大規模系での高速フーリエ変換2

高橋大介
daisuke＠cs.tsukuba.ac.jp
筑波大学計算科学研究センター

講義内容

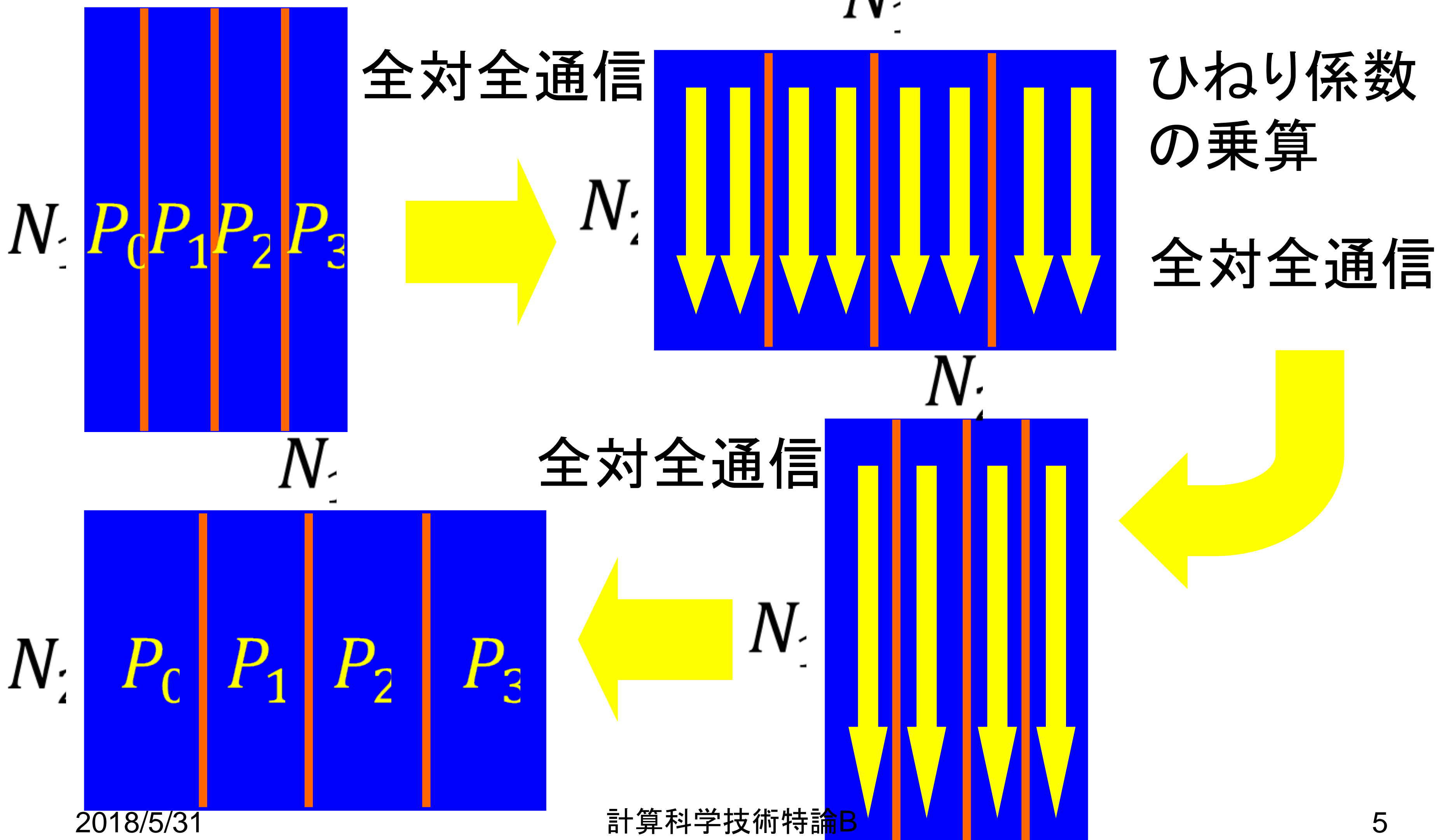
- Xeon Phiクラスタ上の並列FFTにおける通信隠蔽の自動チューニング
- 二次元分割を用いた並列三次元FFTアルゴリズム
- GPUクラスタにおける並列三次元FFT

Xeon Phiクラスタ上の並列FFTにおける通信隠蔽の自動チューニング

背景

- 高速フーリエ変換（fast Fourier transform，以下FFT）は科学技術計算において今日広く用いられているアルゴリズム.
- 分散メモリ型並列計算機においてチューニングを行う際に，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ため，これらのパラメータをその都度手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，自動チューニングを適用したFFTライブラリとしてFFTWやSPIRALなどが提案されている.
- また，並列FFTにおける演算と通信のオーバーラップ手法が提案されている[Doi and Negishi 2010].
- 並列一次元FFTにおいて通信隠蔽のパラメータを自動チューニングしXeon Phiクラスタ上で性能評価を行う.

Six-Step FFT [Bailey 90]に基づいた並列一次元FFTアルゴリズム



Xeon Phiプロセッサにおける最適化

```
      COMPLEX*16 X(N1,N2),Y(N2,N1)
!$OMP PARALLEL DO COLLAPSE(2) PRIVATE(I,J,JJ)
      DO II=1,N1,NB
        DO JJ=1,N2,NB
          DO I=II,MIN(II+NB-1,N1)
            DO J=JJ,MIN(JJ+NB-1,N2)
              Y(J,I)=X(I,J)
            END DO
          END DO
        END DO
      END DO
!$OMP PARALLEL DO
      DO I=1,N1
        CALL IN_CACHE_FFT(Y(1,I),N2)
      END DO
      …
```

最外側ループ長を大きくするために，OpenMPのcollapse節を用いて二重ループを一重化する.

