Vol.2021-HPC-178 No.21
2021/3/16

非圧縮性乱流DNSコードに現れる高速フーリエ変換のSX-Aurora TSUBASAにおける性能評価

武中 裕次郎[1,a)]　横川 三津夫[1]　石原 卓[2]　小松 一彦[3]　小林 広明[4]　今村 俊幸[5]　清水 智也[1]

概要：乱流は自然現象や科学技術の様々な問題で現れるため，その普遍的な性質を解明することが重要となる．フーリエ・スペクトル法による一様等方性乱流の直接数値シミュレーション（DNS）では，その実行時間の大部分を３次元高速フーリエ変換（3D-FFT）が占める．本研究では，2 軸分割による非圧縮一様等方性乱流 DNS コードを開発し，FFT 部分について，最先端のベクトル計算機である SX-Aurora TSUBASA 向け最適化を行なった．最適化としてキャッシュブロッキング，配列のパディングを行なった結果，最大で約 3.3 倍の高速化を達成した．FFT における計算性能は 1 コアあたり 3.38GFlops，対ピーク性能比は 1.1%であり，計算部分のみでは 1 コアあたり 35.9GFlops，11.7%であった．また，大規模な乱流 DNS の実現に向けて通信時間を予測するため，SX-Aurora TSUBASA において，`MPI_alltoall` による通信の実効転送性能を計測した．その結果，VE 数が 16 以上の場合はプロセス数の増加に関わらず，実効転送性能が一定になることがわかった．

1. はじめに

乱流は自然現象や科学技術の様々な問題で現れるため，その普遍的な性質を解明することは非常に重要である．しかし，流体の支配方程式である Navier–Stokes 方程式は非線形性が強いため，解析的に解くことは困難であることが知られている．一方，近年の著しい計算機の発展により，乱流の性質の解明のために数値シミュレーションが有効な手段となっている．

直接数値シミュレーション (Direct Numerical Simulation; DNS) は，モデル化などを行わずに方程式を直接解き，小さい渦まで解像するため，流れの本質を解明するための最も良い方法であるといえる．しかし，高レイノルズ数乱流を再現するためには計算格子点数を増加させる必要があり，それに比例して計算量は膨大となる．1969 年に Orszag によって格子点数 32^3 における DNS[1] が実行されてから，コンピュータの計算能力の向上にしたがって，より多くの格子点数における DNS が実行されてきている．2003 年の地球シミュレータによる DNS や 2016 年の京コンピュータによる DNS など，日本でも当時最高の性能を持つスーパーコンピュータ上で DNS が行われてきた [2], [3], [4]．2019 年には，米国オークリッジ国立研究所（Oak Ridge National Laboratory; ORNL）の Summit 上で GPU を用いて格子点数 18342^3 における DNS が実行されており，これが現在までに行われた最も高解像度な DNS である [5]．しかし，未だに現実世界に現れるほど高いレイノルズ数を持つ乱流の再現には至っていない．

これまでに行われた多くの一様等方性乱流 DNS では，立方体領域の空間方向の離散化手法としてフーリエ・スペクトル法が採用されている．この手法では空間微分を解析的に解くことができるが，実行時間の大部分を３次元高速フーリエ変換（Fast Fourier Transform; FFT）が占める．そのため，DNS 全体の高速化には３次元 FFT の高速化を図ることが不可欠である．

本研究の目標は世界最大規模の DNS の実現である．そのためには，膨大な計算資源が必要であり，現在日本で実行できる計算機システムは「富岳」のみであると考える．一方で，比較的小規模な DNS によって得られる乱流データも未だに不十分である．例えば，同じ格子点数において異なるレイノルズ数を持つ乱流のデータを得ることは有用である．そのため，「富岳」以外のシステムにおける最適化や性能評価を行い，その知見を蓄積する必要がある．本

[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] 岡山大学大学院生命科学研究科
Graduate School of Environmental and Life Science, Okayama University
[3] 東北大学サイバーサイエンスセンター
Cyberscience Center, Tohoku University
[4] 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
[5] 理化学研究所計算科学研究センター
RIKEN Center for Computational Science
[a)] ytakenaka@stu.kobe-u.ac.jp

ⓒ 2021 Information Processing Society of Japan

稿では，DNS 中の 3 次元 FFT 部分に注目して，最先端のベクトル型プロセッサである SX-Aurora TSUBASA で構成された東北大学サイバーサイエンスセンターの計算機システム AOBA-A に対して最適化を行い，性能を評価した．さらに，最適化したプログラムについて，種々の FFT ライブラリとの比較を行なった．また，大規模な DNS の実行に向けた通信時間の予測を行なった．

2. 乱流 DNS

2.1 乱流 DNS の概要

本研究では，非圧縮性乱流を対象とするため，Navier–Stokes 方程式 (1) と非圧縮条件の式 (2) を扱う．

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} = -\frac{1}{\rho}\frac{\partial p}{\partial \boldsymbol{x}} + \nu\Delta\boldsymbol{u} + \boldsymbol{F} \tag{1}$$

$$\nabla\cdot\boldsymbol{u} = 0 \tag{2}$$

ここで，$\boldsymbol{u}$ は速度，p は圧力，ρ は密度，ν は動粘性係数，$\boldsymbol{F}$ は外力である．計算対象は 3 次元空間における $[0,2\pi]\times[0,2\pi]\times[0,2\pi]$ の立方体領域内に発生する乱流（Box 乱流）とする．また，境界条件は全ての方向で周期境界条件とした．

