大規模系での高速フーリエ変換2

高橋大介
daisuke@cs.tsukuba.ac.jp
筑波大学システム情報系
計算科学研究センター

講義内容

- 並列三次元FFTにおける自動チューニング
- 二次元分割を用いた並列三次元FFT アルゴリズム
- GPUクラスタにおける並列三次元FFT

並列三次元FFTにおける自動チューニング

背景

- 並列FFTのチューニングを行う際には，さまざまな性能パラメータが存在する.
- しかし，最適な性能パラメータはプロセッサのアーキテクチャ，ノード間を結合するネットワーク，そして問題サイズなどに依存する.
- これらのパラメータをその都度自動でチューニングすることは困難になりつつある.
- そこで，近年自動チューニング技術が注目されてきている.
  - FFTW，SPIRAL，UHFFT

三次元FFT

- 三次元離散フーリエ変換（DFT）の定義

$$y(k_1, k_2, k_3) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3} \omega_{n_2}^{j_2 k_2} \omega_{n_1}^{j_1 k_1},$$

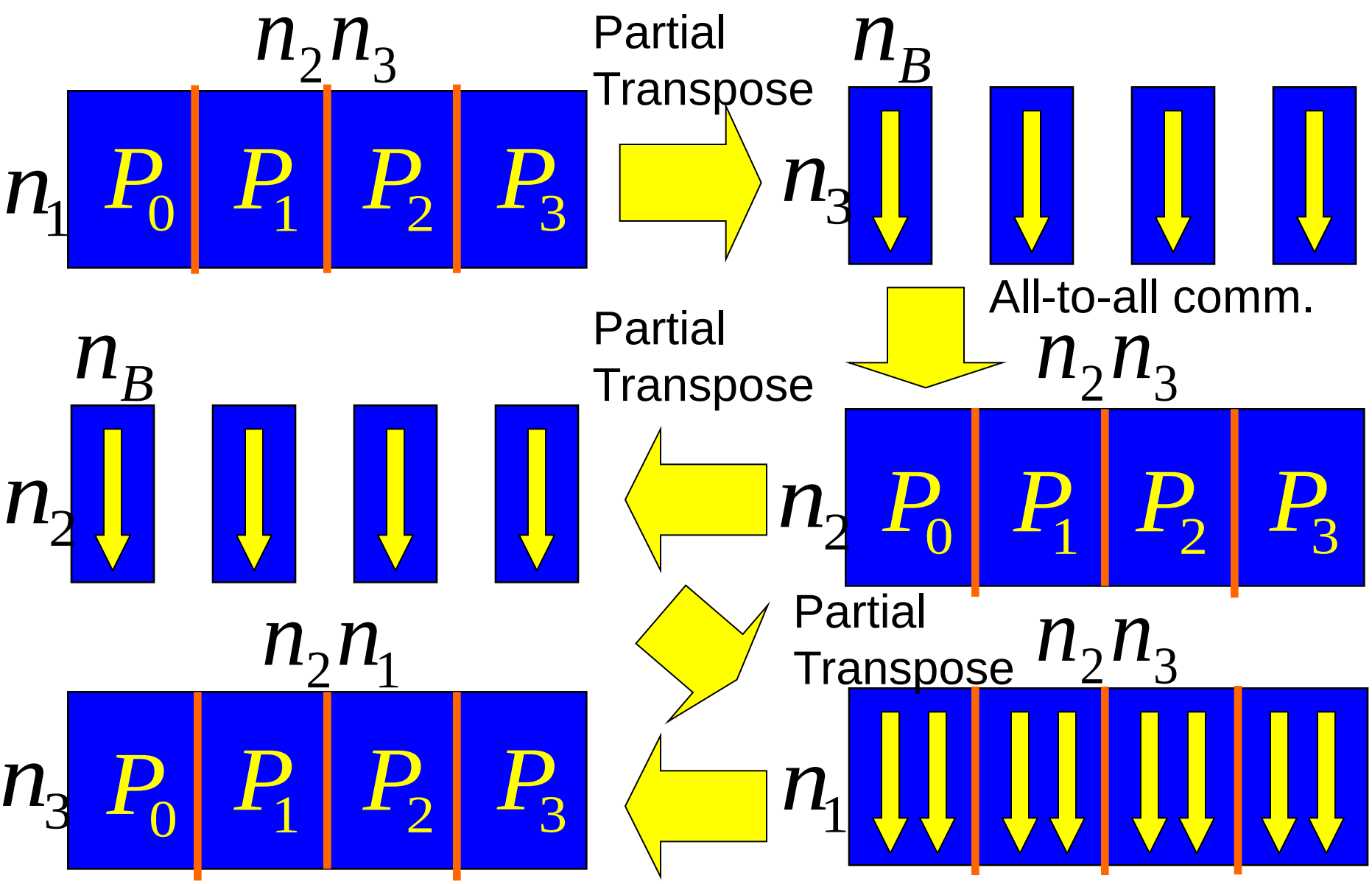
$$0 \le k_r \le n_r - 1,$$

$$\omega_{n_r} = e^{-2\pi i / n_r} \text{ and } i = \sqrt{-1}$$

三次元FFTアルゴリズム

- Step 1: $n_1 n_2$組のn_3点multicolumn FFT
- Step 2: 行列の転置
- Step 3: $n_1 n_3$組のn_2点multicolumn FFT
- Step 4: 行列の転置
- Step 5: $n_2 n_3$組のn_1点multicolumn FFT
- Step 6: 行列の転置

並列ブロック三次元FFTアルゴリズム



自動チューニング手法

- 並列ブロック三次元FFTをチューニングする際には，全体に関わる性能パラメータとして主に以下の2つが存在する.
  - 全対全通信方式
  - ブロックサイズ
- ここで，multicolumn FFTで用いる逐次FFTルーチンは自動チューニング等により十分に最適化されているものとする.

全対全通信方式

- 全対全通信が並列FFTの実行時間に占める割合は非常に大きい.
  - 場合によっては実行時間の大部分を占めることもある.
- これまでに, MPIの集合通信を自動チューニングする研究が行われている[Faraj and Yuan 05].
- InfiniBandで接続されたマルチコアクラスタにおいて, 全対全通信をノード内とノード間の2段階に分けて行うことで, 性能を向上させる手法も知られている[Kumar et al. 08].
- この手法をP個のMPIプロセスが$P = P_x \times P_y$のように分解できる一般的な場合に拡張した「2段階全対全通信アルゴリズム」を構築することができる.

2段階全対全通信アルゴリズム(1/2)

- 各MPIプロセスにおいて，配列の添字の順序を $(N/P^2, P_x, P_y)$ から $(N/P^2, P_y, P_x)$ に入れ替えるようにコピーする．次に，P_x 個のMPIプロセス間における全対全通信を P_y 組行う．
- 各MPIプロセスにおいて，配列の添字の順序を $(N/P^2, P_y, P_x)$ から $(N/P^2, P_x, P_y)$ に入れ替えるようにコピーする．次に，P_y 個のMPIプロセス間における全対全通信を P_x 組行う．

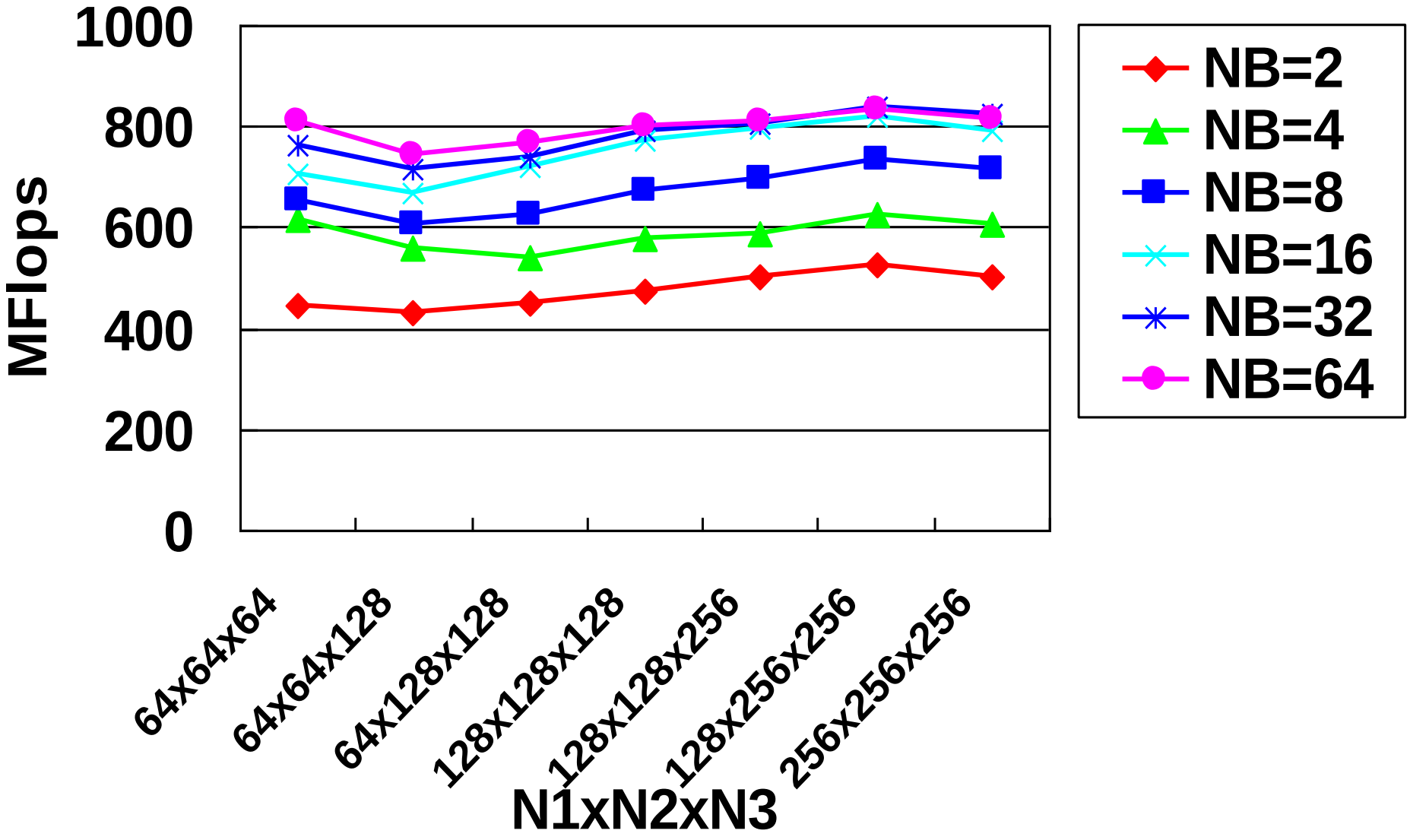
2段階全対全通信アルゴリズム(2/2)

