FrontFlow/blue の勾配計算カーネルの スーパーコンピュータ「京」上でのチューニング

熊畑清† 井上俊介† 南一生†

有限要素法による汎用流体解析ソフト FrontFlow/blue は圧力を節点上に定義するモードと，要素中心に定義するモードの 2 つのモードを備えている．このうち圧力を要素中心に定義するモードでは要素中心の圧力から要素頂点上での圧力勾配ベクトルを計算する部分が計算時間のおよそ 30%以上を費やしており主要な部分(カーネル)となる．この勾配計算カーネルは要素が持つ参照節点リストを介して，節点が持つ物理量を保持する配列にアクセスするリストアクセスとなっておりスーパーコンピュータ「京」で性能を出すためにはキャッシュの利用効率を考慮したチューニングが必要であった．本稿ではこの勾配計算カーネルのスーパーコンピュータ「京」上での性能評価とチューニング技法及び測定結果について述べる．

Performance Tuning for Gradient Kernel of the FrontFlow/blue on the K computer

KIYOSHI KUMAHATA† SHUNSUKE INOUE†
KAZUO MINAMI†

General purpose fluid simulation software FrontFlow/blue based on finite element method has two modes to treat pressure. One defines pressure on node and another defines pressure on element center. For the element pressure mode one, of the kernels that mean governing part of total computation time is calculating pressure gradient. This kernel indirectly access to memory via list-access. Hence an improving performance required special treatment of effective cache utilization. This paper show the way for performance estimation and tuning on the K computer.

1. はじめに

理化学研究所では，スーパーコンピュータ「京」（以降「京」）の共用開始にむけて，実際のアプリケーションを使用したシステム性能の実証を進めてきた．性能実証のためのアプリケーションは「京」の汎用性を活かし，様々な応用分野のアプリケーションが幅広く高い性能を発揮できることを実証できるように選択されている．また今後の計算機開発に役立つように選択に当たっては，基礎方程式・計算アルゴリズムに起因する並列化手法の違い，演算コードの要求演算量及び要求メモリバンド幅といった計算機科学的特性の観点も含めた．これら 2 つの観点を基に，地球科学分野のアプリケーションを 2 本（NICAM[1], Seism3D[2]），ナノ分野のアプリケーションを 2 本（PHASE[3]，RSDFT[4]），工学分野のアプリケーションを 1 本（FrontFlow/blue[5]），物理分野のアプリケーション 1 本（LatticeQCD[6]）の合計 6 本のアプリケーションが選択された．筆者はこれら 6 本のアプリケーションのうち特に有限要素法による汎用流体解析ソフトである FrontFlow/blue（以降 FFB）の「京」上でのチューニングを行っている．

本稿では始めに有限要素法による汎用流体解析ソフト FrontFlow/blue の概要について触れたのち，「京」の CPU 情報を紹介し，本稿で対象とする FFB の主要な演算部である勾配計算部の性能評価と測定結果を示し，チューニング手法及びチューニング手法による性能向上を評価する．

2. FrontFlow/blue

FrontFlow/blue（FFB）は非圧縮性流体の非定常な流れを高精度に予測可能な Large Eddy Simulation（LES）[7]に基づいた汎用流体解析コードである．形状適合性に優れた有限要素法による離散化を採用し，空間及び時間について 2 次精度を持ち，様々な要素タイプ及び座標系・格子系を扱えるためポンプやファン等のターボ機器や複雑形状周りの非定常乱流解析が可能である．さらに Curle の式[8]に基づいた流体騒音の予測や，均質流体モデルによる流体の体積率の時間発展を計算し，キャビテーションを伴う非定常流れを解析可能である．

FFB はこれまでに地球シミュレータや，東京大学の T2K，計算科学振興財団の FOCUS スパコンシステム[9]等の様々な大規模並列計算機での実績があり，これらの計算機上で十分な性能が出せるようチューニングされているが，「京」上での実績がなく，コードにはハードの性能を十分に引き出すチューニングがなされていなかった．そこで我々は「京」上での FFB のチューニングを行った．一般に有限要素法による流体シミュレーションでは並列性能は領域分割によるドメイン間のバランスに依るところが大きいが，FFB は大規模並列計算を指向して開発されており，「京」上においても良好な並列性能を示す．一方で「京」の CPU

*† 理化学研究所
RIKEN

に対するチューニングが十分に行われていないため，本稿では CPU 単体性能の観点から演算性能のチューニングについて述べる．

流体解析では基本的に流速と圧力という 2 種の未知変数を取り扱うが，FFB は圧力の取り扱い方法として，圧力を節点上に定義する節点圧力モードと，圧力を要素中心に定義する要素圧力モードの 2 種類のモードを備えており，パラメータファイル中の記述により切り替えて用いることが可能である．

図 1 は要素圧力モードでおよそ 10 万個の 4 面体要素と 8000 個の 6 面体要素および 600 個の 5 面体要素からなる問題を計算した際の各処理部分の実行時間の内訳の上位 10 件を示したものである．最も多い 14.11 秒（全体のおよそ 30%）の時間を占めている callap._PRL_2_が 4 面体要素の勾配計算部分，続いて 13.04 秒（27%）を占める calaxc_が疎行列ベクトル積，8.32 秒（17%）を占める fld3x._PRL_1 は 4 面体要素の発散計算部分である．ここから判るとおり要素圧力モードでは勾配計算が実行時間のおよそ 30%（6 面体要素の勾配計算も含めると 35%）を占めており最重要な部分であると判る．そこで勾配計算をアプリケーションの核となる部分（カーネル）としてとらえ，勾配計算カーネルの性能評価とチューニングを行った．

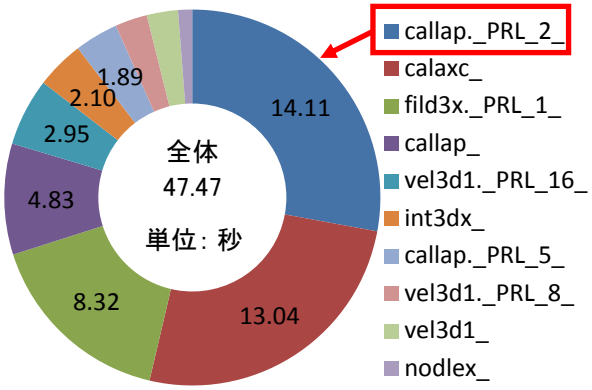

図 1 節点圧力モードでの計算時間内訳（上位 10 件）

なお節点圧力モードにおいて最も支配的となるカーネルは図中 calaxc_と示される疎行列ベクトル積計算カーネルであるが，疎行列ベクトル積計算カーネルの「京」におけるチューニングについては参考文献[10]に詳しい．