空間方向の離散化手法としてフーリエ・スペクトル法，時間方向の離散化手法として 4 段 4 次 Runge–Kutta–Gill 法を用いる．

フーリエ・スペクトル法では，解を以下のようにフーリエ級数の形で近似する．n を計算格子点数とする．

$$\boldsymbol{u} = \sum_{k_1=-n/2}^{n/2-1}\sum_{k_2=-n/2}^{n/2-1}\sum_{k_3=-n/2}^{n/2-1}\hat{\boldsymbol{u}}_{\boldsymbol{k}}(t)\mathrm{e}^{\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}} \tag{3}$$

式 (1)，(2) に対してフーリエ・スペクトル法を適用すると，以下の波数空間における常微分方程式が得られる．

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{u}}_{\boldsymbol{k}} = -\nu|\boldsymbol{k}|^2\hat{\boldsymbol{u}}_{\boldsymbol{k}} + \boldsymbol{k}\cdot\frac{\widehat{(\boldsymbol{k}\cdot(\boldsymbol{u}\cdot\nabla)\boldsymbol{u})}}{|\boldsymbol{k}|^2} - \widehat{((\boldsymbol{u}\cdot\nabla)\boldsymbol{u})}_{\boldsymbol{k}} \tag{4}$$

これを時間発展させることで解が得られる．しかし，右辺第 2，3 項に現れる非線形項の評価の際，エイリアシングエラーと呼ばれる誤差を除去する必要がある．$|\boldsymbol{k}| > \sqrt{2}n/3$ を満たす波数における値を除外し，phase shift 法を用いてエイリアシングエラーを除去できるが，非線形項 1 回の計算につき 3 次元高速フーリエ変換（3D-FFT）が計 18 回必要となり，実行時間の大部分を占める．

2.2 3 次元高速フーリエ変換

高速フーリエ変換は離散フーリエ変換 (5) を高速に計算するアルゴリズムである．

$$y_k = \sum_{j=0}^{n-1} x_j\omega_n^{jk} \tag{5}$$

本研究では，Cooley–Tukey のアルゴリズム [6] による基数 2, 4 の FFT を実装した．また，比較対象として，FFTW[7] 及び FFTE-C[8], [9] を用いたコードも作成した．

2.3 並列化概要

本研究では，並列化手法として 2 軸分割（pencil decomposition）による領域分割を用いた．そのため，DNS 中で各プロセスは一次元方向にのみ連続な領域内のデータのみを扱う．

しかし，3D-FFT では，各プロセス内で計算を行う次元のデータを連続に持つ必要がある．そのため，各方向に FFT を行うたびにデータの転置を行う必要がある（**図 1**）．データの転置では，各プロセス内でのデータの並び替え（転置），全対全通信（`MPI_alltoall`）によるプロセス間のデータ転送を行う．プロセス間のデータ転送の前後にデータを並び替える必要があるため，3 次元 FFT の手順は以下のようになる．

(1) x 方向の FFT
(2) プロセス内データ転置
(3) `MPI_alltoall` によるデータ転送
(4) プロセス内データ転置
(5) y 方向の FFT
(6) プロセス内データ転置
(7) `MPI_alltoall` によるデータ転送
(8) プロセス内データ転置
(9) z 方向の FFT

また，全対全通信について，分割される 2 軸方向のうち，一方の軸方向については独立に通信を行うことができる．例えば，x 方向の FFT を行なった後，次に y 方向の FFT を行うために y 軸方向のデータを集めるが，その際，z 軸方向に分割されたプロセスとは通信を行わず，y 軸方向に分割されたプロセス間のみでの通信を行う．そのため，`MPI_comm_split` によってコミュニケータを分割し，各コミュニケータ内で全対全通信を行う．これによって，コミュニケータ間で通信が干渉し合わないようなシステムでは全対全通信を独立に実行できる．2 方向に分割を行うプロセス数をそれぞれ $p1$，$p2$，1 次元あたりの格子点数を n とすると，1 回の全対全通信で転送するデータの個数 m_1，m_2 はそれぞれ以下のようになる．

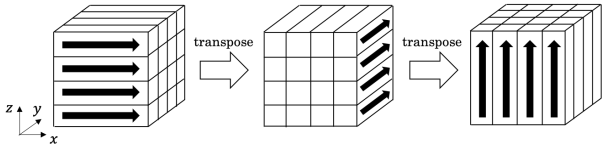

図 1 2 軸分割における 3D-FFT の手順

表 1　AOBA-A の仕様

Total peak performance		1.48PFlops
Number of nodes		72
node		SX-Aurora TSUBASA B401-8
VE		Type 20B
	Number of cores	8
	Performance	2.46TFlops
	Memory	48GB
	Memory bandwidth	1.53TB/s
	L1D cache	32KB/core
	L1I cache	32KB/core
	L2 cache	256KB/core
	LLC	16MB
VH		AMD EPYC7402P

$$m_1 = \frac{n^3}{p1^2 p2} \tag{6}$$

$$m_2 = \frac{n^3}{p1 p2^2} \tag{7}$$

以下では，プロセス内のデータの並び替えを転置部分，全対全通信によるプロセス間データ転送を通信部分，FFT の計算を行うところを計算部分と呼ぶ．

3. SX-Aurora TSUBASA における最適化

3.1 実験環境

本研究では，東北大学サイバーサイエンスセンターの計算機システムである AOBA-A を用いて数値実験を行なった．AOBA-A の仕様を**表 1** に示す．

AOBA-A のプロセッサ SX-Aurora TSUBASA は，主にベクトル演算処理を行う複数の VE（Vector Engine）と主に OS 処理を行う VH（Vector Host）からなる．VE では，多数の要素に対して同じ演算を高速に実行でき，一度に扱える要素数をベクトル長という．また，プログラム中の全演算に対するベクトル演算率をベクトル化率と呼ぶ．ベクトル性能を活かすためには，ベクトル長，ベクトル化率を大きくすることが重要である．