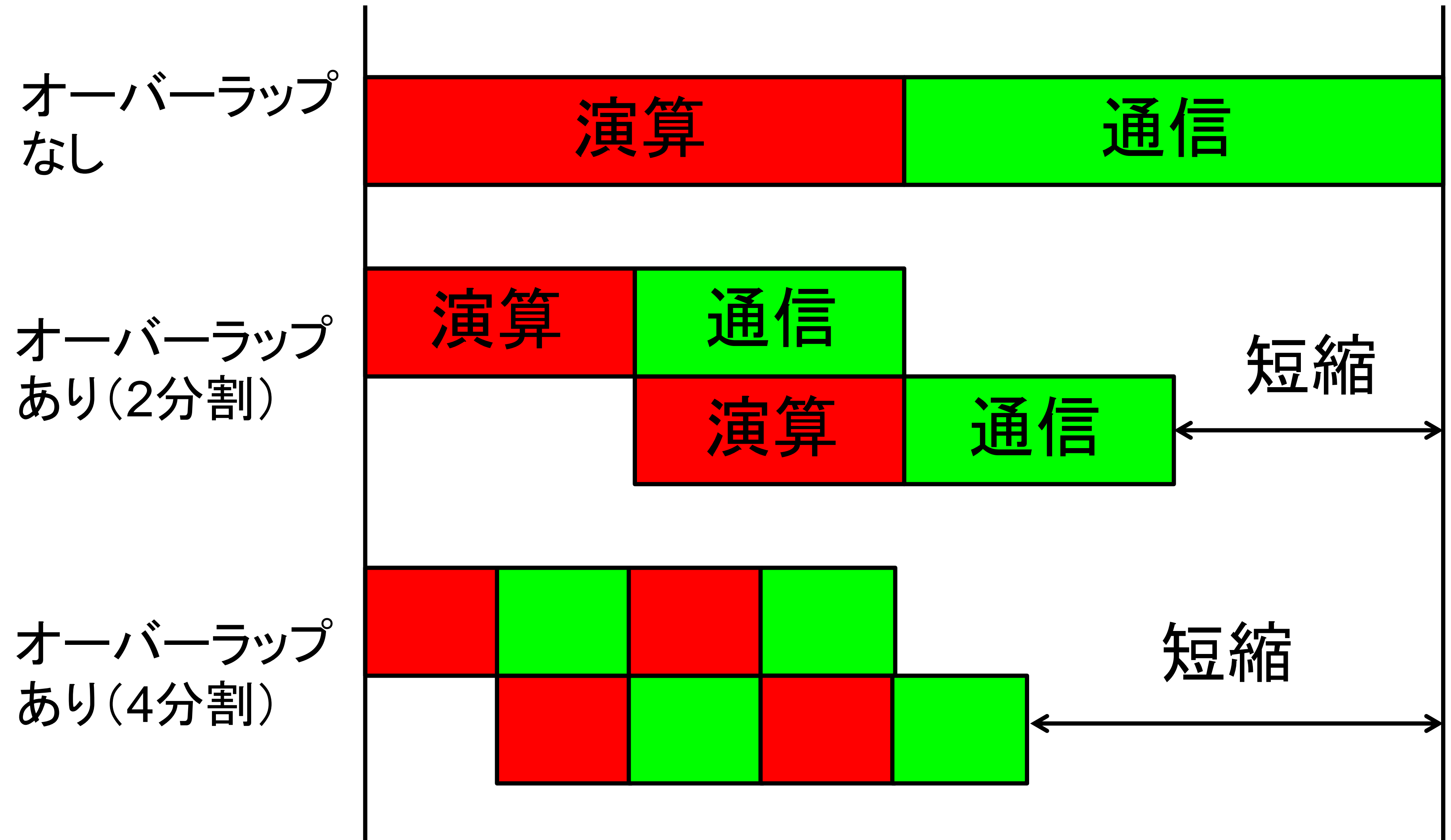
[Idomura et al. 2014]の手法に基づく演算と通信のオーバーラップの記述例

```
      CALL MPI_INIT(IERR)
      …
!$OMP PARALLEL
!$OMP MASTER
        オーバーラップするMPI通信
!$OMP END MASTER      ←同期なし
!$OMP DO SCHEDULE(DYNAMIC)
      DO I=1,N
        オーバーラップする演算
      END DO          ←暗黙の同期
!$OMP DO
      DO I=1,N
        通信結果を使用する演算
      END DO
!$OMP END PARALLEL
      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

←マスタースレッドで通信を実行
（通信が早く終われば演算に参加可能）

←マスタースレッド以外のスレッドで
演算を実行

演算と通信のオーバーラップ（1/2）



自動チューニング手法

- 分散メモリ型並列計算機において並列一次元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4つが存在する.
  (1) 全対全通信方式
  (2) 演算と通信をパイプライン方式でオーバーラップさせる際のパイプラインの段数
  (3) 基底（N_1，N_2）
  (4) ブロックサイズ
- これらの性能パラメータを探索することで，並列一次元FFTの性能をさらに向上させることが可能である.
- 上記の(1)～(2)はMPIプロセス間通信に関するパラメータであり，(3)～(4)はMPIプロセス内の性能に関するパラメータである.
- 今回は(2)～(4)に対して自動チューニングを適用した.

通信メッセージサイズの分割数

- 演算と通信をパイプライン方式でオーバーラップさせる場合，通信メッセージサイズの分割数（パイプラインの段数）を大きくすれば，オーバーラップの割合が高くなる．
- その一方で，通信メッセージサイズの分割数を大きくすれば，通信1回あたりの通信メッセージサイズが小さくなるため，通信バンド幅も小さくなる．
- また，演算と通信をオーバーラップさせる際には，通信によってメモリバンド幅が消費される．
- したがって通信メッセージサイズの分割数には最適な値が存在すると考えられる．
- 通信メッセージサイズの分割数をNDIVとしたとき，NDIV=1（オーバーラップなし）～16の範囲で，通信すべき要素数がNDIVで割り切れるすべての場合について全探索を行う．