- この2段階全対全通信アルゴリズムでは，ノード間の全対全通信が2回行われるため，P 個のMPIプロセス間で単純に全対全通信を行う場合に比べて，トータルの通信量は2倍となる.
- ところが，全対全通信のスタートアップ時間はMPIプロセス数 P に比例するため，N が比較的小さく，かつMPIプロセス数 P が大きい場合にはMPI_Alltoallに比べて有利になる場合がある.
- すべての P_x と P_y の組み合わせについて探索を行うことによって，最適な P_x と P_y の組み合わせを調べることができる.

ブロックサイズ

- 並列ブロック三次元FFTアルゴリズムにおいて，最適なブロックサイズは問題サイズおよびキャッシュサイズ等に依存する.
- ブロックサイズNBについても探索を行うことによって，最適なブロックサイズを調べることができる.
- 今回の実装では，データサイズ $n = n_1 \times n_2 \times n_3$ およびMPIプロセス数 P が2のべき乗であると仮定しているため，ブロックサイズNBも2のべき乗に限定して2，4，8，16，32，64のように変化させている.

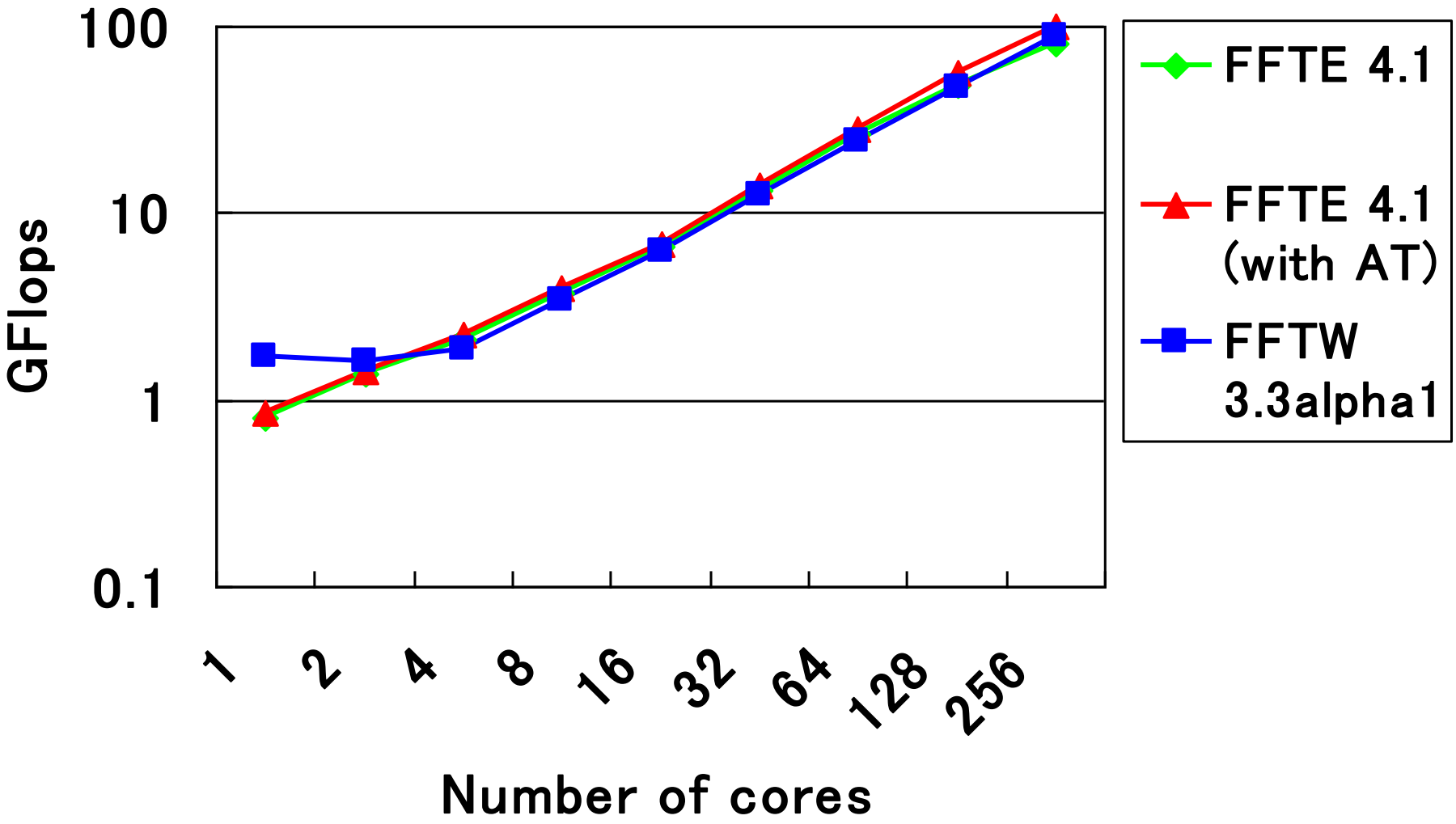
T2K筑波システム（1コア）においてブロックサイズを変化させた場合の性能



性能評価

- 性能評価にあたっては，以下の性能比較を行った.
  - 並列ブロック三次元FFTを用いたFFTライブラリであるFFTE（version 4.1）
  - FFTEに自動チューニングを適用したもの
  - 多くのプロセッサで最も高速なライブラリとして知られているFFTW（version 3.3alpha1）
- 測定に際しては，順方向FFTを連続10回実行し，その平均の経過時間を測定した.
- T2K筑波システム（648ノード，10,368コア）のうち16ノード，256コアを用いた.
- flat MPI（1コア当たり1MPIプロセス）
- MPIライブラリ：MVAPICH 1.4.1

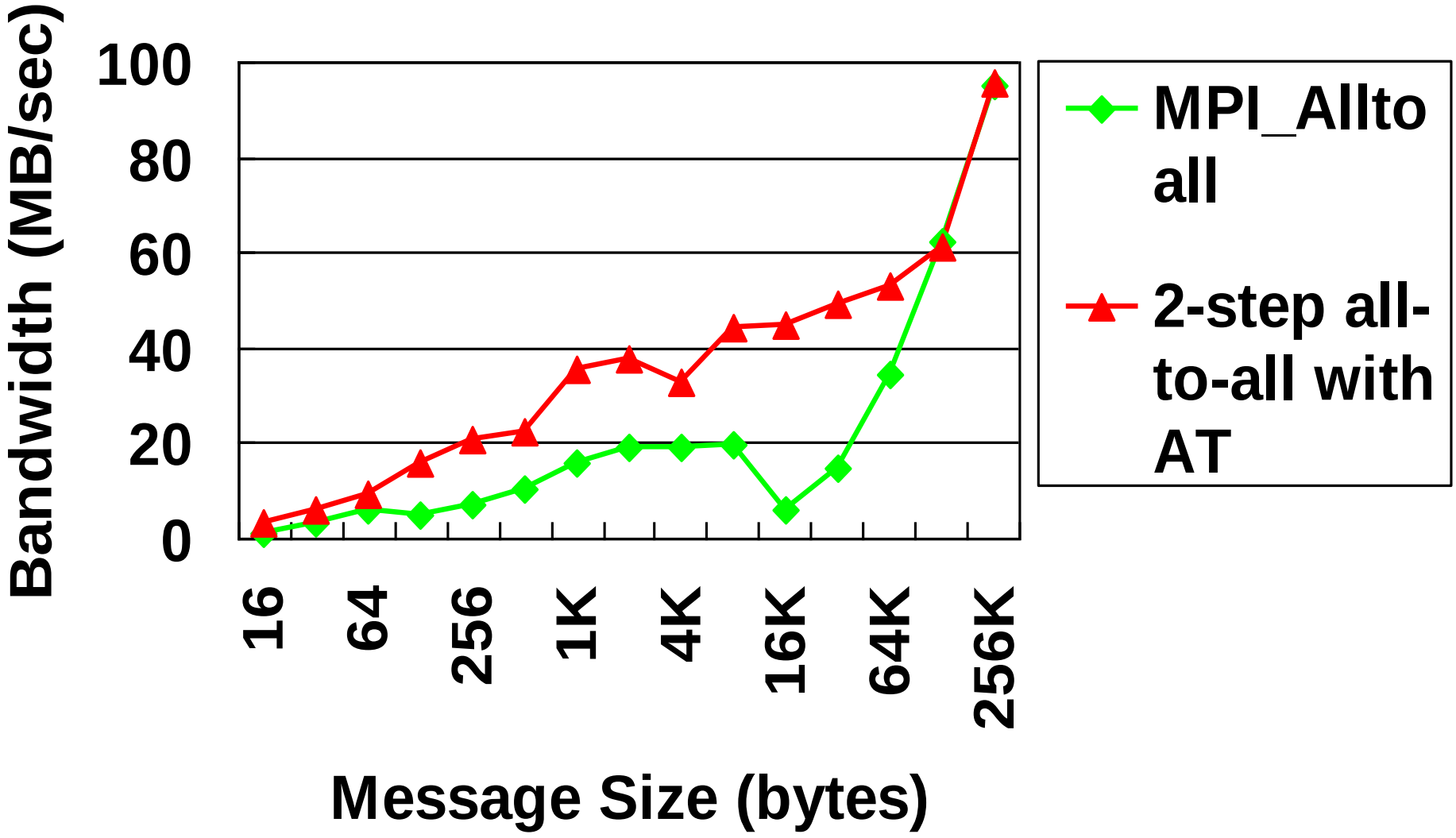
T2K筑波システムにおける並列三次元FFTの性能
（n1×n2×n3 = 2^24×コア数）



考察(1/2)

- FFTE 4.1に自動チューニングを適用することにより性能が向上していることが分かる.
- これは，FFTE 4.1において固定されていた全対全通信方式およびブロックサイズが，自動チューニングにより最適化されたことが理由と考えられる.
- また，4～256コアにおいて，自動チューニングを適用したFFTE 4.1がFFTW 3.3alpha1よりも高速であることが分かる.