また，本研究では Fortran90 でコードを実装し，変数は全て倍精度実数とした．使用したコンパイラは NEC Fortran Ver 3.0.8 である

3.2 最適化に向けた予備実験

AOBA-A における DNS コードの格子点数 $n^3 = 512^3$ における 5000 ステップ経過後の実行時間の内訳を**表 2** に示す．$\Delta t = 1.0 \times 10^{-3}$，動粘性係数 $\nu = 2.8 \times 10^{-4}$ とした．左から全体，3 次元 FFT，時間発展部分，外力計算部分，その他の実行時間を表しており，その他の部分はファイル入出力や MPI の初期化，終了部分などが含まれる．結果より，1 ステップあたり約 3.2 秒，また，3D-FFT が実行時間の 99.5%を占めることがわかる．そのため，DNS コードの高速化には 3D-FFT の高速化が不可欠である．以下では，3D-FFT の実行部分を抜き出して評価を行なった．

表 2　DNS コードの実行時間の内訳（$n^3 = 512^3$, $np = 64$, $t = 5.0$）

Whole[s]	3D-FFT	Integration	Forcing	Other
15897.54	15818.06	35.11	22.31	22.06

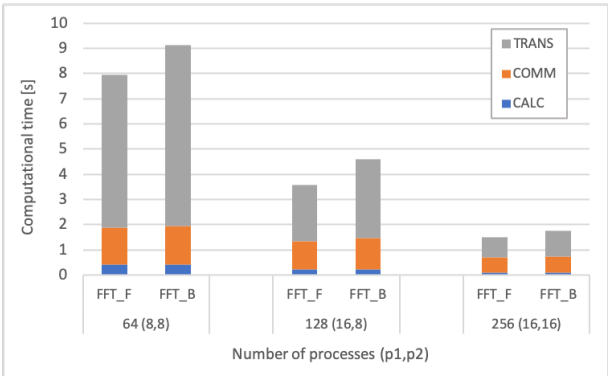

図 2　最適化前の FFT 及び逆 FFT の計算時間（$n^3 = 2048^3$）

表 3　ベクトル長，ベクトル化率，LLC ヒット率（$np = 64$）

n^3	A.V.length	V.Op.ratio	LLC hit ratio[%]
64^3	7.0	35.6	99.4
128^3	14.0	55.7	99.4
256^3	28.0	72.2	9.7
512^3	56.0	83.9	3.4
1024^3	112.0	91.3	16.3
2048^3	224.0	95.4	29.9

図 2 に，$n^3 = 2048^3$ として，MPI プロセス数を 64，128，256 としたときの AOBA-A における FFT（FFT_F），逆 FFT（FFT_B）それぞれ 1 回ずつの実行時間を示す．結果より，すべての部分でプロセス数の増加につれて計算時間が減少している．また，転置部分が実行時間の半分以上を占めていることがわかる．

また，性能解析ツール ftrace を用いて得られる転置部分におけるベクトル長，ベクトル化率，LLC（Last Level Cache）ヒット率を**表 3** に示す．ベクトル長，ベクトル化率，LLC ヒット率は計測区間の平均値である．結果より，格子点数が増加し，配列のサイズが LLC のサイズよりも大きくなると LLC ヒット率は大幅に低下し，実行時間に影響を与えていることがわかる．また，格子点数が大きくなると，転置部分のループ長が大きくなるため，平均ベクトル長，ベクトル化率が大きくなっている．

次に，MPI と OpenMP によるハイブリッド並列において，使用するコア数を 64 と固定し，MPI プロセス数とスレッド数を変化させたときの $n^3 = 2048^3$ における FFT 及び逆 FFT の計算時間の変化を**図 3** に示す．結果より，計算部分についてはスレッド数を変化させても，実行時間にほとんど変化は見られなかった．また，通信部分では，スレッド数によって，1 プロセスあたりの転送量と通信を行う MPI プロセス数が変化するが，実行時間はスレッド数の

ⓒ 2021 Information Processing Society of Japan

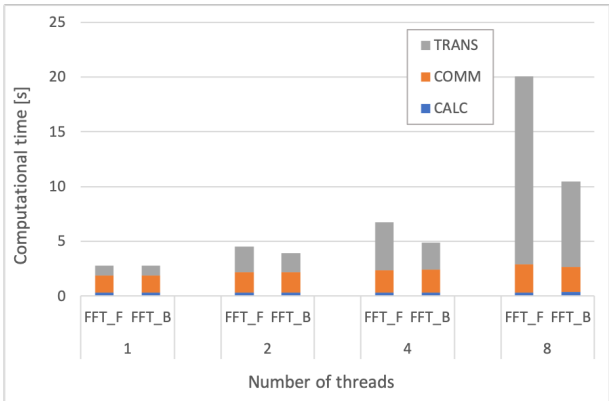

図 3 スレッド数を変化させたときの FFT 及び逆 FFT の計算時間の変化（$n^3 = 2048^3$）

増加にしたがってわずかに増加する傾向がみられた．転置部分では，スレッド数の増加にしたがって実行時間が大きく増加している．これは，MPI プロセス数が減少すると，スレッド並列の対象である転置部分の再外側のループ長がスレッド数よりも短くなり，全てのスレッドによる並列化ができていないことが原因であると考える．より多くのプロセス数を用いる場合，転置部分にかかる実行時間の増加率は小さくなると考えられる．結果より，スレッド数が 1，プロセス数が 64 の flat MPI の場合最速であった．そのため，以下では，スレッド数を 1 とした場合について最適化を行う．

3.3 SX-Aurora TSUBASA に向けた最適化方針

予備実験より，転置部分における LLC ヒット率の低く，実行時間の半分以上を占めているため，転置部分の最適化としてキャッシュブロッキングを適用し，LLC ヒット率の向上を図る．転置の一例（Algorithm 1）にキャッシュブロッキングを適用すると，Algorithm 2 のようになる．一般的に，ブロックサイズは配列がキャッシュに収まるように決定するが，ベクトル長は小さくなる．そのため，ブロックサイズとベクトル長による影響を調べる必要がある．

