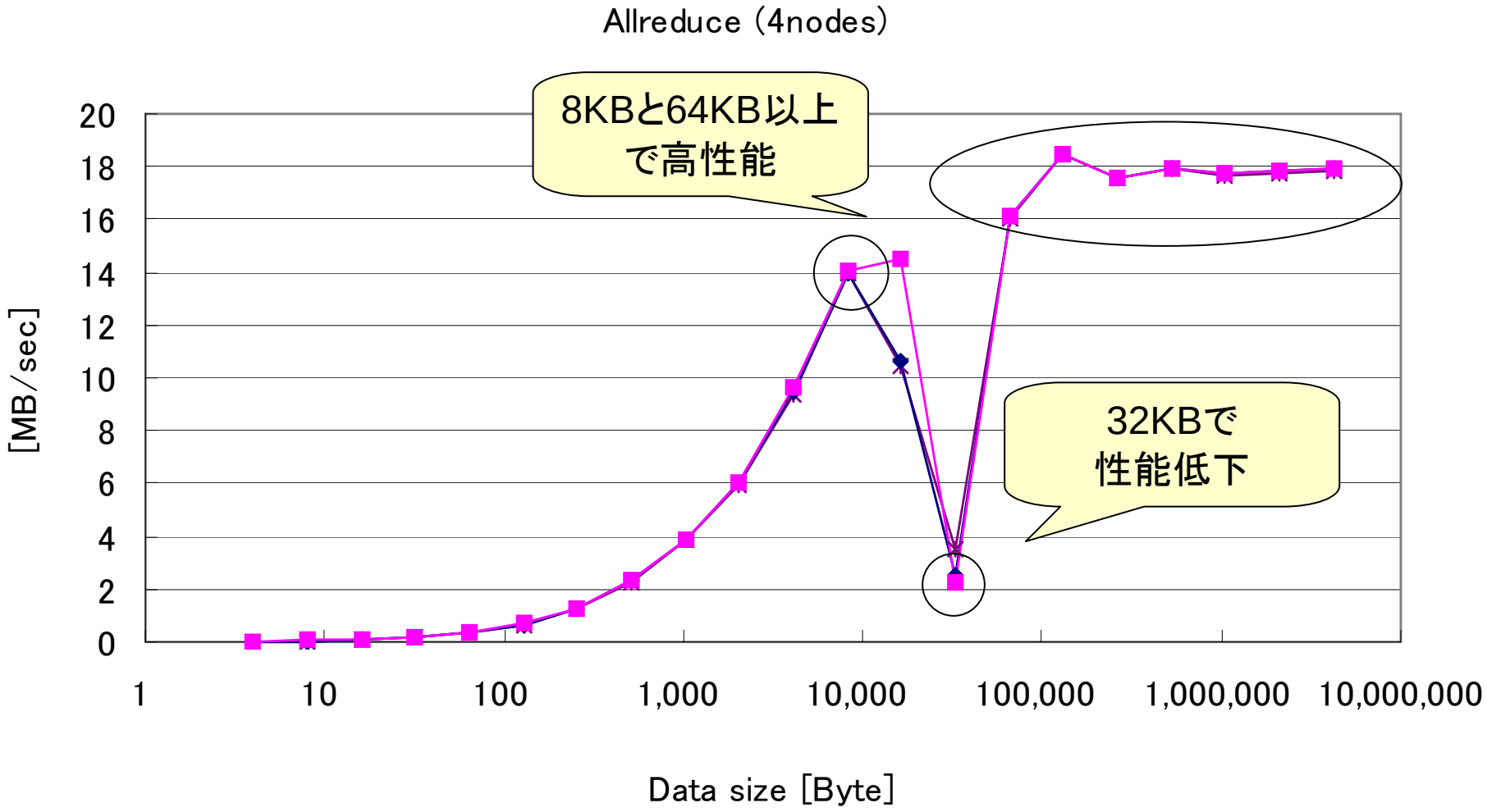
プロセス1の配列　プロセス2の配列　プロセス3の配列　プロセス4の配列　計算結果を全プロセスで保持

Allreduce（4ノード）[時間、リニアスケール]



一見、データサイズに対し比例した時間がかかっているようだが、、、

Allreduce（4ノード）[サイズ/時間、ログスケール]

