情報処理学会論文誌　コンピューティングシステム　Vol.6　No.3　31–42 (Sep. 2013)

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[1,a)]　井上 俊介[1]　南 一生[1]

受付日 2012年12月21日, 採録日 2013年2月9日

概要：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%以上を占めており主要な部分（カーネル）となる．このカーネルの処理は各要素上で計算した値を要素が参照する節点へ分配するという有限要素法および有限体積法で頻出する処理であり実用上重要なものである．このカーネルは有限要素法のデータ構造に依存して，要素が持つ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をしている．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り，キャッシュ利用効率を改善する節点のリオーダリングと要素のブロック化を施し性能向上させた．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，なかでも理論性能の推定法とチューニング技法および測定結果について述べる．

キーワード：有限要素法，性能チューニング，CPU 単体性能，流体解析，リストアクセス

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

Kiyoshi Kumahata[1,a)]　Shunsuke Inoue[1]　Kazuo Minami[1]

Received: December 21, 2012, Accepted: February 9, 2013

Abstract: General purpose fluid simulation software FrontFlow/blue based on the finite element method has two modes to treat pressure. One defines pressure on a node and another defines pressure on an 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 performs the operation in which a value is calculated on an element using variables defined on an element and stores the value into node. Such operation is frequently occurs in finite element method and finite volume method. 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.

Keywords: finite element method, performance tuning, fluid analysis, list access, single CPU performance

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，選択にあたっては，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およ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

[1] 理化学研究所計算科学研究機構
RIKEN, Kobe, Hyogo 650–0047, Japan
[a)] kuma@riken.jp

（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．

本稿でははじ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 情報を述べ，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，チューニング手法およびチューニング手法による性能向上を評価する．

2. 「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^{TM} VIIIfx）[7], [8], [9]，16 GByte 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 6 MByte 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 128 Byte で 2way 構成の 32 KByte 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 1 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あたりクロックサイクルごとに 8 個の浮動小数点演算が可能である．よって CPU 全体で 128 GFLOPS の性能を持つ．理論メモリバンド幅は 64 GByte/秒である．

3. 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）[10] 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，空間および時間について 2 次精度を持ち，様々な要素タイプおよび座標系・格子系を扱えるためポンプやファンなど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 [11] 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ともな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 [12] など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．FFB は大規模並列計算を指向して開発されているが，大規模並列では大域通信である Allreduce の負荷が並列数とともに増大してゆきスケーラビリティが頭打ちになる懸念があった．しかし「京」ではインターコネクトが備えるハードウェアバリア機能により 10 PFLOPS の規模においても高速な大域通信が可能であるため，並列性能の観点では「京」上での大規模並列において良好な性能を示している．一方で「京」の CPU 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．

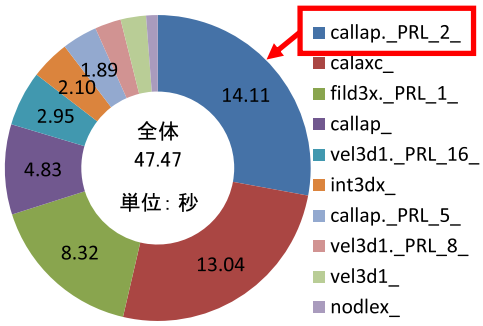

図 1　節点圧力モードでの計算時間内訳（上位 10 件）
Fig. 1　Elapse time distribution of 10 highest kernels.

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 8,000 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．ここから分か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．非圧縮粘性流体の非定常な流れを Fractional-Step 法 [13] に基づいて解く際には圧力 Poisson 方程式 (1) が導出される．

$$\frac{\partial^2 p^{n+1}}{\partial x^2} = \frac{1}{\Delta t}\frac{\partial \tilde{u}}{\partial x} \tag{1}$$

ここで p^{n+1} は次タイムステップの圧力，Δt は積分時間間隔，$\tilde{u}$ は現タイムステップ n の流速から予測した中間流速である．FFB の要素圧力モードでは圧力 Poisson 方程式を全体行列を作らず Element-by-element に Bi-CGSTAB 法で解くが，その際に頻出する係数行列とベクトルとの計算

を，① 勾配を求める，② 求めた勾配の発散を求めるという 2 段階で行っており，callap._PRL_2_ は勾配計算を行う部分である．この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．

