Vol.2018-HPC-163 No.29
2018/3/2

メニーコアプロセッサにおける多軸分割を用いた3次元FFTの性能評価

青木 聖陽[1]　今村 俊幸[2]　横川 三津夫[1]　廣田 悠輔[2]

概要：3 次元 FFT は流体シミュレーションで用いられるアルゴリズムであり，高解像度のシミュレーションの実行のためには高い並列計算性能が求められる．一般的に分散メモリ並列計算環境下で多次元 FFT を実施するためには，分割された軸方向に関するデータを連続もしくは同じプロセス上に保持し直さなくてはならない．3 次元 FFT の場合，1 軸分割 (Slub)，2 軸分割 (Pencil)，3 軸分割 (Cube) の空間分割が存在し，分割法によって通信コストが変化する．プロセス数が大きくなるにつれて全体の計算に対する通信にかかる時間の占める割合が大きくなるため，最適なデータの分割法を選択することが求められる．本稿では，通信コストのモデルから，1 軸分割，2 軸分割および通信方法の異なる 2 種類の 3 軸分割について通信コストを求め，どの分割方法が最適となるかを求める．また，分割方法の異なる 3 次元 FFT の京コンピュータと Oakforest-PACS を用いてその性能を評価する．

Aoki Masaaki[1]　Imamura Toshiyuki[2]　Yokokawa Mitsuo[1]　Hirota Yusuke[2]

1. はじめに

3 次元 FFT は流体シミュレーションで用いられるアルゴリズムであり，高い並列計算性能が求められる．2016 年には石原らが京を用いて格子サイズ $12,288^3$ の非圧縮性乱流のフーリエスペクトル法を用いた直接数値シミュレーション (DNS) を行っている [2]．本研究では，ポスト京において格子サイズ $16,384^3$ での乱流 DNS を目標とし，スペクトル法で用いられる 3 次元 FFT の性能改善を図りたい．分散メモリを用いて 3 次元 FFT の計算を行う場合，高い並列性を保持するためデータを複数次元方向について分割する．3 次元 FFT は各次元方向についての 1 次元 FFT によって計算されるが，1 プロセス内に 1 次元 FFT に必要なデータを集めるためにプロセス間の全対全通信が必要になる．データの分割法によってこの通信コストは変化する．プロセス数が大きくなるにつれて全体の計算に対する通信にかかる時間の占める割合が大きくなるため，最適なデータの分割法を選択することが求められる．また，ポスト京にはメニーコアプロセッサが搭載されるため，メニーコア向けの最適化についても考慮する必要がある．

本稿では，オープンソースの FFT プログラムである FFTE[6] の C 言語移植版 FFTE-C を用いて分割方法と通信の方法が異なる多軸分割の 3 次元 FFT を実装した．各多軸分割 FFT について，Oakforest-PACS と京を用いて性能を測定し，それぞれの計算機の通信コストのモデルから妥当性の評価を行う．本稿の構成は以下の通りである．2 章では，3 次元 FFT の多軸分割のアルゴリズムについて述べる．3 章では，ネットワークトポロジーごとの MPI_Alltoall の通信コストについて述べる．4 章では，多軸分割 FFT の通信コストについて述べる．5 章では，実装した多軸分割 3 次元 FFT の性能評価を行った結果を示す．6 章はまとめの章とする．

2. 3 次元 FFT の多軸分割

2.1 3 次元 FFT

FFT は離散フーリエ変換 (Discrete Fourier Transform; DFT) を高速に計算するアルゴリズムである [1]．データ長を n としたとき，DFT は以下で定義される．

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad \text{for } k = 0, \ldots, n-1 \tag{1}$$

ここで，$\omega_n = \mathrm{e}^{-\frac{2\pi i}{n}}$，$i = \sqrt{-1}$ である．また，3 次元 FFT は各次元方向についての 1 次元 FFT によって計算され，以下で定義される．

[1] 神戸大学大学院システム情報学研究科
Graduate School of System Informatics, Kobe University
[2] 理化学研究所計算科学研究機構
RIKEN Advanced Institute for Computational Science

$$Y(\beta_1,\beta_2,\beta_3) = \sum_{\alpha_1=0}^{n_x-1}\sum_{\alpha_2=0}^{n_y-1}\sum_{\alpha_3=0}^{n_z-1}\omega_{n_1}^{\beta_1\alpha_1}\omega_{n_2}^{\beta_2\alpha_2}\omega_{n_3}^{\beta_3\alpha_3}X(\alpha_1,\alpha_2,\alpha_3) \quad (2)$$

FFT の計算は，キャッシュを有効に利用することが性能向上において重要であると知られている．よって，1 次元 FFT の計算に用いるデータがメモリ上に連続に配置されるようにデータ再分散を行う．本稿では，3 つの軸の方向についての FFT の計算が終了した時点で 3 次元 FFT の計算が終了したとみなし，変換前のデータの並びと一致させるための操作は FFT の計算に含まないものとする．