3. 「京」の CPU 概要

CPU 単体性能チューニングに際して「京」の CPU について述べる．「京」の 1 個の計算ノードは，1 個の CPU（富士通製 SPARC64 TMVIIIfx）[11][12][13][14][15]，16GByte のメモリ，計算ノード間のデータ転送を行うインターコネクト用 LSI（ICC: Inter-Connect Controller）で構成されている．CPU は，8 つのプロセッサコア，コア共有の 6MByte の 2 次キャッシュメモリ，メモリ制御ユニットからなる．各コアはラインサイズ 128Byte，2way 構成の 32KByte の L1D キャッシュ，4 つの積和演算器，256 本の倍精度浮動小数点レジスタを備える．SIMD 命令により一度に 2 つの積和演算器を同時に動作させることができ 2 つの SIMD 命令を同時に発効可能なためコア当たりクロックサイクル毎に 8 個の浮動小数点演算が可能．よって CPU 全体で 128GFLOPS の性能を持つ．理論メモリバンド幅は 64GByte/秒である．

4. 勾配計算カーネル

FORTRAN で記述された 4 面体要素の勾配計算カーネルのソースコードを図 2 に示す．

```
DO ICOLOR=1,NCOLOR(1)
   IES=LLOOP(ICOLOR  ,1)+1
   IEE=LLOOP(ICOLOR+1,1)

   DO IE=IES,IEE・・・・・・・ここで並列
      IP1=NODE(1,IE)
      IP2=NODE(2,IE)
      IP3=NODE(3,IE)
      IP4=NODE(4,IE)
      SWRK=S(IE)

      FX(IP1)=FX(IP1)-SWRK*DNX(1,IE)
      FX(IP2)=FX(IP2)-SWRK*DNX(2,IE)
      FX(IP3)=FX(IP3)-SWRK*DNX(3,IE)
      FX(IP4)=FX(IP4)-SWRK*DNX(4,IE)

      FY(IP1)=FY(IP1)-SWRK*DNY(1,IE)
      FY(IP2)=FY(IP2)-SWRK*DNY(2,IE)
      FY(IP3)=FY(IP3)-SWRK*DNY(3,IE)
      FY(IP4)=FY(IP4)-SWRK*DNY(4,IE)

      FZ(IP1)=FZ(IP1)-SWRK*DNZ(1,IE)
      FZ(IP2)=FZ(IP2)-SWRK*DNZ(2,IE)
      FZ(IP3)=FZ(IP3)-SWRK*DNZ(3,IE)
      FZ(IP4)=FZ(IP4)-SWRK*DNZ(4,IE)
   ENDDO
ENDDO
```

図 2 勾配計算カーネルソースコード

本実装ではデータ依存が発生する隣接要素の同時処理を，互いに隣接する要素を異なるグループ（カラーと呼ぶ）に分け，演算を隣接関係の全くない要素の集合であるカラー毎に行うことで並列に実行可能にするカラーリングを施してある．そのため第一のループ ICOLOR はカラーリングした各カラーについて回転するループ．配列 LLOOP は全要素の中での各カラーに属する要素の開始・終了番号を保持している配列である．第二のループ IE は各カラー内の要素について回転するループであり，配列 NODE は要素 IE について要素が頂点としてどの節点を参照しているか節点番号を保持する参照節点リストであり INTEGER*4 の配列，配列 S は要素 IE について要素中心で定義された圧力を保持している REAL*4 の配列，配列 DNX, DNY, DNZ は要素 IE について 4 頂点それぞれでの形状関数の X, Y, Z 各方向の導関数を保持している REAL*4 の配列，配列 FX, FY, FZ が要素 IE の各頂点上での圧力勾配ベクトルを保持する REAL*4 の配列である．なおここから判る通り単精度計算であるが，これまでにファンによる空力騒音の予測[16]や船体周りの流れ解析[17]などの実績があり精度には問題がない．配列

FX, FY, FZ は配列 NODE からの節点番号 IP1, IP2, IP3, IP4 を介してリストアクセスされる配列である．ここに示したソースでは 2 次元配列である NODE, DNX, DNY, DNZ の 1 次元目は 4 面体要素の頂点数である 4 までしかアクセスしていないが，実装上はこれらの配列は全ての要素タイプで共通して使っているため，多様な要素タイプに対応し，かつベクトル機でのバンクコンフリクトの回避という経緯から 1 次元目のサイズは 9 となっている．カラーリングにより最内ループ IE 内の演算には回転間のデータ依存性がないため，最内ループ IE はマルチスレッドで処理され，複数回転間の演算を重ねて処理する SIMD 命令化とソフトウェアパイプライン化がコンパイラにより適用されているが，最内ループ内において SIMD 命令で処理される積和演算命令の比率が低く効果は小さい．

図 3 に要素に関する配列である LLOOP, NODE, S, DNX, DNY, DNZ の構造を示す．図左側のような互いに隣接していない要素 1,2,3,4 がカラー1 に，要素 5,6,7,8 がカラー2 というようにカラーリングされたメッシュに対して，要素に関する配列は同じカラー内の要素が連続するよう図右側のような並びをしている．なお 1 次元目のサイズが 9 である 2 次元配列の NODE, DNX, DNY, DNZ については図右側で要素 1 個分の情報を意味する 1 個の四角形は 9 個の REAL*4 あるいは INTEGER*4 からなる．

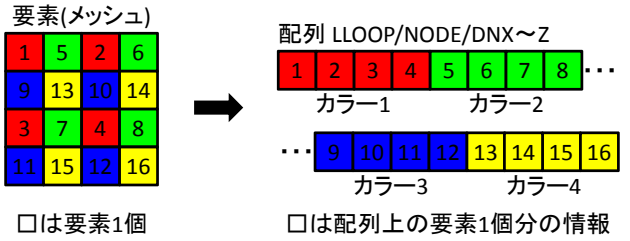

図 3 要素の情報を保持する配列の構造

ここでは本実装の要求バイト量と要求演算量の比（Byte/Flop，B/F 値）から理論性能の上限を参考文献[10]にある手法に基づいて算出する．この手法では L1D キャッシュ・L2 キャッシュ・メインメモリ間のレイテンシを比較し，最もペナルティの高いメモリからのデータ転送量を要求演算量と定義する．はじめにリストアクセス故に入力データ依存でアクセスが不規則となる配列 FX, FY, FZ については，再利用性がある配列であるため，性能上限を推定するに際し再利用性が極めて高く，常に L1D キャッシュ上に乗り続けているという理想的な状態を仮定しメモリプレッシャー無視する．シーケンシャルアクセスである配列 S については最内ループ IE の 1 回転では 4Byte を必要とする．1 回目のアクセスで L1D キャッシュミスし，1 ラインサイズである 128Byte がメモリから転送され，32 回転分のデータを L1D キャッシュでまかなえる．すなわちキャッシュミスの頻度は 32 回転につき 1 回である．配列 NODE についても 1 回目のアクセスで L1D キャッシュミスし，128Byte がメモリから転送されるが，2 次元目のサイズが 9 であり最内ループ IE の 1 回転毎に 36Byte ずつのストライドアクセスとなるため，図 4 に示すように 128Byte では約 3.56 回転を L1D キャッシュのみでまかなえる．配列 S と同様に考えるとキャッシュミスの頻度は 32 回転につき 9 回となる．