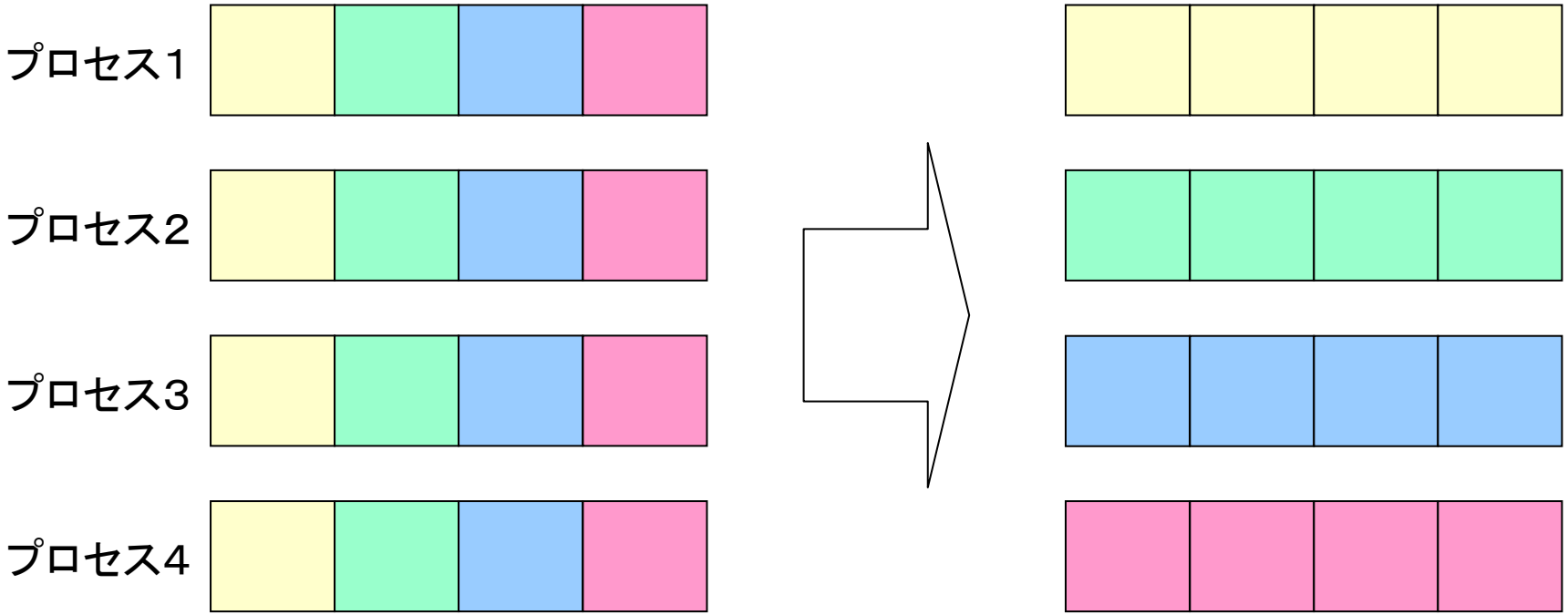


Allreduce（4ノード）の考察

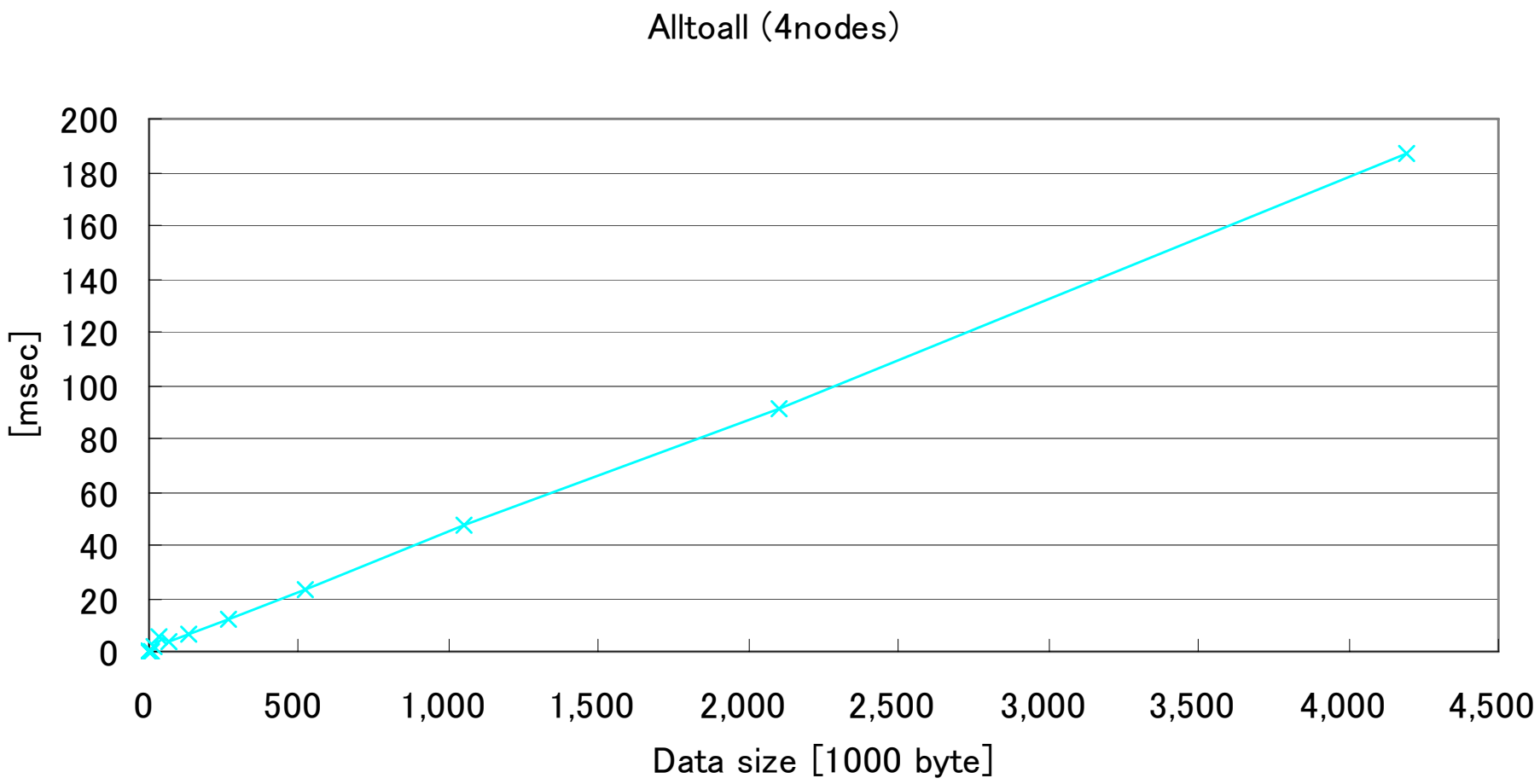
- データサイズは基本的には大きい方が高性能だが、32KBで性能低下
- 8KB、64KB以上では高性能