一般的に分散メモリ並列計算環境下で多次元 FFT を実施するためには，分割された軸方向に関するデータを連続もしくは同じプロセス上に保持し直さなくてはならない．この操作はデータ再分散とも呼ばれており，複数プロセス間での集団的通信を行う必要がある．プロセスの並びが空間グリッドに対して規則的であるとき，データ再分散は All-to-All Personalized Communication，MPI で定義するところの `MPI_Alltoall` で実現できる (以下 All-to-All と記述する)．

3 次元 FFT の場合，1 軸分割 (Slub)，2 軸分割 (Pencil)，3 軸分割 (Cube) の空間分割とともに一つもしくは複数の分割軸に所属するプロセスでの All-to-All 操作による再分散を行う．

2.2 1 軸分割（Slub 分解）

1 軸分割は，1 次元方向にのみデータを分割する最も単純な分割方法であり，FFTW[7] などいくつかの FFT ライブラリで採用されている．入力データの分割方向を z 軸方向としたときの 1 軸分割の FFT における転置操作の概要を**図 1** に示す．

z 軸方向にデータを分割した場合，y 軸方向のデータ連続化はプロセス内での転置操作のみで行えるため，必要な通信は 1 回のみとなる．1 軸分割は通信の回数も少なく実装が容易であるが，分割する次元方向のデータ数以上の並列化を行えず，高い並列性が得られない．よって，より高い並列性を得るためには，複数次元方向の分割が必要となる．

2.3 2 軸分割（Pencil 分解）

2 次元方向にデータを分割する 2 軸分割は，2 次元方向のデータ数までの並列化を行うことができ，1 軸分割より高い並列性を持つ．この分割方法は FFTE[6] や 2decomp[8] などの FFT ライブラリで採用されている．入力データの分割方向を y 軸および z 軸方向としたときの 2 軸分割の FFT における転置操作の概要を**図 2** に示す．

入力データの形状は $n_x \times n_y \times n_z$ とし，y 軸方向および z 軸方向の分割数はそれぞれ p_y, p_z とする．ただし，n_x は p_y で，n_y は p_y および p_z で，n_z は p_z で割り切れるものとする．alltoall_y は p_y プロセス間の通信，alltoall_z は p_z プロセス間の通信となる．

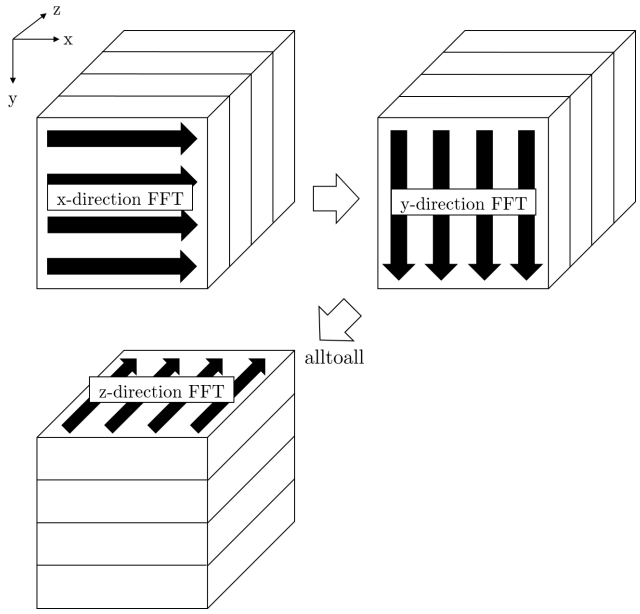

図 1 1 軸分割の FFT における転置操作の概要
Fig. 1 Overview of transposition operations in 1D decomposition FFT

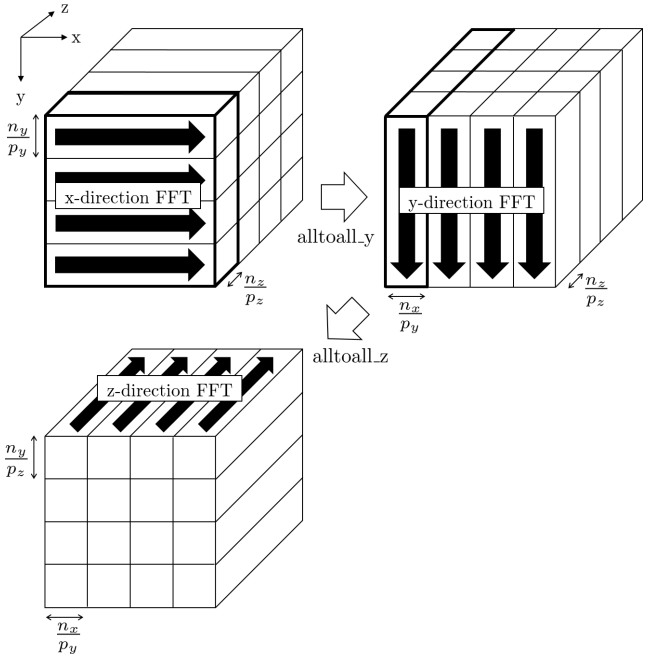

図 2 2 軸分割の FFT における転置操作の概要
Fig. 2 Overview of transposition operations in 2D decomposition FFT

2.4 3 軸分割（Cube 分解）

3 次元方向にデータを分割する 3 軸分割は，FFT の並列性は 2 軸分割と変わらないが，2 軸分割よりも 1 回の通信に関わるプロセス数が減少する．今回は，単純な実装の 5 回の通信を行う 1d_Alltoall 方式と，Jung らの手法 [3] を