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 [14], [15] に詳しい．

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．本カーネルは要素上に定義されたスカラ値と，要素の各頂点上に定義された形状関数の導関数とを乗じ，その値を各節点へ分配するという処理を行っている．このような要素の値から求めた値を節点へと分配する処理は有限要素法や有限体積法には頻出する重要な処理である．

本実装ではデータの依存関係が発生する隣接した要素の同時処理，すなわち隣接した要素間で共有される節点の値を同時に更新することを避けるため，互いに隣接する要素をカラーと呼ぶ異なるグループに分け，演算を隣接関係のまったくない要素の集合であるカラーごと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 1 のループ ICOLOR は各カラーについて回転するループ．配列 LLOOP は要素リスト中での各カラーに属する要素の開始・終了位置を保持している配列であり，第 2 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る．**表 1** に示すように配列 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スカラ値を保持している REAL*4 の配列であり，ここでは勾配計算の対象である圧力である．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．ここから分かるとお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り再利用性がある．またここで扱っているカーネルは 4 面体要素向けの勾配計算カーネルであるため，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，実際のアプリケーションでは頂点数が 5 である 5 面体要素，頂点数

```
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

Fig. 2　Source code of the gradient computation kernel.

表 1　配列の型とサイズ

Table 1　Array definitions.

型	宣言	内容
INTEGER*4	NODE(9,NE)	要素の参照節点リスト
REAL*4	S(NE)	要素中心に定義されるスカラ値 （勾配計算の対象である圧力）
REAL*4	DNX(9,NE)	各要素の形状関数のX方向導関数
REAL*4	DNY(9,NE)	各要素の形状関数のY方向導関数
REAL*4	DNZ(9,NE)	各要素の形状関数のZ方向導関数
REAL*4	FXYZ(3,NP)	各節点上の勾配ベクトル

NEは要素数, NPは節点数

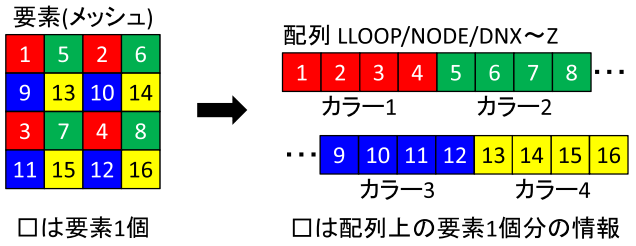

図 3　要素の情報を保持する配列の構造

Fig. 3　Element information structure.

が 8 である 6 面体要素などの多様な要素タイプを統一的に扱うためにサイズは 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る．

図 3 に要素に関する配列である LLOOP，NODE，S，DNX，DNY，DNZ 内がカラーリングによりどのような構造になるかを示す．図左はメッシュ構造すなわち各要素の幾何的な配置を表したもので，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る．対応して図右の各配列内のデータの並びは同じカラーの要素が連続するよう配置される．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．

5. 「京」上での性能評価

ここではカーネルの要求バイト量と要求演算量の比（Byte/Flop，B/F 値）から「京」上での理論性能の上限を参考文献 [16] の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・メインメモリ間のデータ転送にかかるコストを比較し，支配的なペナルティを持つメモリからのデータ転送量を要求バイト量と定義する．

はじめにリストアクセスゆえに不規則にアクセスされる配列 FX，FY，FZ については再利用性がある配列であるため，L1D あるいは L2 キャッシュ上で再利用されていればメモリプレッシャは生じない．再利用性の程度は入力データによって変化するため，性能上限を推定するに際しては，再利用性がきわめて高く，つねに L1D キャッシュ上に載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がないものとする．

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 4 Byte 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 128 Byte がメモリから転送され，128/4=32 回転分のデータを L1D キャッシュでまかなえるため，メモリプレッシャは 32 回転につき 128 Byte である．

配列 NODE についても 1 回目のアクセスでは L1D キャッシュミスし，128 Byte がメモリから転送されるが，1 次元目のサイズが 9 であり最内ループ IE の 1 回転ごとに 36 Byte 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 128 Byte でまかなえる回転数は約 3.56 回転である．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 回であるため，メモリプレッシャは 32 回転につき 1152 Byte 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．

配列 DNX，DNY，DNZ のそれぞれについても NODE と同じく 1 次元目のサイズが 9 であるため，3 つの合計メモリプレッシャは 32 回転につき $1152 \times 3 = 3456$ Byte となる．

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合計 4736 Byte となる．一方で 32 回転分の演算量は $24 \times 32 = 768$ FLOP．以上より本カーネルが要求する B/F の値は $4736/864 = 6.17$ と見積もられる．

