第7回計算科学技術特論B(2026)
大規模系での高速フーリエ変換2

筑波大学
高橋大介

2026年5月28日（木）13:00－14:30

主催：高度情報科学技術研究機構(RIST)
　　次世代HPC・AI研究開発支援センター(HAIRDESC)
共催：東京大学物性研究所
後援：理化学研究所計算科学研究センター、
　　計算物質科学人材育成コンソーシアム(PCoMS)

講義内容

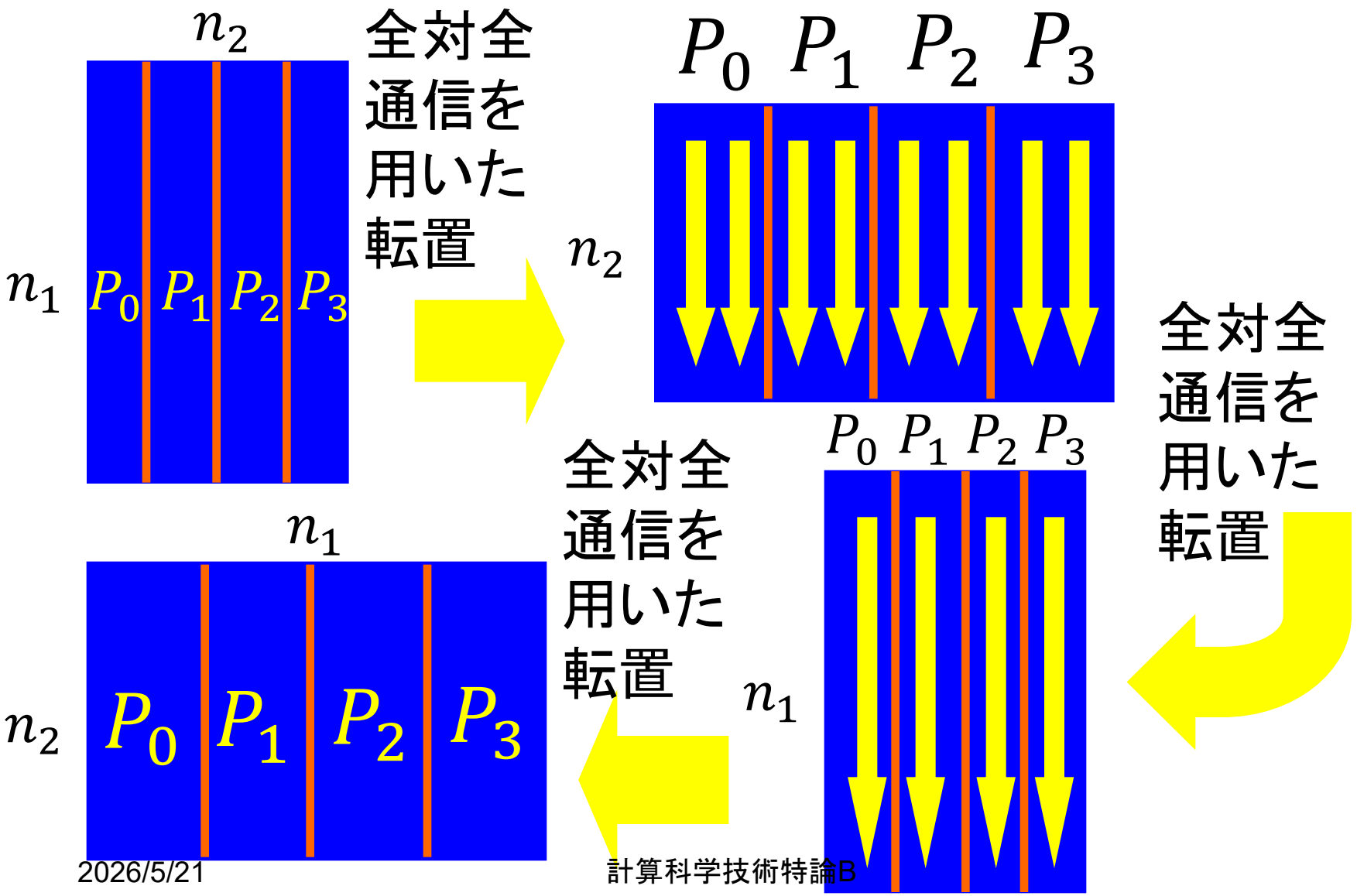
- メニーコアクラスタ上の並列FFTにおける通信隠蔽の自動チューニング
- GPUクラスタにおける二次元分割を用いた並列三次元FFT

メニーコアクラスタ上の並列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メニーコアクラスタ上で性能評価を行う.

並列Six-Step FFTアルゴリズム



メニーコア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
!$OMP END MASTER
!$OMP DO SCHEDULE(DYNAMIC)
      DO I=1,N
        オーバーラップする演算
      END DO
!$OMP DO
      DO I=1,N
        通信結果を使用する演算
      END DO
!$OMP END PARALLEL
      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