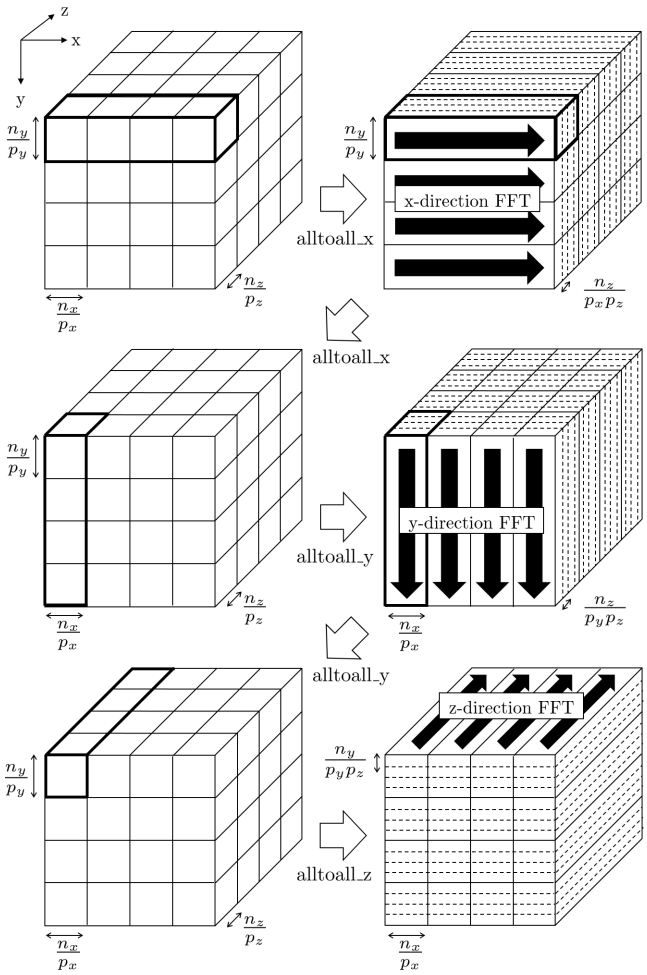

図 3 1d_Alltoall 方式 3 軸分割の FFT における転置操作の概要
Fig. 3 Overview of transposition operations in 1d_Alltoall scheme 3D decomposition FFT

採用し通信を 3 回に減らした 2d_Alltoall 方式の 2 通りの 3 軸分割を実装した．1d_Alltoall 方式 3 軸分割の FFT における転置操作の概要を**図 3** に示す．

入力データの形状は $n_x \times n_y \times n_z$ とし，x 軸方向，y 軸方向および z 軸方向の分割数はそれぞれ p_x, p_y, p_z とする．ただし，n_x は p_x で，n_y は $p_y \times p_z$ で，n_z は $p_x \times p_z$ および $p_y \times p_z$ で割り切れるものとする．alltoall_x は p_x プロセス間の通信，alltoall_y は p_y プロセス間の通信，alltoall_z は p_z プロセス間の通信となる．

同様に，2d_Alltoall 方式 3 軸分割の FFT における転置操作の概要を**図 4** に示す．

入力データの形状および各軸方向の分割数は 1d_Alltoall 方式と同様とする．ただし，n_x, n_y, n_z はそれぞれ $p_x \times p_y, p_y \times p_z, p_x \times p_z$ で割り切れるものとする．

3. 通信コストモデル

All-to-All 通信のコストモデルはネットワークトポロジーと関連することが古くから知られている [4][5]．まず，個々のプロセスが長さ m のメッセージ M_i^q, $(i = 0, \cdots, p-1,\ q$ はプロセス ID) を保持し，p プロセスで構成されるグループでの All-to-All 通信の通信コストモデルを考える．「京」コンピュータに採用されている MPI では特別なアルゴリズムが採用されている場合があるが，今研究では Gupta の本 [4] で紹介されている標準的なアルゴリズムを仮定して議論を進める．

3.1 All-to-All on Ring

「京」コンピュータなどの高次元トーラスネットワークのある一次元要素を取り出すとき，リング構成をしている．リングにおける All-to-All の性能コストは他のケースに対するベースとなる．初期状態として，プロセス q において $(M_0^q, \cdots, M_{p-1}^q)$ のように各メッセージにラベル付けをする．各メッセージはデータ長 m であり，各プロセス上にはプロセス数と同数の p 個メッセージが格納されているとする．

(1) まず，プロセス q において p 個のメッセージから下添え字が q のものを除いた $p-1$ 個のメッセージを通信バッファに格納する．
(2) 個々のプロセスは上位プロセスの向きにシフト通信を行う．
(3) 受信バッファーの中から下添え字が q のメッセージを取り出す．
(4) 通信バッファの要素がなくなるまで 2,3,4 を繰り返す．