また，計算部分の高速化のために配列のパディングを行う．パディングとは，配列の第 1 要素が 2 のべき乗になるのを避けるためにダミー要素を 1 つ加えることである．パディングはベクトル計算機の最適化において広く使われる手法であり，CPU の同じポートを何度も使うことによる遅延（CPU ポート競合）やメモリの同じバンクへのアクセスが頻発すること（メモリバンク衝突）を避けることができる．

3.4 性能評価

ブロックサイズを変化させたときの転置部分の $n^3 = 2048^3$ における実行時間を**図 4** に示す．また，ベクトル長，LLC ヒット率を**表 4** に示す．図 4 より，ブロックサイズ $ibl = 16$ の場合の実行時間が最小であった．ブロックサイズの小さい範囲では LLC ヒット率は高いが，ベクトル長が小さいため，ベクトル演算におけるレイテンシが影響し，実行時間が大きくなっていると考えられる．しかし，転置のそれぞれの部分で実行時間が最小となるブロックサイズは異なるため，それぞれのループで適切なブロックサイズを選択することで最も実行時間を小さくすることができる．転置部分の中でも，格子点数に比例してストライドが大きくなる不連続アクセスが存在するループ（TRANS1F，TRANS3B）では $ibl = 16$ の場合に最速であり，それ以外のループでは $ibl = 256$ の場合，つまり LLC ヒット率よりもベクトル長を大きくしたときに最速であった．また，表 4 より，平均ベクトル長が ibl とほぼ同じ値をとっていることが確認できる．$ibl \leq 128$ では平均ベクトル長が ibl よりも小さい値をとっているが，これは，転置部分のループ長が ibl よりも小さいためである．また，ibl の値が小さい場合，不連続アクセスの幅が小さいため，LLC ヒット率が高く，ibl の増加にしたがって LLC ヒット率が小さくなっていることがわかる．

配列のパディングを行なった結果，計算部分の計算性能は 1 プロセスあたり 27.8GFlops から 35.9GFlops まで約 1.3 倍向上した．しかし，CPU ポート衝突による遅延時間は最適化前後で変わらず，明確な原因を突き止めるためには，さらなる考察が必要である．

AOBA-A における最適化前後の $n^3 = 4096^3$ における

Algorithm 1 Data transposition in one process.

js, je : start and end of j-direction grid point in each process
ks, ke : start and end of k-direction grid point in each process

```
for k = ks to ke do
  for j = js to je do
    for i = 0 to n − 1 do
      g(k, i, j) = f(i, j, k)
    end for
  end for
end for
```

Algorithm 2 Data transposition in one process with cache blocking.

ibl : blocking size

```
for kk = ks to ke; kk = kk + ibl do
  for jj = js to je; jj = jj + ibl do
    for ii = 0 to n − 1; ii = ii + ibl do
      for k = kk to min(kk + ibl − 1, ke) do
        for j = jj to min(jj + ibl − 1, je) do
          for i = ii to min(ii + ibl − 1, n − 1) do
            g(k, i, j) = f(i, j, k)
          end for
        end for
      end for
    end for
  end for
end for
```

ⓒ 2021 Information Processing Society of Japan

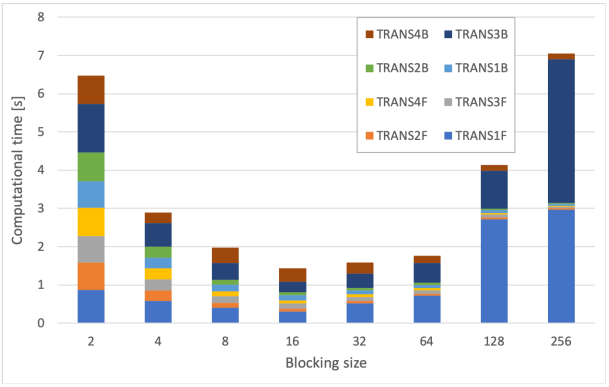

図 4 ブロックサイズによる転置部分の計算時間の変化（$n^3 = 2048^3$）

表 4 転置部分におけるベクトル長，LLC ヒット率（$n^3 = 2048^3$）

ibl	A.V.length	LLC hit ratio[%]
2	2.0	66.0
4	4.0	76.1
8	8.0	66.5
16	16.1	34.6
32	32.2	19.1
64	64.0	20.9
128	123.7	32.6
256	213.2	36.7

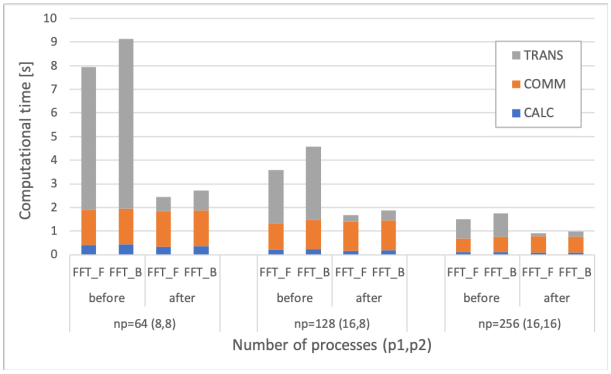

図 5 AOBA-A における最適化前後の FFT 及び逆 FFT の計算時間の比較（$n^3 = 4096^3$）

FFT 及び逆 FFT の計算時間を**図 5** に示す．スレッド数は 1 とし，ブロックサイズはそれぞれのループで適切な値を選択した．主に転置部分の高速化の影響が大きく，全ての場合について高速化が達成された．プロセス数 $np = 64$ の場合に最大約 3.3 倍高速化され，計算性能は 1 コアあたり約 3.38GFlops，ピーク性能比は約 1.1%であった．また，計算部分の計算性能は 1 コアあたり約 35.9GFlops であり，ピーク性能比は約 11.7%であった．計算部分の Flops 値向上のためには，より高い基数の FFT を実装するなどのアルゴリズムレベルの変更が必要である．