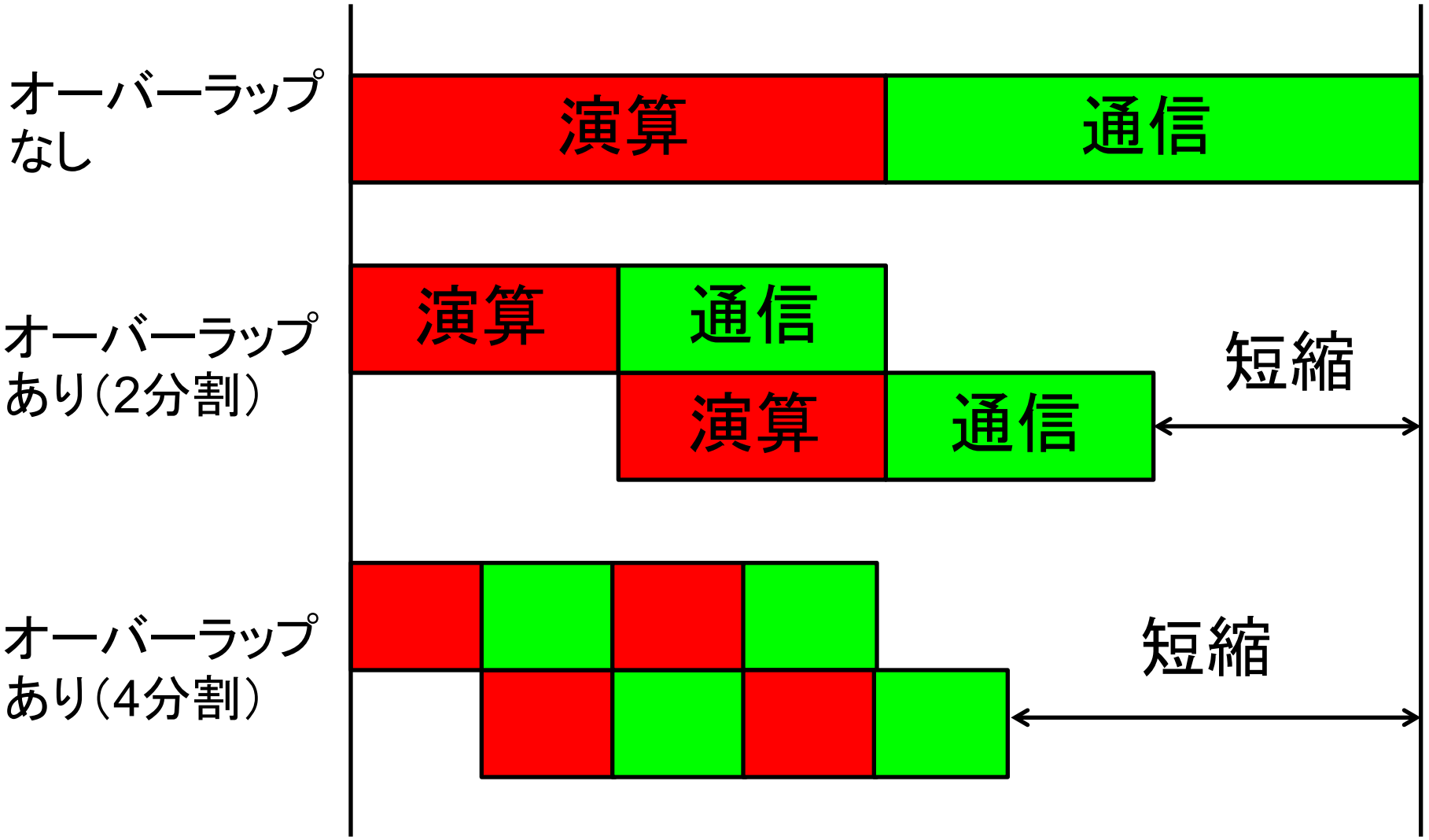
←マスタースレッドで通信を実行
（通信が早く終われば演算に参加可能）

←同期なし

←マスタースレッド以外のスレッドで
演算を実行

←暗黙の同期

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す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$\log_2\left(\sqrt{N}/P\right)$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.