この一連のアルゴリズムでは，第 i ステップで $P-i$ 個のメッセージを隣プロセスに送信することから，通信コストは以下のように定まる．

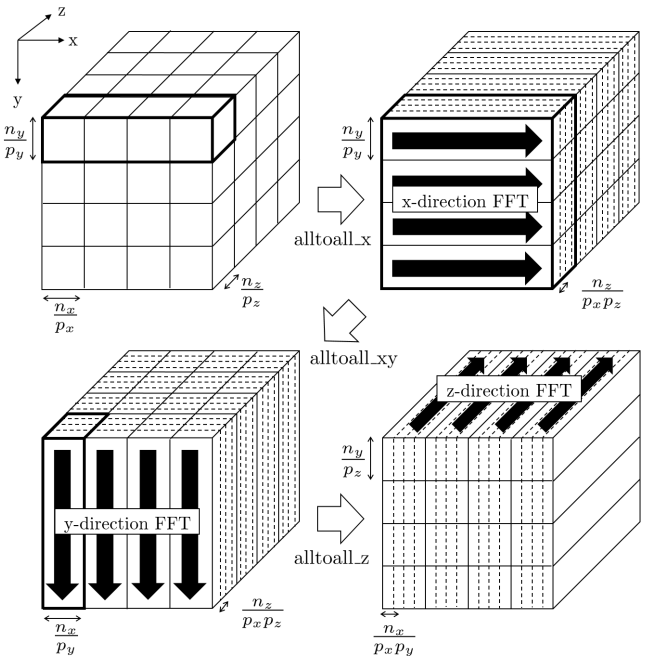

図 4 2d_Alltoall 方式 3 軸分割の FFT における転置操作の概要
Fig. 4 Overview of transposition operations in 2d_Alltoall scheme 3D decomposition FFT

$$T_{\text{Ring}} = \sum_{i=1}^{p-1}(\alpha + \beta m(p-i)) \quad (3)$$
$$= (\alpha + \beta mp/2)(p-1) \quad (4)$$

ここで，α は通信起動コスト (所謂レイテンシ)，β はデータ長あたりの転送時間 (通信速度の逆数) となる．

3.2 All-to-All on Torus/Mesh

2 次元のトーラスネットワーク上ではリングにおける All-to-All のアルゴリズムを列方向，行方向に 2 回実施することで All-to-All を実現できる．したがって

$$T_{\text{Torus}} = 2\sum_{i=1}^{\sqrt{p}-1}(\alpha + \beta m(p - \sqrt{p}i)) \quad (5)$$
$$= 2(\alpha + \beta mp/2)(\sqrt{p}-1) \quad (6)$$

また同様に 3D-Torus では

$$T_{\text{3D-Torus}} = 3(\alpha + \beta mp/2)(\sqrt[3]{p}-1) \quad (7)$$

と見積もることができる．これらのアルゴリズムでは列・行の方向に接続されたネットワークスイッチの何れかの一方向のみが使用されるが，「京」コンピュータに搭載されている Tofu ネットワークでは同時に複数方向の通信が可能であるため，上記アルゴリズムより優れた方法が実装されている．

3.3 その他のネットワークトポロジー

All-to-All は pairwise アルゴリズムで送受信をすることで実現可能であり，仮にすべての送受信プロセスのリンク間で衝突や干渉がないリッチなネットワーク環境を仮定すると，$p-1$ 回の pairwise の 1 対 1 通信が必要となることから

$$\tilde{T} \approx (\alpha_0 + \alpha p) + \beta mp \quad (8)$$

と見積もることができる．上記式は理想的な状況を想定しているため，ここでは All-to-All のコスト下限と考える．以下多軸分割におけるデータ再分散においてチルダをつけコスト下限式とする．

4. 多軸分割プロセスグループにおけるデータ再分散コスト推定

4.1 1 軸分割アルゴリズムのコスト

1 軸分割アルゴリズムでは P プロセス，総データ数 $N = n_x n_y n_z$ の 3 次元データを扱う．$\sqrt[3]{P} \times \sqrt[3]{P} \times \sqrt[3]{P}$ の 3D-Torus 上でのメッセージサイズ $m = N/P^2$ での All-to-All であるので，

$$T_{1D} \approx 3\alpha\sqrt[3]{P} + (3/2)\beta NP^{-2/3} \quad (9)$$
$$\tilde{T}_{1D} \approx (\alpha_0 + \alpha P) + \beta NP^{-1} \quad (10)$$

と見積もられる．

4.2 2 軸分割アルゴリズムのコスト

2 軸分割アルゴリズムでは個々のプロセスは $P = p_y \times p_z \approx \sqrt{P} \times \sqrt{P}$ の 2 次元プロセスグリッドに割り当てられ，ネットワークは 3D-torus($\sqrt[3]{P} \times \sqrt[3]{P} \times \sqrt[3]{P}$) を 2 次元トーラスにマップしたものを想定する．今，$\sqrt{P}$ 個の 1 次元並びのプロセスが $\sqrt[4]{P} \times \sqrt[4]{P}$ の Torus にマップされるとする．[*1] 第一ステップでは，p_y プロセスが $\sqrt{p_y} \times \sqrt{p_y}$ の Torus 上でメッセージサイズ $N/(p_z p_y^2)$ の All-to-All を実施，第二ステップでは p_z プロセスによるメッセージサイズ $N/(p_y p_z^2)$ の All-to-All を実施するので，