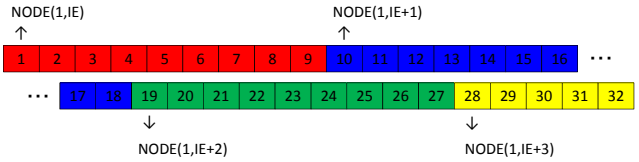

図 4 配列 NODE へのアクセスの際の 1 ライン内分布
128 バイトで 32 個の INTEGER*4 を含むが，ストライドアクセスのため 3.56 回転分しかまかなえない

配列 DNX, DNY, DNZ のそれぞれついても 1 次元目のサイズよりキャッシュミスの頻度は 32 回転につき 9 回である．このようにストライドアクセスになる場合は，ソース上では参照されていない配列要素についてもメモリ転送が生じることを考慮する必要がある．

以上より最内ループ IE の 32 回転でのメモリからの転送量は配列 S で 1 回，配列 NODE, DNX, DNY, DNZ で 9 回ずつの L1D キャッシュミスより $(1+4\times9)\times128\text{Byte}=4736\text{Byte}$ となる．一方で 32 回転分の演算量は $24\times32=768\text{FLOP}$．以上より本カーネルが要求する B/F の値は 4736/864=6.17 と見積もった．

「京」の CPU は 3 節で記したように 128GFLOPS の演算性能と，64GByte/s のメモリバンド幅を持つが，STREAM ベンチマーク[18]で測定された実効メモリバンド幅は 46.6GByte/s であるため，実効的な B/F 値を 46.6/128=0.36 とすると，カーネルの要求 B/F 値 6.17 は「京」の実効 B/F 値 0.36 よりも高いため，演算に対してデータ供給が間に合わず，CPU のピーク性能 128GFLOPS で実行することが出来ない．よって本カーネルの理論性能は「京」のピーク性能 128GFLOPS の 0.36/6.17=5.83%であると見積もられる．

ここでは，性能評価及びチューニングに際して評価対象である 4 面体要素の勾配計算カーネルのみを単体で実行出来るようカーネルソースの切り出しを行い，入力データとしてはおよそ 82 万個の 4 面体要素からなるデータを用いた．測定の結果，何も手を入れていないソースではコンパイラによる自動並列化が適用されずシングルスレッド実行となり 0.6%，コンパイラ指示子の挿入によりマルチスレッド実行を行っても 1.6%，メモリスループットは STREAM ベンチマークでの値 46.6GByte/s に対し 10.29GByte/s であった．

図 5 はプロファイラで取得したマルチスレッド実行時の各スレッドの実行時間内訳である．全てのスレッドでメモリアクセス関連の待ちが占める割合が 67%から 79%と高かった．また「京」の L1D キャッシュは 1 ラインサイズが 128 バイトであり，理想的な条件では 4 バイトのデータ 32

個をキャッシュミスすることなくアクセスすることができるため, L1D キャッシュミス率の理論値は 3.125%となるが, ここでは L1D キャッシュミス率は 21.3%と高い値を示しており，キャッシュメモリの利用効率が悪いことがボトルネックになっていると推測される．

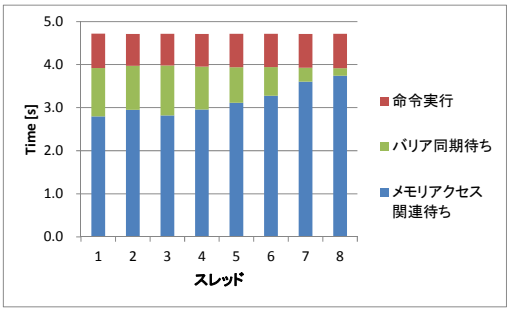

図 5 スレッド毎の実行時間内訳

5. メモリアクセス飛びの低減

キャッシュメモリの利用効率が悪い原因として考えられるのは，完全に理想的な状態では全て L1 キャッシュ上に乗ると見なせるため無視できるはずの配列 FX, FY, FZ へのアクセスが，ここで用いている現実的な入力データでは理想状態から大きくずれており，そのためメモリアクセスに飛びが発生していることが原因と考えられる．これは要素が 4 つの頂点として参照している節点の番号が互いに離れているため，最内ループでの配列 FX, FY, FZ へのリストアクセスの際に L1D キャッシュの 1 ライン分の利用効率が悪いためである．

また最内ループをマルチスレッドで実行するための要素のカラーリングを計算領域全体に対して行っており，1 個のカラーに対しての最内ループ全回転の間に処理する要素は幾何的に遠いものが現れるため，やはりキャッシュのラインアクセス効率が悪くなっていることも考えられる．

そこで幾何的に近い節点はメモリ上でも近い位置に置かれるよう節点のリオーダリングと，最内ループが幾何的に近い要素で完結するよう計算領域の分割を実施した．

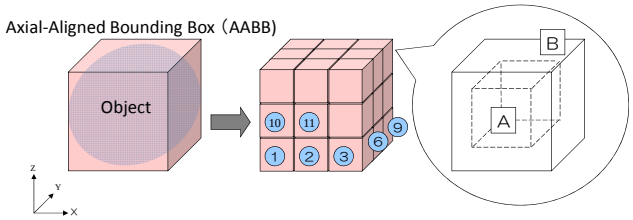

図 6 節点リオーダー

リオーダリングは行列・ベクトル積の前処理としてバンド幅の縮小や，直接法を用いる際の fill-in の抑制を目的として行や列の並びを変更する手法としてよく知られており，Minimum Degree 法 [19]や Nested Dissection 法 [20], グラフ理論を用いた手法 [21]などが知られている．一方で本カーネルは行列を作成せず要素毎に計算する実装となっているため，行や列を並べ替える操作は，そのまま節点の番号を変更する操作に相当する．ここでは節点の座標を元に，近隣の節点は近い番号を持つようリオーダリングを行う．以下に手法を述べる．