最適化後のプログラムについて，代表的な FFT ライブラリである FFTW，神戸大学と理化学研究所が共同で開発した FFTE-C との比較を行なった．FFTW は科学技術計算ライブラリ ASL（Advanced Scientific Library）に含まれて

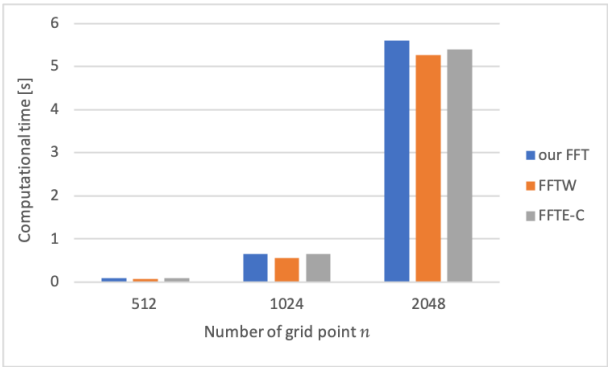

図 6 FFT 及び逆 FFT の計算時間の FFTW，FFTE-C との比較（$np = 64$）

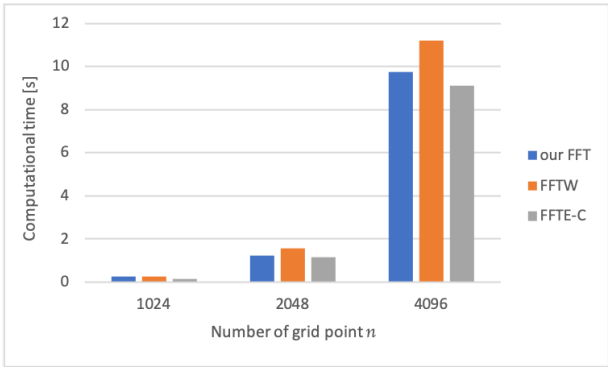

図 7 FFT 及び逆 FFT の計算時間の FFTW，FFTE-C との比較（$np = 512$）

おり，ベクトル計算機に向けて最適化されている．しかし，FFTW では，2 軸分割に対応する FFT ルーチンは存在しないため，1 次元方向に FFT を行う `dfftw_execute_dft`，`dfftw_execute_dft_c2r`，`dfftw_execute_dft_r2c` を使用し，転置部分は独自に実装した．FFTE-C はオープンソースライブラリである FFTE[10] の C 言語移植版に基数 16 の FFT ルーチンを加えたものであり，2 軸分割に対応した 3 次元 FFT ルーチンである `mfft3d2v`，`mrfft3d2v` を用いて 3 次元 FFT を計算することができる．プロセス数 $np = 64$，512 における FFT，逆 FFT それぞれ 1 回ずつの計算時間を**図 6**，**図 7** に示す．

図 6，図 7 より，各ライブラリで実行時間に大きな差はなく，本研究で開発したプログラムは他のライブラリと比較して同等の性能であることが確認された．$n^3 = 2048^3$，$np = 64$ の場合 FFTW，$n^3 = 4096^3$，$np = 512$ の場合 FFTE-C が最も高速であった．FFTW，FFTE-C は基数 8 や 16 などの基数の高い FFT が実装されている．本研究で開発したコードにより高い基数の FFT を実装することで，実行時間の削減が見込める．

DNS コードにライブラリを用いずに FFT を実装することの利点として，メモリ量の削減が挙げられる．大規模な計算の実行にあたって，計算資源の点においてメモリ量を少しでも削減することは非常に重要である．FFT ライブラ

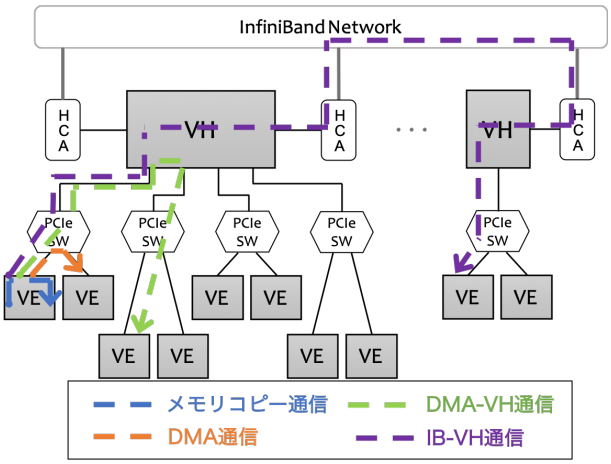

図 8 AOBA-A における 4 種類の通信経路

リで配列として確保している繰り返し用いる定数などを，配列として確保せずに毎回計算することでメモリ量を削減できる．また，phase shift 法で必要な演算を FFT ルーチン内に引き込むなどの工夫を施すこともできる．さらに，本研究で開発したコードやライブラリを用いたコードなど，複数のコードを用意することで計算結果を照らし合わせることもでき，性質解明のために得るデータの信頼性の向上にもつながる．

4. 大規模実行に向けた通信時間の予測

4.1 実効転送性能の計測

大規模な DNS では，計算，転置部分に比べて通信部分の実行時間の割合が非常に大きくなることが考えられる．そのため，FFT の通信時間を予測することで DNS 全体の実行時間を大まかに見積もることは有用である．以下では，AOBA-A における実効転送性能の計測を行い，実行時間の予測を行なった．