$$T_{2D} \approx 2(\alpha + \beta(N/p_z p_y^2)p_y/2)\sqrt{p_y} + 2(\alpha + \beta(N/p_y p_z^2)p_z/2)\sqrt{p_z} \quad (11)$$
$$= 2\alpha(\sqrt{p_y} + \sqrt{p_z}) + \beta N(1/p_z p_y^{1/2} + 1/p_y p_z^{1/2}) \quad (12)$$
$$\approx 4\alpha\sqrt[4]{P} + 2\beta NP^{-3/4} \quad (13)$$
$$\tilde{T}_{2D} \approx (2\alpha_0 + \alpha(p_y + p_z)) + 2\beta(N/p_y p_z) \quad (14)$$
$$\approx 2(\alpha_0 + \alpha\sqrt{P}) + 2\beta NP^{-1} \quad (15)$$

ただし，$\sqrt{P}$ のプロセスが 2D-Torus にマップされても，内部で Ring として扱われた場合には，最悪以下のような評価になることも考えられる．

$$T'_{2D} = 2\alpha\sqrt{P} + \beta NP^{-1/2} \quad (16)$$

4.3 3 軸分割+1d_Alltoall 方式のコスト

3 軸分割+1d_Alltoall 方式では個々のプロセスは $P = p_x \times p_y \times p_z \approx \sqrt[3]{P} \times \sqrt[3]{P} \times \sqrt[3]{P}$ の 3 次元プロセスグリッドに割り当てられ，ネットワークは 3D-torus を素直に 3 次元にマップしたものを想定する．各ステップで，p_* 個のプロセスによるメッセージサイズ $N/(p_x p_y p_z p_*)$ の Ring 上の All-to-All を実施しているゆえに，

$$T_{3D+1d} \approx 2(\alpha + \beta(N/p_x^2 p_y p_z)p_x/2)p_x + 2(\alpha + \beta(N/p_x p_y^2 p_z)p_y/2)p_y + (\alpha + \beta(N/p_x p_y p_z^2)p_z/2)p_z \quad (17)$$
$$= \alpha(2p_x + 2p_y + p_z) + \beta N(1/p_y p_z + 1/p_z p_x + 1/2p_x p_y) \quad (18)$$
$$\approx 5\alpha\sqrt[3]{P} + (5/2)\beta NP^{-2/3} \quad (19)$$
$$\tilde{T}_{3D+1d} \approx (5\alpha_0 + \alpha(2p_x + 2p_y + p_z)) + 5\beta(N/p_x p_y p_z) \quad (20)$$
$$\approx 5(\alpha_0 + \alpha\sqrt[3]{P}) + 5\beta NP^{-1} \quad (21)$$

4.4 3 軸分割+2d_Alltoall 方式のコスト

3 軸分割+2d_Alltoall 方式は，第二ステップにおいて

[*1] $\sqrt[6]{P} \times \sqrt[6]{P} \times \sqrt[6]{P}$ にマップされることもありうるが，「京」のマッピングでは 2 次元が自然であることから，本研究では $\sqrt[4]{P} \times \sqrt[4]{P}$ で議論を進める．

表 1　多軸分割 FFT における All-to-All のコスト
Table 1　All-to-All cost for multidimensional decomposition FFT

	Lower limit	Upper limit
T_{1D}	$(\alpha_0+\alpha P)+\beta NP^{-1}$	$3\alpha\sqrt[3]{P}+(3/2)\beta NP^{-2/3}$
T_{2D}	$2(\alpha_0+\alpha\sqrt{P})+2\beta NP^{-1}$	$4\alpha\sqrt[4]{P}+2\beta NP^{-3/4}$
		$2\alpha\sqrt{P}+\beta NP^{-1/2}$
T_{3D+1d}	$5(\alpha_0+\alpha\sqrt[3]{P})+5\beta NP^{-1}$	$5\alpha\sqrt[3]{P}+(5/2)\beta NP^{-2/3}$
T_{3D+2d}	$3\alpha_0+\alpha(2\sqrt[3]{P}+P^{2/3})$ $+3\beta NP^{-1}$	$4\alpha\sqrt[3]{P}+2\beta NP^{-2/3}$

$p_x\times p_y$ の 2D-Torus 上でメッセージサイズ $N/(p_xp_y)^2p_z$ の All-to-All を行うので,

$$\begin{aligned}T_{3D+2d} &\approx (\alpha+\beta(N/p_x^2p_yp_z)p_x/2)p_x \\ &\quad +2(\alpha+\beta(N/p_x^2p_y^2p_z)p_xp_y/2)\sqrt{p_xp_y} \quad (22)\\ &\quad +(\alpha+\beta(N/p_xp_yp_z^2)p_z/2)p_z \\ &= \alpha(p_x+2\sqrt{p_xp_y}+p_z) \\ &\quad +\beta N(1/2p_yp_z+1/\sqrt{p_xp_y}p_z+1/2p_xp_y) \quad (23)\\ &\approx 4\alpha\sqrt[3]{P}+2\beta NP^{-2/3} \quad (24)\\ \tilde{T}_{3D+2d} &\approx (3\alpha_0+\alpha(p_x+p_xp_y+p_z)) \\ &= \alpha(2p_x+2p_y+p_z) \\ &\quad +3\beta(N/p_xp_yp_z) \quad (25)\\ &\approx (3\alpha_0+\alpha(2\sqrt[3]{P}+P^{2/3}))+3\beta NP^{-1} \quad (26)\end{aligned}$$