T2K筑波システム（64ノード，1024コア，flat MPI）における全対全通信の性能



2段階全対全通信の自動チューニング結果

Message Size (bytes)	Px	Py
16	8	128
64	8	128
256	16	64
1024	32	32
4096	8	128
16384	64	16
65536	4	256
262144	1	1024

考察(2/2)

- メッセージサイズが64KB以下の範囲で，2段階全対全通信アルゴリズムが選択されており，MPI_Alltoallよりも高速になっている.
- $P_x = 1, 2, 4, 8, 16$ の場合には，1段目においてノード内に閉じた通信がP_x個のMPIプロセス間で行われることになる.
- メッセージサイズが16KBの場合には，$P_x = 64, P_y = 4$ が選択されており，ノード間で2段階通信が行われている.

二次元分割を用いた並列三次元FFTアルゴリズム

背景

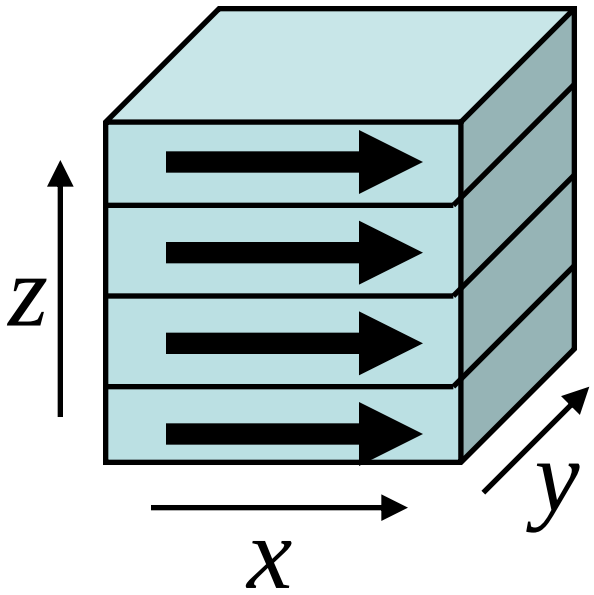
- 2013年11月のTop500リストにおいて，4システムが10PFlopsの大台を突破している.
  - Tianhe-2（Intel Xeon E5-2692 12C 2.2GHz，Intel Xeon Phi 31S1P）：33.862 PFlops（3,120,000 Cores）
  - Titan（Cray XK7，Opteron 6274 16C 2.2GHz，NVIDIA K20x）: 17.590 PFlops (560,640 Cores)
  - Sequoia（BlueGene/Q，Power BQC 16C 1.6GHz）: 17.173 PFlops (1,572,864 Cores)
  - K computer（SPARC64 VIIIfx 2.0GHz）：10.510 PFlops (705,024 Cores)
- 今後出現すると予想される，エクサフロップス級マシンは，ほぼ確実に1000万コアを超える規模のものになる.

方針

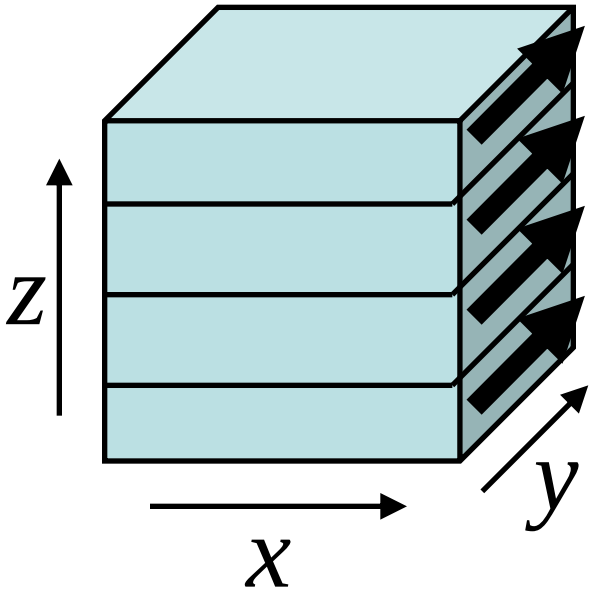
- 並列三次元FFTにおける典型的な配列の分散方法
  - 三次元（x, y, z方向）のうちの一次元のみ（例えばz方向）のみを分割して配列を格納.
  - MPIプロセスが1万個の場合，z方向のデータ数が1万点以上でなければならず，三次元FFTの問題サイズに制約.
- x, y, z方向に三次元分割する方法が提案されている[Eleftheriou et al. ’05, Fang et al. ’07].
  - 各方向のFFTを行う都度，全対全通信が必要.
- 二次元分割を行うことで全対全通信の回数を減らしつつ，比較的少ないデータ数でも高いスケーラビリティを得る.

z方向に一次元ブロック分割した場合の並列三次元FFT

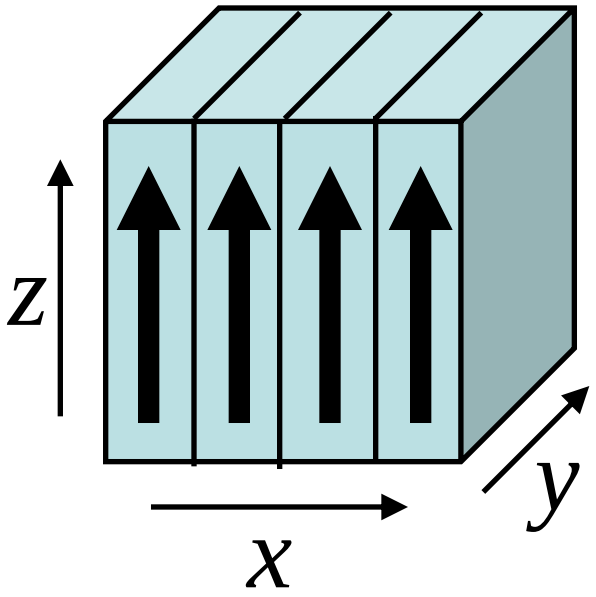
1. x方向FFT



2. y方向FFT



3. z方向FFT



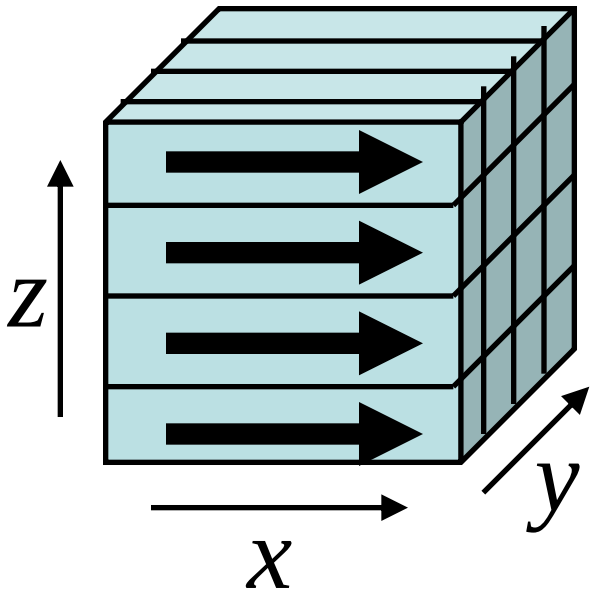
各プロセッサでslab形状に分割

三次元FFTの超並列化

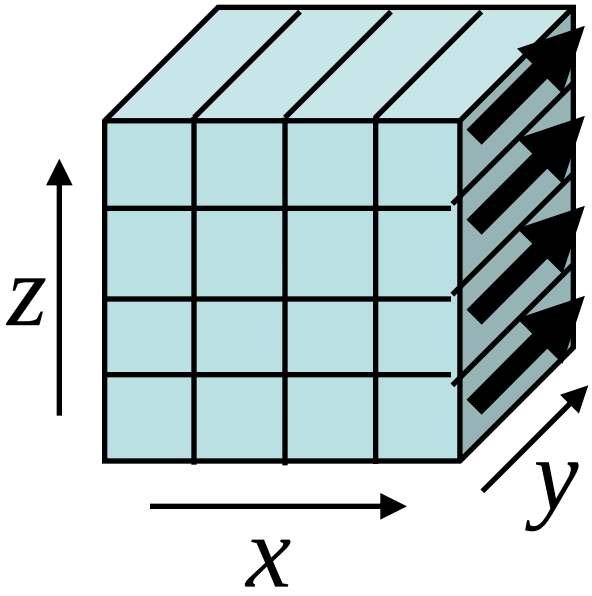
- 並列アプリケーションプログラムのいくつかにおいては，三次元FFTが律速になっている.
- x，y，zのうちz方向のみに一次元分割した場合，超並列化は不可能.
  - 1,024×1,024×1,024点FFTを2,048プロセスで分割できない（1,024プロセスまでは分割可能）
- y，zの二次元分割で対応する.
  - 1,024×1,024×1,024点FFTが1,048,576（=1,024×1,024）プロセスまで分割可能になる.