図 6 は節点リオーダリングの概要図である．まず図左に示すよう Axis-Aligned Bounding Box (AABB)と呼ばれる，形状を包含する最小の直方体を定義する．この AABB は，仮想的に図中にあるように複数の矩形に区切られ，各矩形は X, Y, Z 軸の順にシーケンシャルな番号が付けられる．ここでは AABB を 10×10×10 に分割した．初期状態では矩形内に位置する節点の番号は入力データとして与えられるメッシュデータに依存して不連続となり得るが，矩形番号 1 から順に図右側のように矩形内を外側領域と内側領域に分け，矩形内部の節点の番号を各内側領域内，外側領域内の順で連続するよう更新する．矩形内を内外に分けるのは矩形間をまたいだ節点参照をする要素について節点番号の離れ方を小さくする狙いがあり，ここでは矩形 1 辺の長さを L とした際, 矩形表面から厚さ L/10 未満の領域を外側領域，それ以外を内側領域と定義した．次の矩形内での節点番号は先の矩形で最後に付与した節点番号＋ 1 から始める．このように節点番号を更新した後，配列 NODE で保持される各要素の頂点番号を更新された節点番号へ更新する．

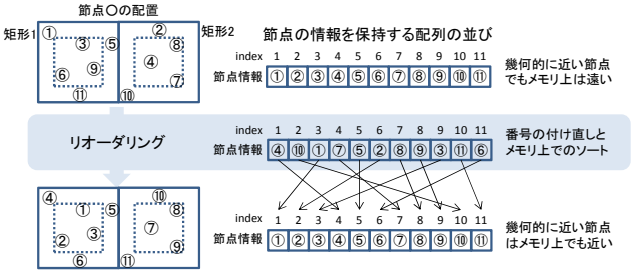

図 7 メモリ配置
リオーダリングにより幾何的に近い節点の
情報はメモリ上でも近い場所に置かれる

さらに配列 FX, FY, FZ 等の節点の情報を保持する配列については図 7 に示すように，更新後の節点番号順にソートすることで幾何的に近い節点に関する情報はメモリ上でも近い位置に置かれるようにする．

次いで最内ループが幾何的に近い要素で完結するように計算領域の分割を行う．図 8 は計算領域の分割の概要図である．図左側に示される計算領域は図 6 と同様に AABB である．この AABB は図中央に示されるように複数の小領域に分割される．なお矩形分割では入力データの形状によっては小領域間で大きなインバランスを引き起こす可能性があるため，汎用のグラフ分割ライブラリである METIS[22]を用いて，要素数がバランスするよう小領域分割を行うコードも開発しているが，今回扱った検証用の入力データが元々直方体に近い形状であったため本稿では矩形分割を採用している．

マルチスレッド実行のためのカラーリングはこれまでは計算領域全体で要素のカラーリングを行ってきたが，ここでは図右側のように分割した各小領域単位に行う．そのためループ構造はこれまでのカラーについてのループとカラー内の要素についてのループの 2 重ループ構造から図 9 に示すように小領域についてのループ IGROUP，小領域内のカラーについてのループ ICOLOR，そしてカラー内の要素についてのループ IE の 3 重ループ構造へと変更される．コンパイラによる最内ループ IE 内演算の SIMD 命令化とソフトウェアパイプライン化は引き続き適用されている．

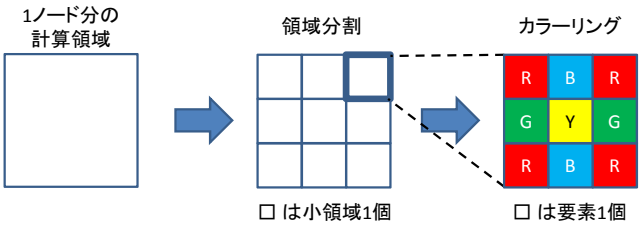

図 8 領域分割（処理単位の局所化）

```
DO IGROUP=1,NGROUP(1)
  DO ICOLOR=1,NCOLOR(IGROUP,1)
    IES=LLOOP(IGROUP,ICOLOR  ,1)+1
    IEE=LLOOP(IGROUP,ICOLOR+1,1)

    DO IE=IES,IEE・・・・・・・ここで並列
      IP1=NODE(1,IE)
      IP2=NODE(2,IE)
      IP3=NODE(3,IE)
      IP4=NODE(4,IE)
      SWRK=S(IE)

      FX(IP1)=FX(IP1)-SWRK*DNX(1,IE)
      FX(IP2)=FX(IP2)-SWRK*DNX(2,IE)
      FX(IP3)=FX(IP3)-SWRK*DNX(3,IE)
      FX(IP4)=FX(IP4)-SWRK*DNX(4,IE)

      FY(IP1)=FY(IP1)-SWRK*DNY(1,IE)
      FY(IP2)=FY(IP2)-SWRK*DNY(2,IE)
      FY(IP3)=FY(IP3)-SWRK*DNY(3,IE)
      FY(IP4)=FY(IP4)-SWRK*DNY(4,IE)

      FZ(IP1)=FZ(IP1)-SWRK*DNZ(1,IE)
      FZ(IP2)=FZ(IP2)-SWRK*DNZ(2,IE)
      FZ(IP3)=FZ(IP3)-SWRK*DNZ(3,IE)
      FZ(IP4)=FZ(IP4)-SWRK*DNZ(4,IE)
    ENDDO
  ENDDO
ENDDO
```

図 9 領域分割適用後カーネルソース

以上の処理により最内ループで参照する要素・節点が空間的・メモリ的に局所化され，メモリアクセスが局所化される．表 1 に今回用いた領域分割数，各分割数で 1 小領域当たりに含まれる平均要素数，平均要素数分だけ最内ループの演算を実行するのに要するメモリ量，8 スレッド（8 コア）で処理する際の 1 スレッド当たりの必要メモリ量を示す．ここで必要メモリ量は，最内ループで 1 要素当たりに実際にアクセスするデータ量が配列 NODE，S，DNX，DNY, DNZ で 148Byte，FX, FY, FZ については 4 節点×X, Y, Z の 3 成分で 48Byte より合計 198Byte となることから求めた．また「京」の各コアが持つ L1D キャッシュサイズは 32KByte であるため，ケース 1 の分割数はスレッド当たり必要メモリ量が約 L1D キャッシュに収まるように定め，それ以降のケースでは約 2 倍ずつ増大するよう設定した．ケース 11 は領域分割と節点リオーダーを行っていないオリジナルコードの状態に相当する．

図 10 は最内ループで配列 FX，FY，FZ にアクセスする際のインデックスの飛び量を表した図である．横軸は小領域当たりの平均要素数，縦軸は最内ループ 1 回転当たりについて配列 FX，FY，FZ のインデックスである IP1，IP2，IP3，IP4 の値の最大値・最小値の差の，全回転についての平均値であり，この値が小さいほど 1 個の要素から参照している節点番号が近い事を意味する．図で左側，すなわち領域分割数が多いほどインデックスの飛びが小さく収まることが示された．これによりキャッシュ利用効率の大幅な改善が期待でき，事実，最も領域分割数が多いケースでの実測の L1D キャッシュミス率は 5.44%と，先の 21.3%と比して大きく改善した．一方でピーク性能比は 0.13%，メモリスループットは 1.08GByte/s と大幅に低下した．