基底

- Six-step FFTアルゴリズムでは，データ数Nを$N = N_1 \times N_2$と分解して，N_1組のN_2点FFTと，N_2組のN_1点FFTをそれぞれ計算する．
- ここで，N_1とN_2を基底と呼ぶことにする．
- N_1とN_2の値は，$N = N_1 \times N_2$を満たしていれば任意に選ぶことができる．
- ただし，MPIプロセス数をPとした場合，$N_1, N_2 \geq P$を満たす必要がある．
- 通常は$N_1 \approx N_2 \approx \sqrt{N}$となるように$N_1$と$N_2$を選ぶことが多いが，性能が最も高くなるように$N_1$と$N_2$を選ぶことができる．
- なお，データ数Nが2のべき乗になる場合には，すべてのN_1とN_2の組み合わせを試行したとしても，探索空間は$\log_2(\sqrt{N}/P)$となる．

ブロックサイズ

- Six-step FFTアルゴリズムにおいては行列の転置が必要になるが，この行列の転置はキャッシュブロッキングを行うことで効率よく実行できることが知られている.
- その際，最適なブロックサイズNBは，問題サイズおよびキャッシュサイズ等に依存する.
- 今回の実装では，ブロックサイズNBを2のべき乗に限定して4，8，16，32，64と変化させている.

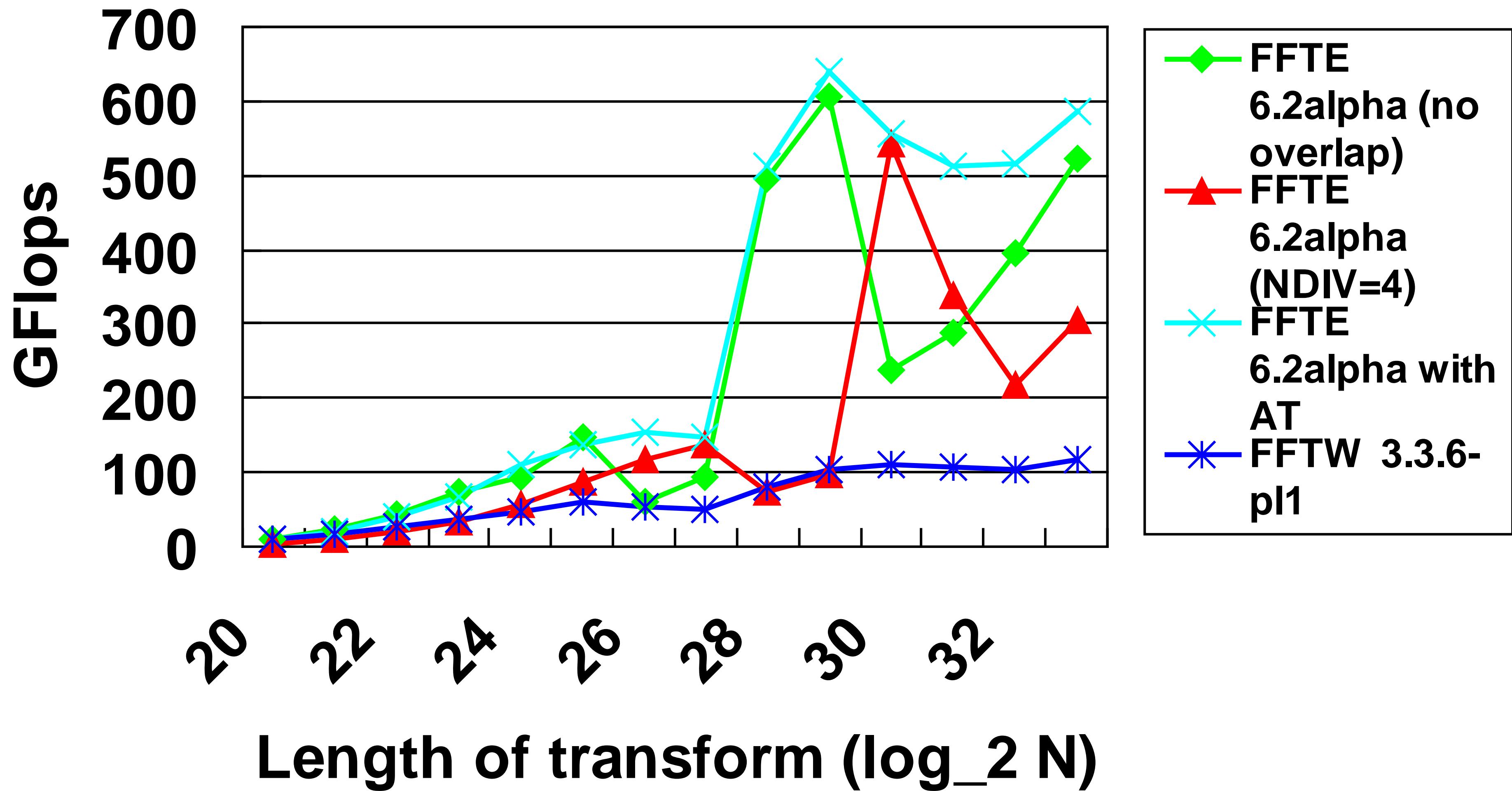
性能評価

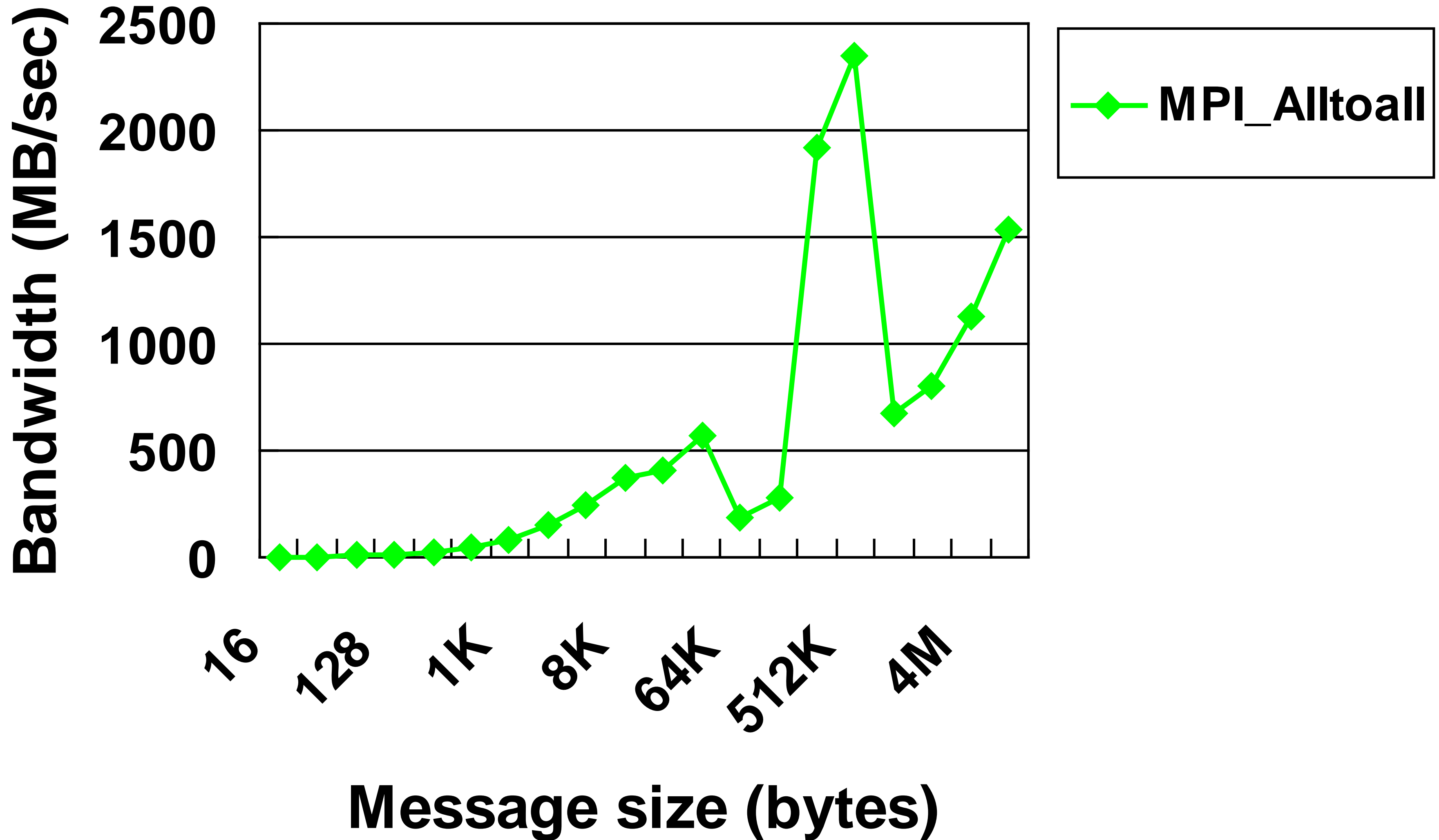
- 性能評価にあたっては，並列FFTライブラリであるFFTE 6.2alpha（http://www.ffte.jp/）と，自動チューニング手法をFFTE 6.2alphaに適用したもの，そしてFFTW 3.3.6-pl1との性能比較を行った．
- 順方向FFTを1～128MPIプロセス（1ノードあたり1MPIプロセス）で連続10回実行し，その平均の経過時間を測定した．
- Xeon Phiクラスタとして，最先端共同HPC基盤施設（JCAHPC）に設置されているOakforest-PACS（8208ノード）の128ノードを用いた．
  - CPU：Intel Xeon Phi 7250 (68 cores, Knights Landing 1.4 GHz)
  - インターコネクト：Intel Omni-Path Architecture
  - コンパイラ：Intel Fortran compiler 17.0.1.132 (FFTE)
Intel C compiler 17.0.1.132 (FFTW)
  - コンパイラオプション：“-O3 -xMIC-AVX512 -qopenmp”