y, z方向に二次元ブロック分割した場合の並列三次元FFT

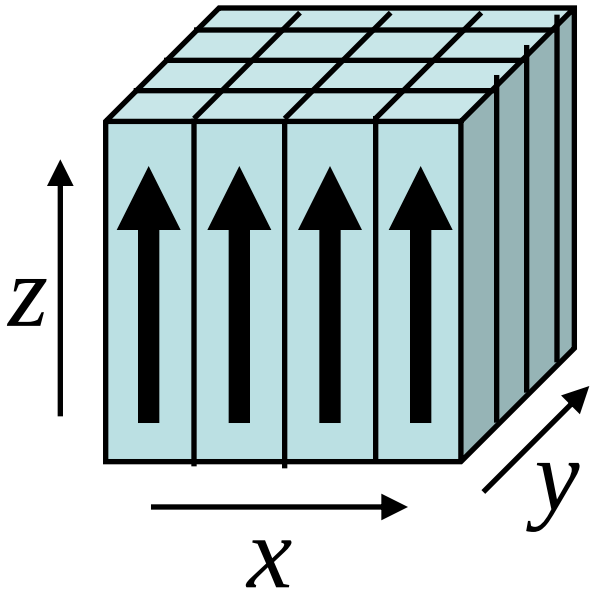
1. x方向FFT



2. y方向FFT



3. z方向FFT



各プロセッサで直方体形状に分割

二次元分割による並列三次元FFTの実装

- 二次元分割した場合, $P \times Q$ 個のプロセッサにおいて,
  - P 個のプロセッサ間で全対全通信を Q組
  - Q個のプロセッサ間で全対全通信を P 組

  行う必要がある.
- MPI_Comm_Split()を用いてMPI_COMM_WORLDをy方向（P プロセッサ）とz方向（Qプロセッサ）でコミュニケータを分割する.
  - 各コミュニケータ内でMPI_Alltoall()を行う.
- 入力データがy, z方向に, 出力データはx, y方向に二次元ブロック分割されている.
  - 全対全通信はy方向で1回, z方向で1回の合計2回で済む.

一次元分割の場合の通信時間

- 全データ数をN, プロセッサ数を$P \times Q$, プロセッサ間通信性能を W(Byte/s), 通信レイテンシを L (sec)とする.
- 各プロセッサは $N /(PQ)^2$ 個の倍精度複素数データを自分以外の $PQ-1$ 個のプロセッサに送ることになる.
- 一次元分割の場合の通信時間は

$$T_{1\mathrm{dim}} = (PQ-1)\left(L + \frac{16N}{(PQ)^2 \cdot W} \right)$$

$$\approx PQ \cdot L + \frac{16N}{PQ \cdot W} \quad (\mathrm{sec})$$

二次元分割の場合の通信時間

- y方向の P 個のプロセッサ間で全対全通信を Q 組行う.
  - y方向の各プロセッサは $N/(P^2Q)$ 個の倍精度複素数データを，y方向の $P-1$ 個のプロセッサに送る.
- z方向の Q 個のプロセッサ間で全対全通信を P 組行う.
  - z方向の各プロセッサは $N/(PQ^2)$ 個の倍精度複素数データを，z方向の $Q-1$ 個のプロセッサに送る.
- 二次元分割の場合の通信時間は

$$T_{2\dim} = (P-1)\left(L + \frac{16N}{P^2Q \cdot W}\right) + (Q-1)\left(L + \frac{16N}{PQ^2 \cdot W}\right)$$

$$\approx (P+Q)\cdot L + \frac{32N}{PQ \cdot W} \text{（sec）}$$

一次元分割と二次元分割の場合の通信時間の比較(1/2)

- 一次元分割の通信時間

$$T_{1\mathrm{dim}} \approx PQ \cdot L + \frac{16N}{PQ \cdot W}$$

- 二次元分割の通信時間

$$T_{2\mathrm{dim}} \approx (P+Q) \cdot L + \frac{32N}{PQ \cdot W}$$

- 二つの式を比較すると，全プロセッサ数 $P \times Q$ が大きく，かつレイテンシ L が大きい場合には，二次元分割の方が通信時間が短くなることが分かる.

一次元分割と二次元分割の場合の通信時間の比較(2/2)

- 二次元分割の通信時間が一次元分割の通信時間よりも少なくなる条件を求める.

$$(P+Q)\cdot L+\frac{32N}{PQ\cdot W}<PQ\cdot L+\frac{16N}{PQ\cdot W}$$

を解くと,

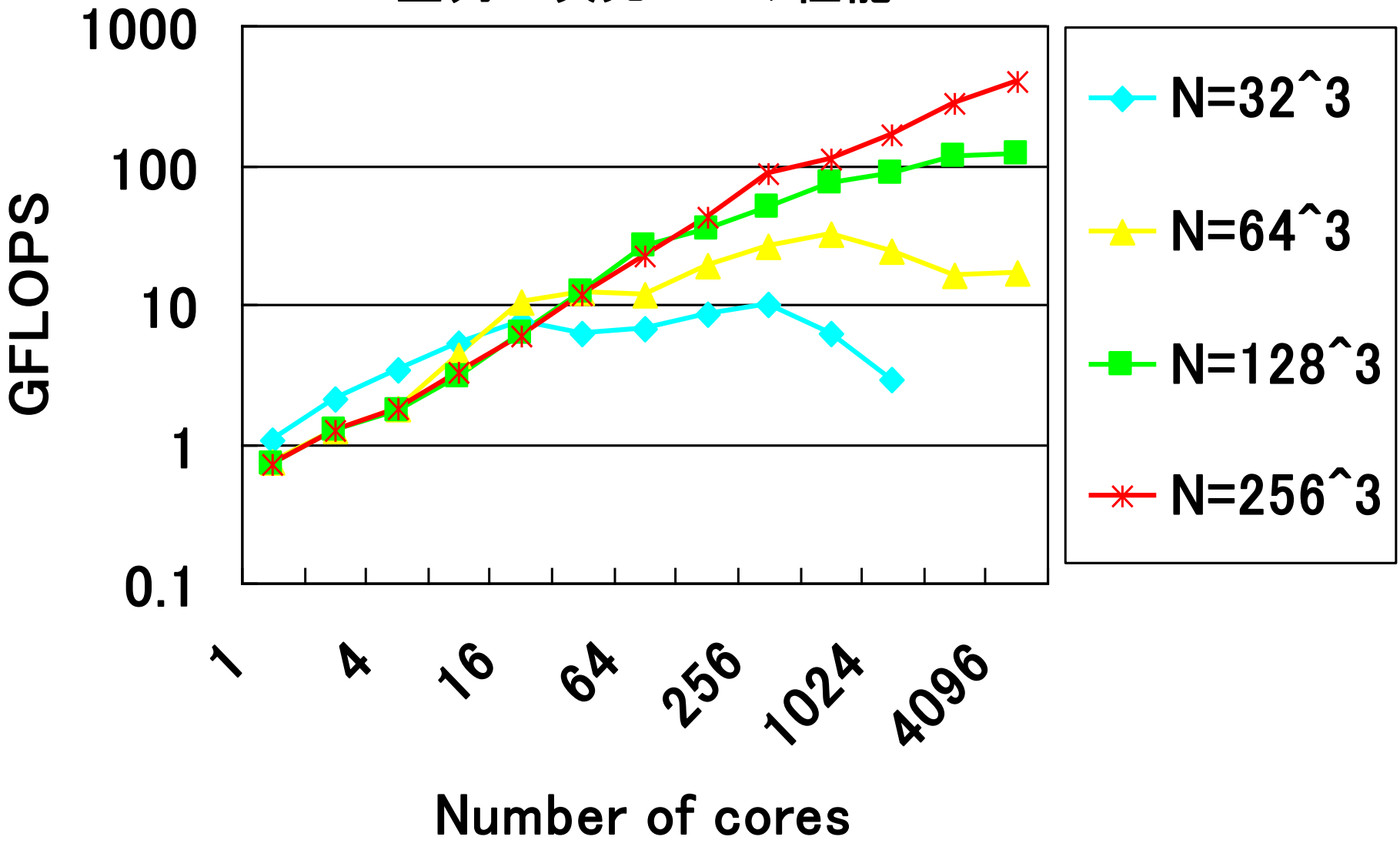
$$N<\frac{(LW\cdot PQ)(PQ-P-Q)}{16}$$

- 例えば, $L=10^{-5}$ (sec), $W=10^{9}$ (Byte/s), $P=Q=64$ を上の式に代入すると, $N<10^{10}$ の範囲では二次元分割の通信時間が一次元分割に比べて少なくなる.

