平行平板間乱流の大規模直接数値シミュレーション

阿部浩幸*, 松尾裕一*

Large-scale direct numerical simulation of a fully developed turbulent channel flow

Hiroyuki ABE* and Yuichi MATSUO*

Abstract

Direct numerical simulation of a fully developed turbulent channel flow has been carried out using Numerical Simulator III at JAXA. The Reynolds number is set to be Re_τ=1020 where Re_τ is the Reynolds number based on the friction velocity and the channel half width. To the authors' knowledge, this is one of the largest computations ever simulated in conjunction with this configuration, where about 1.4 billion grid points are required. In the present study, to perform large-scale DNS more efficiently, the parallelized computational program has been modified, and its computational performance is presented.

1. 緒言

近年，計算機の性能が飛躍的に向上したことにより，大規模数値シミュレーションが徐々に実行されるようになった．乱流の分野では，直接数値シミュレーション(Direct Numerical Simulation; DNS)による大規模計算が，実験室の実験に相当する数値実験として，数多く行われつつある(Yokokawa et al. [1])．

本研究の対象となる平行平板間乱流(チャネル乱流)の DNS は，形状が非常に単純で，航空機の翼のような壁面に接する乱流(壁乱流)の基礎的性質を有することから，1987 年の Kim ら[2]の壁面摩擦速度 u_τ，チャネル半幅δに基づくレイノルズ数 $Re_\tau = u_\tau\delta/\nu$ =180 の DNS 以降，数多くの計算が行われてきた．我々のグループでも，1997 年から，宇宙航空研究開発機構(旧航空宇宙技術研究所)の数値シミュレータを駆使して，平行平板間乱流の DNS を行ってきた．1999 年には，ベクトル型計算機を使用し，当時世界最高のレイノルズ数 Re_τ =640 の計算を約 3300 万点の格子点数を用いて達成した [3]．そして，2003 年に，この形態で初めて Re_τで 10^3 のオーダーのレイノルズ数となる Re_τ =1020 の平行平板間乱流の大規模 DNS を，スカラー型計算機を使用し，約 14 億点の所要格子点数を用いて達成した [4), 5)]．この際，計算機の種類がベクトル型からスカラー型へ変更になったことに伴い，高効率な計算を実現するために，計算コードをスカラー計算機用にチューニングを行った．そこで，本報では，この大規模 DNS 実行に際して行った並列計算プログラムのチューニング方法，チューニングにより得られた並列計算パフォーマンスについて報告する．

2. 計算手法

流れは十分発達した無限に長い平行平板間乱流を想定し，平均圧力勾配により駆動されるものとした．計算アルゴリズムには，フラクショナルステップ法を用いた．時間進行は，壁垂直方向の粘性項に対しては，2次精度クランクニコルソン法を，その他の項は3次精度ルンゲクッタ法を用いた．他方，空間的離散化には有限差分法を用いた．差分精度としては，流れ方向，スパン方向には 4 次精度中心差分を用い，壁垂直方向は 2 次精度で扱った．設定されたレイノルズ数は，Re_τ =1020 である．計算領域は，$L_x \times L_y \times L_z$=12.8$\delta$×2$\delta$×6.4$\delta$，所要格子数は，$N_x \times N_y \times N_z$ = 2048×448×1536，空間解像度は，Δx^+ = 6.38, Δy^+= 0.15～7.32, Δz^+ = 4.25 に設定した．計算手法の詳細は，Abe et al. [3), 5)]に記した．

3. 計算プログラムの構成

本研究の計算プログラムは，使用言語に Fortran 77，並列化言語には XPFortran[6]を用いた約 20,000 行のプログラムで構成されている．ここで，XPFortran とは並列化言語 VPP Fortran の仕様を包含する並列化言語であり，ディレクティブ挿入型の並列言語である．主として，3 次元 Navier-Stokes 方程式を離散化した 3 重 DO ループの計算から構成されている．本計算にて，特記する解法は，陰的に扱う必要がある部分に対しては 3 重対角行列の直接解法(Tri-Diagonal Matrix Algorithm; TDMA)を用い，ポアソン方程式の解法には，計算アルゴリズム上計算量を低減して高速化を図るために，高速フーリエ変換(Fast Fourier Transformation; FFT) を用いている．