「京」の CPU は 2 章で記したように 128 GFLOPS の演算性能と，64 GByte/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 46.6 GByte/s であるため，「京」が持つ実効的な B/F 値の上限を $46.6/128 = 0.36$ 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 128 GFLOPS で実行することができない．よって本カーネルの理論性能は「京」のピーク性能 128 GFLOPS の $0.36/6.17 = 5.83\%$ であると見積もられる．

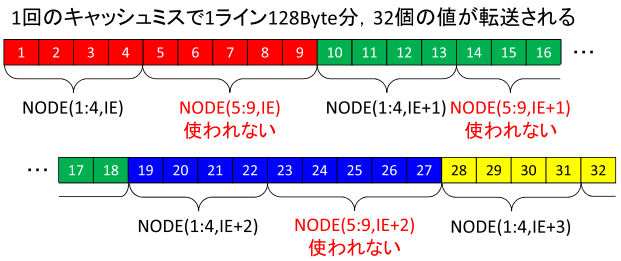

図 4　NODE(1, IE) のロードによりメモリから転送される 1 ライン内の分布．1 ライン 128 バイトで 32 個の INTEGER*4 がメモリから転送されるが，ストライドアクセスのためソース上は使われないものもメモリプレッシャとなる

Fig. 4　Data distribution in a cache line transferring from memory by loading NODE(1, IE). An array element that will not be used also becomes a memory pressure due to stride access.

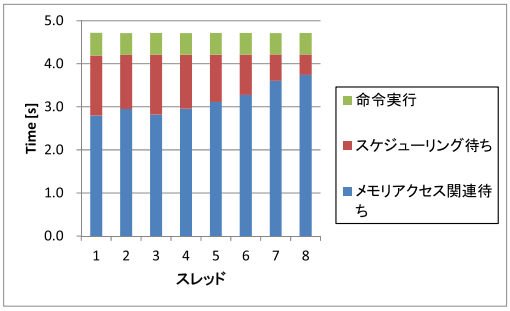

図 5　実行時間内訳

Fig. 5　Elapse time distribution on 8-threads.

ここでは，性能評価およ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でき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，およそ 21 万個の節点を含むデータを用いた．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 46.6 GByte/s に対し 10.29 GByte/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．すべてのスレッドにおいて図中青で示される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．また図中赤で示されるバリア同期や命令スケジューリングによる待ちも多く，図中緑で示される実行時間のうち演算などの命令実行に要する時間の割合はわずかであった．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ことが分かった．

6. 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すべて L1D キャッシュ上に載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ずれているため，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く，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．この問題は入力データの規模が大きくなるにつれ顕著になり，L2 キャッシュ上の再利用性にも影響が出てくると考えられる．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要素のブロック化を実施した．

6.1 節点リオーダリング

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ごと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もとに，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直方体に区切られ，各直方体は 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 $10 \times 10 \times 10$ に分割した．初期状態では直方体内に位置する節点の番号は入力データと

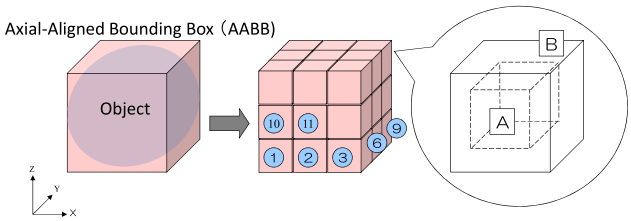

図 6　節点リオーダ
Fig. 6　Node number reordering.

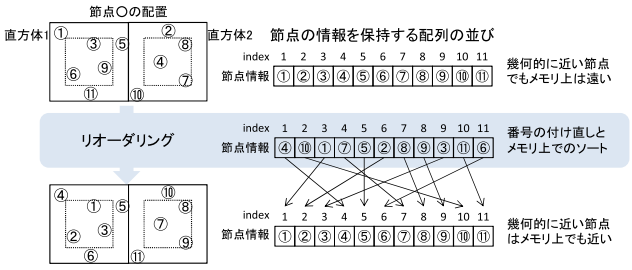

図 7　メモリ配置．リオーダリングにより幾何的に近い節点の情報はメモリ上でも近い場所に置かれる
Fig. 7　Node information arrangement on memory. Information of geometrically near nodes will locate near position on memory.

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うるが，直方体番号 1 から順に図右側のように直方体内を外側領域と内側領域に分け，直方体内部の節点の番号を内側領域内，外側領域内の順で連続するよう更新する．直方体を内外に分けるのは直方体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，ここでは直方体の 1 辺の長さを L とした際，直方体表面から厚さ 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．次の直方体内での節点番号は先の直方体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．

さらに配列 FX，FY，FZ など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．ソート後は個々の直方体の情報は不要である．

6.2 要素のブロック化

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内の要素のブロック化を行う．**図 8** は要素のブロック化の概要図である．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ブロックへと分割される．前述のように節点リオーダリングの際に行った分割情報は節点情報のソート後は不要であるのに対し，要素ブロック化のための分割はループ構造に反映される情報であるため，この分割

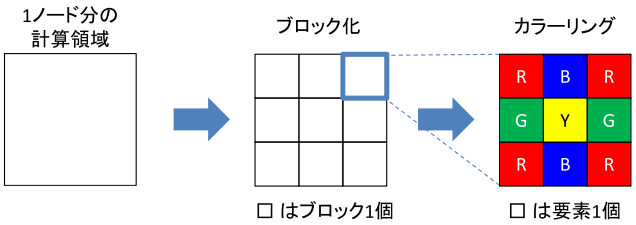

図 8　要素のブロック化（処理単位の局所化）
Fig. 8　Element blocking (localize elements unit).