Alltoall

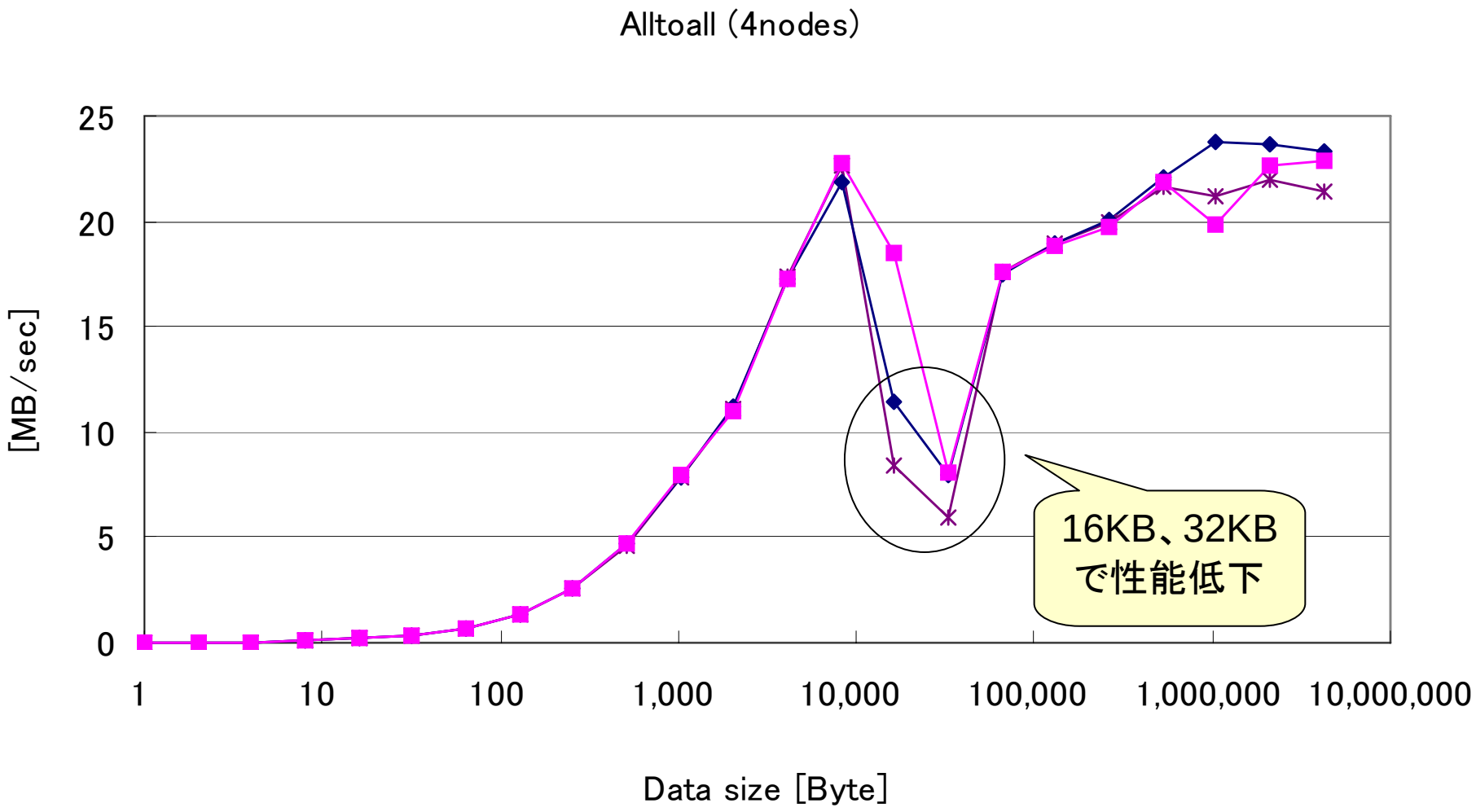
- 行列の転置に相当する集団通信



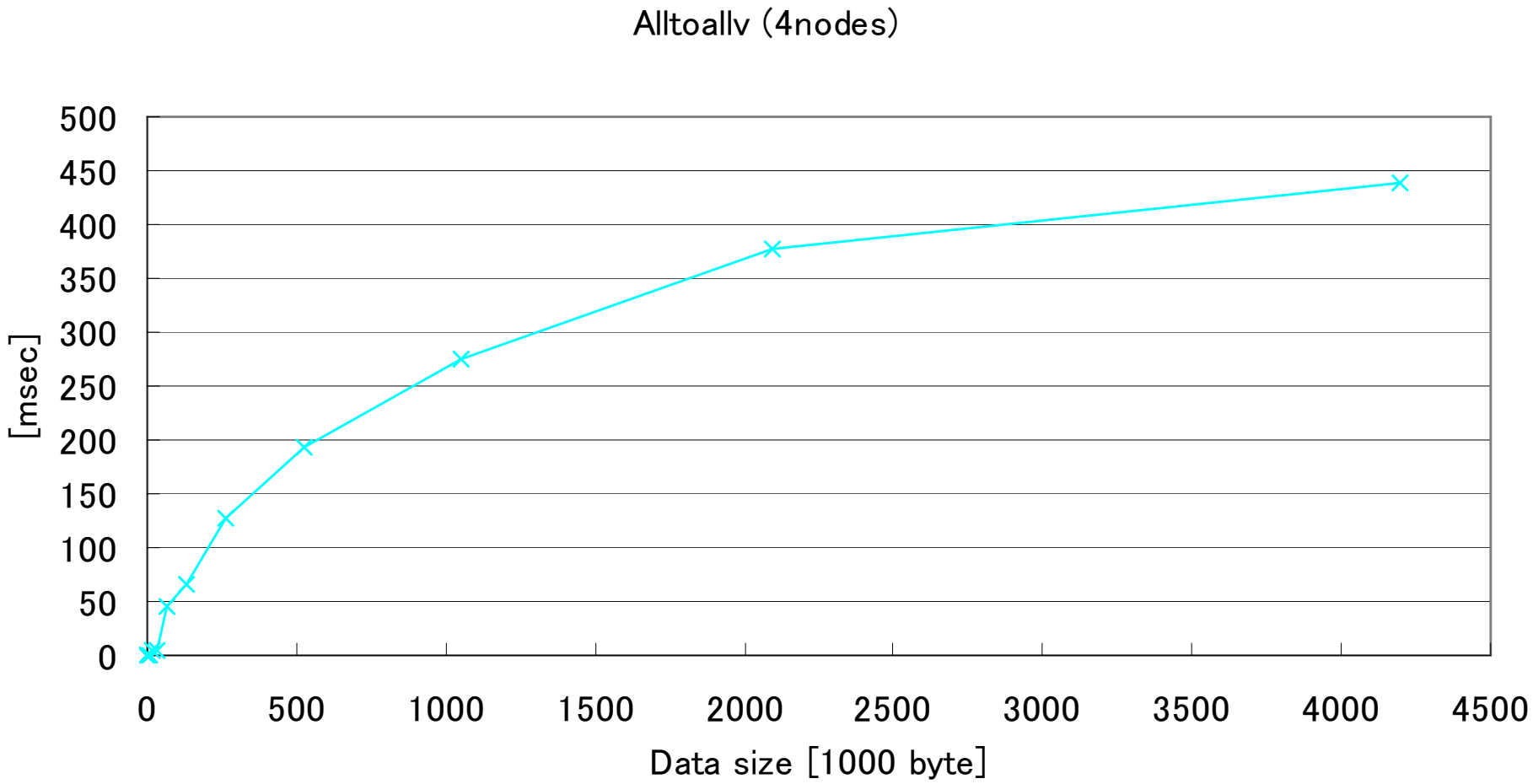