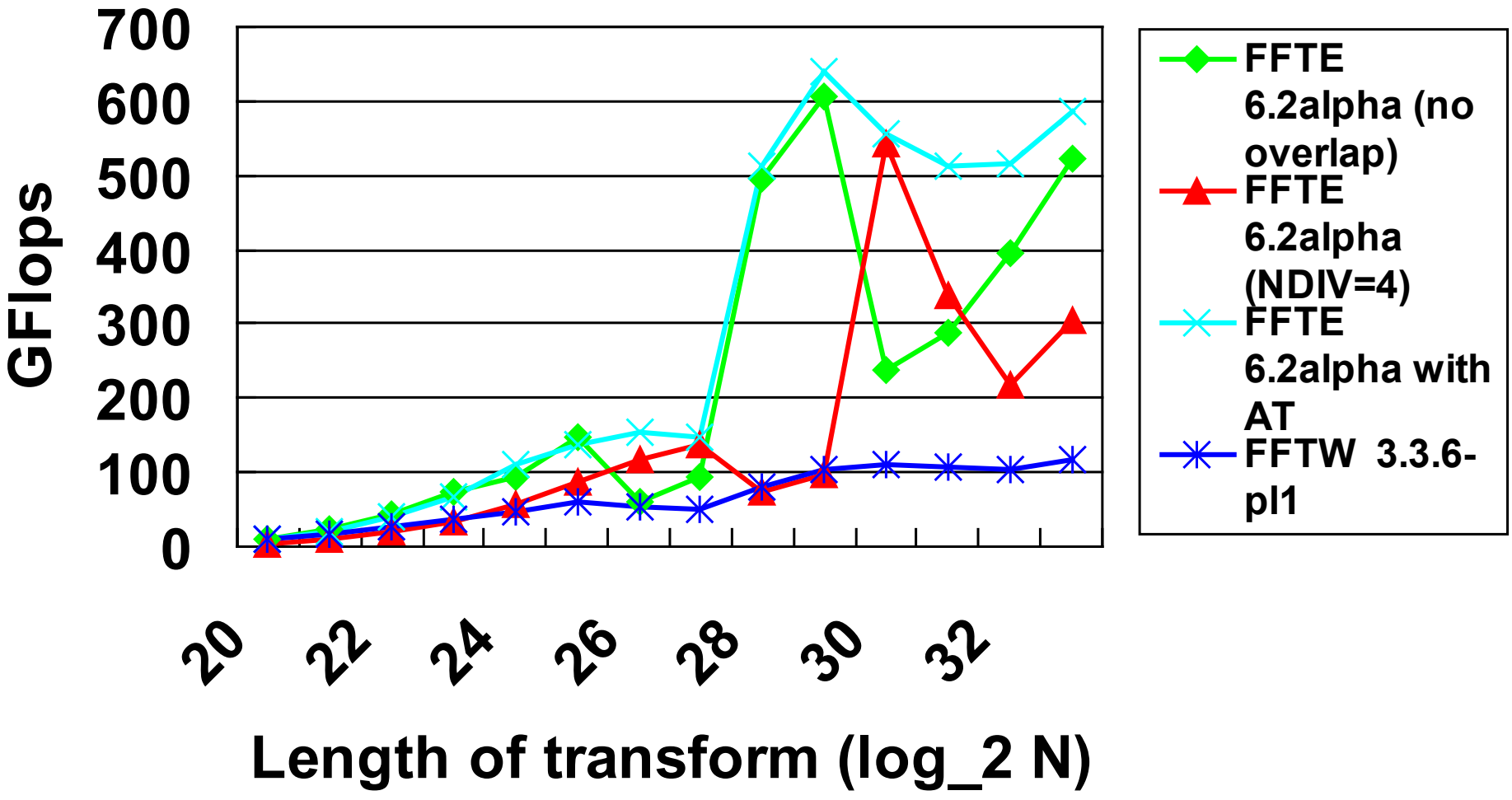
性能評価

- 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.fft.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

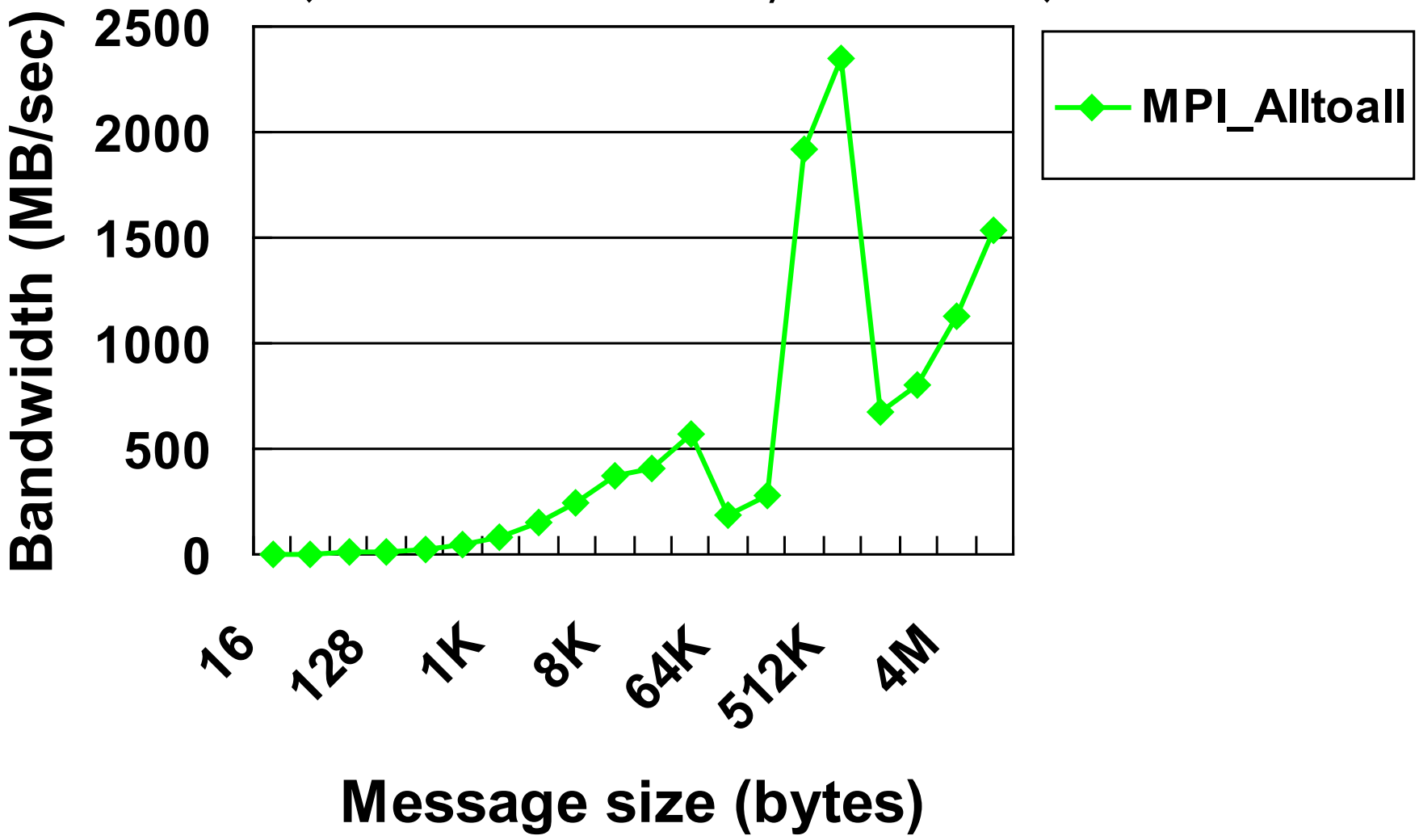
並列一次元FFTにおける自動チューニングの結果（Oakforest-PACS，128ノード）