  - MPIライブラリ：Intel MPI 2017.1.132
  - flat/quadrantモードおよびMCDRAMのみを用い，KMP_AFFINITY=compact
  - 各ノードあたりのスレッド数は64に設定

並列一次元FFTにおける自動チューニングの結果（Oakforest-PACS，128ノード）

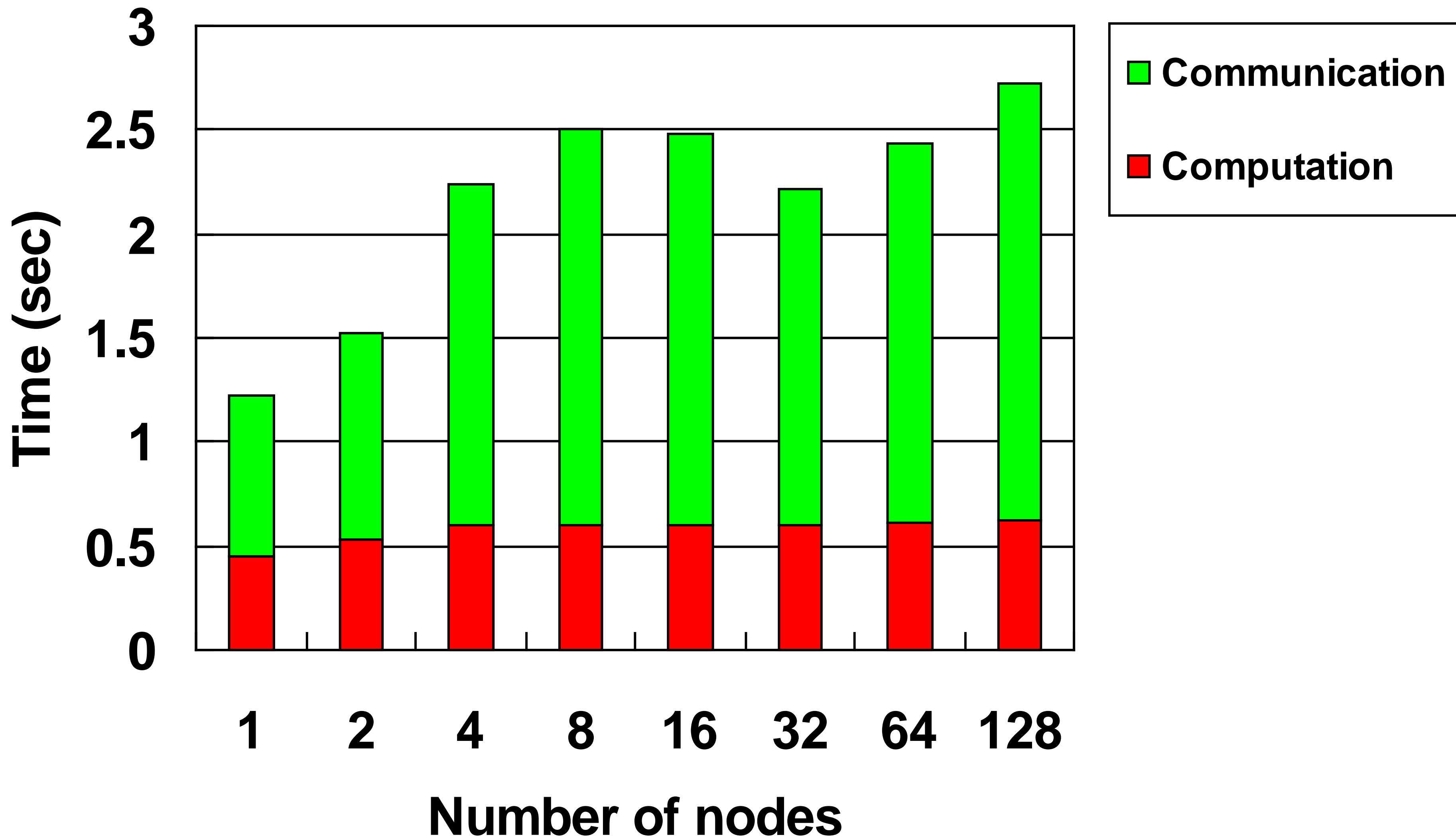
	FFTE 6.2alpha					FFTE 6.2alpha with AT				
N	N1	N2	NB	NDIV	GFlops	N1	N2	NB	NDIV	GFlops
16M	4K	4K	32	4	57.8	4K	4K	32	1	109.4
64M	8K	8K	32	4	116.9	8K	8K	16	2	154.8
256M	16K	16K	32	4	73.3	8K	32K	16	1	513.8
1G	32K	32K	32	4	541.7	32K	32K	64	4	554.9
4G	64K	64K	32	4	217.0	64K	64K	32	16	516.5

並列一次元FFTの性能
（Oakforest-PACS，128ノード）




全対全通信の性能
（Oakforest-PACS, 128ノード）
Bandwidth (MB/sec)
2500
2000
1500
1000
500
0
16
128
1K
8K
64K
512K
4M
Message size (bytes)
MPI_Alltoall

FFTE 6.2alpha（no overlap）の実行時間の内訳
（Oakforest-PACS, N=2^26×ノード数）



二次元分割を用いた並列三次元FFTアルゴリズム

背景

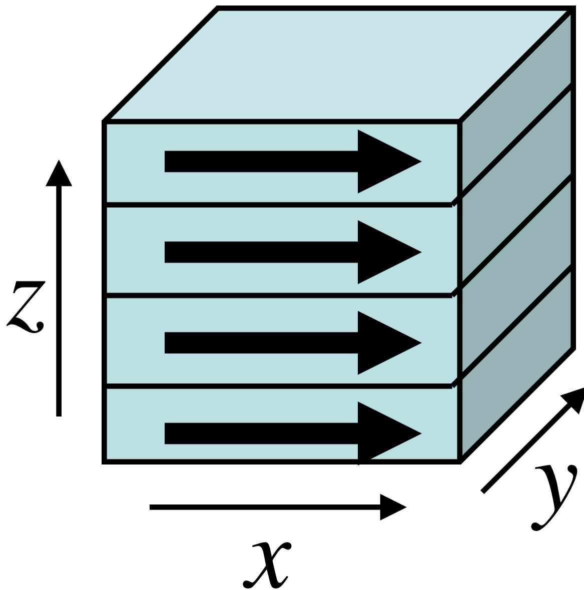
- 2017年11月のTop500リストにおいて，10システムが10PFlopsの大台を突破している.
  - Sunway TaihuLight（Sumway SW26010 260C 1.45GHz）：93.014 PFlops（10,649,600 Cores）
  - Tianhe-2（Intel Xeon E5-2692 12C 2.2GHz，Intel Xeon Phi 31S1P）：33.862 PFlops（3,120,000 Cores）
  - Piz Daint（Xeon E5-2690v3 12C 2.6GHz，NVIDIA Tesla P100）：19.590 Pflops（361,760 Cores）
- 第1位のSunway TaihuLightは，1000万コアを超える規模になっている.