表 1 領域分割数と要素数，必要メモリ量

ケースNo	小領域分割数	平均要素数/小領域	アクセスデータ量 [KB]	1スレッド当りメモリ [KB]
1	640	1291	247.1	30.9
2	320	2583	494.4	61.8
3	160	5166	988.8	123.6
4	80	10333	1977.8	247.2
5	40	20666	3955.6	494.5
6	20	41332	7911.2	988.9
7	10	82665	15822.6	1977.8
8	5	165331	31645.4	3955.7
9	3	275552	52742.4	6592.8
10	2	413328	79113.6	9889.2
11	1	826656	158227.1	19778.4

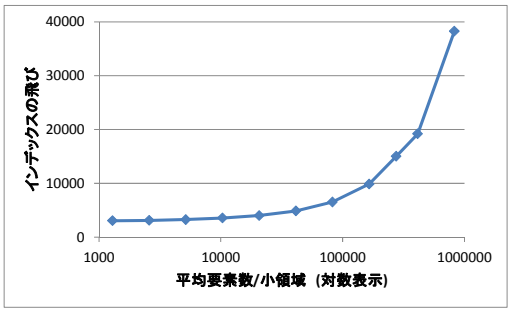

図 10 配列 FX, FY, FZ のインデックス飛びの低減
分割数が多いほど（図左側ほど）インデックス飛びが低減

図 11 は領域の分割を変化させた際の際のカーネルのピーク性能比を示したものである．図右端の何も分割していない場合に相当するケースが最も性能が良く前述の 1.6%程度であり，領域分割が細かいほどピーク性能比が低下してゆく結果となった．

表 2 は各領域分割サイズにおける小領域当たりの平均カラー数，カラー当たりの平均要素数，および 1 スレッド当たりの最内ループの平均回転数を示した物である．領域分

割が細かいケースでは，領域内の要素数が少なく，少ない要素数に対しカラーリングを行ったことでカラー内の要素数が極端に少なくなり，その結果各スレッドで最内ループの平均回転数は極端に小さな値となっていることが判る．

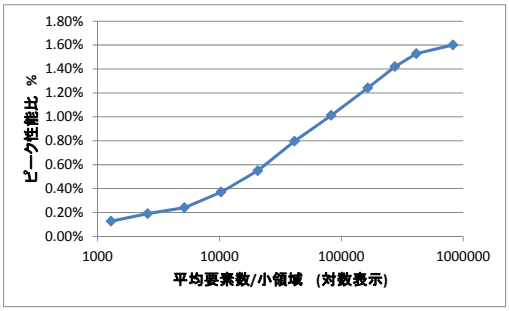

図 11 各分割数に応じたピーク性能比

表 2 小領域 1 個の平均カラー数と
カラー1 個当たりの平均要素数

ケースNo	小空間分割数	要素数/小領域	平均カラー数/小領域	平均要素数/カラー	平均最内回転数/スレッド
1	640	1291	37.4	34.0	4
2	320	2583	38.7	66.0	8
3	160	5166	39.7	130.0	16
4	80	10333	41.0	252.0	32
5	40	20666	41.5	498.0	62
6	20	41332	42.1	981.0	123
7	10	82665	42.2	1958.0	245
8	5	165331	42.6	3881.0	485
9	3	275552	43.7	6310.0	789
10	2	413328	43.5	9501.0	1188
11	1	826656	44.0	18787.0	2348

キャッシュの利用効率を改善するための節点リオーダーと領域分割により L1D キャッシュミス率は 21.3%から 5.44%まで向上するも性能自体は低下してしまった．これは領域分割とカラーリングの併用により小領域内カラーに含まれる要素数が小さくなりすぎ，その結果最内ループの回転数が不足し演算スケジューリングの効率が低下したためと推測される．

6. 回転数不足の回避

ここでは領域分割と節点リナンバーによるメモリアクセスの局所性とカラーリングによるマルチスレッド実行を生かしたまま最内ループの回転数の不足を回避するためカラーリングされる対象の変更を行った．5 節で述べた節点リオーダリングと同様に，行列に対してはキャッシュアクセスの効率を向上させる Block multi-color ordering [23]等の手法が知られているが，本カーネルは行列を作成せず要素毎に計算する実装となっているため，行列をブロック化しカラーリングを行う操作は，要素をそのままブロック化しカラーリングする操作に相当する．

図 12 はカラーリング対象変更の概要図である．これまでと同様に図左側に示す計算領域全体は，図中央に示されるように複数の小領域へと分割される．これまでの手法ではカラーリングは各小領域の中の要素に対して行っており，そのためカラー当たりが含む要素数が少なくなったが，本手法ではカラーリングの対象は各小領域とし，要素に対して行ったのと同様に，互いに隣接する小領域を別のカラーへと所属させる．これはマルチスレッドで処理される単位が要素から小領域に変わることを意味し，小領域内には十分な数の要素が含まれることから，十分な数の最内ループの回転数を確保するという狙いである．

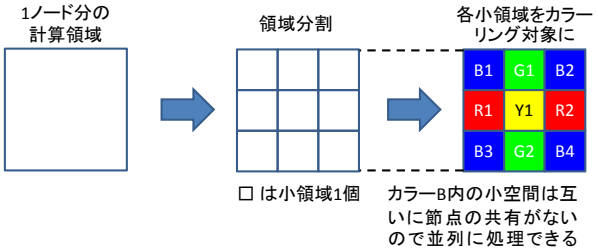

図 12 カラーリング対象の変更

そのためループ構造は，図 13 に示すようにカラーについてのループ ICOLOR，カラー内の小領域についてのループ IGROUP，そして小領域内の要素についてのループ IE という三重ループ構造となる．マルチスレッドは最内のループ IE について行っていたが，ここでは一段上位のループ IGROUP について行う．