	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

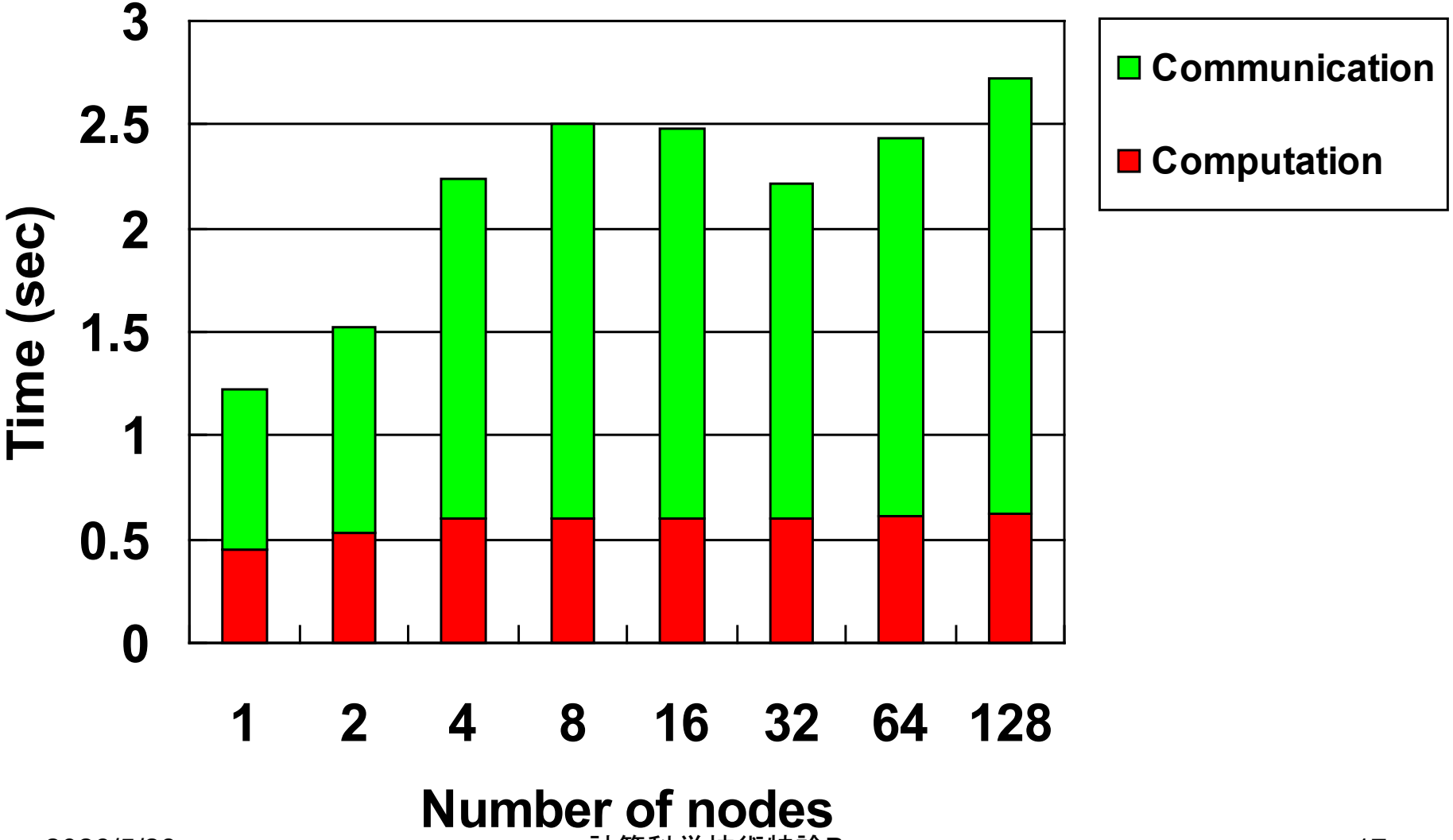
並列一次元FFTの性能
（Oakforest-PACS，128ノード）



全対全通信の性能
（Oakforest-PACS，128ノード）



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



GPUクラスタにおける二次元分割を用いた並列三次元FFT

背景

- 2025年11月のTop500リストにおいて，4システムが1EFlopsの大台を突破している.
  - El Capitan: 1,809.0 PFlops (11,340,000 Cores)
  - Frontier: 1,353.0 PFlops (9,066,176 Cores)
  - Aurora: 1,012.0 PFlops (9,264,128 Cores)
  - JUPITER Booster: 1,000.0 PFlops (4,801,344 Cores)
- 第1位のEl Capitan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.