性能評価

- 性能評価にあたっては，二次元分割を行った並列三次元FFTと，一次元分割を行った並列三次元FFTの性能比較を行った.
- Strong Scalingとして$N = 32^3, 64^3, 128^3, 256^3$ 点の順方向FFTを1～4,096MPIプロセスで連続10回実行し，その平均の経過時間を測定した.
- 評価環境
  - T2K筑波システムの256ノード（4,096コア）を使用
  - flat MPI（1core当たり1MPIプロセス）
  - MPIライブラリ：MVAPICH 1.2.0
  - Intel Fortran Compiler 10.1
  - コンパイルオプション：”ifort –O3 –xO”（SSE3ベクトル命令）

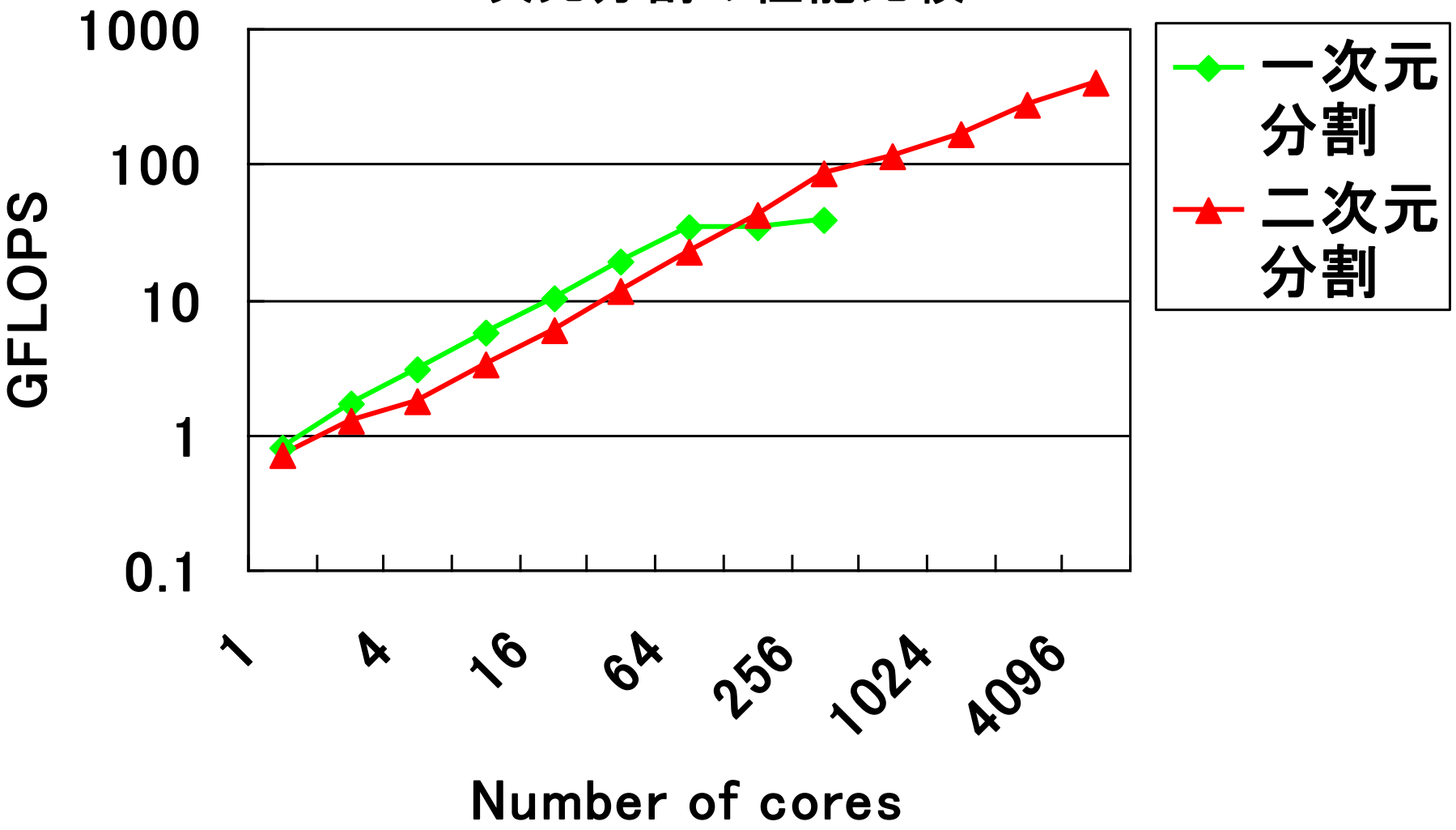
二次元分割を行ったvolumetric 並列三次元FFTの性能



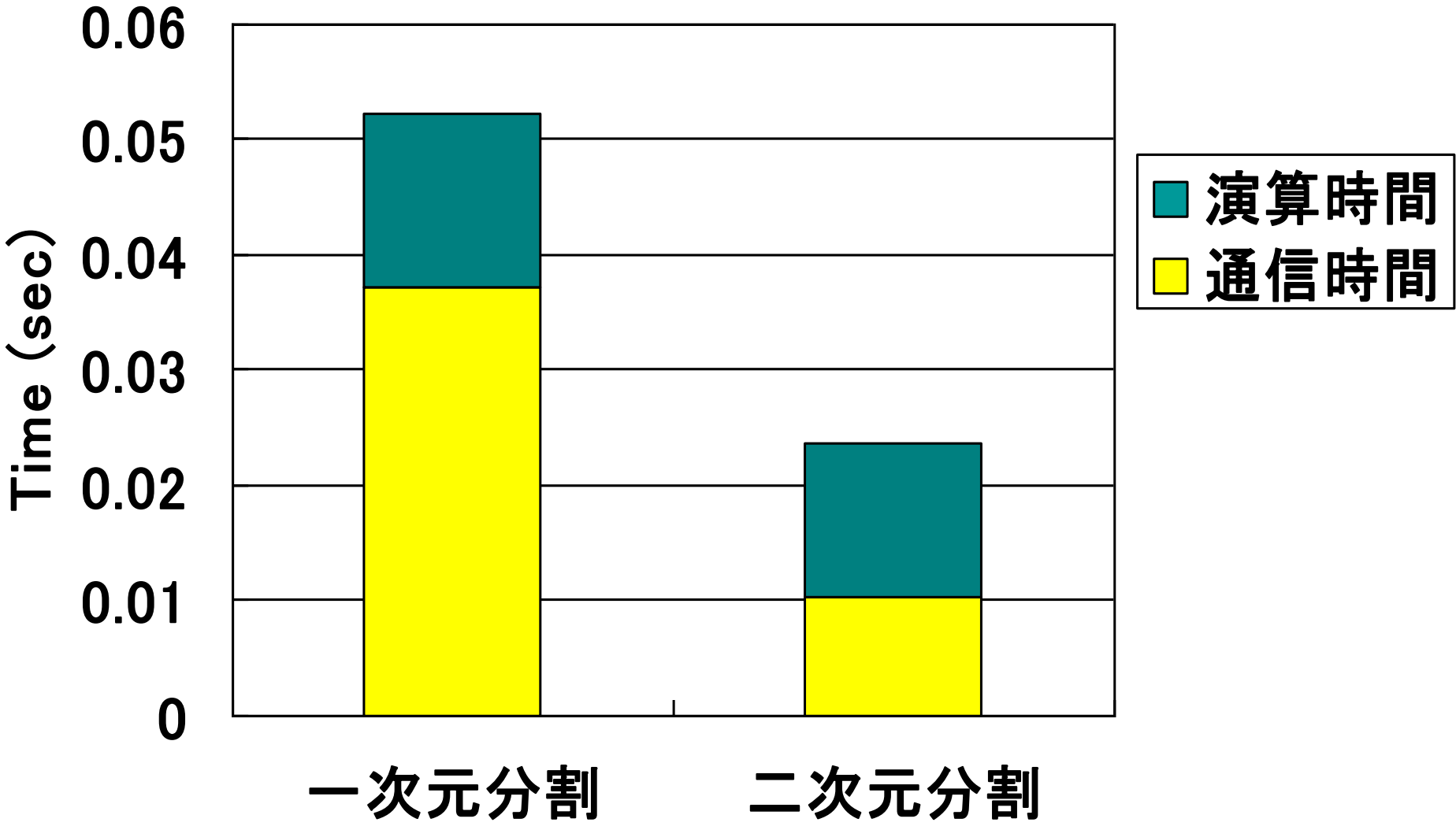
考察(1/2)

- $N = 32^3$ 点FFTでは良好なスケーラビリティが得られていない.
- これは問題サイズが小さい(データサイズ:1MB)ことから, 全対全通信が全実行時間のほとんどを占めているからであると考えられる.
- それに対して, $N = 256^3$ 点FFT(データサイズ:512MB)では4,096コアまで性能が向上していることが分かる.
  - 4,096コアにおける性能は約401.3 GFlops (理論ピーク性能の約1.1%)
  - 全対全通信を除いたカーネル部分の性能は約10.07 TFlops (理論ピーク性能の約26.7%)

256^3点FFTにおける一次元分割と二次元分割の性能比較



並列三次元FFTの実行時間の内訳
(256cores, 256^3点FFT)



考察(2/2)

- 64コア以下の場合には，通信量の少ない一次元分割が二次元分割よりも性能が高くなっている.
- 128コア以上では通信時間を少なくできる二次元分割が一次元分割よりも性能が高くなっていることが分かる.
- 二次元分割を行った場合でも，4,096コアにおいては96%以上が通信時間に費やされている.
  - 全対全通信において各プロセッサが一度に送る通信量がわずか1KBとなるため，通信時間においてレイテンシが支配的になるためであると考えられる.
- 全対全通信にMPI_Alltoall関数を使わずに，より低レベルな通信関数を用いて，レイテンシを削減する工夫が必要.