```
DO IBLOCK=1,NBLOCK(1)
  DO ICOLOR=1,NCOLOR(IBLOCK,1)
    IES=LLOOP(IBLOCK,ICOLOR  ,1)+1
    IEE=LLOOP(IBLOCK,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　ブロック化適用後カーネルソース
Fig. 9　Source code after element blocking.

は節点リオーダリングの分割とは独立に行っている．

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ブロック単位に行う．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 IBLOCK，ブロック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．

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 2** に今回用いたブロック数で 1 ブロックあ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参照するデータ量，8 スレッド（8 コア）で処理する際の 1 スレッドあたりの必要データ量を示す．ここで必要メモリ量は，最内ループで 1 要素あ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 148 Byte，FX，FY，FZ については 4 節点 × X，Y，Z の 3 成分で 48 Byte より合計 198 Byte となることから求めた．

表 2　ブロック数と要素数，必要メモリ量
Table 2　Number of blocks, average number of element, and required memory.

ケースNo	ブロック分割数	平均要素数/ブロック	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

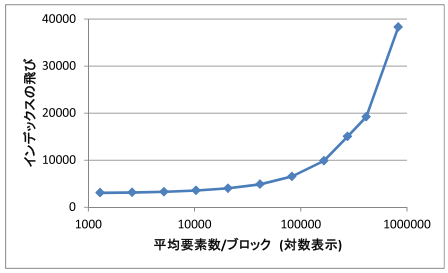

図 10　配列 FX，FY，FZ のインデックス飛びの低減．ブロックサイズが小さいほど（図左側ほど）インデックスの飛びが低減される
Fig. 10　Decreasing jumping of arrays FX, FY and FZ indices. An index jumping is small on small block size.

また「京」の各コアが持つ L1D キャッシュサイズは 32 KByte であるため，ケース No.1 のブロック数はスレッドあたりの必要メモリ量がおよそ 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メモリ量が約 2 倍ずつ増大するよう設定した．ケース No.11 は計算領域全体が 1 つのブロックとなっており，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．横軸はブロックあたりの平均要素数で表したブロックサイズ，縦軸は最内ループ 1 回転あたりについて配列 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ことを意味する．図で左側，すなわちブロックサイズが小さくブロックあたりに含む要素数が少な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，最もメモリアクセスの跳びが小さいブロックサイズが最小であるケースで測定した L1D キャッシュミス率は理想値 3.125%に対して 5.44%と，先の結果の 21.3%と比べて大きく改善された．その一方でピーク性能比は理想値 5.85%に対してわず