これまでのコードでは最内ループをマルチスレッド化するために要素単位でのカラーリングを行っていたため，最内ループの演算にはデータ依存性がなくコンパイラにより SIMD 化とソフトウェアパイプライン化が適用されていたが，この改善では小領域単位でのカラーリングであり最内ループの演算にはデータ依存性が生ずるため SIMD 化とソフトウェアパイプラインは適用されない．

```
DO ICOLOR=1,NCOLOR(1)
  DO IGROUP=1,NGROUP(ICOLOR,1)・・・ここで並列
    IES=LLOOP(IGROUP,ICOLOR   ,1)+1
    IEE=LLOOP(IGROUP,ICOLOR+1,1)
    DO IE=IES,IEE
      IP1=NODE(1,IE)
      IP2=NODE(2,IE)
      IP3=NODE(3,IE)
      IP4=NODE(4,IE)
      SWRK=S(IE)
      FX(IP1)=FX(IP1)-SWRK*DNX(1,IE)
      FX(IP2)=FX(IP2)-SWRK*DNX(2,IE)
      FX(IP3)=FX(IP3)-SWRK*DNX(3,IE)
      FX(IP4)=FX(IP4)-SWRK*DNX(4,IE)
      FY(IP1)=FY(IP1)-SWRK*DNY(1,IE)
      FY(IP2)=FY(IP2)-SWRK*DNY(2,IE)
      FY(IP3)=FY(IP3)-SWRK*DNY(3,IE)
      FY(IP4)=FY(IP4)-SWRK*DNY(4,IE)
      FZ(IP1)=FZ(IP1)-SWRK*DNZ(1,IE)
      FZ(IP2)=FZ(IP2)-SWRK*DNZ(2,IE)
      FZ(IP3)=FZ(IP3)-SWRK*DNZ(3,IE)
      FZ(IP4)=FZ(IP4)-SWRK*DNZ(4,IE)
    ENDDO
  ENDDO
ENDDO
```

図 13 カラーリング対象変更後ソース

表 3 は各領域分割サイズにおける 1 スレッド当たりの最内ループ平均回転数を，カラーリング対象変更前後で比較したものである．カラーリング対象変更前では領域分割とカラーリングの併用により小領域内カラーに含まれる要素数が小さくなりすぎ，その結果最内ループの回転数は非常に小さな値となったが，カラーリング対象変更により，最内ループに十分な回転数を確保することが出来た．

表 3 最内ループ平均回転数

			平均最内回転数/スレッド	
ケースNo	小領域分割数	要素数/小領域	カラーリング対象変更前	カラーリング対象変更後
1	640	1291	4	161
2	320	2583	8	323
3	160	5166	16	646
4	80	10333	32	1292
5	40	20666	62	2583
6	20	41332	123	5167
7	10	82665	245	10333
8	5	165331	485	20666
9	3	275552	789	34444
10	2	413328	1188	51666
11	1	826656	2348	103332

図 14 はカラーリング対象の変更後の性能を示したものである．図左側に位置する細かい領域分割によりキャッシュの利用効率が改善されているケースで性能は大きく向上し，1 スレッド当たりの最内ループの平均回転数が 323 回転となる分割数で最もよい性能値 3.79%まで向上した．このときの L1D キャッシュミス率は 4.00%，メモリスループットは 32.0GByte/s であった．

領域分割を粗くしてゆくと，1 個の小領域に含まれる要素数が増加し 1 スレッド当たりの最内ループの平均回転数が増加するため演算スケジューリング効率の改善が期待されるが，他方で小領域が大きくなったことによりメモリアクセスの局所性が低下し，小領域の個数の減少により 1 カラー内に含まれる小領域の個数が減少．その結果ロードインバランスが増大することで性能は劣化し，ケース 6 以降は第 4 節末で示したオリジナルの性能 1.6%と比しても低い性能となった．

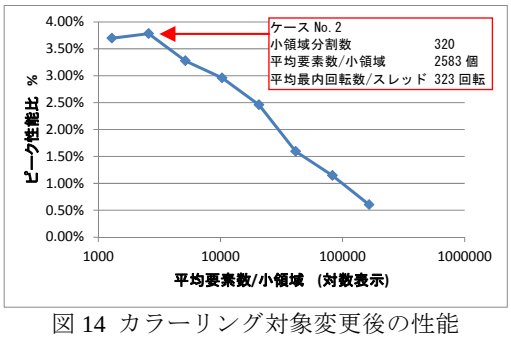

図 14 カラーリング対象変更後の性能

7. 配列融合

ここまでのチューニングにより最良で理論値 5.83%のおよそ 65%に相当する 3.79%の性能まで到達した．前述のように L1D キャッシュミス率は理想値 3.125%であるが，4.00%とやや高く，メモリスループットも理想値 46.6GByte/s に対して 32.0GByte/s と低くまだ向上の余地が見込まれる．

本カーネルでは最内ループ IE で回転毎に NODE, S, DNX, DNY, DNZ, FX, FY, FZ の 8 種の配列にアクセスしているが，これらの中で節点上の勾配ベクトルを保持する配列 FX, FY, FZ は 3 つとも同じインデックス IP1, IP2, IP3, IP4 を介して同様のパターンでアクセスされているため融合することでメモリスループットを向上できる可能性がある．また要素の形状関数の導関数を保持している配列 DNX, DNY, DNZ についても同様にそれぞれ同一のインデックスによりアクセスされるため融合できる可能性がある．ここでは DNX,DNY,DNZ を融合したパターンを 1 種，FX,FY,FZ を融合したパターン 2 種及び DNX,DNY,DNZ の融合と FX,FY,FZ の融合を同時に適応したパターンの計 4 パターンを検討した．

```
FX(IP1)=FX(IP1)-SWRK*DNXYZ(1,1,IE)
FX(IP2)=FX(IP2)-SWRK*DNXYZ(1,2,IE)
FX(IP3)=FX(IP3)-SWRK*DNXYZ(1,3,IE)
FX(IP4)=FX(IP4)-SWRK*DNXYZ(1,4,IE)

FY(IP1)=FY(IP1)-SWRK*DNXYZ(2,1,IE)
FY(IP2)=FY(IP2)-SWRK*DNXYZ(2,2,IE)
FY(IP3)=FY(IP3)-SWRK*DNXYZ(2,3,IE)
FY(IP4)=FY(IP4)-SWRK*DNXYZ(2,4,IE)

FZ(IP1)=FZ(IP1)-SWRK*DNXYZ(3,1,IE)
FZ(IP2)=FZ(IP2)-SWRK*DNXYZ(3,2,IE)
FZ(IP3)=FZ(IP3)-SWRK*DNXYZ(3,3,IE)
FZ(IP4)=FZ(IP4)-SWRK*DNXYZ(3,4,IE)
```