三次元FFT

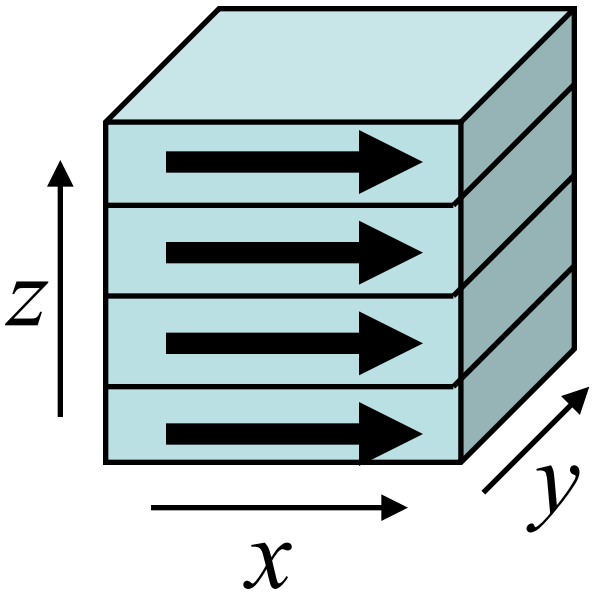
- 三次元離散フーリエ変換（DFT）の定義

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

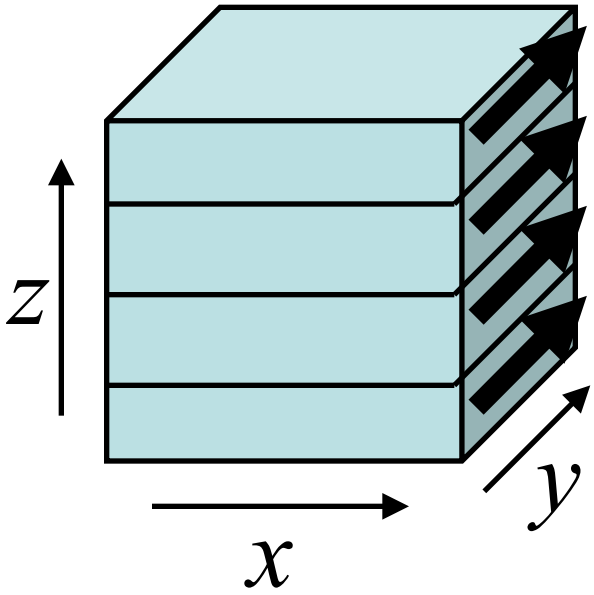
$$0 \leq k_r \leq n_r - 1, \omega_{n_r} = e^{-2\pi i / n_r}$$

z方向に一次元ブロック分割した場合の並列三次元FFT

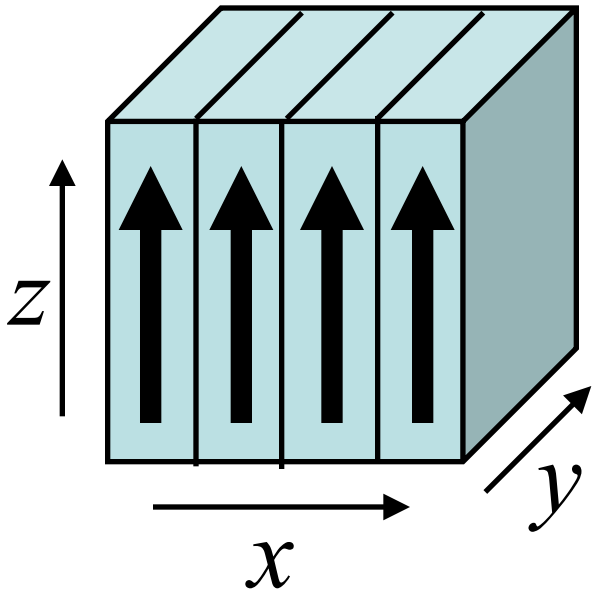
1. x方向FFT



2. y方向FFT



3. z方向FFT



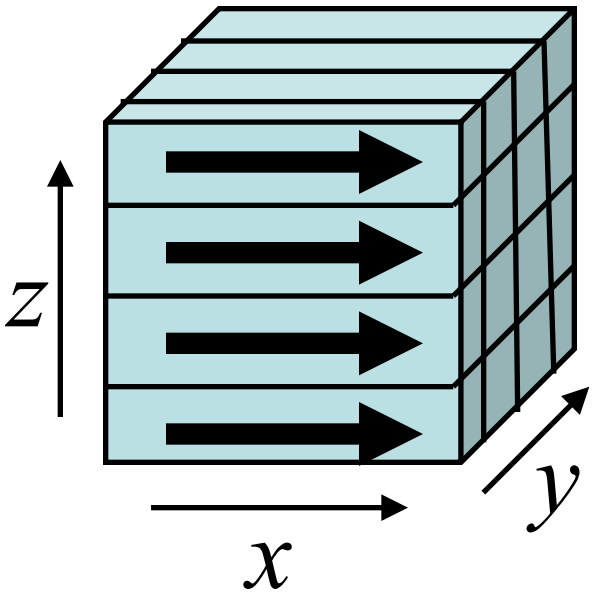
各MPIプロセスでslab形状に分割

三次元FFTの超並列化

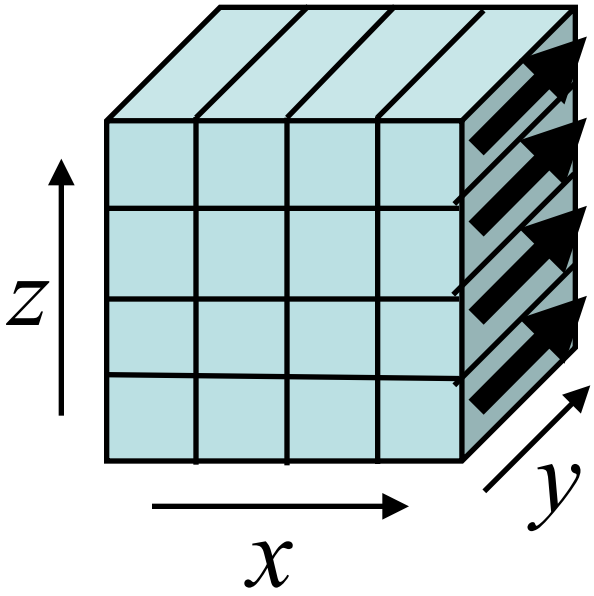
- 並列アプリケーションプログラムのいくつかにおいては，三次元FFTが律速になっている.
- x，y，zのうちz方向のみに一次元分割した場合，超並列化は不可能.
  - 1,024×1,024×1,024点FFTを2,048プロセスで分割できない（1,024プロセスまでは分割可能）
- y，zの二次元分割で対応する.
  - 1,024×1,024×1,024点FFTが1,048,576（=1,024×1,024）プロセスまで分割可能になる.