か 0.13%，メモリスループットは理想値 46.6 GByte/s に対してわずか 1.08 GByte/s と大幅に低下してしまった．

図 11 はブロックサイズを変化させた際のカーネルのピーク性能比を示したものである．図左端が最も小さいブロックのケースで，右へゆくほどブロックサイズは大きくなる．性能は図右端のオリジナルコードに相当するケースで最も性能が良く，性能比は前述のオリジナルと同じ 1.6%であり，ブロックサイズが小さいほどピーク性能比が低下してゆく結果となった．

表 3 は各ブロックサイズにおけるブロックあたりの平均カラー数，カラーあたりの平均要素数，および 1 スレッドあたりの最内ループの平均回転数を示したものである．ブロックサイズが小さい，すなわちブロック内の要素数が少ない場合，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た．

結果として，キャッシュの利用効率を改善するための節点リオーダと要素のブロック化により L1D キャッシュミス率は 21.3%から 5.44%まで向上したが，性能自体は低下してしまった．これはブロック化とカラーリングの併用によりブロック内の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．

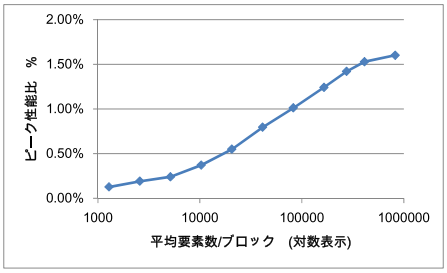

図 11　各ブロックサイズに応じたピーク性能比
Fig. 11　Peak performance ratio according to the block sizes.

表 3　ブロックあたりの平均カラー数とカラーあたりの平均要素数
Table 3　Average number of color per block and average number of element per color.

ケースNo	ブロック分割数	平均要素数/ブロック	平均カラー数/ブロック	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

7.　回転数不足の回避

ここではブロック化と節点リナンバ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 章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 [22] など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ごと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を囲む AABB は，図中央に示されるように複数のブロックへと分割される．これまでの手法ではカラーリングは各ブロック中の要素に対して行っており，そのためカラーあたりの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ブロックとし，要素に対して行ったのと同様に，互いに隣接するブロック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ブロックに変わることを意味し，ブロック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．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，カラー内のブロックについてのループ IBLOCK，そしてブロック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 1 段上位のループ IBLOCK について行う．

表 4 は各ブロックサイズにおける 1 スレッドあ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ブロック化とカラーリングの併用により各ブロック内の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できた．

図 14 はカラーリング対象変更後の性能を示したものである．ブロックサイズが小さいケースではキャッシュの利用効率が向上し，かつ最内ループの回転数不足も回避され

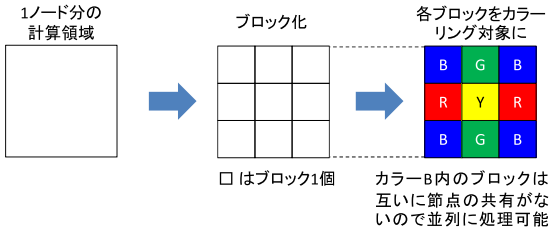

図 12　カラーリング対象の変更
Fig. 12　Changing coloring target to blocks from elements.

たため性能は大きく向上し，1 スレッドあたりの最内ループの平均回転数が 323 回転となるブロックサイズで最も良い性能値 3.78%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 32.0 GByte/s であった．一方でブロックサイズを大きくしてゆくと，1 個のブロックに含まれる要素数が増加し 1 スレッドあ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，反してブロック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，かつ全体のブロック数の減少により 1 カラー内に含まれるブロック数が減少．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 No.6 以降は 4 章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．

```
DO ICOLOR=1,NCOLOR(1)
  DO IBLOCK=1,NBLOCK(IBLOCK,1)・・・ここで並列
    IES=LLOOP(IBLOCK,ICOLOR   ,1)+1
    IEE=LLOOP(IBLOCK,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

Fig. 13　Source code after changing coloring target.

表 4　最内ループ平均回転数

Table 4　Average loop iterations within innermost loops.

ケースNo	ブロック分割数	平均要素数/ブロック	平均最内回転数/スレッド	
			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

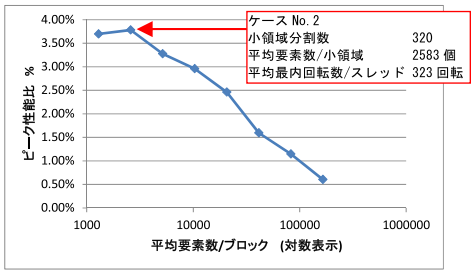

図 14　カラーリング対象変更後の性能

Fig. 14　Peak performance ratio after changing coloring target.

8.　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 3.78%の性能まで到達した．L1D キャッシュミス率については 4.00%と，前述の理想値 3.125%よりやや高く，メモリスループットも理想値 46.6 GByte/s に対して 32.0 GByte/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 内で回転ごとに 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させることが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 種および 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．

パターン No.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 4 章に示した計算法に従うと 32 回転で，配列 DNX，DNY，DNZ で 3456 Byte 必要であるが，配列 DNXYZ では 32 回転で 27 回 L1D ミスし $128 \times 27 = 3456$ Byte となりカーネルの理論性能は変化しない．

パターン No.2 では配列 FX，FY，FZ をパターン No.1