並列化手法には，プロセス並列に XPFortran，スレッド並列に自動並列，一部に OpenMP [7]のハイブリッド型の手法を用いた．プロセス並列の分割の概略図を図 1 に示す．FFT の計算を含めた通常の並列計算は，壁面に並行な面(y 軸に垂直な方向)で分割し(図 1(a))，一方，3 重対角行列(TDMA)の並列計算は，スパン方向(z 軸に垂直な面)に対して分割を行った(図 1(b))．なお，分割軸の変更には，3 次元データを一括で転置転送させて行った．一方，プロセス並列のオペレーションを施していないループに対しては，自動並列化を施した．さらに，コンパイラで自動並列化されない部分，最内側ループが並列化されて効率が悪い部分等については，OpenMP を適用した．

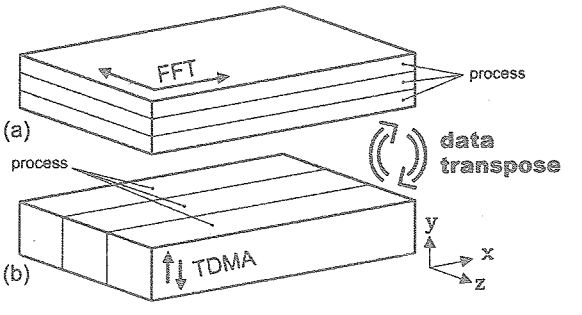

Fig.1 Schematics of domain decomposition for process parallelization: (a) FFT; (b) TDMA.

* 宇宙航空研究開発機構 情報技術開発共同センター

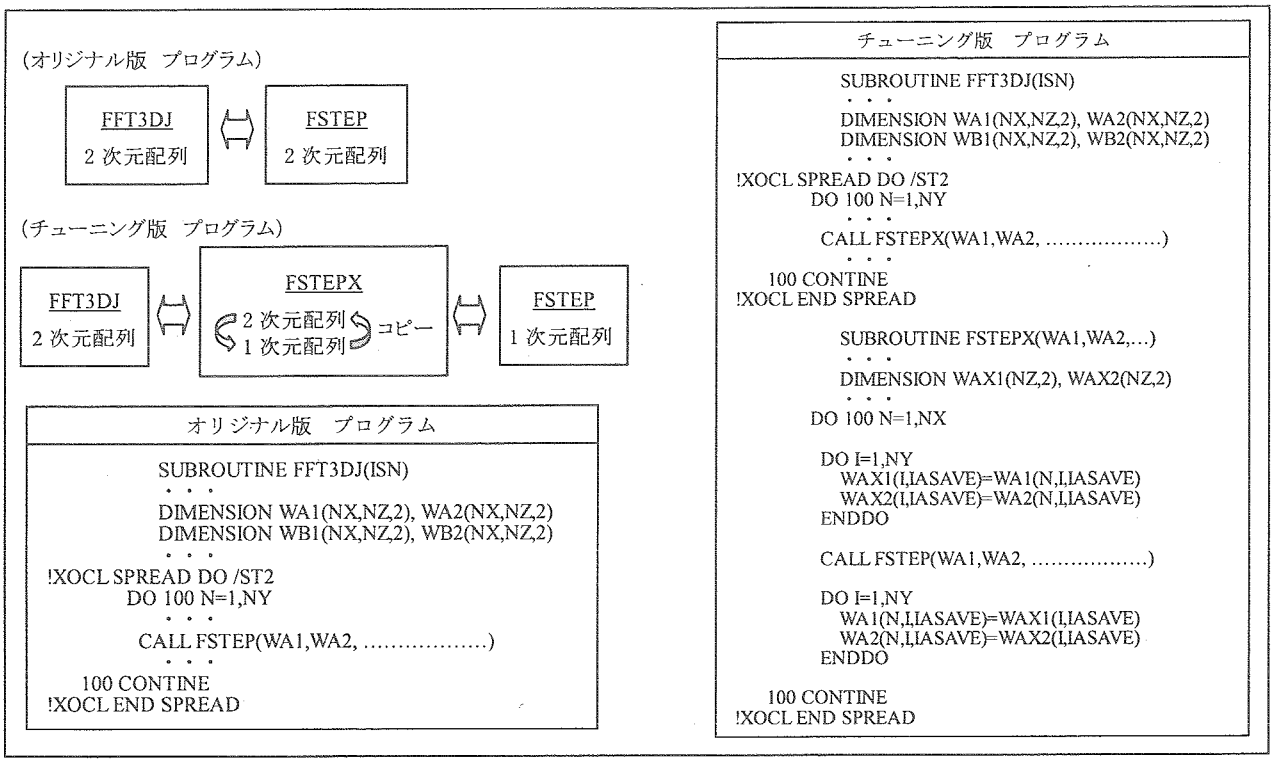

Fig.2 On the modification of the computational code for FFT.