4.5　多軸分割 All-to-All まとめ

ここまでに様々な軸分割 FFT に必要な All-to-All のコストを評価してきた．それらは通信コスト下限上限として表 1 のようにまとめることができる．評価式からは明確に上下限で最良な分割方式を 1 つに定められないが，上限と下限との入れ子関係から問題サイズやプロセス数に応じて適切な分割方法の候補を絞り込むことは可能である．図 5 は下限式のレイテンシ項による各手法の順序と上限式の漸近項 (β に係る項) で比較した順序関係を区間で入れ子関係にし図示したものである．上下限の入れ子関係からは 2 軸分割が比較的少ないコストの可能性を示唆するが，3 軸分割+2d_Alltoall も高い性能を出す可能性を残している．

5.　実装と性能評価

5.1　FFTE-C

本稿では, 1 次元 FFT に高橋によって作成されたオープンソースの FFT プログラムである FFTE[6] の C 言語移植版 (FFTE-C) を用いる．C 言語に移植するに当たり, 長さ n の複素数データを配列 a に格納する場合, $a[2k], 0\le k\le n-1$ に実数部，$a[2k+1], 0\le k\le n-1$ に虚数部を格納するものとした．FFTE-C は，基数 2，3，4，5，8 の FFT を用いてデータ長 $2^p3^q5^r$ の複素数データに対する FFT および逆 FFT を計算する．

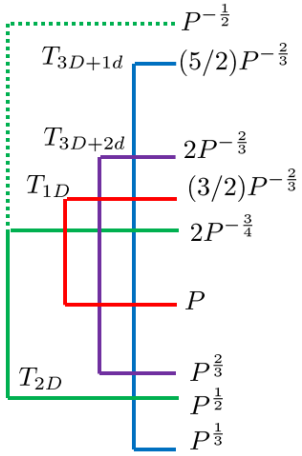

図 5　下限式のレイテンシ項と上限式の漸近項の関係
Fig. 5　Relationship between the latency term of the lower limit formula and the asymptotic term of the upper limit formula

表 2　京コンピュータの仕様
表 3　Specification of K computer

CPU		SPARC64 ™ VIII fx
	Cores	8
	Operating frequency	2.0 GHz
	L1 cache (each core)	Instruction cache : 32KB/2way Data cache : 32KB/2way
	L2 cache	6MB/12way
Node	Chip (LSI)	1
	Performance	128 GFLOPS
	Memory	16GB
	Memory throughput	46GB/s (8 cores)
Rack	Performance	12.3TFLOPS

今回は，3 次元の複素数データに対する 3 次元 FFT を実装し，AVX-512 によるベクトル化および MPI+OpenMP によるハイブリッド並列化を行った．AVX-512 によるベクトル化は ivdep プラグマを用いたコンパイルによる自動ベクトル化である．

5.2　実行環境

本稿では，京コンピュータと Oakforest-PACS を用いて性能評価を行った．京コンピュータの使用を**表 3** に，Oakforest-PACS の仕様を**表 4** に，Oakforest-PACS で用いられる KNL の仕様を**表 5** に示す．

クラスターモードは Quadrant，メモリモードは Flat を選択する．コンパイラは Intel コンパイラ (`mpiicc`, Version 18.0.1.163 Build 20171018) としコンパイラオプションは `-O3 -qopenmp -fma -axMIC-AVX512` を使用した．

5.3　京コンピュータにおける性能

京コンピュータを用いて, 2 軸分割, 3 軸分割+1d_Alltoall 方式および 3 軸分割+2d_Alltoall 方式の 3 種類の 3 次元

表 4　Oakforest-PACS の仕様
Table 4　Specification of Oakforest-PACS.

Total peak performance		25.004 PFlops
Number of nodes		8208
Node	Product	Fujitsu PRIMERGY CX1640 M1
	Peak performance	3.0464 TFlops
	CPU	Intel ® Xeon Phi ™ 7250
Interconnect	Product	Intel ®Omni-Path
	Link speed	100 Gbps
	Topology	Full-bisection Fat Tree

表 5　Intel Xeon Phi 7250 の仕様
Table 5　Specification of Intel Xeon Phi 7250.