Alltoall [時間、リニアスケール]



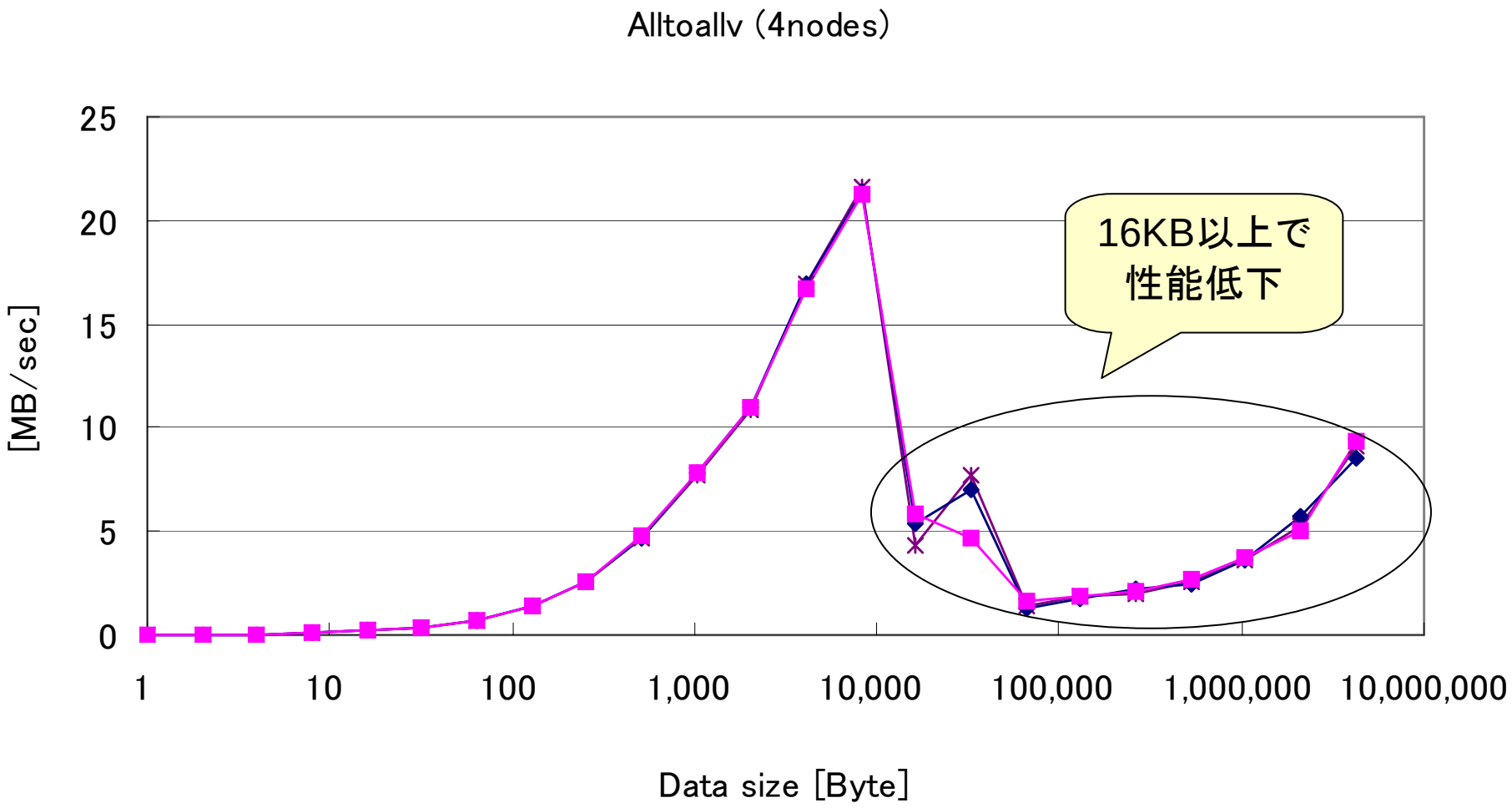
Alltoall [サイズ/時間、ログスケール]