図 15 パターン 1 配列 DNX, DNY, DNZ の融合

パターン 1 として DNX, DNY, DNZ を 1 次元目が 1 なら X 方向導関数，2 なら Y 方向導関数，3 なら Z 方向導関数となる 3 次元配列 DNXYZ として融合した．図 15 に最内ループ IE での演算を示す．融合されたことによるカーネルの要求バイト数は第 4 節に示した計算法に従うと 32 回転で配列 DNX,DNY,DNZ で 3456Byte 必要であるが，配列 DNXYZ では 27 回 L1D ミスし 3456Byte となりカーネルの理論性能は変化しない．

```
FXYZ(1,IP1)=FXYZ(1,IP1)-SWRK*DNX(1,IE)
FXYZ(1,IP2)=FXYZ(1,IP2)-SWRK*DNX(2,IE)
FXYZ(1,IP3)=FXYZ(1,IP3)-SWRK*DNX(3,IE)
FXYZ(1,IP4)=FXYZ(1,IP4)-SWRK*DNX(4,IE)

FXYZ(2,IP1)=FXYZ(2,IP1)-SWRK*DNY(1,IE)
FXYZ(2,IP2)=FXYZ(2,IP2)-SWRK*DNY(2,IE)
FXYZ(2,IP3)=FXYZ(2,IP3)-SWRK*DNY(3,IE)
FXYZ(2,IP4)=FXYZ(2,IP4)-SWRK*DNY(4,IE)

FXYZ(3,IP1)=FXYZ(3,IP1)-SWRK*DNZ(1,IE)
FXYZ(3,IP2)=FXYZ(3,IP2)-SWRK*DNZ(2,IE)
FXYZ(3,IP3)=FXYZ(3,IP3)-SWRK*DNZ(3,IE)
FXYZ(3,IP4)=FXYZ(3,IP4)-SWRK*DNZ(4,IE)
```

図 16 パターン 2 配列 FX, FY, FZ の融合①

パターン 2 では配列 FX, FY, FZ をパターン 1 と同様に 1 次元目が X, Y, Z のインデックスとなるよう融合した．図 16 に最内ループ IE での演算を示す．第 4 節で述べたよう

にリストアクセスされる配列はここではリストアクセスされる配列はオンキャッシュと仮定するため，カーネルの要求 B/F 値は変化せず理論性能は 5.83%である．

図 17 示すパターン 3 では配列 FXYZ についてメモリを連続的に参照するよう 1 次元目が先に動くよう演算順序を変更した．すなわちカーネルの要求 B/F 値は変化せず理論性能は 5.83%である．

```
FXYZ(1,IP1)=FXYZ(1,IP1)-SWRK*DNX(1,IE)
FXYZ(2,IP1)=FXYZ(2,IP1)-SWRK*DNY(1,IE)
FXYZ(3,IP1)=FXYZ(3,IP1)-SWRK*DNZ(1,IE)

FXYZ(1,IP2)=FXYZ(1,IP2)-SWRK*DNX(2,IE)
FXYZ(2,IP2)=FXYZ(2,IP2)-SWRK*DNY(2,IE)
FXYZ(3,IP2)=FXYZ(3,IP2)-SWRK*DNZ(2,IE)

FXYZ(1,IP3)=FXYZ(1,IP3)-SWRK*DNX(3,IE)
FXYZ(2,IP3)=FXYZ(2,IP3)-SWRK*DNY(3,IE)
FXYZ(3,IP3)=FXYZ(3,IP3)-SWRK*DNZ(3,IE)

FXYZ(1,IP4)=FXYZ(1,IP4)-SWRK*DNX(4,IE)
FXYZ(2,IP4)=FXYZ(2,IP4)-SWRK*DNY(4,IE)
FXYZ(3,IP4)=FXYZ(3,IP4)-SWRK*DNZ(4,IE)
```

図 17 パターン 3 配列 FX, FY, FZ の融合②

```
FXYZ(1,IP1)=FXYZ(1,IP1)-SWRK*DNXYZ(1,1,IE)
FXYZ(2,IP1)=FXYZ(2,IP1)-SWRK*DNXYZ(2,1,IE)
FXYZ(3,IP1)=FXYZ(3,IP1)-SWRK*DNXYZ(3,1,IE)

FXYZ(1,IP2)=FXYZ(1,IP2)-SWRK*DNXYZ(1,2,IE)
FXYZ(2,IP2)=FXYZ(2,IP2)-SWRK*DNXYZ(2,2,IE)
FXYZ(3,IP2)=FXYZ(3,IP2)-SWRK*DNXYZ(3,2,IE)

FXYZ(1,IP3)=FXYZ(1,IP3)-SWRK*DNXYZ(1,3,IE)
FXYZ(2,IP3)=FXYZ(2,IP3)-SWRK*DNXYZ(2,3,IE)
FXYZ(3,IP3)=FXYZ(3,IP3)-SWRK*DNXYZ(3,3,IE)

FXYZ(1,IP4)=FXYZ(1,IP4)-SWRK*DNXYZ(1,4,IE)
FXYZ(2,IP4)=FXYZ(2,IP4)-SWRK*DNXYZ(2,4,IE)
FXYZ(3,IP4)=FXYZ(3,IP4)-SWRK*DNXYZ(3,4,IE)
```

図 18 パターン 4 配列 FX, FY, FZ の融合②及び
配列 DNX, DNY, DNZ の融合

図 18 に示すパターン 4 ではパターン 3 に加えてパターン 1 の配列 DNX, DNY, DNZ の配列 DNXYZ への融合も加えたもので，カーネルの要求 B/F 値は他のパターンと同様で変化しない．

表 4 測定結果

パターン	ピーク性能比%	L1Dキャッシュミス率%	メモリスループット GB/s
1	3.95%	3.98%	35.70
2	4.05%	3.69%	35.91
3	4.05%	3.69%	35.88
4	4.41%	3.37%	38.80

表 4 にこれまでの検討で最良のピーク性能比を示した図 14 に示すケース 2 の領域分割数における，各パターンでのピーク性能比，L1D キャッシュミス率，メモリスループットを示す．パターン 1 については融合により最内ループ IE の回転毎にアクセスされる配列数が 2 つ減少しわずかに性能向上した．節点リオーダーと領域分割により配列 FX, FY, FZ のインデックス飛びの大幅な低減がなされたとはいえ，リストアクセス故のメモリアクセスの飛びは完全には除去できず，3 つの配列 FX, FY, FZ についての不連続なアクセスによりキャッシュミスが発生するが，パターン 2 と 3 については不連続にアクセスされる 3 つの配列が 1 つに融合したことによるキャッシュミスの削減効果が有効に作用し 4%台のピーク性能比に到達した．パターン 4 については最内ループ IE の回転内でアクセスされる配列数が大きく減少したため，キャッシュラインの競合が低減されたためパターン 2/3 よりもさらに高い性能を示したと考えられる．

図 18 にここまでで最良の結果を得た配列融合パターン 4 の各スレッドの実効時間内訳を示す．図 2 に示したオリジナルコードと比して計算時間全体に占めるメモリアクセス関連の待ちとバリア同期待ちの割合が大幅に削減され，ほとんどの時間を命令実行が占めるようになった．全体の実効時間も 4.72 秒から 1.76 秒へと 2.7 倍向上した．