4．計算プログラムのチューニング

大規模 DNS を長時間効率よく実行するためには，計算プログラムのチューニングが必要不可欠となる．本研究の計算プログラムは，ベクトル型計算機用にチューニングを行っていた．そこで，本研究では，スカラー型計算機上で高い実行性能を実現させるために，各種のチューニングを行った．

(1)圧力ポアソン方程式の解法のチューニング
(2)スカラーチューニング
(3)ノード間のデータ転送のコスト削減

以降，本章では，これらのチューニングの詳細について記す．

4．1 圧力ポアソン方程式の解法のチューニング

圧力ポアソン方程式は

$$\nabla^2\phi = Q \tag{1}$$

と表される．式(1)の解法としては，一般的にはSOR法を用いて収束計算が行われる．この場合，値が収束するまでに加速係数に依存した(最低約4，5回程度)計算回数を要する．一方，本計算対象の場合，流れ方向(x 方向)，スパン方向(z 方向)には周期境界条件を用いているため，2 次元 FFT を用いて計算の高速化を図ることが可能である．ここで，式(1)をフーリエ級数展開すると，式(2)のように表され，3 次元の収束計算が 1 次元の収束計算となる．なお，k_x, k_z は，流れ方向，スパン方向の波数であり，式(3)，(4)は，4 次精度差分を用いた場合の等価波数である．また，この波数空間で計算する方法は，高次精度の差分を用いた場合でも，等価波数の値が変化するのみで，実質的な FFT の演算量は変わらない．さらに，本研究においては，壁垂直方向(y 方向)に対しては2次精度差分を用いているため，式(2)に対して 3 重対角行列解法を用いれば，収束計算を要しない．従って，この FFT，行列解法を用いた計算方法は，計算アルゴリズム上で大幅な高速化が行われたことになり，この種の計算では広く用いられている[8),9)]．

4．2 スカラーチューニング

計算中のデータアクセスの順番としては，1次キャッシュ(L1)，2次キャッシュ(L2)，主メモリーの順番になる．ここで，主メモリーへアクセスを行うと，アクセス速度が非常に遅いために，計算速度が著しく低下する．

$$\frac{\partial^2\tilde{\phi}(k_x,j,k_z)}{\partial y^2}+(\lambda_{x,4}+\lambda_{z,4})\tilde{\phi}(k_x,j,k_z)=\tilde{Q}(k_x,j,k_z) \tag{2}$$

$$\lambda_{x,4}=\frac{1}{288(\Delta x)^2}\left\{\cos\frac{6\pi k_x}{N_x}-54\cos\frac{4\pi k_x}{N_x}+783\cos\frac{2\pi k_x}{N_x}-730\right\} \tag{3}$$

$$\lambda_{z,4}=\frac{1}{288(\Delta z)^2}\left\{\cos\frac{6\pi k_z}{N_z}-54\cos\frac{4\pi k_z}{N_z}+783\cos\frac{2\pi k_z}{N_z}-730\right\} \tag{4}$$

オリジナル版　プログラム

```
          . . .
        PARAMETER (NX=1024, NY=256)
        DIMENSION D(NX,NY), E(NX,NY)
          . . .
      DO 100 J=1,NY
      DO 100 I=1,NX
        B2(I,J)=B(I,J)-C(J)*E(I,J-1)
        E(I,J)=A(J)*PB2(I,J)
        D(I,J)=(-D(I,J)+C(J)*D(I,J-1))*PB2(I,J)
100 CONTINE
```

チューニング版　プログラム

```
          . . .
        PARAMETER (NX=1024, NY=256)
        DIMENSION D(NX+4,NY), E(NX+4,NY)

!$OMP PARALLEL PRIVATE(I,J)
          . . .
        DO J=1,NY
!$OMP DO
        DO 100 I=1,NX
          B2(I,J)=B(I,J)-C(J)*E(I,J-1)
          E(I,J)=A(J)*PB2(I,J)
          D(I,J)=(-D(I,J)+C(J)*D(I,J-1))*PB2(I,J)
  100 CONTINE
!$OMP END DO NOWAIT
        ENDDO
          . . .
!$OMP END PARALLEL
```

Fig.3 On the modification of the computational code for TDMA.