方針

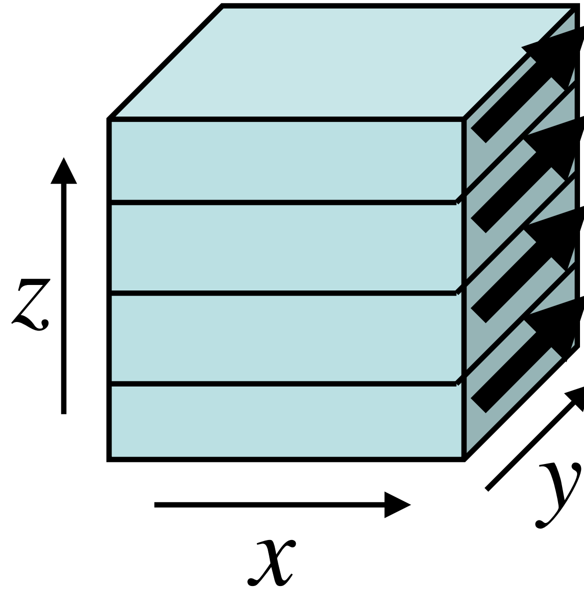
- 並列三次元FFTにおける典型的な配列の分散方法
  - 三次元（x, y, z方向）のうちの一次元のみ（例えばz方向）のみを分割して配列を格納.
  - MPIプロセスが1万個の場合，z方向のデータ数が1万点以上でなければならず，三次元FFTの問題サイズに制約.
- x, y, z方向に三次元分割する方法が提案されている [Eleftheriou et al. '05, Fang et al. '07].
  - 各方向のFFTを行う都度，全対全通信が必要.
- 二次元分割を行うことで全対全通信の回数を減らしつつ，比較的少ないデータ数でも高いスケーラビリティを得る.

z方向に一次元ブロック分割した場合の並列三次元FFT

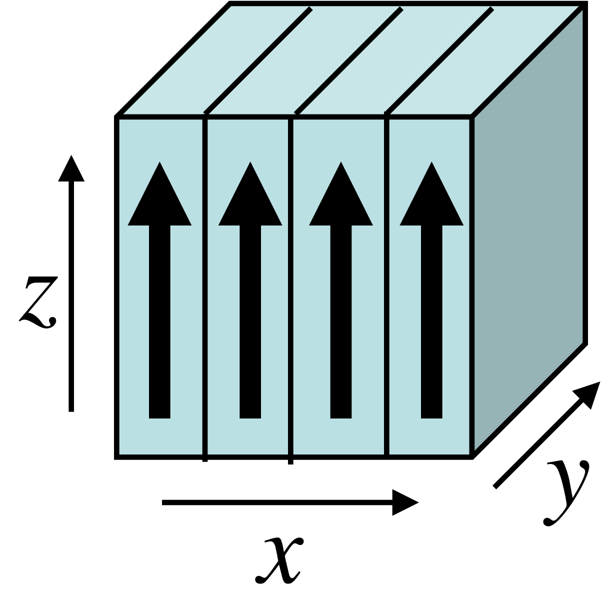
1. x方向FFT



2. y方向FFT



3. z方向FFT



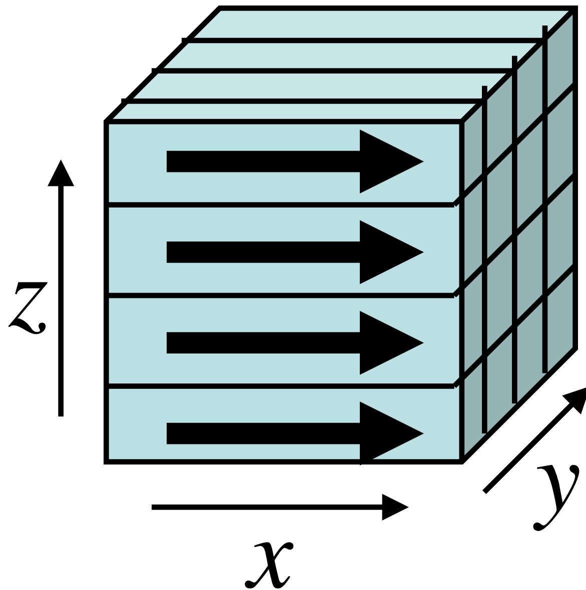
各プロセッサでslab形状に分割

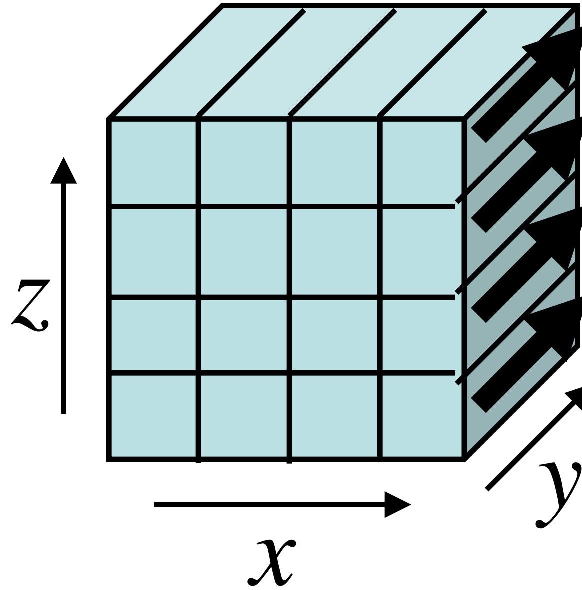
三次元FFTの超並列化

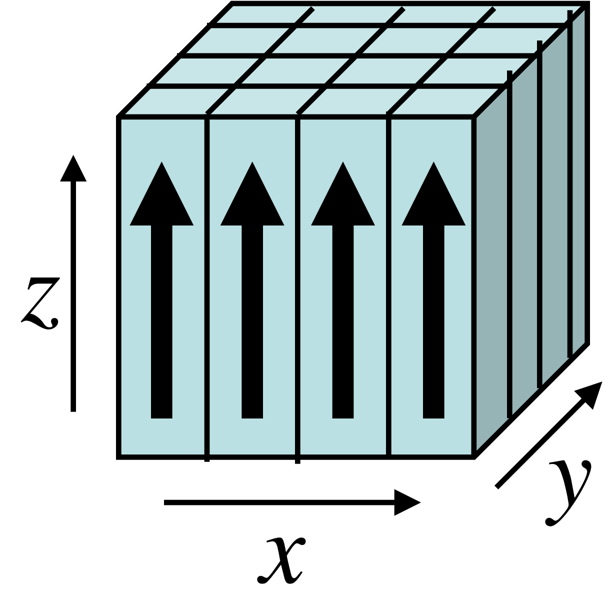
- 並列アプリケーションプログラムのいくつかにおいては，三次元FFTが律速になっている.
- x，y，zのうちz方向のみに一次元分割した場合，超並列化は不可能.
  - 1,024×1,024×1,024点FFTを2,048プロセスで分割できない（1,024プロセスまでは分割可能）
- y，zの二次元分割で対応する.
  - 1,024×1,024×1,024点FFTが1,048,576（=1,024×1,024）プロセスまで分割可能になる.

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

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

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

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 $N/(P^2 Q)$ 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

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