```
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 No.1．配列 DNX，DNY，DNZ の融合

Fig. 15　Pattern No.1: Merging array DNX, DNY and DNZ.

```
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 No.2．配列 FX，FY，FZ の融合①

Fig. 16　Pattern No.2: Merging array FX, FY and FZ (case 1).

```
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 No.3．配列 FX，FY，FZ の融合②

Fig. 17　Pattern No.3: Merging array FX, FY and FZ (case 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 No.4．配列 FX，FY，FZ の融合②および配列 DNX，DNY，DNZ の融合

Fig. 18　Pattern No.4: Merging pattern No.3 and No.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．4 章で述べたようにリストアクセスされる配列はここで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 に示すパターン No.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．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 18 に示すパターン No.4 ではパターン No.3 に加えてパターン No.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．いずれのパターンでも最良の結果は，これまでの検討で最良を示したケース No.2（ブロック数 320，平均要素数で示したブロックサイズは 2583 個）のブロックサイズで示された．**表 5** に測定を実施した結果のピーク性能比，L1D キャッシュミス率，メモリスループットを示す．

表 5　測定結果（ケース No.2）

Table 5　Measurement result (block size, case No.2).

パターンNo.	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

パターン No.1 については融合により最内ループ IE の回転ごとにアクセスされる配列数が 2 つ減少し，わずかに性能が向上した．節点リオーダと要素のブロック化により配列 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，実際の入力データではリストアクセスゆえ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 No.2 と No.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 No.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 No.2 と No.3 よりもさらに高い性能を示したと考えられる．

9.　ブロック間の要素数バランスの改善

ここまでの検討で性能向上への寄与が大きいのは 7 章で導入した回転数不足の回避を施した要素のブロック化であった．このブロック化手法は図 12 に示したように計算領域を含む最小の直方体である AABB を定義し，AABB を直方体形状に分割してブロック化を行っていた．この手法は単純なブロック化手法にもかかわらずピーク性能比 1.6%から 3.79%までの向上をもたらした．しかし直方体分割では入力データの形状によっては含む要素数が少ないブロック，あるいはまったく要素を含まないブロックが生ずるといったブロック間での要素数すなわち演算量のインバランスを引き起こす可能性がある．そこで汎用のグラフ分割ライブラリである METIS [23] を用いて，要素数がバランスするようブロック分割を行うことを検討した．

図 19 にブロック化の概念図を示す．この手法では任意形状を AABB で囲わずに，METIS を用いて各ブロックが均等な数の要素を含むようにブロック化する．その後隣接するブロックが別のカラーとなるようカラーリングする．

表 6 に指定ブロック数に応じた，新旧のブロック化手法でのブロックあたりの平均要素数，最大要素数，最小要素数を記す．各ブロック数のケースで旧手法ではブロックご

とに含む要素数に大きな違いが出ていた．一方で新ブロック化手法ではすべてのケースにおいてブロック間で要素数ほぼ等しくなった．一方で，6 章で述べた節点リオーダリングの際にも直方体分割を行っているが，これは要素ブロック化では直方体のサイズが最内ループの回転数に影響を与えるのとは異なり，節点の番号およびメモリ上での位置を並べ替える際のみに一時的に行う分割であるため，演算量のインバランスは生じないため，引き続き直方体分割を用いた．

