新ベンチマークソフトウェアHPCGの スーパーコンピュータ「京」上での性能改善

Performance Improvement of the New Benchmark Software HPCG on the K computer

理化学研究所計算科学研究機構 運用技術部門
熊畑　清、南　一生

世界で最も高速なコンピュータシステムの上位500位までを定期的にランク付けし、評価するプロジェクトとしてTOP500がある。TOP500プロジェクトは、1993年に発足したが、そこで用いられているLINPACKベンチマークプログラムが浮動小数点演算性能の評価に重きを置いているために、近年ではベンチマーク結果で示された性能と、実際のアプリケーションを実行した際の性能の乖離が指摘されるようになってきた。そこでTOP500プロジェクトの提唱者であるJ. Dongarra博士より、実アプリケーションで使われる計算手法の性能評価に重きを置いたベンチマークプログラムとしてHigh Performance Conjugate Gradient (HPCG)ベンチマークが提案された。2014年12月に開催されたSC14において、日本の「京」を含む世界のトップクラスのスーパーコンピュータのHPCGベンチマーク結果のランキングが発表され「京」は２位となった。本稿では我々が行った、「京」上でのHPCGベンチマークへの取り組みについて報告する。

１．はじめに

世界中のスーパーコンピュータの性能をランキングするTOP500プロジェクトで用いられるLINPACKベンチマークは長年にわたりスーパーコンピュータの性能指標として使われてきたが、本ベンチマークは浮動小数点演算性能の評価に特化しているため、通信性能やメモリ性能などの重要な特性が測定結果に反映されにくく、結果として、近年では実際のアプリケーションを実行した際の性能指標としては適切ではないという声がある。そこでTOP500プロジェクトの提唱者であるJ. Dongarra博士により、新しいベンチマークプログラムとしてHigh Performance Conjugate Gradientベンチマーク（以下HPCG）が提唱された［１］。我々は「京」のHPCG性能の評価と改善を行っており、その結果、2014年11月に開催されたSC14において世界のトップクラスのスーパーコンピュータのHPCG性能が発表された際に、「京」は世界２位にランクされた。本稿では我々が行った、「京」上でのHPCGベンチマークへの性能改善の取り組みについて報告する。

２．HPCG

HPCGは有限要素法などの実用アプリケーションで頻出する疎で対称な係数行列を持つ連立一次方程式を、ガウス・ザイデル法をスムーサーとして用いたマルチグリッド法前処理付き共役勾配法［２］で解く際の性能を測定するベンチマークである。

計算科学的な特長から言うとHPCGは疎行列・ベクトル積であり、アプリケーションが要求するB/F値が大きいタイプの計算となるため、CPU単体性能が出しにくいアプリケーションである。また、ベクトルへのアクセスがリスト値を介した不連続なアクセスとなるため、通常の要求B/F値が大きいアプリケ

ーションよりさらに性能を出しにくい。この意味では要求B/F値が小さいLINPACKとは対極にあるベンチマークプログラムであると言える。

プログラムとしてのHPCGの実行は、①パラメータに応じたサイズの行列生成、②行列データの最適化、③最適化の検証、④測定、⑤結果出力の５つのフェーズに大別することが出来る。②の行列データの最適化フェーズはデフォルトの状態ではなにも実行せず、ユーザが独自に実装する。③の最適化の検証フェーズは②でユーザが行った行列データの最適化が正当であるかを検証する。④ではCGを複数回実行しその時間を測定するのであるが、複数回実行する前に１回だけ実行した際の時間から、複数回実行に要する時間がパラメータファイルで指定した時間以上となるように実行回数を動的に決定する。正式には実行時間が１時間を超える必要がある。⑤では③の検証結果や、④の測定結果を結果提出用のファイルに出力する。

性能チューニングに際して手を加えて良いのは③の最適化フェーズと、④で呼ばれるCG関数及び行列･ベクトル積や、内積、マルチグリッド前処理でスムーサーとして用いるガウス･ザイデル関数などに限定されており、なおかつ数値解法自体の変更や、特定の行列データに特化したチューニングは行えない。