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

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,

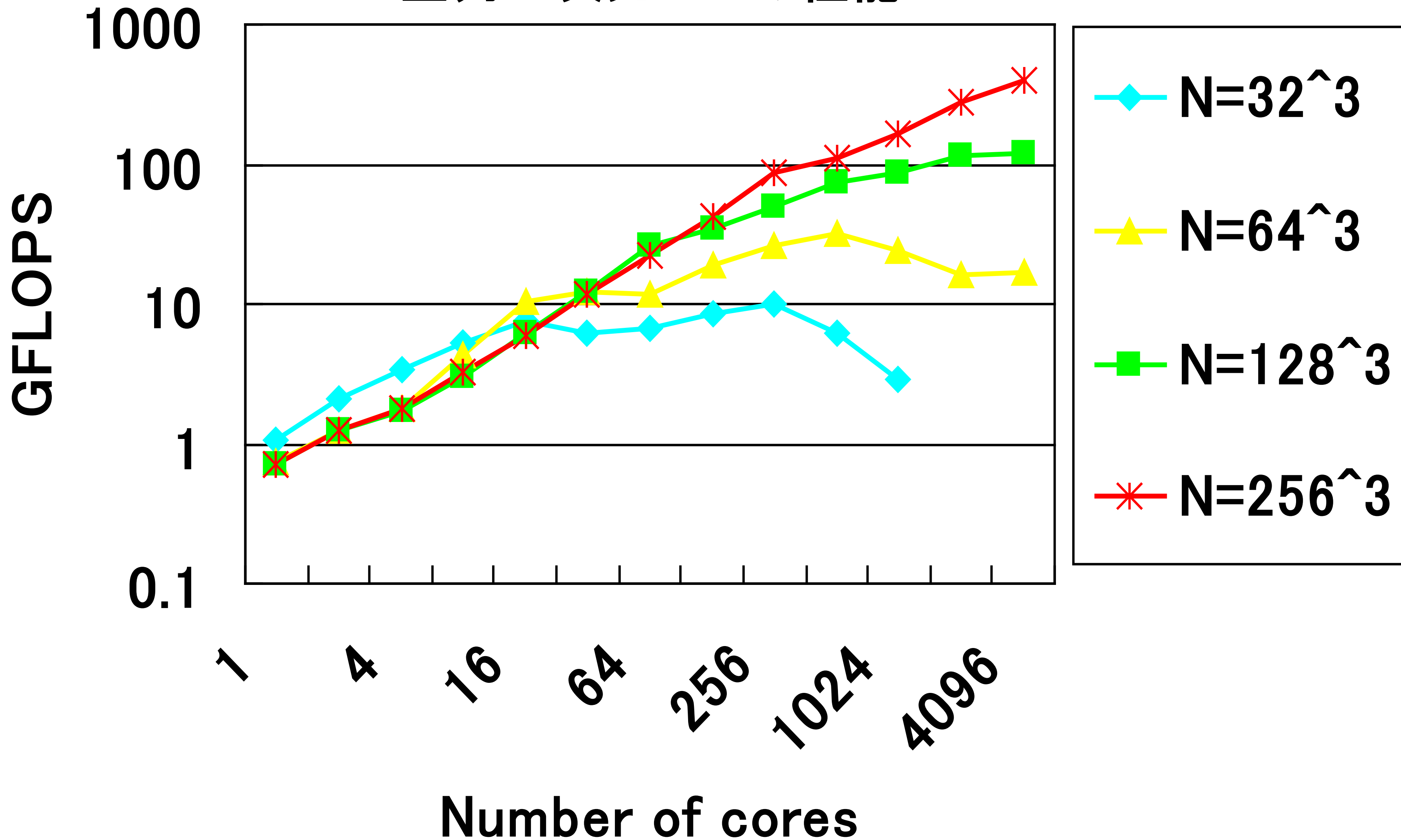
$$N<\frac{(LW\cdot PQ)(PQ-P-Q)}{16}$$

- 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

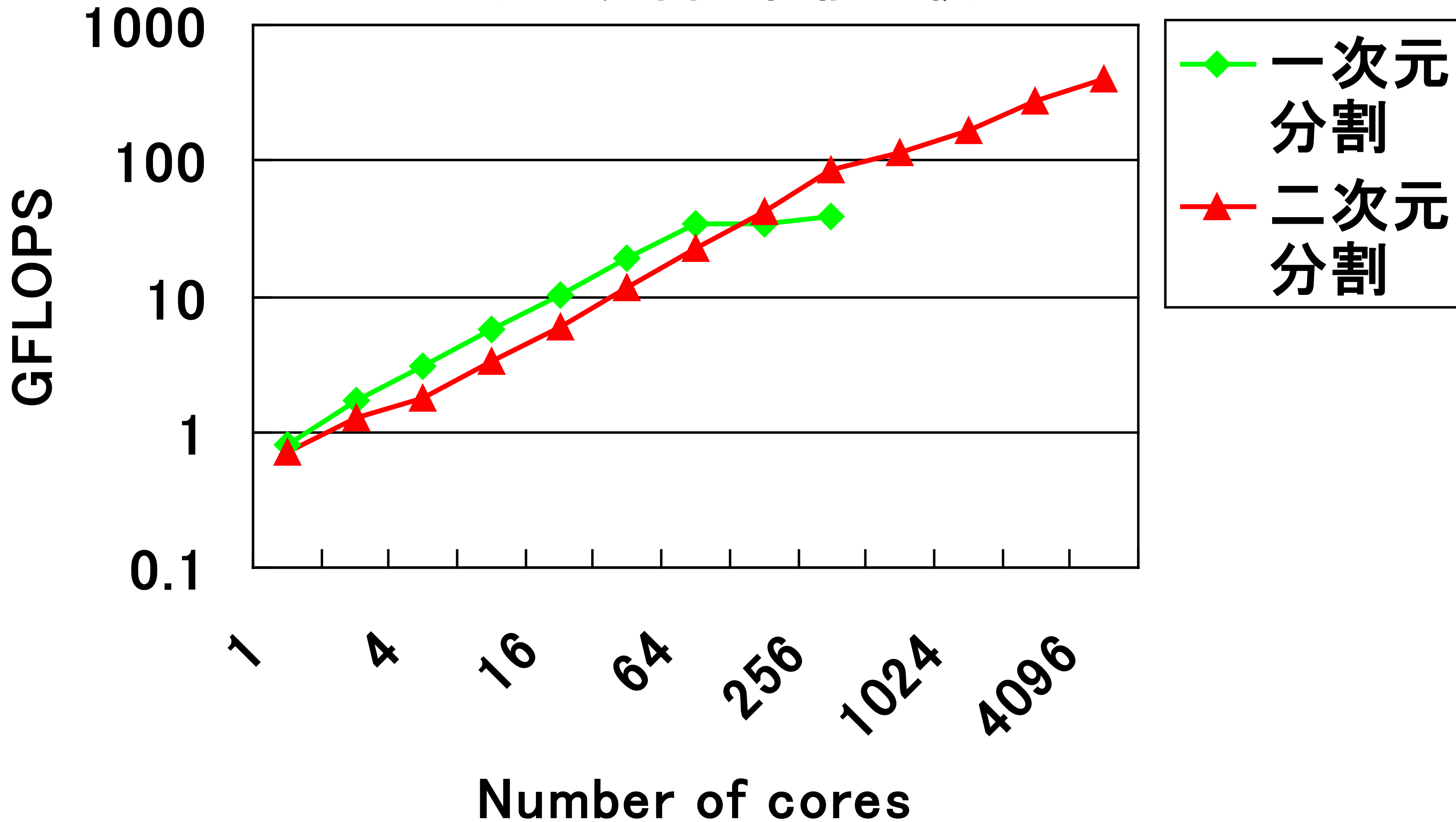
- 性能評価にあたっては，二次元分割を行った並列三次元FFTと，一次元分割を行った並列三次元FFTの性能比較を行った．
- Strong Scalingとして $N = 32^3, 64^3, 128^3, 256^3$ 点の順方向FFTを1～4,096MPIプロセスで連続10回実行し，その平均の経過時間を測定した．
- 評価環境
  - T2K筑波システムの256ノード（4,096コア）を使用
  - flat MPI（1core当たり1MPIプロセス）
  - MPIライブラリ：MVAPICH 1.2.0
  - Intel Fortran Compiler 10.1
  - コンパイルオ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

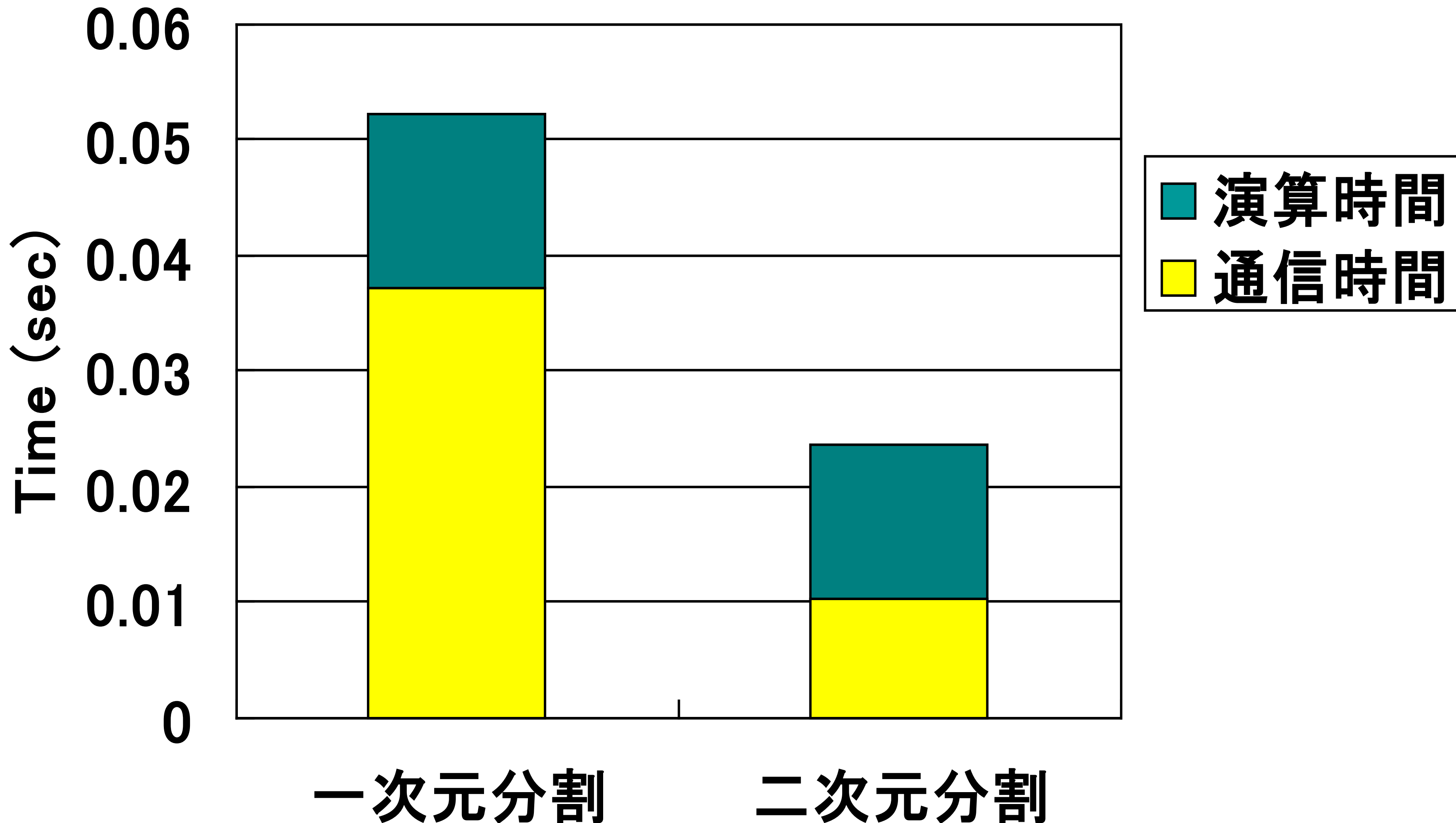


考察（1/2）

- $N = 32^3$ 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
1000
100
10
1
0.1
GFLOPS
1
4
16
64
256
1024
4096
Number of cores
一次元分割
二次元分割

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

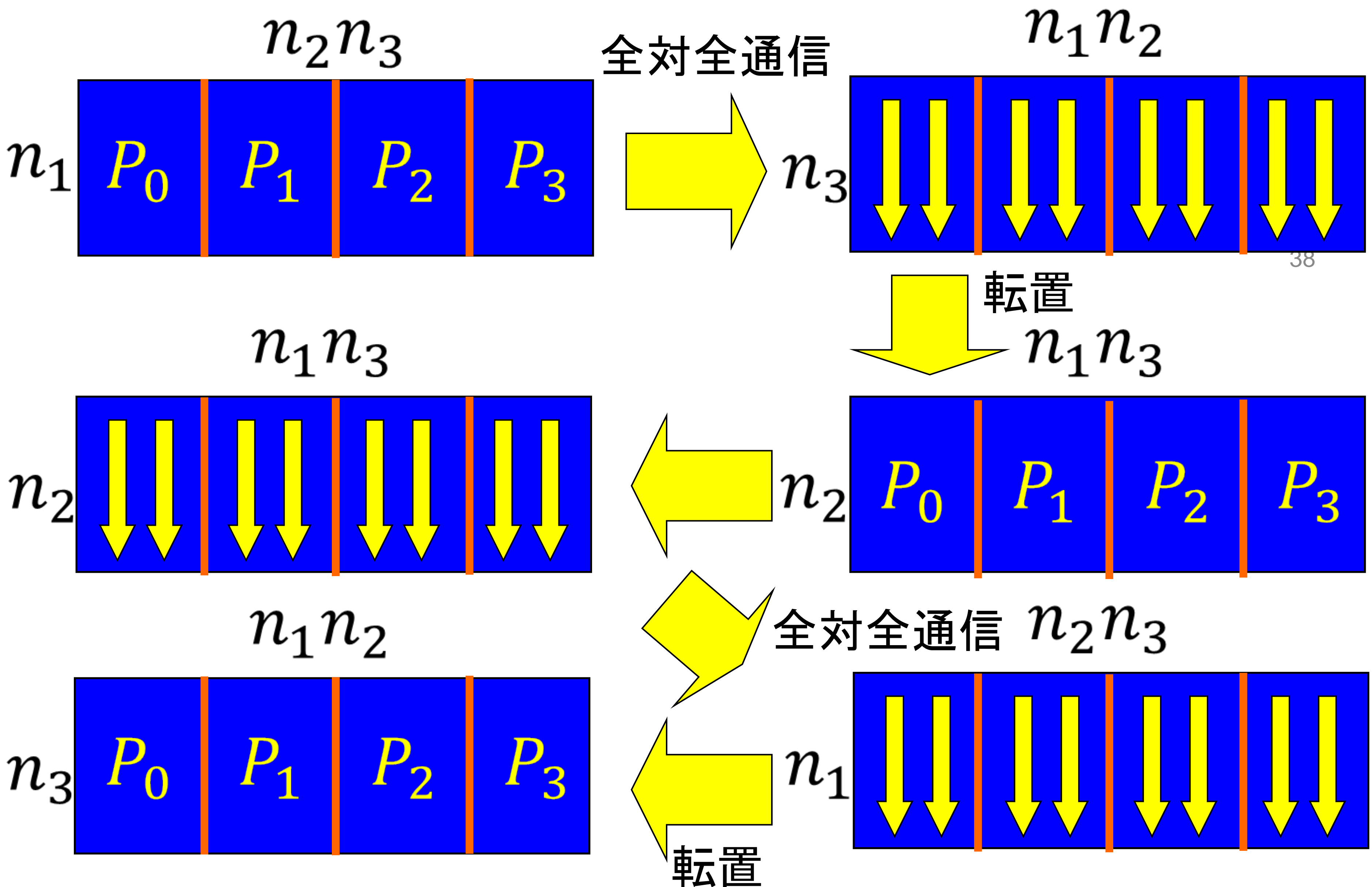
背景

- 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，2017年11月のTOP500リストではNVIDIA Tesla P100 GPUを搭載したPiz Daintが第3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

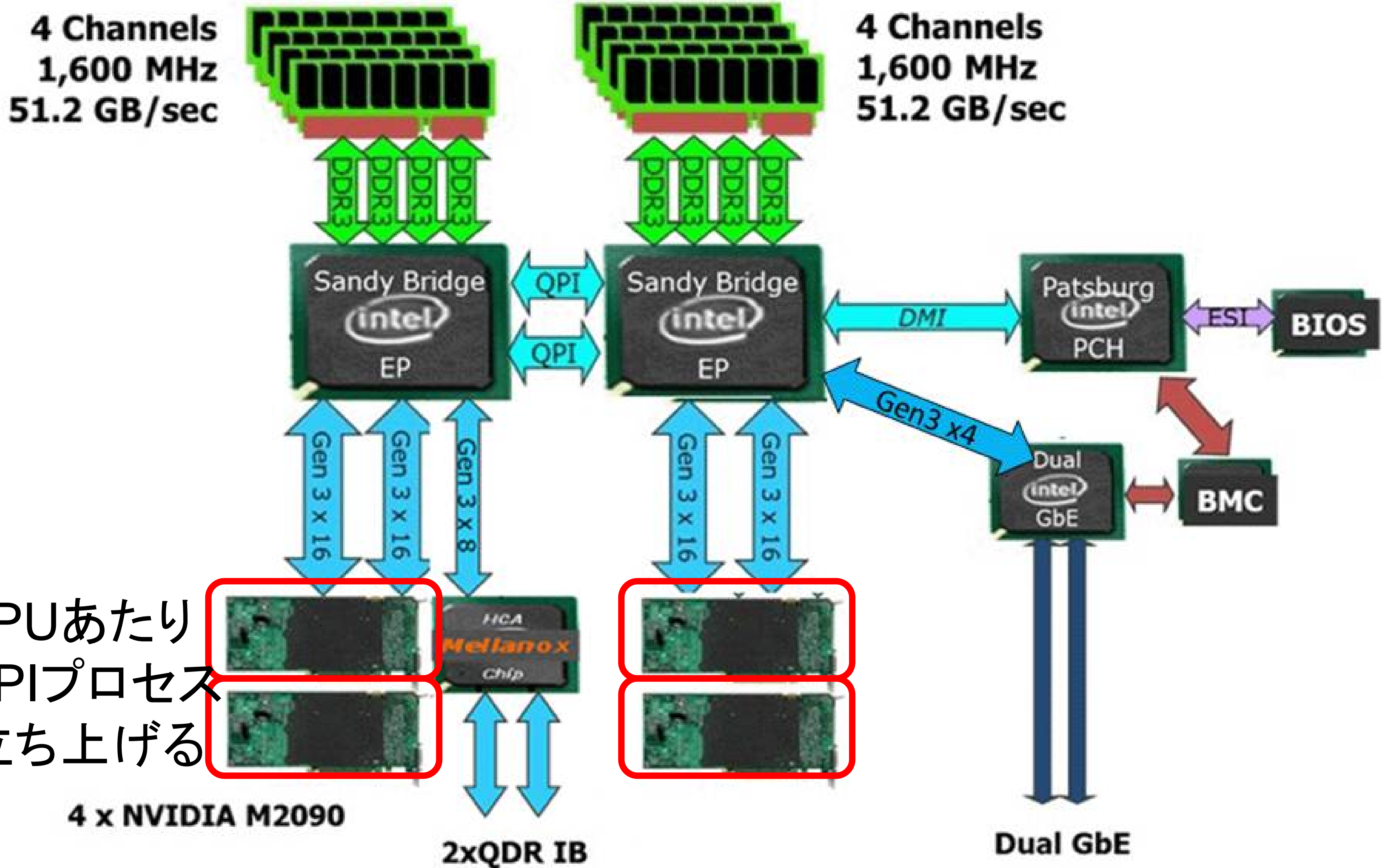
```
At Sender:
MPI_Send(s_device, size, …);