GPUクラスタにおける
並列三次元FFT

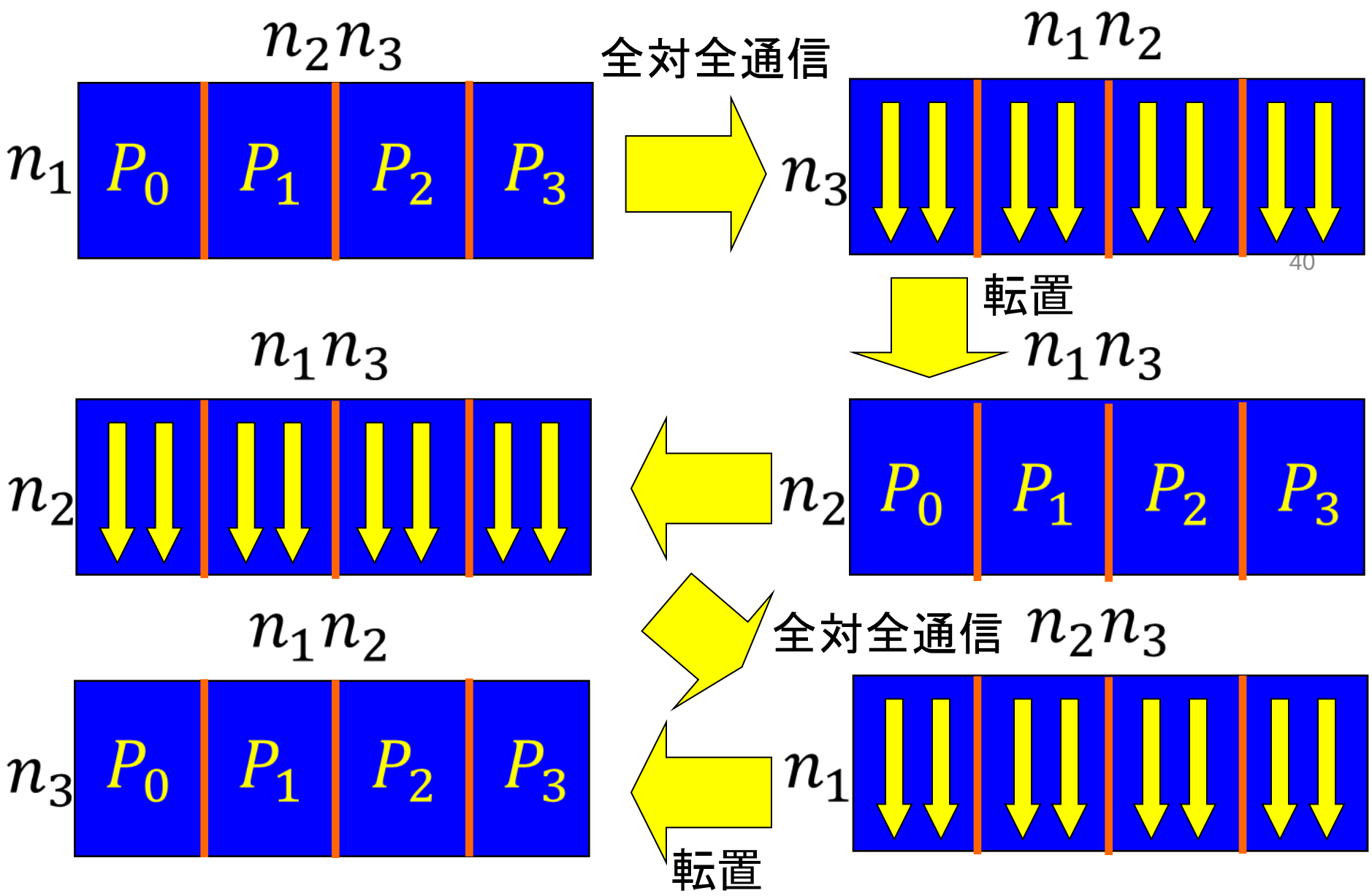
背景

- 近年，GPU（Graphics Processing Unit）の高い演算性能とメモリバンド幅に着目し，これを様々なHPCアプリケーションに適用する試みが行われている.
- また，GPUを搭載した計算ノードを多数接続したGPUクラスタも普及が進んでおり，2013年11月のTOP500リストではNVIDIA Tesla K20X GPUを搭載したTitanが第2位にランクされている.
- これまでにGPUクラスタにおける並列三次元FFTの実現は行われている[Chen et al. 2010, Nukada et al. 2012]が，一次元分割のみサポートされており，二次元分割はサポートされていない.

方針

- CPU版とGPU版を同一インターフェースとするため，入力データおよび出力データはホストメモリに格納する.
  - FFTライブラリが呼び出された際に，ホストメモリからデバイスメモリに転送し，FFTライブラリの終了時にデバイスメモリからホストメモリに転送する.
- 計算可能な問題サイズはGPUのデバイスメモリの容量が限度になる.
  - ホストメモリのデータを分割してデバイスメモリに転送しながらFFT計算を行うことも可能であるが，今回の実装ではそこまで行わないこととする.

並列三次元FFTアルゴリズム



GPUクラスタにおける並列三次元FFT（1/2）

- GPUクラスタにおいて並列三次元FFTを行う際には，全対全通信が2回行われる.
- 計算時間の大部分が全対全通信によって占められることになる.
- CPUとGPU間を接続するインターフェースであるPCI Expressバスの理論ピークバンド幅はPCI Express Gen 2 x 16レーンの場合には一方向あたり8GB/sec.
- CPUとGPU間のデータ転送量をできるだけ削減することが重要になる.
  - CPUとGPU間のデータ転送はFFTの開始前と終了後にそれぞれ1回のみ行う.
  - 行列の転置はGPU内で行う.

GPUクラスタにおける並列三次元FFT（2/2）

- GPU上のメモリをMPIにより転送する場合，以下の手順で行う必要がある.
  1. GPU上のデバイスメモリからCPU上のホストメモリへデータをコピーする.
  2. MPIの通信関数を用いて転送する.
  3. CPU上のホストメモリからGPU上のデバイスメモリにコピーする.
- この場合，CPUとGPUのデータ転送を行っている間はMPIの通信が行われないという問題がある.
- そこで，CPUとGPU間のデータ転送とノード間のMPI通信をパイプライン化してオーバーラップさせることができるMPIライブラリであるMVAPICH2を用いた.

MPI + CUDAでの通信

- 通常のMPIを用いたGPU間の通信

```
At Sender:
cudaMemcpy(sbuf, s_device, …);
MPI_Send(sbuf, size, …);

At Receiver:
MPI_Recv(rbuf, size, …);
cudaMemcpy(r_device, rbuf, …);
```

・cudaMemcpyを行っている間はMPIの通信が行われない.
・メモリをブロックで分割し, CUDAとMPIの転送をオーバーラップさせることも可能.
→プログラムが複雑になる.

- MVAPICH2-GPUを用いたGPU間の通信

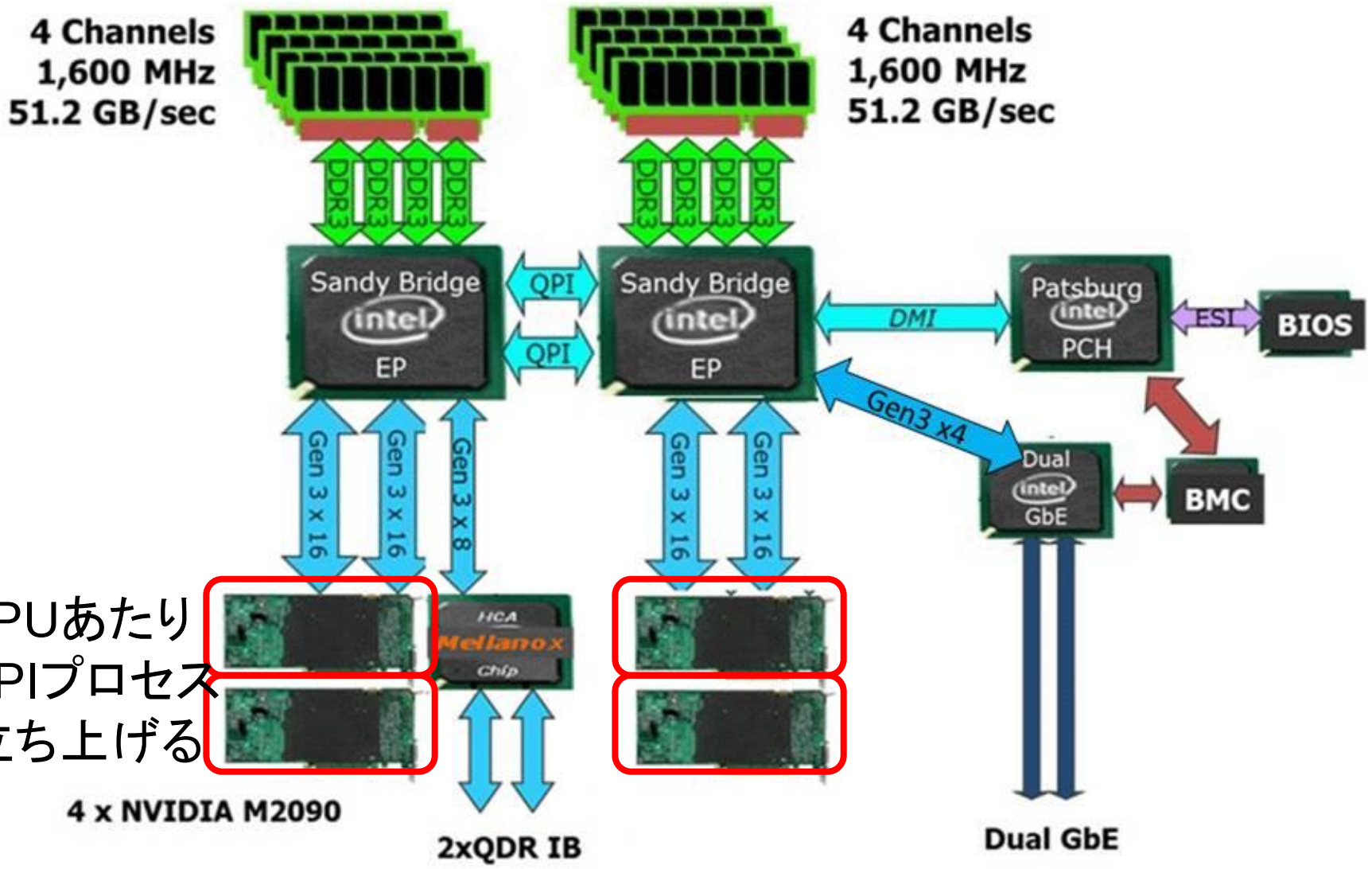
```
At Sender:
MPI_Send(s_device, size, …);

At Receiver:
MPI_Recv(r_device, size, …);
```

・デバイスメモリのアドレスを直接MPI関数に渡すことが可能.
・CUDAとMPIの転送のオーバーラップをMPIライブラリ内で行う.