AOBA-A には，以下の 4 種類の通信経路が存在する（**図 8**）．

- メモリコピー通信
- DMA（Direct Memory Access）通信
- DMA-VH 通信
- IB-VH 通信

メモリコピー通信は同一の VE 内のコアによる通信，DMA 通信は同一の PCIe スイッチに接続された 2 つの VE 内のそれぞれのコアによる通信，DMA-VH 通信は異なる PCIe スイッチに接続された VE 内のコアによる VH 経由の通信，IB-VH 通信は異なるノードの VE 内のコアによる VH 経由の通信である．VH-PCIe スイッチ間，PCIe スイッチ-VE 間は PCIe Gen3 によって接続されており，ピーク転送性能は 12.5GB/s である．また，VH-HCA 間は PCIe Gen4 で接続されており，転送性能は 12GB/s である．

それぞれの通信経路での `MPI_sevdrecv` による 1 対 1 通信について，送受信するメッセージサイズを変化させた時の実行時間，実効転送性能の変化を**図 9**，**図 10** に示す．それぞれのグラフは，図 8 に示す経路と同じ色で記載している．ここで，メッセージサイズを m[bytes]，転送にかかる実行時間を t[s] とすると，`MPI_sendrecv` による実効転送性能は以下のように計算できる．また，プロセス数が多い場合の予測を行うため，レイテンシは無視する．

$$Throughput[\text{bytes/s}] = \frac{m}{t} \tag{8}$$

図 9 より，それぞれの通信経路で通信時間が異なり，メッセージサイズが大きくなるにつれて実行時間が増加していることがわかる．図 10 より，4 種類の通信経路における実効転送性能はそれぞれ約 54.6GB/s，10.6GB/s，6.1GB/s，2.7GB/s であった．メモリコピー通信はデータ転送を必要としないため，他の通信経路に比べて非常に高速である．DMA 通信の対ピーク性能は 85.6%であり，VH，IB を経由することで転送性能は低下した．また，DMA 通信，DMA-VH 通信ではメッセージサイズが 256KB と 32MB のとき，IB-VH 通信では 1MB，32MB のときに性能の低下が見られ，通信メッセージサイズによって通信プロトコルが切り替わっていることが考えられるが，さらなる測定などによって明らかにしたい．

次に，`MPI_alltoall` による集団通信について，送受信するメッセージサイズを変化させた時の実行時間，実効転送性能を**図 11**，**図 12** に示す．なお，プロセス数の増加にしたがってプロセスあたりのメモリ量は減少するため，メッセージサイズの最大値は小さくなる．プロセス数 $np = 8$，16，32，64 のときは 8VE を使って割り当てており，$np = 128$，256，512 における VE 数はそれぞれ 16，32，64 である．ここで，`MPI_alltoall` による実効転送性能を以下とする．

$$Throughput[\text{bytes/s}] = \frac{np \cdot m}{t} \tag{9}$$

図 11 より，プロセス数が増加するにつれて実行時間は増加し，図 12 より，プロセス数の増加にしたがって実効転送性能は低下していることがわかる．また，8VE 内での実行ではメッセージサイズが 256KB のときに性能が低下し，8VE 以上では 64KB のときに性能低下がみられた．これは，1 対 1 通信と同様，通信プロトコルが切り替わっていることが原因だと考えられる．使用する VE 数が 16 以上の場合，使用する VE 数，プロセス数に関わらず，実効転送性能は約 0.05GB/s で一定となる傾向がみられる．

また，$np = 256$, $p1 = p2 = 16$ において，`MPI_comm_split` によって 2 つの方向に分割したコミュニケータ $com1$, $com2$ における `MPI_alltoall` による全対全通信の実行時間とその実効転送性能を**図 13**, **図 14** に示す．比較として，$np = 16$，128，256，512 における実行時間，実効転送性能を点線で示す．分割されたコミュニケータ $com1$ では，隣接した 2VE 内の 16 プロセスにおける全対全通信が 16 組独立して行