従って，スカラー型計算機では，極力，L1，L2 キャッシュを使用して計算を実行する必要がある．本研究では，このうち L2 キャッシュミスを極力低減するようチューニングを行った．具体的には，FFT のルーチンにおいては，従来のベクトル計算機用の計算プログラムではループ長を長くするために大きな配列で計算を行ってきたが，今回，ワーク用の小さな配列にコピーして計算することにより，メモリアクセスを局所化し，キャッシュミスを削減した(図 2)．さらに，従来の基底 2 の複素型 2 次元 FFT を，混合基底の複素型 2 次元 FFT に変更している．この際，複素共役の関係[9),10)]を用いることにより，演算量を，従来の全モードの計算の約半分まで低減し，さらに，内部の配列のコピーについても，最適化を行っている．これらの FFT のチューニングで従来のプログラムに対して，約 2 倍の高速化を達成した．次に，3 重対角行列のルーチンにおいては，1) コンパイラが標準でアンローリングしない DO ループのうち，アンローリングした方が性能向上するものについて，強制的にアンローリングをかける，2) 配列に対してストライドアクセスを行う処理で，ストライドの間隔が 2 のべき乗では，メモリーからキャッシュへのロードの際に，キャッシュラインの使用に偏りが出る場合があるので，配列の宣言要素数を調整することにより偏りを回避する，3) キャッシュミス率をさらに低減させるためにメモリーアクセスが連続になる内側ループに対して OpenMP 化を行う，というチューニングを行った(図 3)．これらの TDMA のチューニングで従来のプログラムに対して，約 30%の高速化を達成した．

プロセス並列向けのチューニングとしては，4．3にて述べるデータ転送と計算のオーバーラップの他に，ロードバランスの不均一による性能劣化を改善するため，1 プロセスずつシリアルに実行する処理を融合し

オリジナル版　プログラム

```
          . . .
        DIMENSION U11(0:NY), U33(0:NY)
          . . .
!XOCL SPREAD DO /ISE
        DO 100 J=1,NY
        DO 100 K=1,NZ
        DO 100 I=1,NX
          U11(J)=U11(J)+UT(I,J,K,1)**2
          U33(J)=U33(J)+UT(I,J,K,3)**2
  100 CONTINE
!XOCL END SPREAD
```

チューニング版　プログラム

```
          . . .
        PARAMETER (NPAD=8)
        DIMENSION U11(NPAD,0:NY), U33(NPAD,0:NY)
          . . .
!XOCL SPREAD DO /ISE
        DO 100 J=1,NY
        DO 100 K=1,NZ
        DO 100 I=1,NX
          U11(1,J)=U11(1,J)+UT(I,J,K,1)**2
          U33(1,J)=U33(1,J)+UT(I,J,K,3)**2
  100 CONTINE
!XOCL END SPREAD
```

Fig.4 On the modification of the computational code for thread parallelization.

て，複数プロセスを同時に実行することにより同期待ちのコストを改善している(図省略)．

スレッド並列向けのチューニングとしては，1 次元配列へのアクセスをスレッド並列化する際に，スレッドあたりの要素数が少ない場合は，複数スレッド間のキャッシュライン競合(False Sharing)が発生することから，配列に 1 次元追加して 1 キャッシュライン分(64 バイト)のすき間を空けることで，False Sharing を回避している(図 4)．

4．3 ノード間のデータ転送のコスト削減

分割軸を変更する場合は，計算中に全データを転置転送しなければならない．本計算においては，TDMA の計算の前後で，1 物理量約 11GB のデータを複数個，転置転送させている．そのため，1 ステップ当たり(ルンゲクッタ 3 段階)のデータ転送量は，約 406GB と非常に大きなものとなっている．従って，データ転送のコストは，全計算の中でも大きなウェイトを占め，従来の計算は，全計算時間の約 25%がデータ転送のために費やされていた．よって，データ転送のアルゴリズム自体は変更せずに，可能な範囲で，データを転送させながら同時に計算を実行させるようにチューニングを行った(図 5)．結果として，従来の計算と比べて，約 5%の転送待ち時間を削減した．

5．結果及び考察

5．1 性能測定条件

性能測定は，JAXA 数値シミュレータⅢの全 56 ノードのうち，52 ノードを占有した状態で行った．この時のピーク性能は，約 8.7TFLOPS となる．数値シミュレータⅢのシステム構成の詳細については，Matsuo et al.[11)]に記した．並列規模は，プロセス並列数は，28，56，112，224 の 4 種類，スレッド並列は，1，2，4，8，16 の 5 種類とし，両者を組み合わせて，28～448 の CPU 数