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.

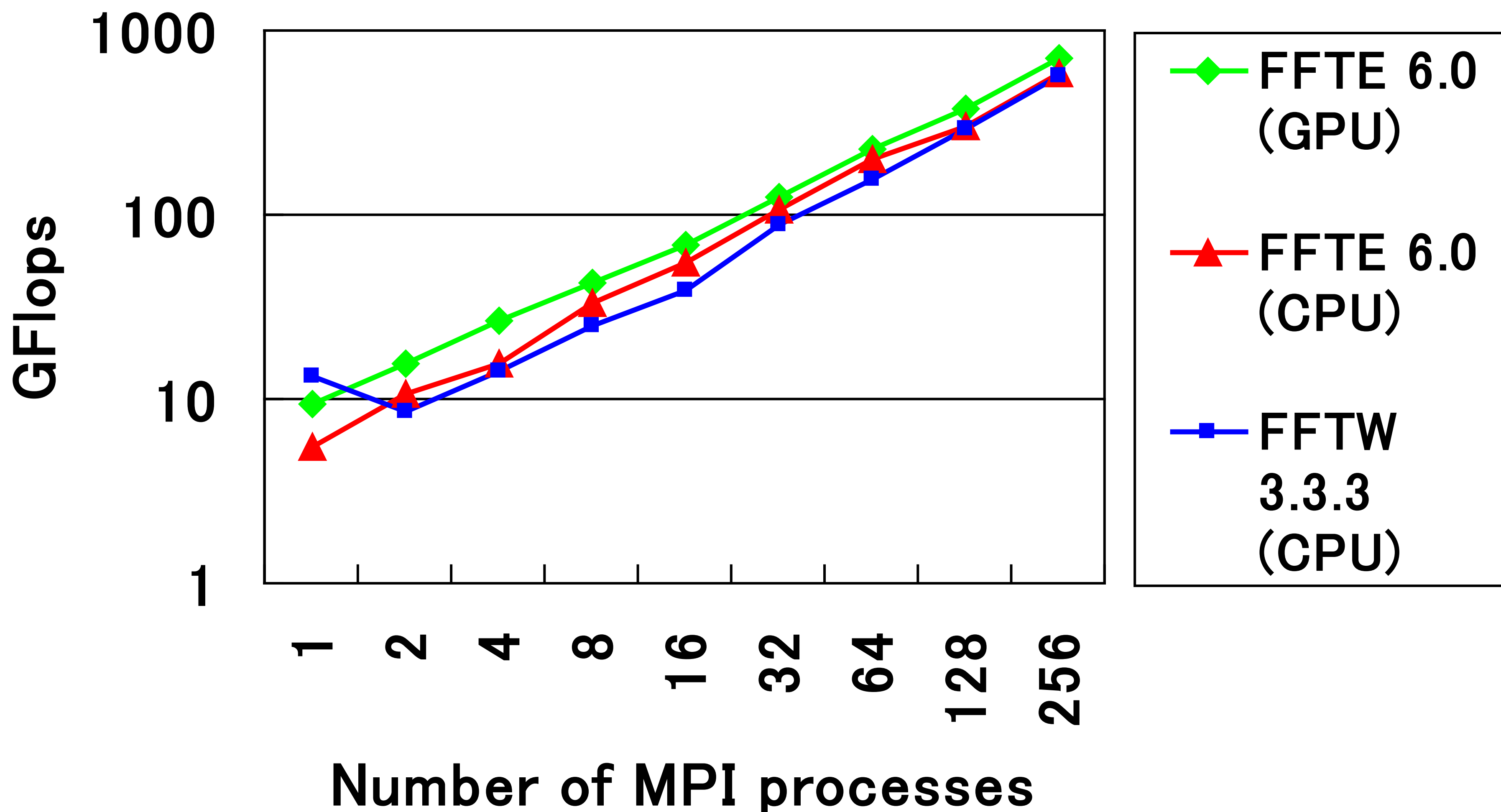
性能評価

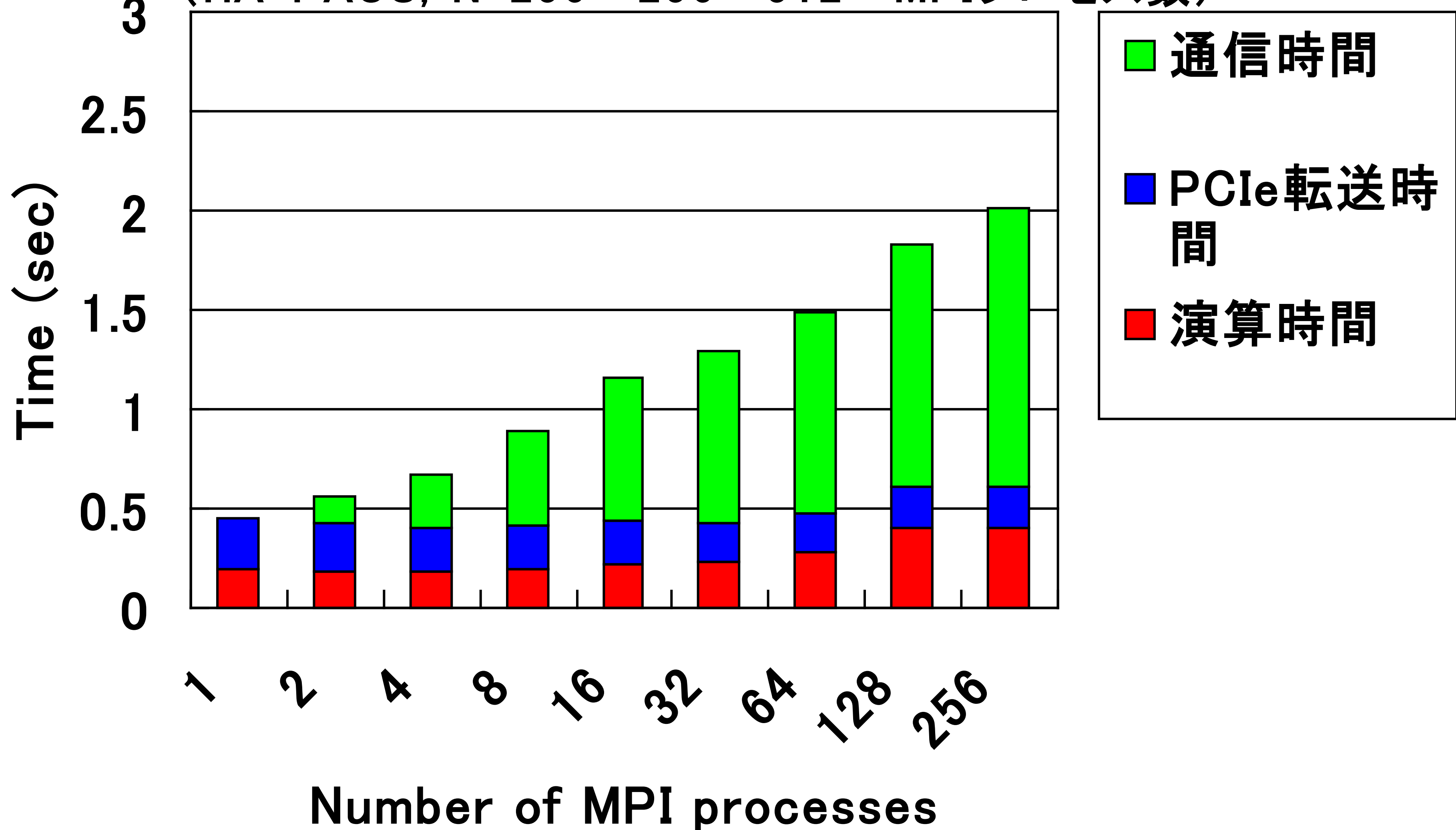
- 性能評価にあたっては，以下のFFTライブラリについて性能比較を行った．
  - FFTE 6.0（http://www.ffte.jp/，GPUを使用）
  - FFTE 6.0（http://www.ffte.jp/，CPUを使用）
  - FFTW 3.3.3（http://www.fftw.org/，CPUを使用）
- 順方向FFTを1～256MPIプロセス（1ノードあたり4MPIプロセス）で連続10回実行し，その平均の経過時間を測定した．
- HA-PACSベースクラスタ（268ノード，4288コア，1072GPU）のうち，1～64ノードを使用した．
  - 各ノードにIntel Xeon 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）