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

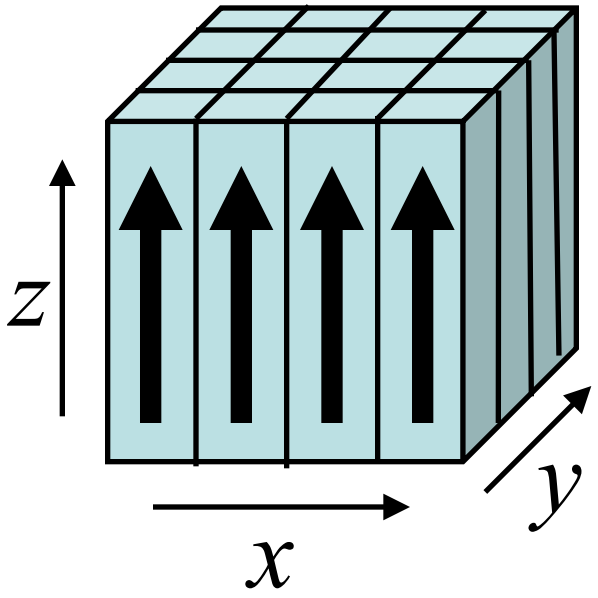
1. x方向FFT



2. y方向FFT



3. z方向FFT



各MPIプロセスでpencil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MPIプロセスにおいて，
  - P個のMPIプロセス間で全対全通信をQ個
  - Q個のMPIプロセス間で全対全通信をP個

  行う必要がある.
- MPI_Comm_Split()を用いてMPI_COMM_WORLDをy方向（P個のMPIプロセス）とz方向（Q個のMPIプロセス間）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$N = N_1 \times N_2 \times N_3$, MPIプロセス数を$P \times Q$, MPIプロセス間通信性能を$W$（byte/sec）, 通信レイテンシを$L$（sec）とする.
- 各MPIプロセスは$N/(PQ)^2$個の倍精度複素数データを自分以外の$PQ - 1$個のMPIプロセスに送ることになる.
- 一次元分割の場合の通信時間は

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

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

二次元分割の場合の通信時間

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

$$T_{2\mathrm{dim}} \approx (P-1)\left(L+\frac{16N}{P^2Q\cdot W}\right)+(Q-1)\left(L+\frac{16N}{PQ^2\cdot W}\right)$$
$$\approx (P+Q)\cdot L+\frac{32N}{PQ\cdot W}\ (\mathrm{sec})$$

通信時間の比較

- 一次元分割の場合の通信時間

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

- 二次元分割の場合の通信時間

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

- 二つの式を比較すると，全MPIプロセス数$P \times Q$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.

GPUクラスタにおける並列三次元FFT

- GPUクラスタにおいて並列三次元FFTを行う際には，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．
- GPU上のメモリをMPIにより転送する場合，通常のMPIでは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．

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, …);

- GPUDirect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

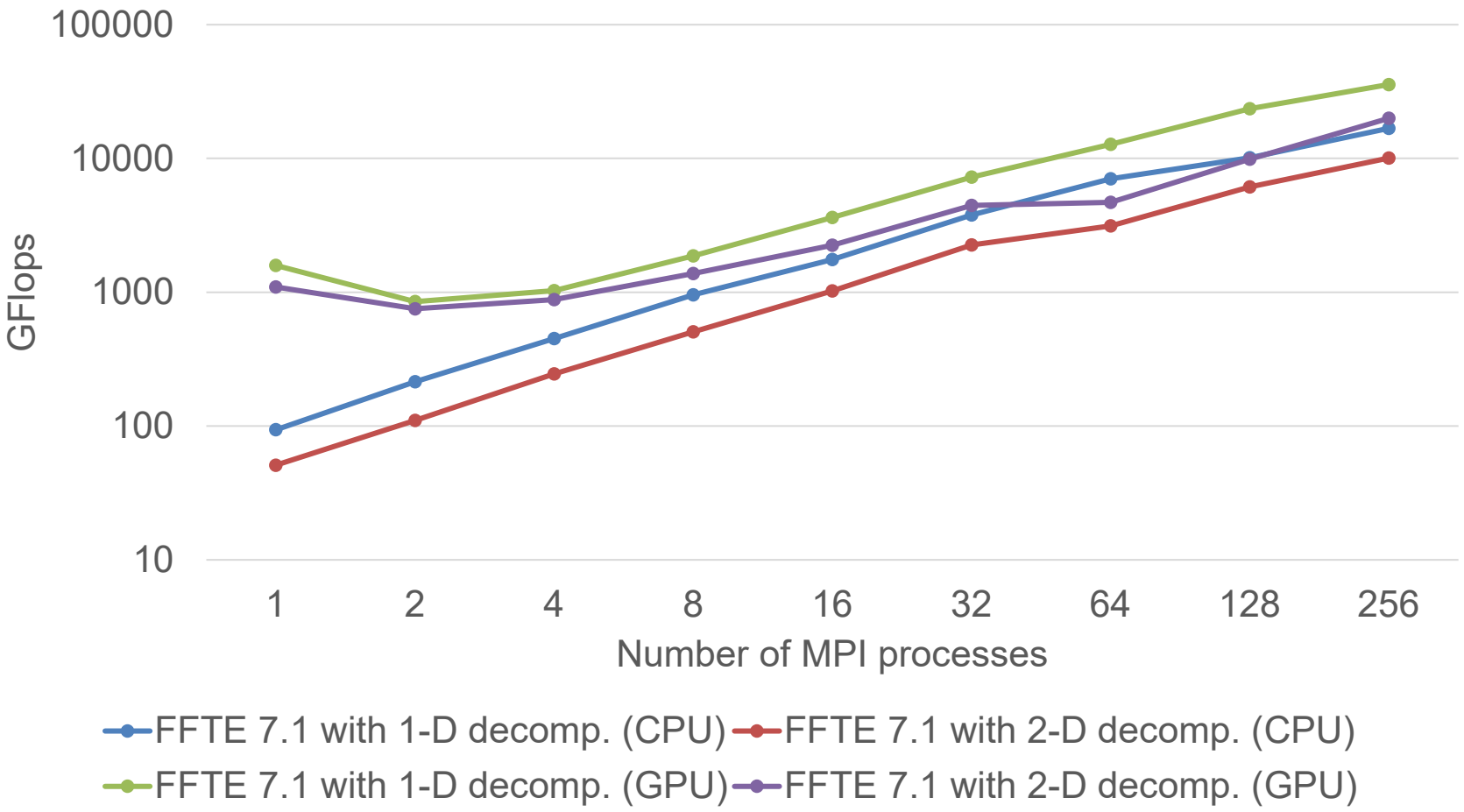
- 性能評価にあたっては，
  - 一次元分割を行った並列三次元FFTのCPU実装
  - 二次元分割を行った並列三次元FFTのCPU実装
  - 一次元分割を行った並列三次元FFTのGPU実装
  - 二次元分割を行った並列三次元FFTのGPU実装