図 20 は新ブロック化手法を用いた際の性能を示したものである．図 14 の旧ブロック化手法と同様に新ブロック化手法を用いた場合も，ブロックサイズが小さいケースの方がキャッシュの利用効率が良くなるため高い性能を示す傾向は一致するが，新ブロック化手法を用いた方が高い性能を示し，最良の場合で 4.43%となった．このときの L1D キャッシュミス率は 3.40%，メモリスループットは 40.23 Gbyte/s に達した．ブロック化手法の変更によりブロック間のインバランスを回避できるようになったのをふまえて，ブロックサイズを変えた測定を行った結果，最良の値としてブロックサイズ 1500 要素/ブロックでピーク性能比 4.58%，L1D キャッシュミス率 3.41%，メモリスループット 41.22 GByte/s という結果を得た．

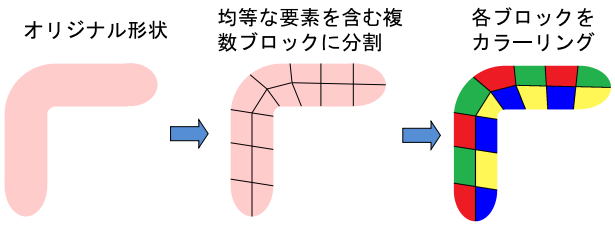

図 19　ブロック間の要素数バランスの改善
Fig. 19　Improve element balance betwen blocks.

表 6　ブロック間の要素数バランス新旧手法比較
Table 6　Comparing element balance between blocks by the old way and the new way.

ケースNo.	ブロック分割数	旧ブロック化手法			新ブロック化手法		
		平均要素数/ブロック	最大要素数/ブロック	最小要素数/ブロック	平均要素数/ブロック	最大要素数/ブロック	最小要素数/ブロック
1	640	1291	2380	463	1291	1293	1290
2	320	2583	4116	1054	2583	2585	2582
3	160	5166	7578	2367	5166	5168	5166
4	80	10333	13388	5232	10333	10334	10332
5	40	20666	26383	14840	20666	20668	20665
6	20	41332	50268	36685	41332	41334	41332
7	10	82665	100453	74098	82666	82666	82665
8	5	165331	199998	151979	165531	165532	165531
9	3	275552	355171	165154	277552	277552	277552
10	2	413328	432332	394324	411328	411328	411328
11	1	826656	826656	826656	826656	826656	826656

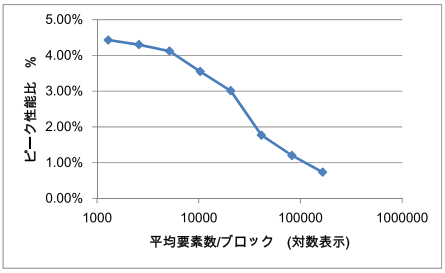

図 20　ブロック化手法変更後の性能
Fig. 20　Peak performance ratio after improving element balance.

10. むすび

図 21 にオリジナルコードと各章で達成した性能向上した結果についてプロファイラで得た実行時間内訳を示す．

図中 ORG は 5 章で述べたオリジナルコードである．図中青で示されるメモリアクセスに起因する待ちが支配的で，図中赤で示されるバリア同期や命令スケジューリングによる待ちも多く，図中緑で示される実行時間のうち演算などの命令実行に要する時間の割合はわずかであった．図中 Sec.7 は 7 章で述べた回転数不足の回避のためのカラーリング手法の変更を行ったものである．6 章で述べた節点リオーダリングと要素ブロッキングの効果でメモリアクセスに起因する待ちが大きく減少した．このデータ転送の効率化によりバリア同期や命令スケジューリングによる待ちもおよそ半減した．図中 Sec.8 は 8 章で述べた配列融合を行ったもので，最内ループでアクセスされる配列数が減りキャッシュライン競合が低減したためメモリアクセスに起因する待ちはさらに削減された．図中 Sec.9 は 9 章で述べたブロック間の要素数インバランス回避のためにブロック化手法を変更したもので，ブロック間のインバランスが大きく改善されバリア同期が削減された．

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．キャッシュ利用効率を向上させるために，リストアクセスゆえに生じる不連続なメモリアクセスを低減させる節点のリナンバと要素のブロック分割，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，メモリアクセス効率の向上のための配列融

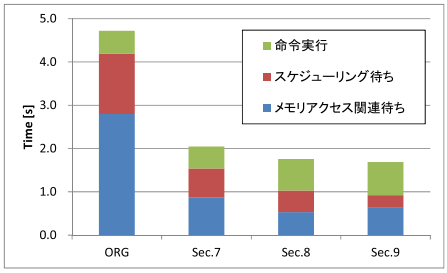

図 21　性能向上比較（スレッド 1）．Org：オリジナル，Sec.7：回転数不足回避，Sec.8：配列融合，Sec.9：ブロック化手法改善
Fig. 21　Comparing performance by employed way. Org is by original code. Sec.7 is improving lacks of iterations. Sec.8 is by array merge. Sec.9 is by improving element balance.