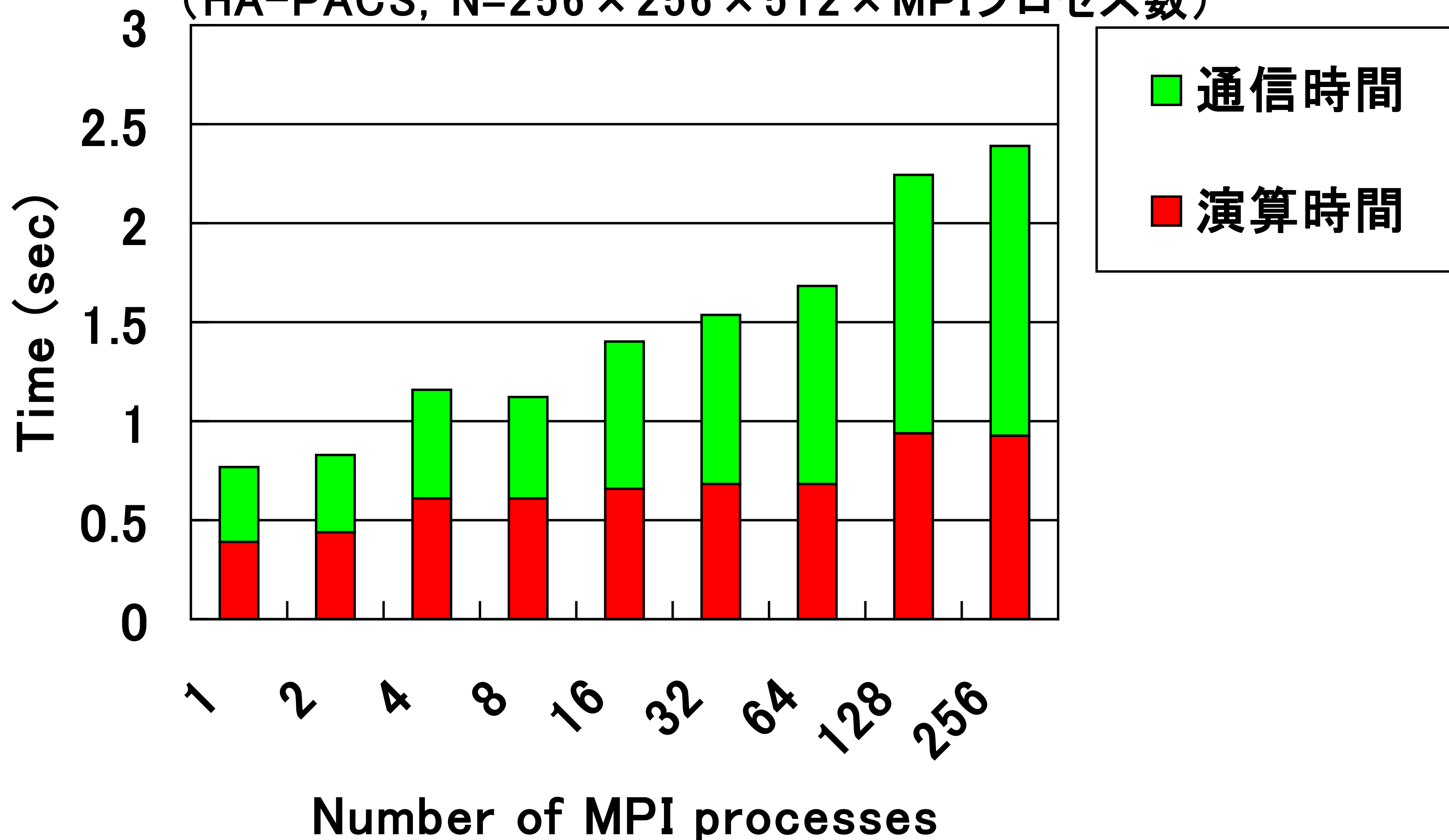
HA-PACSベースクラスタのノード構成

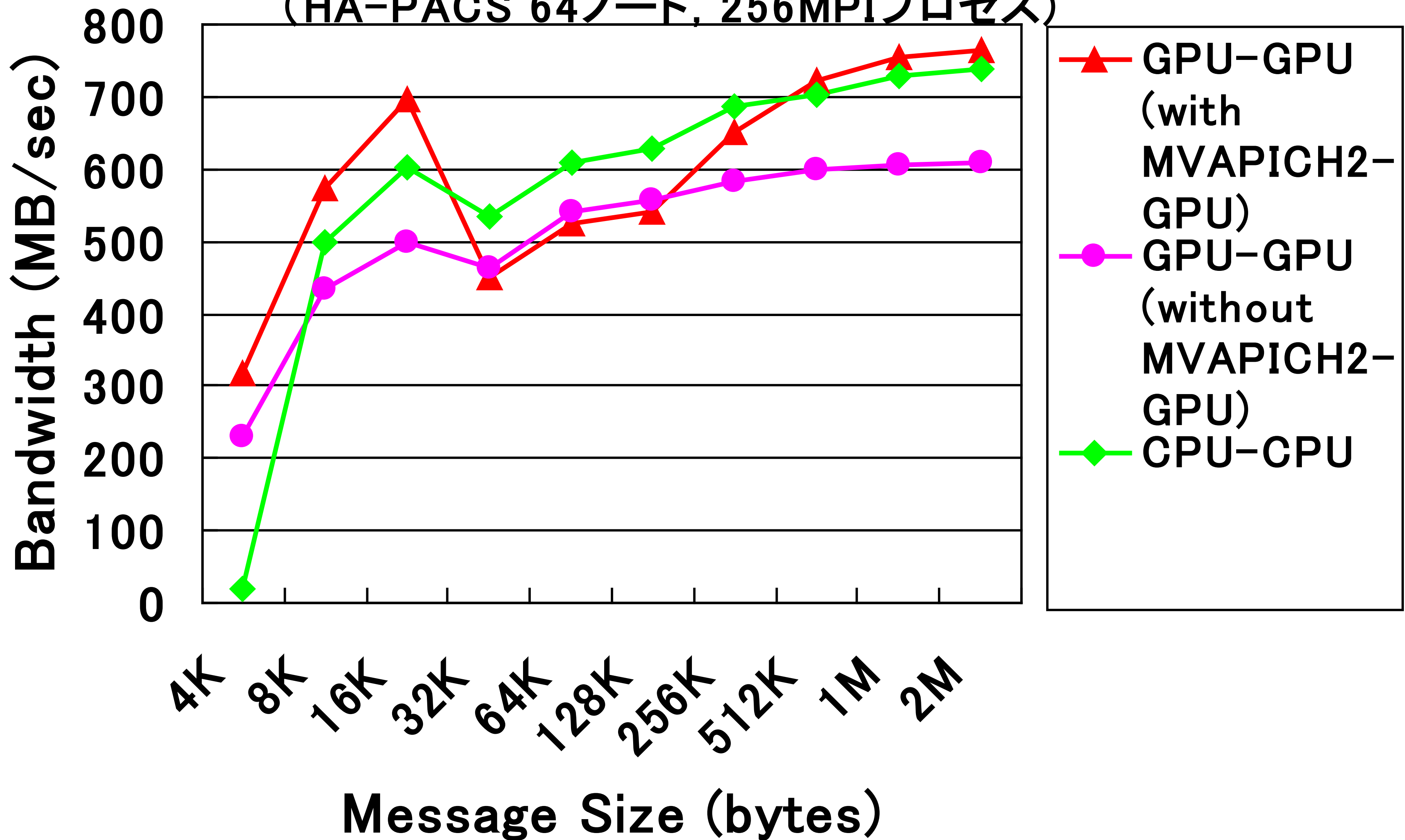


並列三次元FFTの性能




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）
通信時間
PCIe転送時間
演算時間
Time (sec)
Number of MPI processes
0
0.5
1
1.5
2
2.5
3
1
2
4
8
16
32
64
128
256


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）
通信時間
演算時間
Time (sec)
3
2.5
2
1.5
1
0.5
0
1
2
4
8
16
32
64
128
256
Number of MPI processes


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，通信の隠蔽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.