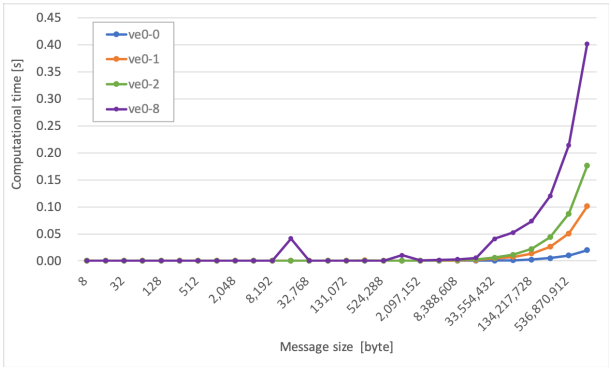

図 9 MPI_sendrecv による通信の実行時間

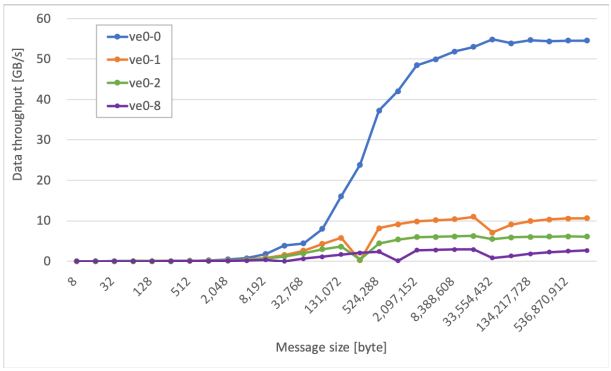

図 10 MPI_sendrecv による通信の実効転送性能

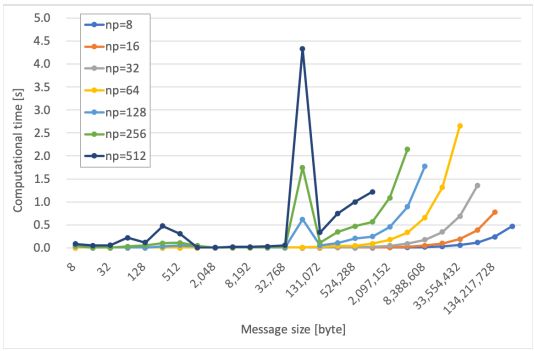

図 11 MPI_alltoall による集団通信の実行時間

われ，$com2$ では，mod(MPI ランク番号,16) の等しいプロセス，つまり 2VE につき 1 プロセスをとった 16 プロセスにおける全対全通信が 16 組行われる．図 14 より，$com1$ における実効転送性能が $np = 16$ における値と近いことがわかる．これは，$com1$ における通信がそれぞれ 2VE 内で済み，他の VE の通信と干渉しないためである．また，$com2$ の実効転送性能は約 0.45GB/s であり，$np = 128$, 256, 512 における値と同様の値になっている．そのため，MPI_comm_split によって分割されたコミュニケータ内の通信についても，使用する VE 数が 16 以上の場合には実効転送性能は同様の値をとるといえる．

4.2 通信時間の予測

実効転送性能を用いることで，AOBA-A 上で

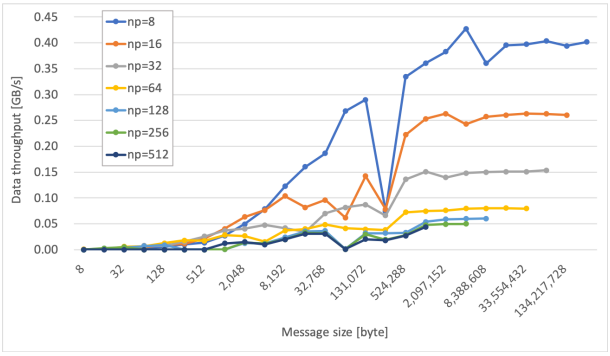

図 12 MPI_alltoall による集団通信の実効転送性能

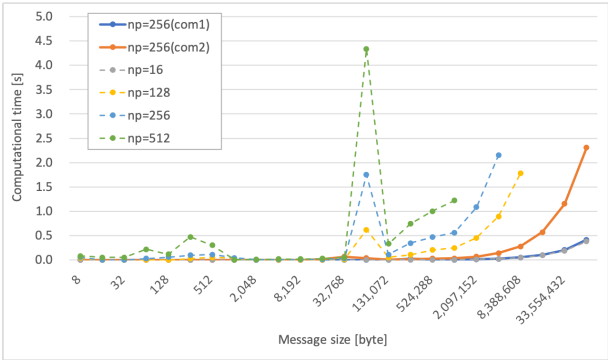

図 13 分割されたコミュニケータ $com1$, $com2$ における MPI_alltoall を用いた集団通信の実行時間

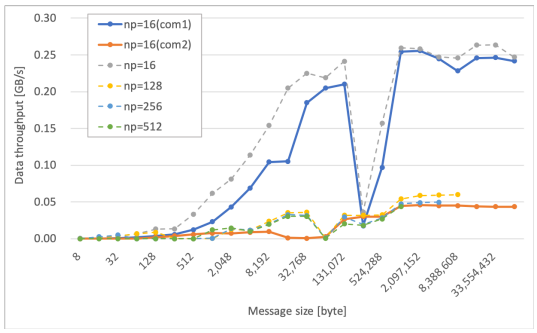

図 14 分割されたコミュニケータ $com1$, $com2$ における MPI_alltoall を用いた集団通信の実効転送性能

MPI_alltoall による通信時間の予測を試みた．MPI_comm_split によってコミュニケータを分割する場合，それぞれのコミュニケータにおける全対全通信のメッセージサイズ $m1$, $m2$ は (6), (7) で求められる．実効転送性能は (9) で計算できるため，実効転送性能を tp[byte/s] とすると，それぞれのコミュニケータにおける通信時間 t_1, t_2 は以下のようになる．ここで，転送するデータは倍精度実数とする．

$$t_1 = \frac{p1 \times m1}{tp} = \frac{p1 \times \frac{n^3}{p1^2 \cdot p2}}{tp} \times 8 \quad (10)$$

$$= \frac{8n^3}{tp \cdot p1 \cdot p2}[s] \quad (11)$$

$$t_2 = \frac{p2 \times m2}{tp} = \frac{p2 \times \frac{n^3}{p1 \cdot p2^2}}{tp} \times 8 \quad (12)$$

$$= \frac{8n^3}{tp \cdot p1 \cdot p2}[s] \quad (13)$$

ここで例として，これまでに実行された最高レイノルズ数における DNS の格子点数 $n^3 = 18,432^3 = (2^{11} \times 3^2)^3$ における FFT の通信時間を考える．DNS 実行のために必要な 1 コアあたりのメモリ量は以下の通りである．また，簡単のために，DNS で確保する必要のある配列データの中で最もサイズの大きいもののみを考える．プログラム中ではそのような配列が 19 個必要となるため，配列のサイズに 19 を掛けている．

$$(2^{11} \times 3^2)^3/np \times 8 \times 19 = \frac{886,464}{np}[\text{Gbyte}] \quad (14)$$

SX-Aurora TSUBASA B401-8 の使用メモリ容量は 1 コアあたり 6GB である．従って，$\frac{886,464}{np} > 6$ を満たし，分割を行える最小のプロセス数（コア数）は，$np = 131,072 (p1 = 512,\ p2 = 256)$ である．1 ノード 64 コアであるため，2048 ノードを用いることで，これまでに実現された最も高いものと同等のレイノルズ数における DNS が実行できると考えられる．この場合，実効転送性能を 0.05GB/s とすると，1 回の FFT における通信時間は以下のように計算できる．

$$t_1 = t_2 = \frac{8 \times 18,432^3}{50,000,000 \cdot 512 \cdot 256} \simeq 7.64[s]$$