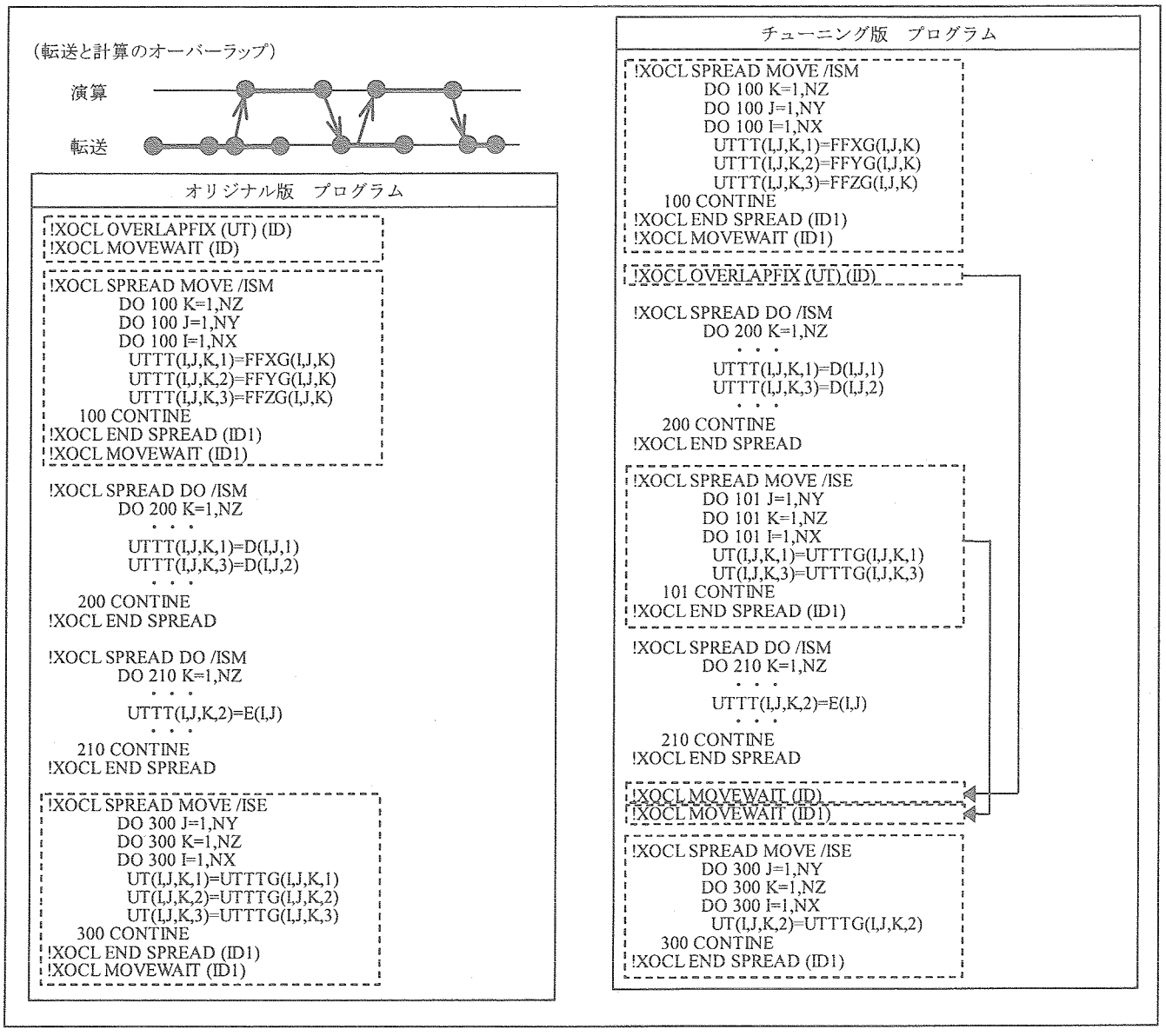

Fig.5 On the modification of the computational code on large data transpose.

の範囲で，2 回の計算回数に対して性能測定を行い，各種の評価を行った．なお，プロセス数は，ロードバランスを重視し，分割方向の格子数(N_y = 448)が均等に分割される数を選んでいる．

5. 2 性能測定結果

図 6 に，28 プロセス×1 スレッドの測定結果を基準に纏めた，スピードアップ性能の測定結果を示す．結果として，プロセス並列の性能の増加率は，スレッド数によらず，56 プロセスまでは比較的良いが，112 プロセスを超えると性能の低下が著しい(図 6(a))．一方，スレッド並列の性能の増加率に関しては，各プロセス数で，4〜8 スレッドまで良いスケーラビリティを示している(図 6(b))．ここで，図 7 に同一 CPU 数の性能を検討すると，448CPU の場合，56 プロセス×8 スレッドの組み合わせが最も性能が高く，224 プロセス×2 スレッドの組み合わせに比べて 20%以上の性能向上が見られた(図 7(b))．この傾向は，112，224CPU で CPU 数を固定して性能を比較しても同様に見受けられ，両者ともに 56 プロセスに性能のピークが現れた．この結果は，本計算の場合，プロセス数を増やし 1 プロセス当たりのデータ転送量を削減するよりも，プロセス数を減らした方が，計算コストを低減できることを示唆している．この原因については，JOB 実行の際の DTU (Data Transfer Unit;数は，各ノード 1 個，I/O ノードのみ 2 個)資源の割り当て方や XPFortran のデータ転送性能の特性に関係しているものと考えられ，5. 4 で再度検討する．