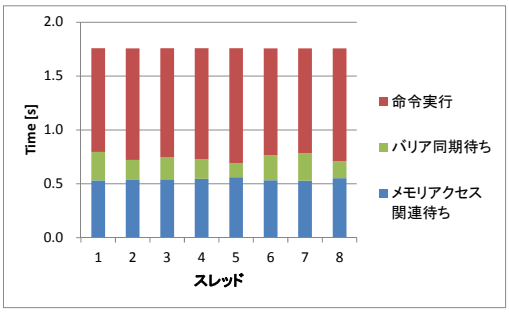

図 18 配列融合パターン 4 計算時間内訳
4.81 秒から 1.76 秒まで 2.37 倍に向上

8. むすび

有限要素法による汎用流体解析ソフト FrontFlow/blue の要素圧力モードにおけるカーネルである勾配演算カーネルについてスーパーコンピュータ「京」上での性能値の評価とチューニングを行った．手法としてはマルチスレッドで実行するための基本的なカラーリングを行った後，リストアクセス故に生じる不連続なメモリアクセスを低減する節点のリナンバーと小領域への分割，カラーリングと小領域への分割の結果生じた最内ループの回転数不足を回避する並列処理対象とする処理単位の変更及びメモリアクセス効率の向上のための配列融合であった．これらを適用した結果，勾配計算カーネルは理論性能 5.83%に対しておよそ 79%の 4.41%まで性能向上した．このとき L1D ミス率は理想値 3.125%に対し 3.37%，メモリスループットは 46.6GB/s に対して 38.8GB/s であった．ピーク性能比は理論値に対してまだ 20%程度低いが，この不足分は節点リナンバリングによっても完全に除去できないリストアクセスによるメモリアクセスの飛びによると考えられる．事実リスト値として実際の入力データではなく，メモリアクセスの飛びが一切無いよう作った理想的なリスト値を用いると理論値に対し 93%である 5.43%まで向上した．本稿で述べた手法は FrontFlow/blue の勾配計算カーネルのみならず，有限要素法や有限体積法などでよく出現する要素あるいは Control Volume から節点に対する演算を行う部分の改善に有効で

あると考えられる．現在はさらなる性能向上案として，配列サイズ縮小によるカーネルの要求 B/F の削減と，本稿で紹介したカラーリング対象変更の結果適用されなくなった最内ループ内の SIMD 化及びソフトウェアパイプライン化を適用させる策を検討中している．

謝辞 本報告に際し，FrontFlow/blue の開発を行った東京大学生産技術研究所の加藤千幸教授，みずほ情報総研の山出吉伸氏並びに革新的シミュレーションソフトウェアの研究開発プロジェクトの諸兄，理化学研究所計算科学研究機構運用技術部門の諸氏，理化学研究所計算科学研究機構に常駐している富士通株式会社 SE の皆様に感謝します．本論文の結果は，理化学研究所計算科学研究機構が保有するスーパーコンピュータ「京」の試験利用によるものです．

参考文献

1) Satoh, M., Matsuno, T., Tomita, H., Miura, H., Nasuno, T. and Iga, S.: Nonhydrostatic Icosahedral Atmospheric Model (NICAM) for global cloud resolving simulations, Journal of Computational Physics, the special issue on Predicting Weather, Climate and Extreme events, 227, pp.3486-3514, doi:10.1016/j.jcp.2007.02.006 (2008).
2) Furumura, T. and Chen, L.: Parallel simulation of strong ground motions during recent and historical damaginge earthquakes in Tokyo, Japan, Parallel Computing, 31, pp149-165, (2005).
3) http://www.ciss.iis.u-tokyo.ac.jp/rss21/theme/multi/material/material_softwareinfo.html#01
4) Iwata, J., Takahashi, D., Oshiyama, A., Boku, T., Shiraishi, K. and Okada, S.: A massively-parallel electronic-structure calculations based on real-space density functional theory, Journal of Computational Physics 229, pp2339-2363, (2010).
5) http://www.ciss.iis.u-tokyo.ac.jp/rss21/theme/multi/fluid/fluid_softwareinfo.html
6) Aoki, S., Ishikawa, K.I., Ishizuka, N., Izubuchi, T., Kadoh, D., Kanaya, K., Kuramashi, Y., Namekawa, Y., Okawa, M., Taniguchi, Y., Ukawa, A., Ukita, N. and Yoshie, T.:2+1 Flavor Lattice QCD toward the Physical Point, Physical Review D79, 034503, (2009).
7) Smagorinsky, J.: General Circulation Experiments with the Primitive Equations, Monthly Weather Review, Vol. 91, Issue 3, pp.99-164 (1963).
8) Curle, N.: The influence of solid boundaries upon aerodynamic sound, Proc. of the Royal Society, Series A, Vol.231, pp.505-514 (1955).
9) http://www.j-focus.or.jp/
10) 南一生, 井上俊介, 堤重信, 前田拓人, 長谷川幸弘, 黒田明義, 寺井優晃, 横川三津夫:「京」コンピュータにおける疎行列とベクトル積の性能チューニングと性能評価, 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.2012, pp.23-31 (2012).
11) Maruyama, T.: SPARC64 VIIIfx: Fujitsu's New Generation Octo-core Processor for Peta Scale Computing., Hot Chips 21 (2009).
12) Maruyama, T.: SPARC64 VIIIFX: A New-Generation Octocore Processor for Petascale Computing, IEEE micro, Vol.30, No.2, pp30-40 (2010).
13) SPARC International, The SPARC Architecture Manual (Version 9), Prentice-Hall (1994).
14) Sparc Joint Programming Specification (JPS1): Commonality, architecture manual., Sun Microsystems and Fujitsu Ltd. (2002).
15) SPARC64VIIIfx Extensions, Fujitsu Ltd., architecture manual (2008).
16) 岩瀬拓: 空調用ファンにおける空力騒音の計算, 第 4 回「イノベーション基板シミュレーションソフトウェアの研究開発」シンポジウム講演集, pp.165-173 (2012).
17) 西川達雄: 境界層を完全に解像したラージ・エディー・シミュレーションの船舶への適用, SRC News No.90, pp.6-7 (2012).
18) http://www.cs.virginia.edu/stream/ref.html
19) Tinney, W.F. and Walker, J.W.: Direct solutions of sparse network equations by optimally ordered triangular factorization, Proc. IEEE, Vol.55, pp.1801-1809 (1967).
20) George, A.: Nested dissection of a regular finite-element mesh, SIAM J. Numerical Analysis, Vol.10, pp.345-363 (1973).
21) Pinar. A. and Heath,T.M.: Improving performance of sparse satrix–vector multiplication, Proc. ACM/IEEE Conference on Supercomputing, Portland, Oregon. (1999).
22) Abou-Rjeili, A and Karypis, G.: Multilevel Algorithms for Partitioning Power-Law Graphs, IEEE International Parallel & Distributed Processing Symposium (IPDPS), (2006).
23) Iwashita, T., Nakashima, H. and Takahashi, Y.: Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in ICCG method, Proceedings of 26th IEEE International Parallel & Distributed Processing Symposium (2012).