Cores	68
Operating frequency	1.4 GHz
L1 cache (each core)	Instruction cache : 32KB Data cache : 32KB
L2 cache	34MB
Memory	96GB(DDR4) 16GB(MCDRAM)

FFT について，All-to-All，データ再分散，1 次元 FFT の各区間の実行時間の計測を行った．変換するデータの長さは $128^3, 256^3, 512^3$ とし，使用するノード数を 128 から (128^3 は 64 から)1024 まで変化させる．その結果をそれぞれ**図 6**，**図 7**，**図 8** に示す．

5.4　Oakforest-PACS における性能

Oakforest-PACS を用いて，2 軸分割，3 軸分割+1d_Alltoall 方式および 3 軸分割+2d_Alltoall 方式の 3 種類の 3 次元 FFT について，演算性能を測定した．変換するデータの長さは 512^3 とし，使用するノード数を 1 から 1024 まで変化させる．その結果を**図 9** に示す．3 次元 FFT の FLOPS は，1 次元 FFT が基数 2 のみの利用を想定し，問題サイズが $n_x \times n_y \times n_z$，実行時間が t のとき，$5n_xn_yn_z\log_2(n_xn_yn_z)/t$ で算出する．

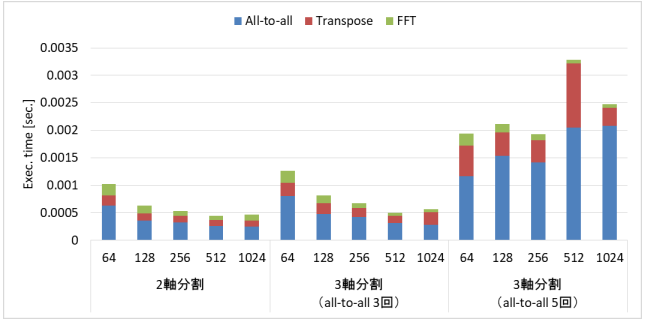

図 6　京におけるデータ長 128^3 の 3 次元 FFT の実行時間
Fig. 6　Execution time of 3D FFT with 128^3 grid on K computer

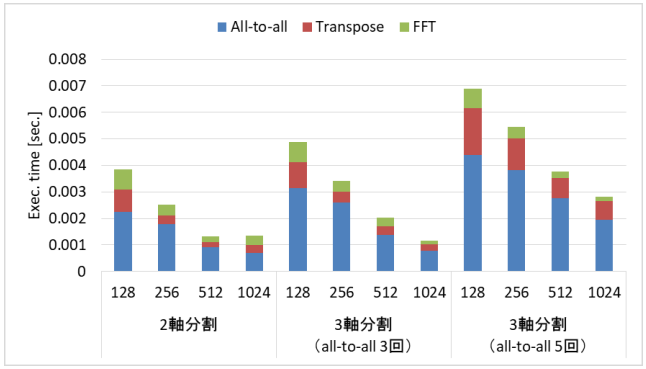

図 7　京におけるデータ長 256^3 の 3 次元 FFT の実行時間
Fig. 7　Execution time of 3D FFT with 256^3 grid on K computer

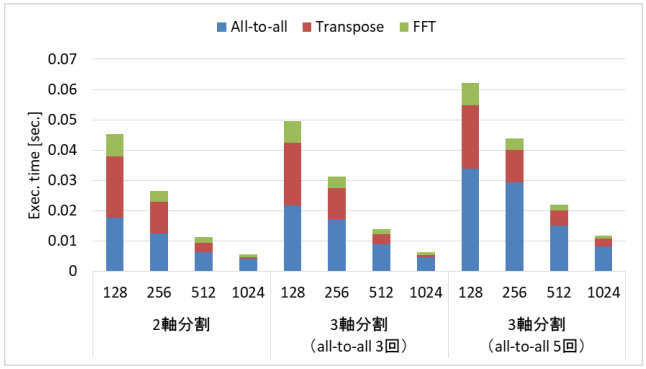

図 8　京におけるデータ長 512^3 の 3 次元 FFT の実行時間
Fig. 8　Execution time of 3D FFT with 512^3 grid on K computer

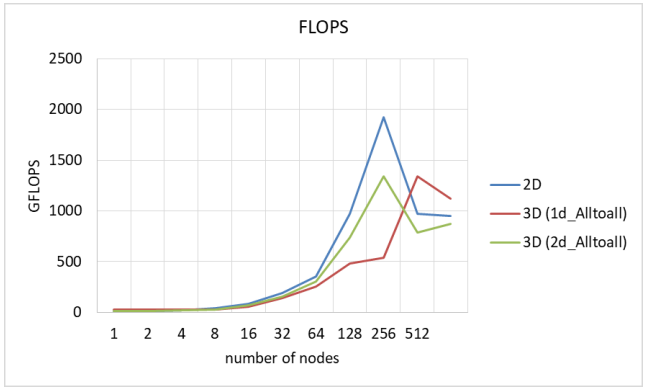

図 9　Oakforest-PACS における 3 次元 FFT の演算性能
Fig. 9　Performance of 3D FFT on Oakforest-PACS

また，Oakforest-PACS の MPI_Alltoall1 回の実行時間の計測を行った．使用するノード数は 2 ノードから 32 ノードまでとし，データ長を 2 の累乗数で変化させる．その結果を**図 10** に示す．

プロセス数 $p = 2, 4, \ldots, 32$ の場合について，3 章で述べた Ring, 3D mesh/torus, 理想的 pairwise 通信のモデルの α, β を線形回帰による推定を行った．推定の結果として，Oakforest-PACS の通信モデルとして理想的な pairwise 通信がもっとも近いモデルであった．これは，通信に関わる