性能評価

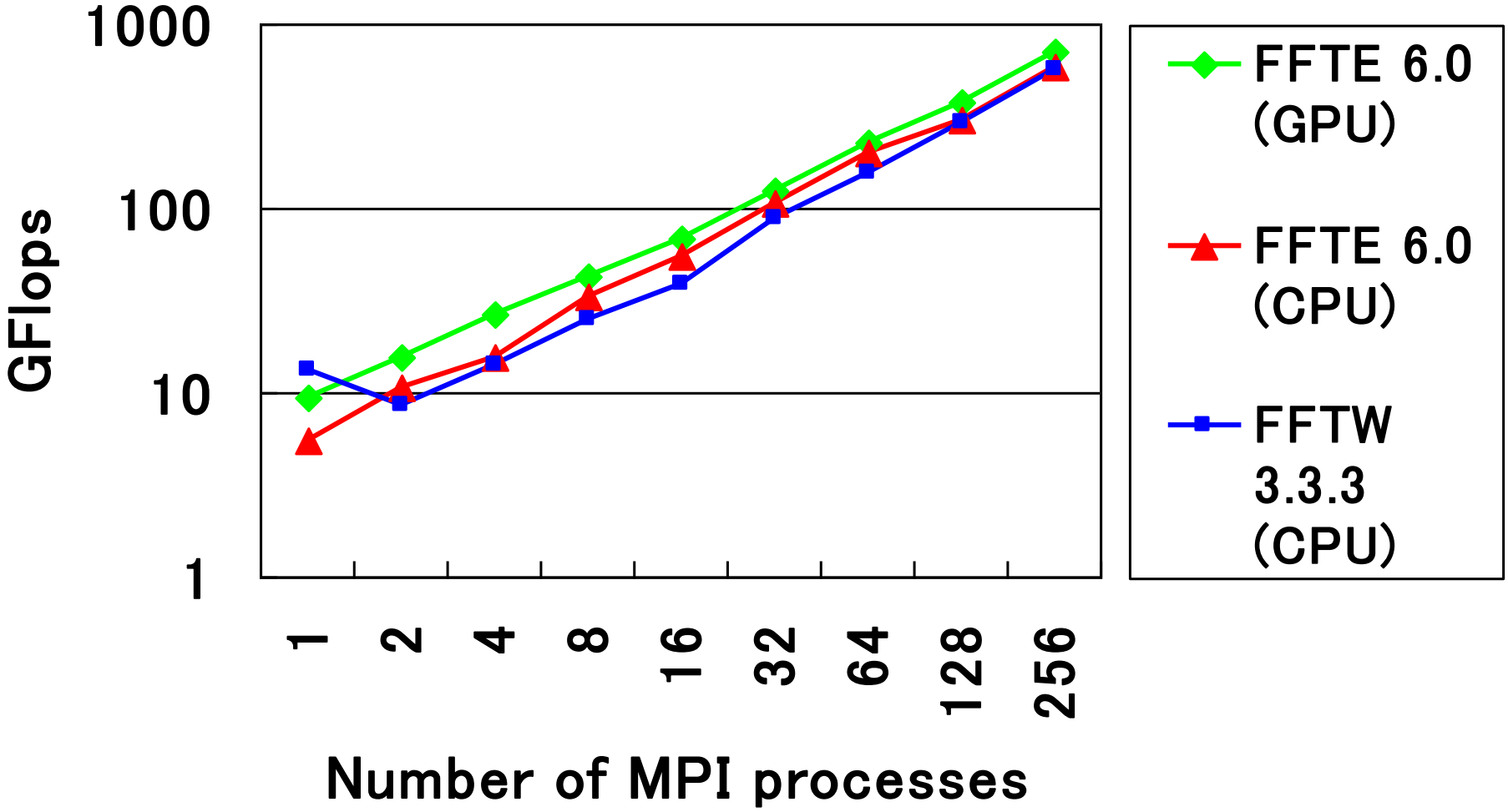
- 性能評価にあたっては，以下のFFTライブラリについて性能比較を行った.
  - FFTE 6.0（http://www.ffte.jp/，GPUを使用）
  - FFTE 6.0（http://www.ffte.jp/，CPUを使用）
  - FFTW 3.3.3（http://www.fftw.org/，CPUを使用）
- 順方向FFTを1～256MPIプロセス（1ノードあたり4MPIプロセス）で連続10回実行し，その平均の経過時間を測定した.
- HA-PACSベースクラスタ（268ノード，4288コア，1072GPU）のうち，1～64ノードを使用した.
  - 各ノードにIntel Xeon E5-2670（Sandy Bridge-EP 2.6GHz）が2ソケット，NVIDIA Tesla M2090が4基
  - ノード間はInfiniBand QDR（2レール）で接続
  - MPIライブラリ：MVAPICH2 2.0b
  - PGI CUDA Fortran Compiler 14.2 + CUDA 5.5 + CUFFT
  - コンパイラオプション：“pgf90 -fast -Mcuda=cc2x,cuda5.5”（FFTE 6.0，GPU），“pgf90 –fast -mp”（FFTE 6.0，CPU），”pgcc -fast”（FFTW 3.3.3）