Alltoallv [時間、リニアスケール]



Alltoallv [サイズ/時間、ログスケール]



Alltoall(v)の考察

- Alltoallは、データサイズは基本的には大きい方が高性能だが、16KB～32KBで性能低下
  - 8KB、64KB以上では高性能
  - Allreduceと同様
- Alltoallvは、8KBを超えると極端に性能低下
  - 不必要なメモリコピーのためと思われる
  - 性能最適化がなされていない

プロファイラ

- プログラムの挙動を把握する
  - 呼び出し回数の多い関数
  - 処理に時間がかかっている関数
  - 関数の呼び出し関係
  - 関数のメモリ使用量など
- 実行時間の多くが費やされているコードの特定
- 並列プログラムにおける同期待ち、負荷不均衡の把握
  - プログラムの実行に影響しないことが望ましい
  - 軽量のプロファイラが必須

tlog – time log

- **実行プロファイルをとるための軽量ライブラリ（佐藤センター長@筑波大）**
  - 1イベントあたり16バイト
  - 各プロセスのメモリに保持
- **単発イベント、区間イベント各9種類のログ**
  - イベント番号は8ビットなので拡張可能
- **tlog_initializeからの経過時間（秒）を記録**
  - tlog_initializeでノード間の時刻差を測定し補正
  - 並列プロセスにおける「絶対」相対時間
- **暫定ダウンロードURL**
  - http://www.ccs.tsukuba.ac.jp/workshop/HPCseminar/2007/software/tlog.tar.gz

tlog - 主要API

void tlog_initialize(void)

初期化。**MPI_Initの後で呼ぶこと**

void tlog_log(int event)

eventで指定されたイベントを記録する

void tlog_finalize(void)

ログを**trace.log**に出力。**MPI_Finalize()の前に呼ぶこと**

```
tlog_initialize();
...
tlog_log(TLOG_EVENT_1_IN);
/* EVENT 1 */
tlog_log(TLOG_EVENT_1_OUT);
...
tlog_finalize();
```

例 - cpi.c

- πを計算するテストプログラム

```
MPI_Init(&argc, &argv);
tlog_initialize();
tlog_log(TLOG_EVENT_1_IN);
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
tlog_log(TLOG_EVENT_1_OUT);
/* mypiの部分計算 */
tlog_log(TLOG_EVENT_2_IN);
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
tlog_log(TLOG_EVENT_2_OUT);
if (rank == 0) /* 結果表示 */
tlog_log(TLOG_EVENT_1_IN);
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
tlog_log(TLOG_EVENT_1_OUT);
tlog_finalize();
MPI_Finalize();
```

例 - cpiのコンパイル

- **tlogライブラリをリンク**

```
% mpicc -O -o cpi cpi.c -ltlog
```

- **tlogライブラリのインストール**

/usr/localにインストールする例

```
% cd lib
% make
% sudo cp libtlog.a /usr/local/lib
% sudo cp tlog_dump /usr/local/bin
```

- **tlogviewのインストール**

```
% sudo mkdir /usr/local/tlog/tlogview
% sudo cp tlogview/tlogview.jar /usr/local/tlog/tlogview
% sudo cp bin/tlogview /usr/local/bin
```

例 - cpiの実行結果

```
$ mpiexec -hostfile hosts -n 4 cpi
adjust i=1,t1=0.011781,t2=0.011886,t0=0.011769,diff=6.7e-05
adjust i=2,t1=0.012911,t2=0.013015,t0=0.012877,diff=8.8e-05
adjust i=3,t1=0.014441,t2=0.014548,t0=0.014392,diff=0.000115
adjust i=1,t1=0.01623,t2=0.016335,t0=0.016285,diff=-2e-06
adjust i=2,t1=0.017314,t2=0.017418,t0=0.017367,diff=-2e-06
adjust i=3,t1=0.018401,t2=0.018504,t0=0.018454,diff=2.5e-06
tlog on ...
Process 0 on exp0.omni.hpcc.jp
pi is approximately 3.1416009869231249, Error is 0.0000083333333318
wall clock time = 0.000213
tlog finalizing ...
Process 3 on exp3.omni.hpcc.jp
Process 1 on exp1.omni.hpcc.jp
Process 2 on exp2.omni.hpcc.jp
tlog dump done ...
```

ノード間の時間差測定(デバッグ時に出力)