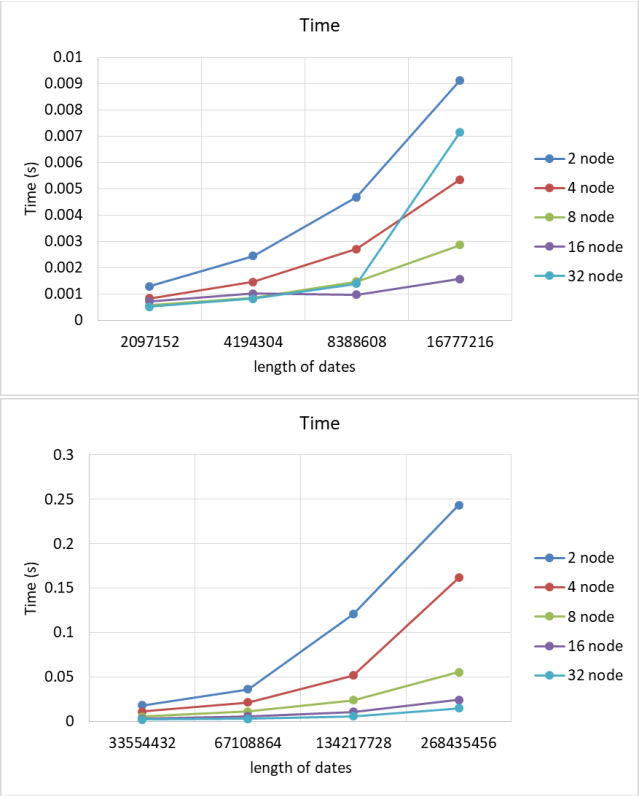

図 10 Oakforest-PACS における MPI_Alltoall の実行時間
Fig. 10 Execution time of MPI_Alltoall on Oakforest-PACS

プロセス数が少なかったからであると考えられる．

6. おわりに

本稿では，1 軸分割，2 軸分割と通信方法が異なる 2 種類の 3 軸分割 (1d_Alltoall 方式と 2d_Alltoall 方式) の 3 次元 FFT について，3D-Torus における All-to-All のモデルをコスト上限，理想的な pairwise 通信のモデルをコスト下限としてそれぞれの軸分割の All-to-All のコストを求めた．結果として，2 軸分割が比較的少ないコストの可能性を示唆したが，2d_Alltoall 方式の 3 軸分割も高い性能を出す可能性が残されていることが分かった．そして，2 軸分割と 2 種類の通信方法の 3 軸分割の 3 次元 FFT を，1 次元 FFT に FFTE の C 言語移植版 FFTE-C を用いて実装し，京コンピュータと Oakforest-PACS を用いてその性能を測定した．また，Oakforest-PACS 上で MPI_Alltoall 関数の実行時間を測定し，Oakforest-PACS の通信モデルの推定を行った．結果として，ノード数が 32 までの場合では理想的な pairwise 通信モデルがもっとも近い通信モデルであることが分かった．

謝辞 本研究は科学研究費補助金 基盤研究 (B)15H02709 の支援を得て実施しています．

参考文献

[1] Cooley J.W. and Tukey, J.W.: *An Algorithm for the Machine Calculation of Complex Fourier Series* Mathematics of Computation, vol. 19, pp. 297–301 (1965).
[2] Ishihara, T., Morishita, K., Yokokawa, M., Uno, A., Kaneda, Y.: *Energy spectrum in high-resolution direct numerical simulations of turbulence*, Phys. Rev. Fluids vol.1, pp. 082403 (2016).
[3] Jung, J., Kobayashi, C., Imamura, T., Sugita, Y.: *Parallel implementation of 3D FFT with volumetric decomposition schemes for efficient molecular dynamics simulations*, Computer Physics Communications, vol. 200, Elsevier, pp. 57–65 (2016).
[4] Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing, Addison Wesley, (2013).
[5] Hoefler, T., Gropp, W., Thakur, R., Träff, J. L.: *Toward Performance Models of MPI Implementations for Understanding Application Scaling Issues*, EuroMPI, pp. 21–30 (2010).
[6] Takahashi, D.: FFTE: A Fast Fourier Transform Package available from ⟨http://www.ffte.jp/⟩ (updated 2014).
[7] Frigo, M. and Johnson, S.G.: *The Design and Implementation of FFTW3*, Proceedings of the IEEE, vol. 93, pp. 216-231 (2005).
[8] Li N., Laizet S.: *2DECOMP&FFT A highly scalable 2D decomposition library and FFT interface*, Cray User Group 2010 conference, Edinburgh, (2010).
[9] Canning A. Scalable Parallel 3d FFTs for Electronic Structure Codes. In: Palma J.M.L.M., Amestoy P.R., Dayd M., Mattoso M., Lopes J.C. (eds) High Performance Computing for Computational Science - VECPAR 2008. VECPAR 2008. Lecture Notes in Computer Science, vol 5336. pp 280–286 (2008)
[10] 北澤好人 黒田明義 志田直之 安達知也 南一生, ”スループットを用いた「京」における MPI 通信性能の評価”, ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, Vol.2017, pp.17–25 (2017)
[11] 北澤好人 黒田明義 南一生 庄司文由, ”スーパーコンピュータ「京」における MPI 通信性能の評価”, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2016-HPC-153, No.34, pp.1–7 (2016)