したがって，1 回の FFT における通信時間は 7.64 秒と予測できる．

フーリエ・スペクトル法では非線形項の評価のために FFT が 18 回必要であり，Runge–Kutta–Gill 法を用いると，1 ステップあたり 72 回，合計で 550 秒かかる．コア数が非常に膨大であるため，本実験よりもさらに多くのコア数を用いた計測を行うことにより，さらに精度の高い予測できると考えられる．

5. おわりに

本研究では，フーリエ・スペクトル法による非圧縮性乱流 DNS コードを開発し，実行時間の大部分を占める 3D-FFT について SX-Aurora TSUBASA 向けの最適化を行なった．LLC ヒット率を高めるため，転置部分に対してキャッシュブロッキングを適用した．格子点数に比例してストライドが大きくなるループにおいてキャッシュブロッキングは効果的であるが，それ以外のループではベクトル長を大きくする場合が高速であった．また，配列のパディングによって計算部分の計算性能は約 1.3 倍向上した．最適化の結果，格子点数 $n^3 = 4096^3$，プロセス数 $np = 64$ において最大約 3.3 倍高速化された．

AOBA-A において FFT ライブラリである FFTW，FFTE-C との比較を行い，本研究で開発したコードが同等の性能が得られることが確認された．FFT ライブラリに比べて，本研究で開発したコードでは，使用するメモリ量の削減や FFT ルーチンに phase shift 法で必要な演算を引き込むなどの工夫ができるといった利点が挙げられる．

また，FFT における通信時間の予測のため，AOBA-A の 4 種類の通信経路の 1 対 1 通信，複数プロセスでの集団通信の実効転送性能を計測した．`MPI_alltoall` による通信では，使用する VE 数が 16 以下のときにはプロセス数が増加につれて実効転送性能が低下するが，16 以上の場合，プロセス数に関わらず，実効転送性能は同じ値に近づく傾向がみられた．また，`MPI_comm_split` によって分割されたコミュニケータにおける通信でも同様の傾向がみられた．実験で得られた実効転送性能を用いることで，1 回の FFT における通信にかかる時間を予測することができ，これまでに実行された最高レイノルズ数における DNS と同じ格子点数による FFT の通信時間は 7.64 秒であると予測された．

より大規模な DNS 実行の際には，さらに多くのコア数が必要となり，通信時間の割合が大きくなると考えられる．そのため，今後の課題として，コアのマッピングによる通信性能の計測などの通信部分の最適化，コンパクト差分法などの全対全通信を必要としないアルゴリズムを用いたコードの実装が挙げられる．また，本稿の通信時間の予測は，富岳での実行に向けた実行時間の予測にも役立つと考える．

謝辞 本研究の一部は，学際大規模情報基盤共同利用・共同研旧拠点の支援 (課題番号：jh200021)，JSPS 科研費 JP18K11325，及び文部科学省「次世代領域研究開発」（高性能汎用計算機高度利用事業費補助金）量子アニーリングアシスト型次世代スーパーコンピューティング基盤の開発の助成を受けて実施したものです．

参考文献

[1] S. A. Orszag: *Numerical Methods for the Simulation of Turbulence*, THE PHYSICS OF FLUIDS SUPPLEMENT II, pp. 250–257 (1969), https://doi.org/10.1063/1.1692445.

[2] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara and Y. Kaneda: *16.4-Tflops Direct Numerical Simulations of Turbulence by a Fourier Spectral Method on the Earth Simulator*, Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, pp. 16–22 (2002), https://doi.org/10.1109/SC.2002.10052.

[3] Y. Kaneda, T. Ishihara, M. Yokokawa, K. Itakura

ⓒ 2021 Information Processing Society of Japan

and A. Uno: *Energy dissipation rate and energy spectrum in high resolution direct numerical simulations of turbulence in a periodic box*, PHYSICS OF FLUIDS, Vol. 15, No. 2, pp. 21–24 (2003), https://doi.org/10.1063/1.1539855.

[4] T. Ishihara, K. Morishita, M. Yokokawa, A. Uno and Y. Kaneda: *Energy spectrum in high-solution direct numerical simulations of turbulence*, PHISICAL REVIEW FLUIDS, 1(8), 082403 (2016), https://doi.org/10.1063/1.1539855.

[5] K. Ravikumar, D. Appelahms, and P. K. Yeung: *GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism*, SC' 19, November 17-22, (2019), https://doi.org/10.1145/3295500.3356209.

[6] J. W. Cooley and J. W. Tukey: *An Algorithm for the Machine Calculation of Complex Fourier Series*, Mathematics of Computation, Vol. 19, No. 90, pp. 297–301 (1905), https://doi.org/10.1090/S0025-5718-1965-0178586-1.

[7] M. Frigo and S. G. Johnson: *The Design and Implementation of FFTW3*, Proceedings of the IEEE, vol. 93, pp. 216-231 (2005), https://doi.org/10.1109/JPROC.2004.840301.

[8] T. Imamura, M. Aoki, and M. Yokokawa: *Batched 3D-Distributed FFT Kernels Towards Practical DNS Codes*, Parallel Computing: Technology Trends, I. Foster et al. (Eds.), pp. 169 – 178, IOS Press (2020), https://doi.org/10.3233/APC200038.

[9] 青木 聖陽, 今村 俊幸, 横川 三津夫, 廣田 悠輔: 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, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2018-HPC-163, No. 29, pp. 1–7, (2018).

[10] D. Takahashi: FFTE: A Fast Fourier Transform Package, 入手先 ⟨http://www.ffte.jp/⟩ (2021/2/9 閲覧).

ⓒ 2021 Information Processing Society of Japan