デバッグ時の出力

プログラムの出力

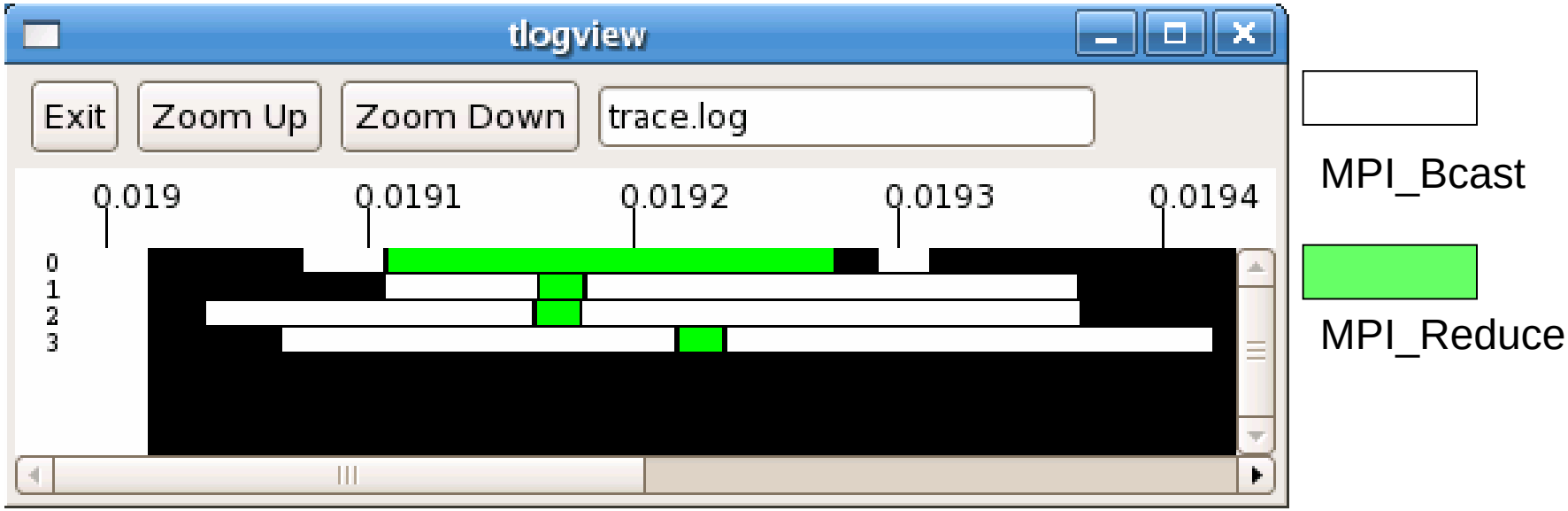
デバッグ時の出力

cpiのプロファイル結果（1）

- **tlogview – tlogの可視化ツール**

```
% tlogview trace.log
```

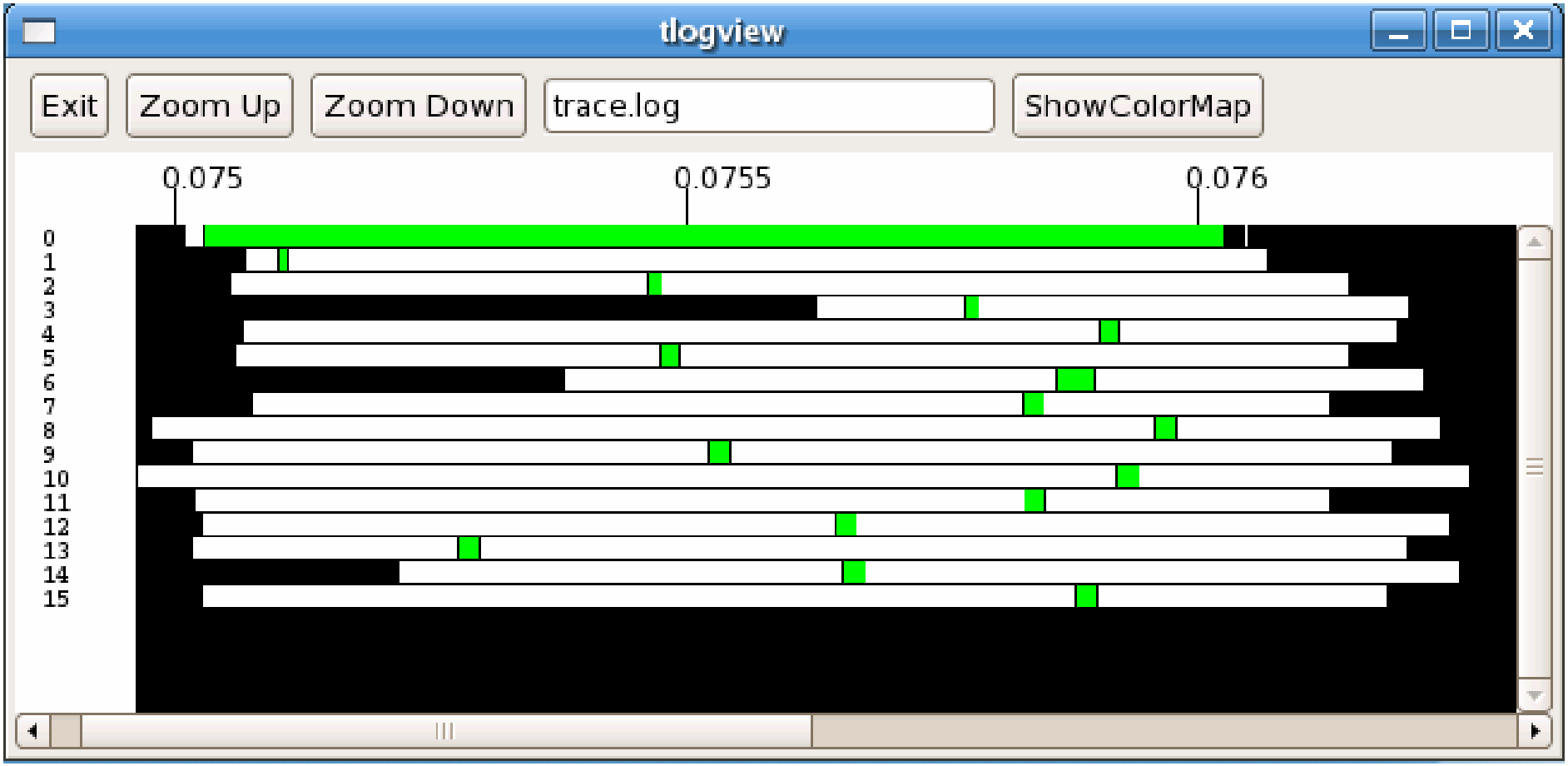
- **4プロセス（4ノード）での実行プロファイル**



tlog_initializeからの経過時間（秒）。ノード間の時刻差修正済。

cpiのプロファイル結果（2）

- **16プロセス（4ノード×4プロセス）のプロファイル**



MPI_Bcast　　MPI_Reduce

通信最適化

- 通信の削減*
- 通信遅延隠蔽
- 通信ブロック
- 負荷分散*

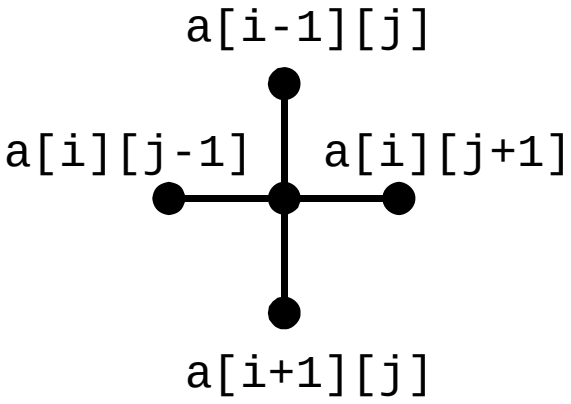
通信ブロッキング