  の性能を比較した.
- Weak scaling（$N = 512 \times 512 \times 512 \times$ MPI processes）およびstrong scaling（$N = 512 \times 512 \times 512$）の性能を測定した.
- 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計測した.

評価環境

- GPU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Miyabi-G（1120ノード）のうち256ノードを用いた.
  - 1120 nodes，Peak 78.8 PFlops
  - CPU：Nvidia Grace（72コア，3.0 GHz，3.456 TFlops）
  - GPU：NVIDIA H100（66.9 TFlops in FP64 Tensor Core）
  - インターコネクト：InfiniBand NDR
  - コンパイラ：NVIDIA HPC Compilers 24.9
  - MPIライブラリ：OpenMPI 4.1.7a1
  - コンパイルオプション：“-fast -mp” (for CPU)
    “-fast -cuda -gpu=cc90” (for GPU)
- 各ノードあたりのMPIプロセス数は1に設定した.
- GPU実装では，NVIDIAのcuFFTライブラリを呼び出してmulticolumn FFTを実行している.

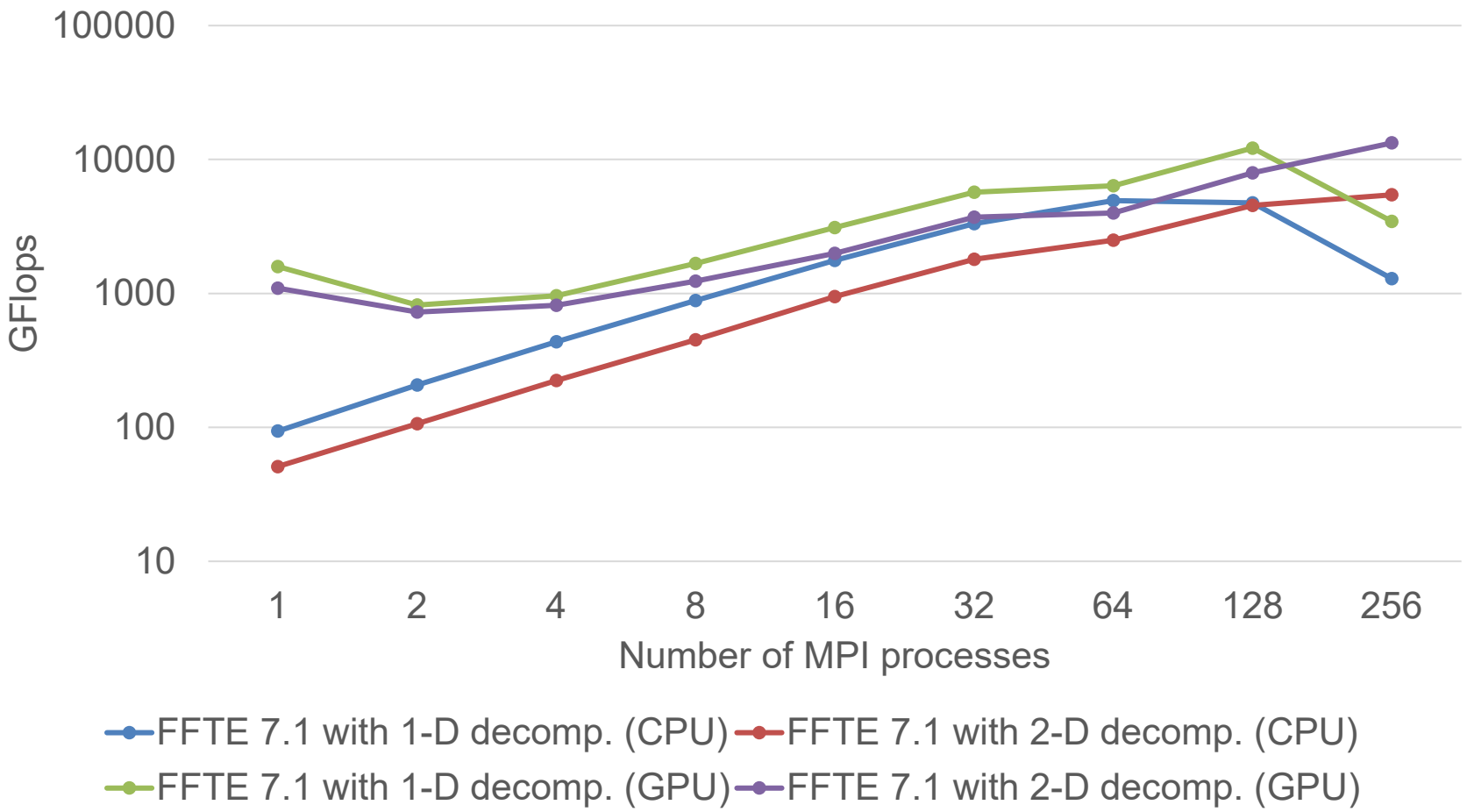
並列三次元FFTの性能
（$N = 512 \times 512 \times 512 \times$ MPI processes）