図 8 に，本計算プログラムのメモリアクセスとデータ転送負荷の関係を示す．なお，メモリアクセス比は，平均メモリアクセス時間を平均経過時間で除した値，データ転送比は，データの平均送受信時間を平均経過時間で除した値として定義しており，両者ともにプロセス当たりの値を求めている．図 8 より，本研究の計算プログラムは，航空宇宙分野で広く用いられる CFD コードに比して，メモリアクセス負荷は同程度であるのに対し，データ転送負荷は 100 倍以上高くなっており，データ転送の負荷が高い計算であることが分かる．

Table 1 Computational performance using 448CPUs (56processes with 4threads).

全体(全プロセス)	50GFLOPS
全体(各プロセス)	300MFLOPS
各ルーチン(各プロセス)	通常:800～1000MFLOPS
	FFT:300～400MFLOPS
	TDMA:90～120MFLOPS

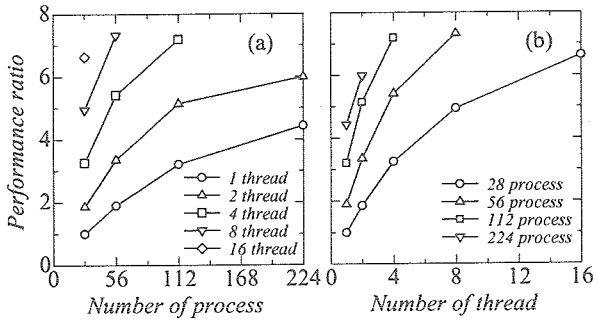

Fig. 6 Performance ratio: (a) as a function of number of process; (b) as a function of number of thread. The performance has been evaluated based on that using 28 CPUs (28processes with 1 thread).

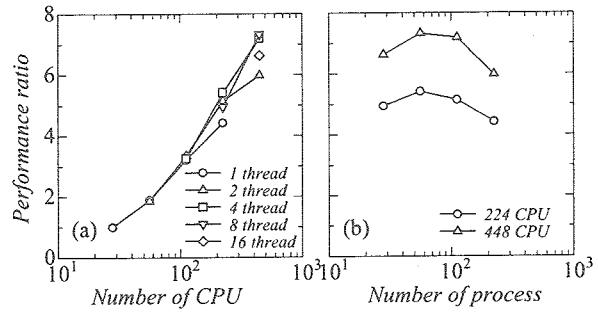

Fig. 7 Performance ratio: (a) as a function of number of cpu; (b) as a function of number of process. The performance has been evaluated based on that using 28 CPUs (28processes with 1 thread).

5．3 本計算プログラムの実行性能(FLOPS 値)

本計算の代表的な実行形態(56 プロセス×8 スレッド)の各ルーチンの実行性能(FLOPS 値)を表 1 に示す．表 1 より，各ルーチンのプロセス単位の性能は，TDMA 以外の各ルーチンで 400MFLOPS を超えており，他の CFD のプログラムと比しても遜色のない性能を示している[11]．一方，TDMA のルーチンでは，90～120MFLOPS まで極端に計算性能が低下している．これは，今回行ったデータ転送のチューニングによっても，依然として，データ転送のコストを隠し切れなかったことが原因と考えられる．この TDMA の性能改善案としては，データ転送負荷を低減する方法としてパイプライン法やハイパープレーン法[12]などの適用が考えられる．今後は，これらの手法が本計算に適用可能かも含めて検討していきたい．

5．4 DTU 数と実行性能との関係

5．2で示したように，本計算プログラムは，56 のプロセス数を用いた場合にピーク性能が現れる傾向がある．ここで，ノード間のデータ転送の仕組みについて考える．図 10 に JAXA 数値シミュレータⅢにおける DTU 数とプロセス数の関係を示す．ノード間のデータ

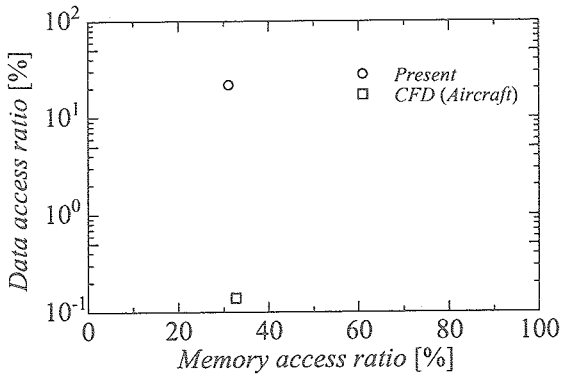

Fig. 8 Relation between memory and data access ratio.