HA-PACSベースクラスタのノード構成

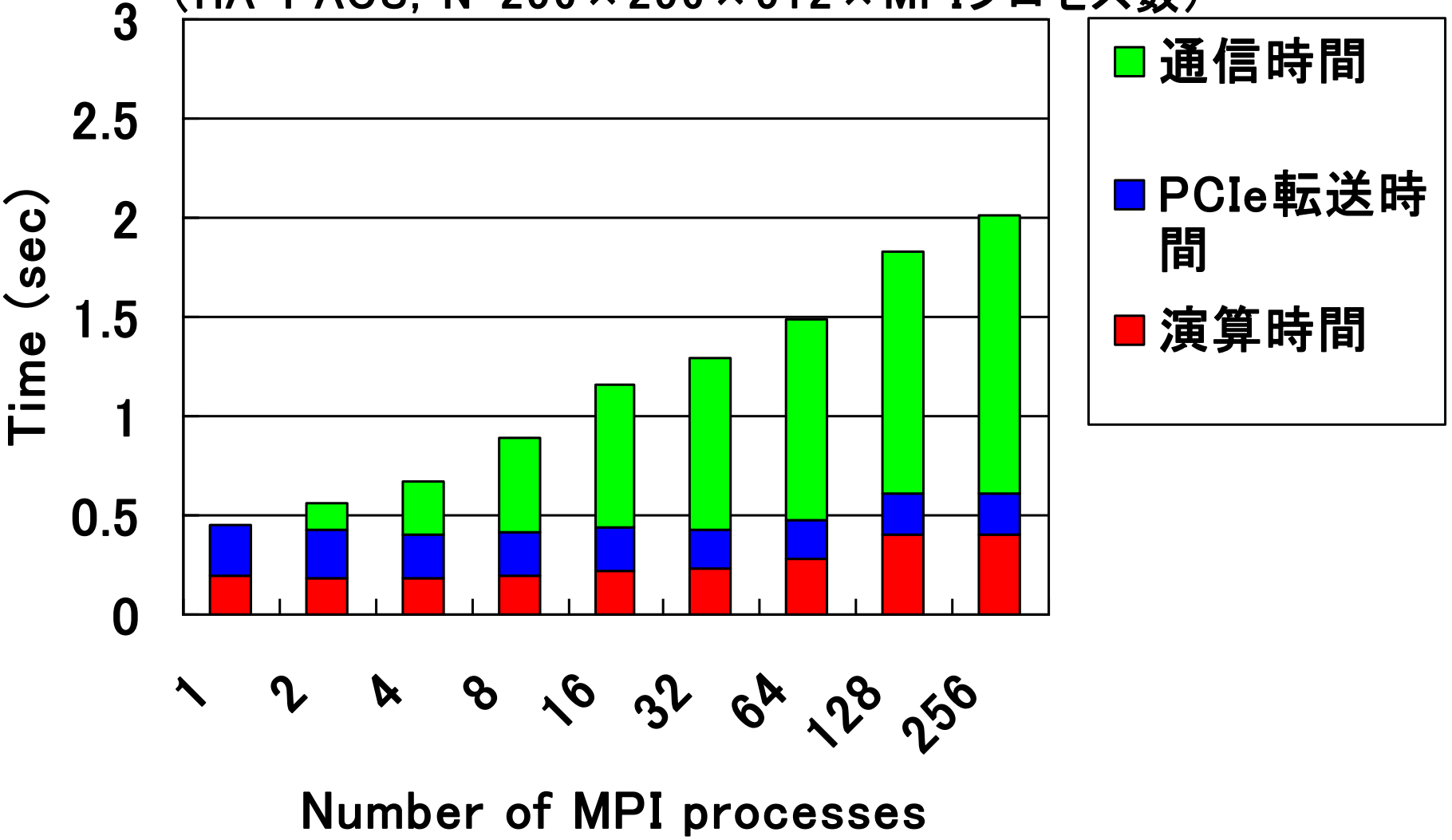


並列三次元FFTの性能

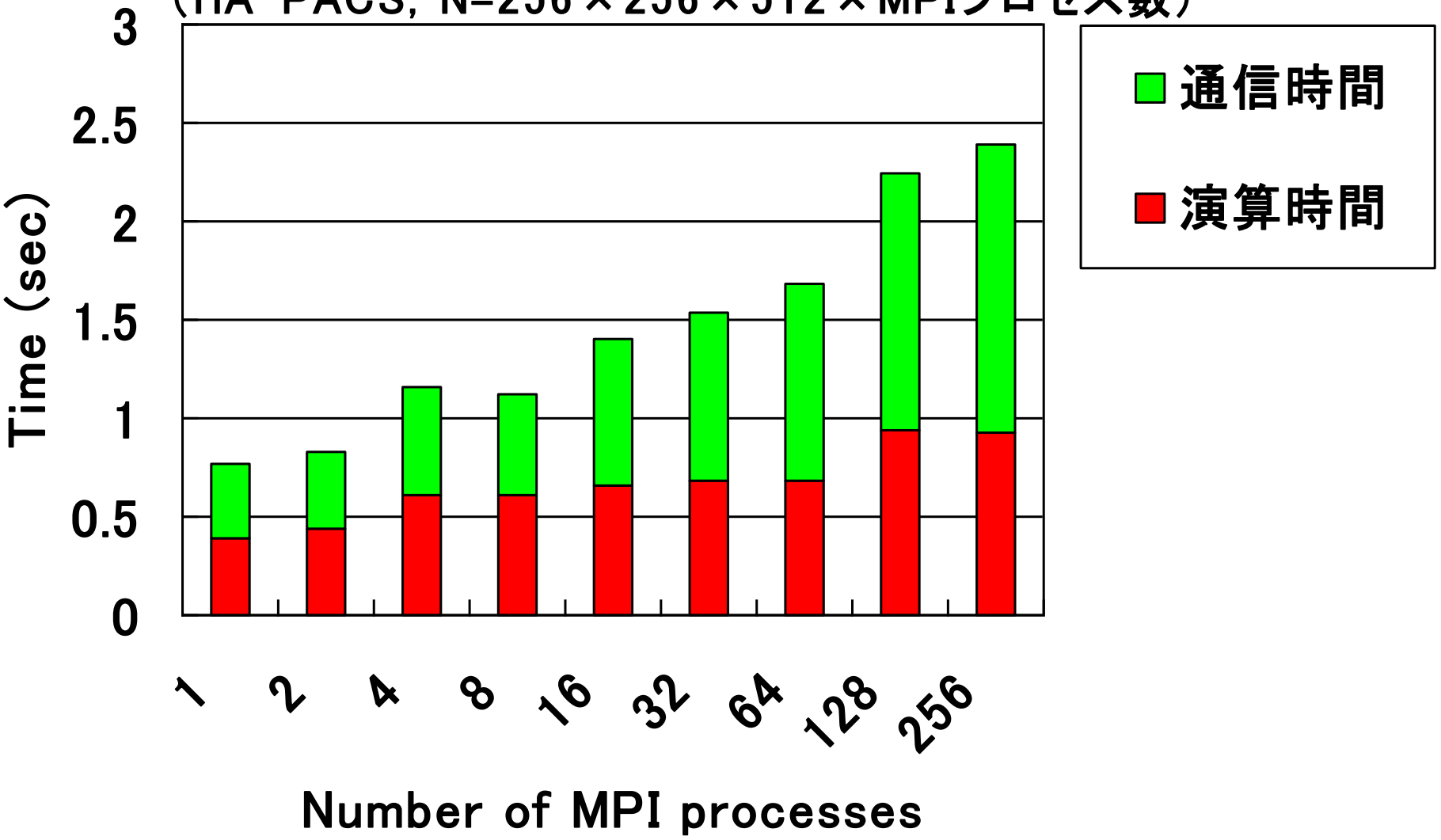
（HA-PACS，N=256×256×512×MPIプロセス数）

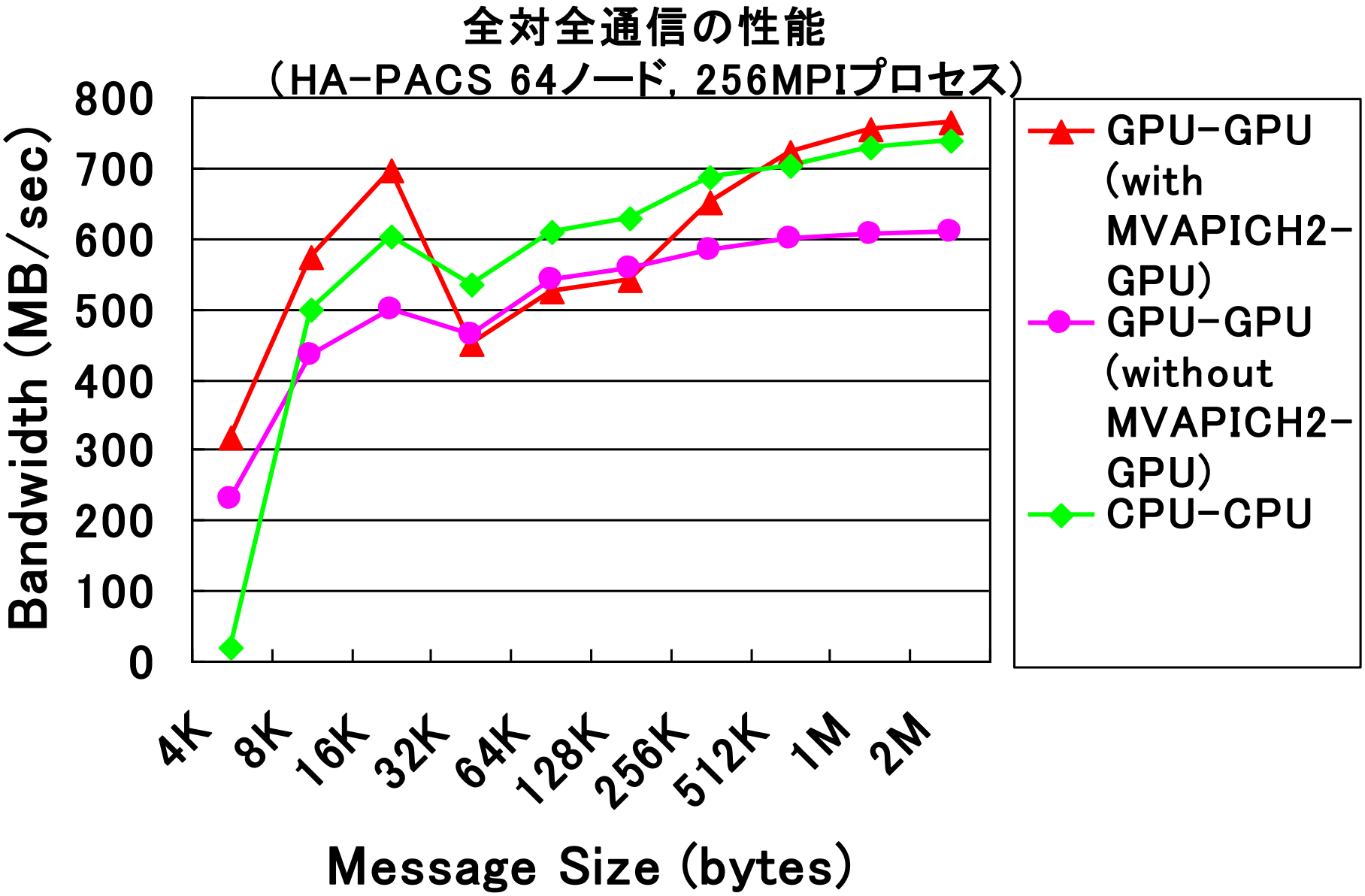


FFTE 6.0（GPU版）の並列三次元FFTの実行時間の内訳
（HA-PACS, N=256×256×512×MPIプロセス数）



FFTE 6.0（CPU版）の並列三次元FFTの実行時間の内訳
（HA-PACS, N=256×256×512×MPIプロセス数）




全対全通信の性能
（HA-PACS 64ノード, 256MPIプロセス）
Bandwidth (MB/sec)
800
700
600
500
400
300
200
100
0
4K
8K
16K
32K
64K
128K
256K
512K
1M
2M
Message Size (bytes)
GPU-GPU (with MVAPICH2-GPU)
GPU-GPU (without MVAPICH2-GPU)
CPU-CPU

まとめ(1/2)

- 物質科学の実アプリケーションにおいて使われることが多い，高速フーリエ変換(FFT)について紹介した.
- これまで並列FFTで行われてきた自動チューニングでは，基数の選択や組み合わせ，そしてメモリアクセスの最適化など，主にノード内の演算性能だけが考慮されてきた.
- ノード内の演算性能だけではなく，全対全通信の最適化においても自動チューニングが必要になる.
- 今後，並列スーパーコンピュータの規模が大きくなるに従って、FFTの効率を向上させることは簡単ではない.
  - 二次元分割や三次元分割が必要がある.

まとめ(2/2)

- GPUを用いた場合にはCPUに比べて演算時間が短縮される一方で，全実行時間における通信時間の割合が増大する.
  - HA-PACSベースクラスタの64ノード，256MPIプロセスを用いた場合，2048^3点FFTにおいて実行時間の約70%が全対全通信で占められている.
- MPIライブラリであるMVAPICH2の新機能(MVAPICH2-GPU)を用いることで，PCIe転送とノード間通信をオーバーラップさせた際のプログラミングが容易になるとともに通信性能も向上した.