考察(1/2)

- Weak scalingの場合，CPU実装およびGPU実装のいずれにおいても，一次元分割の性能は二次元分割よりも優れている.
- これは，一次元分割の総通信量が二次元分割の半分であるのが理由である.
- Weak scalingの場合，全対全通信のメッセージサイズが大きいことから，通信時間が総通信量にほぼ比例する.
- MPIプロセス数が増加するに従って全対全通信のメッセージサイズが小さくなり，通信バンド幅が低下する.
- したがって，CPU実装とGPU実装の性能差は，MPIプロセス数が増加するに従って小さくなる.

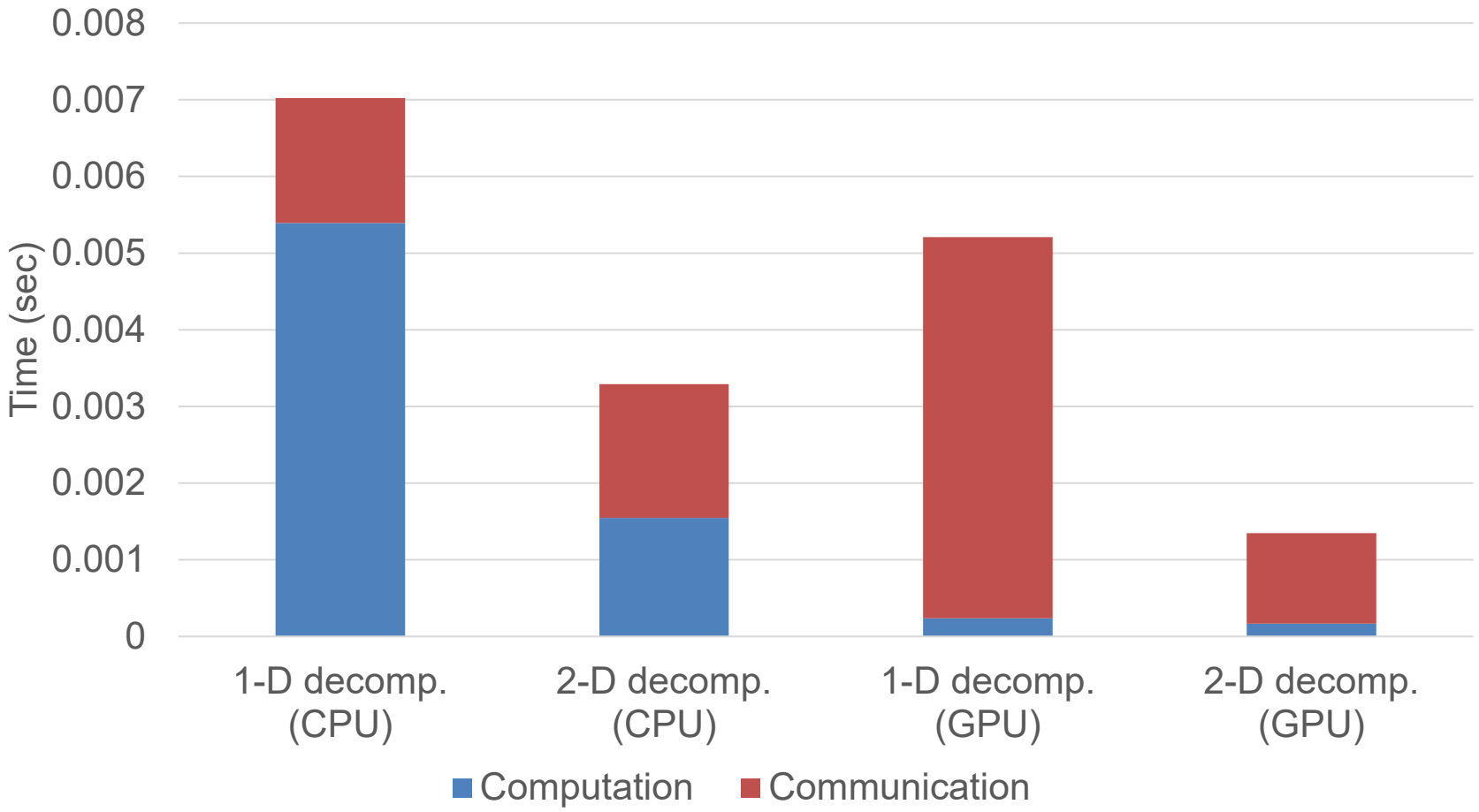
並列三次元FFTの性能
$(N = 512 \times 512 \times 512)$



考察(2/2)

- Strong scalingの場合，MPIプロセス数が128以下ではCPUおよびGPUの実装のいずれにおいても，一次元分割の性能が二次元分割よりも優れている.
- 一方，MPIプロセス数が256の場合，CPUおよびGPUの実装のいずれにおいても，二次元分割が一次元分割よりも高速である.
- これは，512^3点FFTにおいて，一次元分割では全対全通信のメッセージサイズがわずか32 KBであるのに対し，2次元分解ではそれぞれ1 MBおよび2 MBになるためである.

並列三次元FFTの実行時間の内訳
（$N = 512^3$, 256 MPI processes）



まとめ

- 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.
- 並列三次元FFTにおいて，二次元分割を用いることにより，MPIプロセス数が増加した場合でも通信時間を短縮し，性能を効果的に向上させることができることを示した.