合，およびブロック間のインバランスを改善するブロック分割手法の高度化であった．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 4.58%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 3.41%，メモリスループットは 46.6 GB/s に対して 41.22 GB/s まで近づいた．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起因すると考えられる．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いっさいないように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だけでなく，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．

謝辞　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なら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および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．本稿の結果は，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*, Vol.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*, Vol.31, pp.149–165 (2005).

[3] ナノ・物質・材料・マルチスケール機能シミュレーションシステム，入手先 ⟨http://www.ciss.iis.u-tokyo.ac.jp/rss21/theme/multi/material/⟩.

[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*, Vol.229, pp.2339–2363 (2010).

[5] マルチフィジックス流体シミュレーション・システム，入手先 ⟨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] Maruyama, T.: SPARC64 VIIIfx: Fujitsu's New Generation Octo-core Processor for Peta Scale Computing, *Hot Chips 21* (2009).

[8] Maruyama, T.: SPARC64 VIIIFX: A New-Generation Octocore Processor for Petascale Computing, *IEEE micro*, Vol.30, No.2, pp.30–40 (2010).

[9] SPARC64 VIIIfx Extensions, Fujitsu Ltd., architecture manual (2008).

[10] Smagorinsky, J.: General Circulation Experiments with the Primitive Equations, *Monthly Weather Review*, Vol.91, Issue 3, pp.99–164 (1963).

[11] Curle, N.: The influence of solid boundaries upon aerodynamic sound, *Proc. Royal Society, Series A*, Vol.231, pp.505–514 (1955).

[12] 公益財団法人計算科学研究機構，入手先 ⟨http://www.j-focus.or.jp/⟩.

[13] Chorin, A.J.: On the Convergence of Discrete Approximations to the Navier-Stokes Equations, *Math. Comp.*, Vol.23, pp.341–353 (1969).

[14] 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).

[15] 南　一生，井上俊介，熊畑　清，黒田明義，寺井優晃，長谷川幸弘：アプリケーションの性能と最適化，情報処理，Vol.53, No.8, pp.787–794 (2012).

[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] STREAM Benchmark Reference Information, available from ⟨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 matrix–vector multiplication, *Proc. ACM/IEEE Conference on Supercomputing*, Portland, Oregon (1999).

[22] Iwashita, T., Nakashima, H. and Takahashi, Y.: Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in ICCG method, *Proc. 26th IEEE International Parallel & Distributed Processing Symposium* (2012).

[23] Abou-Rjeili, A. and Karypis, G.: Multilevel Algorithms for Partitioning Power-Law Graphs, *IEEE International Parallel & Distributed Processing Symposium* (*IPDPS*) (2006).

熊畑 清

2008 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．専門は計算流体力学．2012 年より理化学研究所計算科学研究機構にてスーパーコンピュータ「京」におけるアプリケーション高速化に従事．博士（情報科学）．

井上 俊介

1999 年横浜国立大学教育学部卒業．同年株式会社富士通長野システムエンジニアリング（現，富士通システムズ・イースト）入社．2010 年理化学研究所次世代スーパーコンピュータ開発実施本部に出向．現在，スーパーコンピュータ「京」におけるアプリケーション高速化に従事．

南 一生

1981 年日本大学理工学部物理学科卒業．同年富士通株式会社入社．主に原子力分野のシミュレーションコードのスパコンへの性能最適化の仕事に従事．2000 年財団法人高度情報科学技術研究機構入社．地球シミュレータ用ソフトウェア性能最適化研究に従事．2008 年理化学研究所次世代スーパーコンピュータ開発実施本部開発グループアプリケーション開発チームリーダー，2012 年理化学研究所計算科学研究機構運用技術部門ソフトウェア技術チームヘッド．2011 年ゴードン・ベル賞受賞．