- 1対1通信、集団通信はデータサイズによって通信性能が大きく変化する
- 通信ブロッキングは、通信データをまとめてデータサイズを変更する（大きくする）手法
  - データのブロック分散
  - 複数反復をまとめる

通信ブロッキングの例:ヤコビ法

- 二次元ポアソン方程式を5点差分で離散化した連立一次方程式の解法

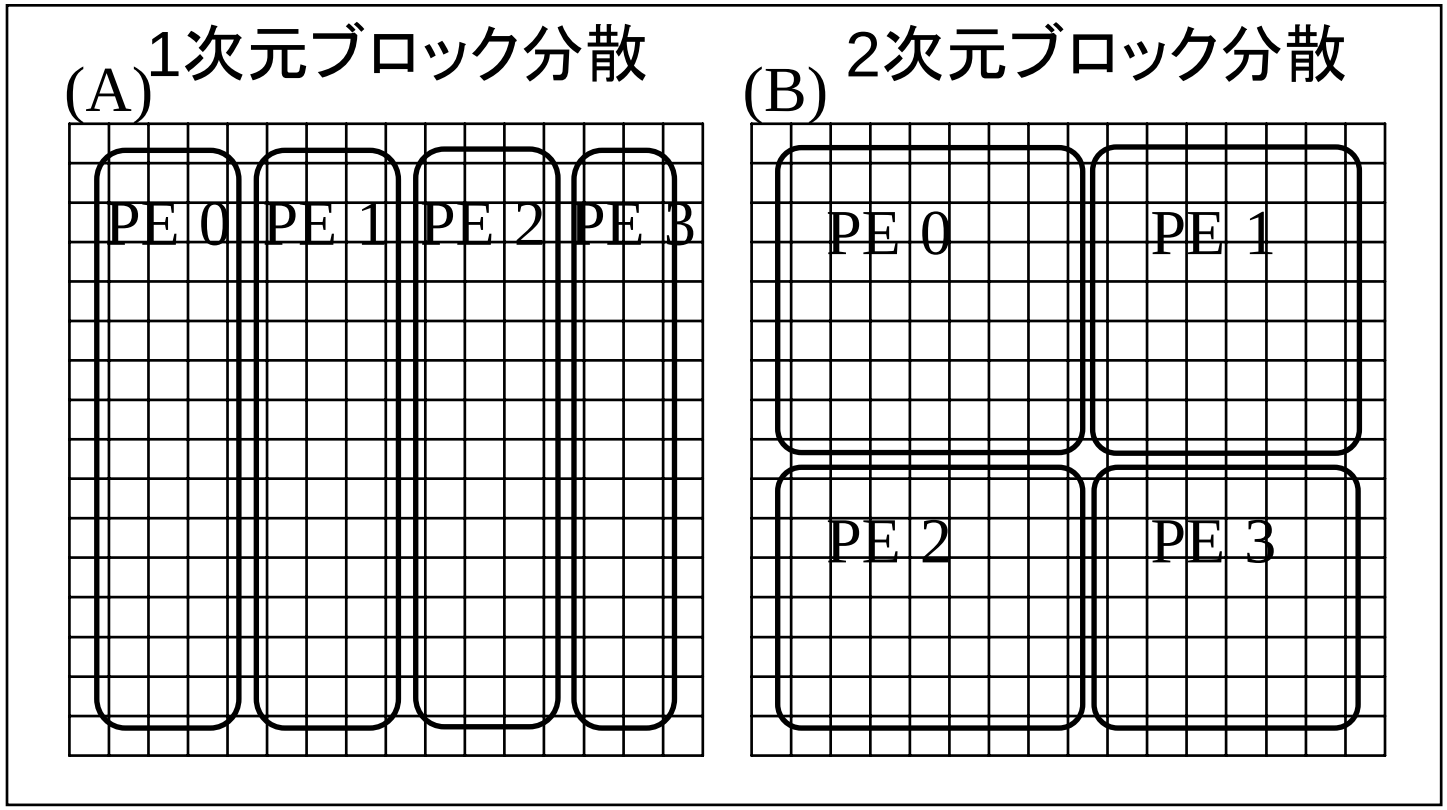
```
jacobi() {
  while (!converge) {
    for(i = 1; i < N - 1; ++i)
      for(j = 1; j < N - 1; ++j)
        b[i][j] = .25 *
              (a[i - 1][j] + a[i][j - 1]
               + a[i][j + 1] + a[i + 1][j]);
    /* 収束テスト */
    /* bをaにコピー */
  }
}
```