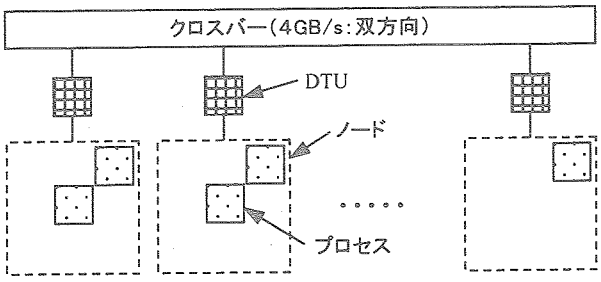

Fig.9 On the way to transpose data using the data transfer unit (DTU).

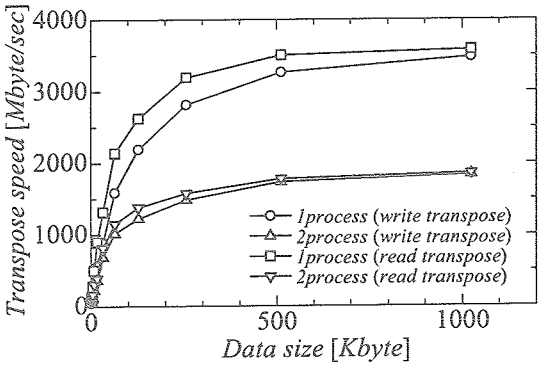

Fig. 10 Data transpose performance for XPFortran using the SPREAD MOVE command.

転送を行う場合，通常は，高速にデータを転送するために DTU を用いる．この場合，データは，まず，各ノード 16 個用意されている DTU の箱へ一度格納される．格納されたデータは，その後一度に 16 個ではなく，順次 1 個ずつ DTU を介して他のノードへ転送される．従って，今回，52 ノードの実行環境で性能測定を行った訳だが，この場合，理論的には最大 52 個のデータが同時に転送できることになる．一方，52 個以上のデータを転送する場合は，必ず，何処かのノードに 2 プロセス以上割り当てられることになる．この場合，自分自身の JOB によって，DTU 資源待ちが生じることになる．よって，DTU を用いた JOB の形態は，以下の 2 つに分類される．

1) 1 ノード，1 プロセスで構成される場合
　ノード間のデータ転送時に自分の JOB による DTU 資源待ちは生じない

2) 1 ノード，2 プロセス以上で構成される場合
　ノード間のデータ転送時に自分の JOB による DTU 資源待ちが生じる

Table 2 Process of generating DNS database

Stage	Grid number	Memory[GB]	I/O[GB]
Stage1	1024×256×1024	164	22.2
Stage2	1024×448×1024	301	38.6
Stage3	2048×448×1024	531	77.0
Stage4	2048×448×1536	819	138.5

1)の場合では，DTU の性能に応じたデータ転送性能が期待され，本計算では，並列化言語に XPFortran を用いているので，特に，SPREAD MOVE コマンドの性能に依存することになる．ここで，図 10 に SPREAD MOVE コマンドの基本性能を示す．図 10 から，プロセス数が多くなるにつれデータ転送性能が低下する傾向が見受けられる．これは，MPI では見られなかった現象であり[11]，XPFortran のオーバーヘッドによるものと考えられる．この結果は，1)の場合でも，プロセス数が増加するにつれて，データ転送のオーバーヘッドが増加することを示唆している．一方，2)の場合では，1)のデータ転送時間に加えて，DTU 資源待ちの時間が加算される．この場合，1回のデータ転送量が小さい時には，DTU 資源待ち時間はさほど問題にならないが，今回のように1回のデータ転送量が数十 GB にも及ぶ問題規模の計算では，1ノードに複数個のプロセスが存在すると，自分自身の JOB により有意な DTU 資源待ち時間が生じ，計算パフォーマンスの劣化を引き起こすことになる．従って，大容量のデータ転送を有した計算では，極力，52 プロセスに近くなるようプロセス数を低減させる必要がある．

以上を総括すると，本計算の場合は，ピーク性能が 56 プロセスに現れることから，上記 2)の影響が非常に大きかったものと考えられる．従って，本計算のようにデータ転送量が非常に多い計算では，自分自身の JOB による DTU 資源待ちを生じないように，可能な限りプロセス数を少なくする必要がある．一方，高速に計算を実行するには，数多くの CPU を用いなければならない．この場合，今回のような大容量のデータ転送を有する計算では，プロセス並列度数を増加させることは実行性能を低下させることから，スレッド並列度数を増加させる必要があり，この時，スレッド並列化効率を向上させる必要がある．

5．5 DNS データベースの作成及び本計算の実行

平行平板間乱流の大規模 DNS の本計算を行う前に初期データとなる DNS データベースを構築しなければならない．本研究では，低レイノルズ数の DNS データベースをもとに表 2 に示した 4 段階を経て Re_τ=1020 までレイノルズ数を上げた．DNS データベース構築に要した総計算ステップ数は，約 50 万ステップ，総計算時間は約 5 ヶ月である．その後，作成した DNS データベースを用いて，平均流速などの各種乱流統計量を収集するための DNS を実行した．この計算も統計的に定常な状態まで計算を継続する必要があるので，非常に長い計算時間を要する．本研究においては，統計量を収集するのに約 67,500 ステップの計算，約 5 ヶ月間の計算時間を要した．

6．結言

本報では，宇宙航空研究開発機構（JAXA）の数値シミュレータ III を用いて行った平行平板間乱流の大規模直接数値シミュレーションの実行に際して行った並列計算プログラムのチューニング方法，並列計算性能について示した．

計算プログラムのチューニング方法に関しては，データの局所化に対する方法，L2 キャッシュミスの低減方法，大容量データ転送のチューニング方法などの実例を示し，それぞれのチューニングが有効であることを示した．

チューニング後の計算プログラムの実行性能に関して，各種の評価を行い，本計算では，プロセス数を単に増加させた計算よりも，プロセス並列とスレッド並列を組み合わせた計算の方が高い実行性能が得られることを示した．この原因を調べた所，本計算は，航空宇宙分野の CFD に比べて 100 倍以上のデータ転送量を有し，これが実行性能に大きく影響していることが判明した．また，56 プロセスを用いた場合にピーク性能が得られる理由については，DTU 数，XPFortran の転送性能の双方から検討した所，DTU の資源待ち時間が大きく影響していることが分かった．すなわち，1ノード 1 プロセスの状態に近い 56 プロセスを用いた場合に，DTU 資源待ち時間が最小となり，ピーク性能が現れている．

今後はさらに大規模な計算を行う予定であり，より一層計算効率を向上させる必要がある．特に，データ転送方法に関しては，より大きな問題規模の計算に適用できる手法を開発していきたい．また，今回は大規模計算の並列計算プログラムのチューニング及び並列計算の実行に焦点をあてたが，このような大規模計算では，大容量のデータが算出されるため，後処理方法も確立して行きたい．

本研究の計算は，航空宇宙研究開発機構の数値シミュレータIIIを用いて行った．

謝辞

本研究は，東京理科大学 河村洋教授との共同研究のもとに行われた．また，計算全般にわたり富士通株式会社の稲荷智英氏に多大なご支援を頂いた．記して謝意を表す．

参考文献

1) M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and K. Kaneda: http://www.sc-2002.org/paperpdfs/pap.pap273.pdf (2002).
2) J.Kim, P.Moin & R.Moser: J. Fluid Mech. 177 (1987) 133.
3) H.Abe, H.Kawamura & Y.Matsuo: Trans. ASME J. Fluids Eng. 123 (2001) 382.
4) 阿部浩幸，松尾裕一，河村洋：航空宇宙数値シミュレーション技術シンポジウム 2003 (2003) 22.
5) H.Abe, H.Kawamura & Y.Matsuo: Int. J. Heat and Fluid Flow 25 (2004) 404.
6) 富士通 PRIMEPOWER ソフトウェア製品紹介(XPFortran): http://primeserver.fujitsu.com/primepower/products/soft/opt/xpf_2x/.
7) OpenMP: http://www.openmp.org/.
8) 梶島岳夫: 乱流の直接数値シミュレーション，養賢堂 (1999).
9) P.Moin: Fundamentals of engineering numerical analysis, Cambridge Univ. Press, (2001).
10) C.V.Loan: Computational frameworks for the Fast Fourier Transform, SIAM (1992).
11) Y.Matsuo: Proc. ISHPC 2003, LNCS 2858, pp.39-53, Springer-Verlag (2003).
12) M.Kobayashi: 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 HPCS2005 (2005), pp.73-80.