データ依存関係

*本当はヤコビ法ではなくRB-SOR法などを使って欲しい

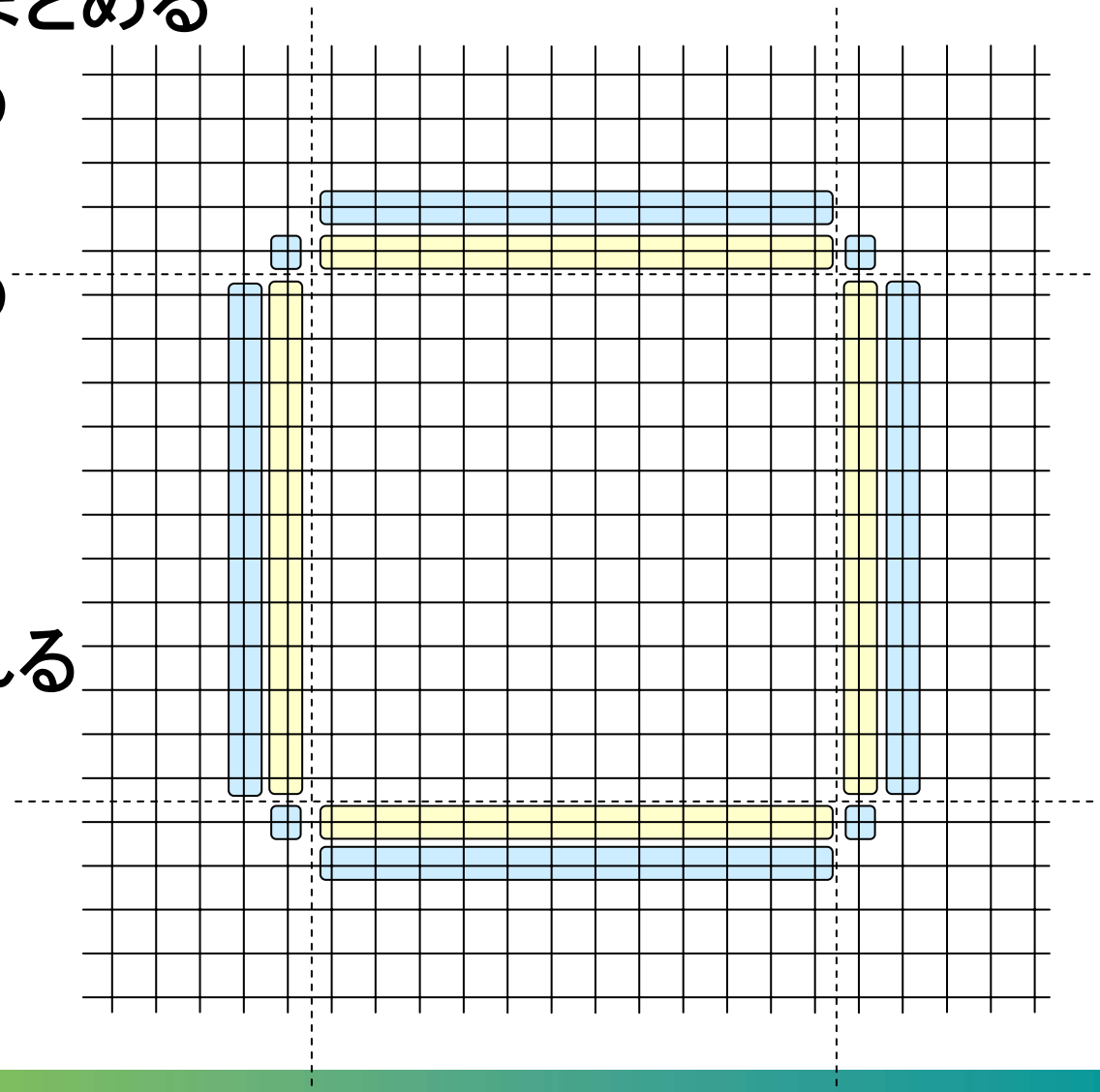
データのブロック分散



- データをブロック分割することにより通信データサイズを大きくできる
  - 1次元ブロック分散では　n
  - 2次元ブロック分散では　$n/\sqrt{p}$

複数反復をまとめる

- ヤコビ法二反復をまとめる
- 一反復目では の データが必要
- 二反復目では の データが必要
- と のデータを転送することにより二反復をまとめられる
  - 1次元　$2n$
  - 2次元　$2n/\sqrt{p}$



まとめ

- 基本通信性能
  - 1対1通信
  - 集団通信
- プロファイラ
- 通信最適化
  - 通信の削減
  - 通信遅延隠蔽
  - 通信ブロック
  - 負荷分散

おまけ - Open Source MPI

- **OpenMPI**
  - http://www.open-mpi.org/
- **MPICH2**
  - http://www-unix.mcs.anl.gov/mpi/mpich2/
- **YAMPII**
  - http://www.il.is.s.u-tokyo.ac.jp/yampii/

OpenMPIでのアプリケーションの実行

```
% cat hosts
host1 slots=4
host2 slots=4
host3 slots=4
host4 slots=4
% mpiexec –hostfile hosts –n 16 program
```

OpenMPIでのStdin/stdoutの扱い

- **mpiexec program < stdin > stdout**
  - StdinはMPI_COMM_WORLDのランク0が受け取る
  - Stdoutは全プロセスの標準出力