3.「京」上での性能評価とチューニング方針

HPCGコードの「京」上での特性を把握し、性能チューニングの方針を決めるために、初めに何も手を加えていないオリジナルコードを「京」上で実行し性能評価を行った。

まず初めに１プロセス当たりの問題サイズを固定にしたまま並列プロセス数を増やすウィークスケーリング測定により並列性能の調査を１プロセス当たりの問題サイズを128^3として行った。前述したとおり測定に際し結果を正式に提出するためには最低１時間の実行時間が必要であるが、ここではパラメータファイルを書き換えて測定時間を10分間とした。また、この際コンパイルオプションとしては「京」で典型的に用いられる推奨オプションである“-Kfast,openmp”を用いた。これによりSIMD化やソフトウェアパイプライニングなどのコンパイラによる自動最適化と、OpenMPによるマルチスレッド化が適用される。

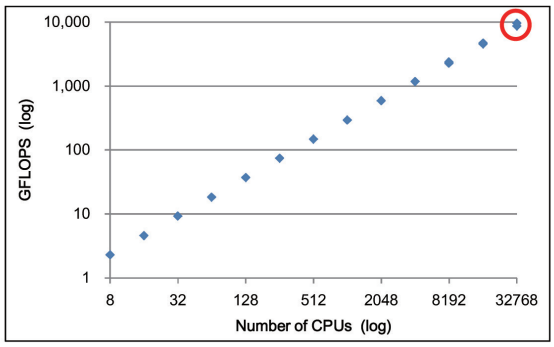

図１　「京」上でのオリジナルコードのスケーラビリティ（赤丸は32768プロセスで約9500GFLOPS）

測定結果を図１に示す。図中横軸は利用したCPU数（１プロセス当たり１CPU利用）、縦軸はHPCGの最終出力ファイルに記載されたGFLOPS値である。一見して判るとおり非常によいスケーラビリティが得られており、並列化効率は100%に近い。それゆえ我々は並列性能に関するチューニングは必要ないと判断した。一方でGFLOPS値は低く、各CPU利用時のピークGFLOPSに対する効率はいずれの場合も0.23%弱であったためCPU単体性能のチューニングが必要であることは明白である。

次いで単体性能のチューニングのため、プロファイラを用いてホットスポットの調査を行った。

なお測定条件はウィークスケール測定時と同様である。CGの全実行時間に対する、各

下位処理の実行時間が占める割合を図2に示す。これによると全実行時間の約98%程度が二つの処理で占められていることが判る。

ここで“ComputeSPMV_ref”は疎行列ベクトル積（以下SPMV）であり、“ComputeSYMGS_ref”はマルチグリッド前処理のスムーサーとして用いる対称ガウス・ザイデル法（以下SYMGS）である。これら2つのルーチンは計算のみを行うものである。また、各ルーチン名に付いているサフィックス”_ref”は参照実装であることを示しており、ユーザはこれらを書き換えて性能チューニングを行う。

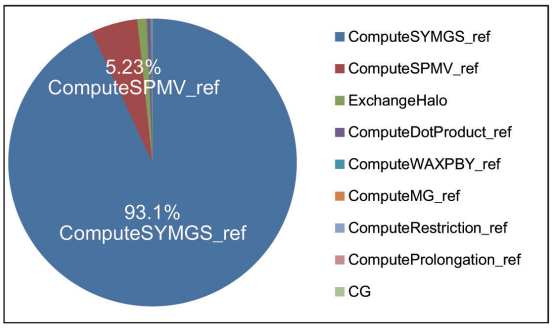

図2　CG実行中の各下位処理の時間分布

我々はこれら2種の処理に対し、メモリ配置の改善、データアクセス順序の改善、マルチスレッド化、ループ最適化などを施し性能改善を試みた。

4.「京」のCPU概要

ここでは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，16GB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6MB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で2way構成の32KBのL1データキャッシュ、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である。よってCPU全体で128GFLOPSの性能を持つ。理論メモリバンド幅は64GB/sである。

5．メモリ配置の改善

SPMVとSYMGSは共に処理としては疎行列とベクトルの積を実行するものであり、行列データが頻繁にアクセスされる。HPCGでは行列データを疎行列の格納形式の一つであるELL形式で保持しているが、各行の非ゼロ値と非ゼロ値の列番号を保持する配列のメモリを確保するに際して、各行ごとに動的メモリ確保を行っている。そのため、メモリ空間上では図3に示すように各行の情報が連続して配置されず、SPMV及びSYMGSの処理時には処理が次の行に移るごとに不連続なメモリアクセスが発生する。

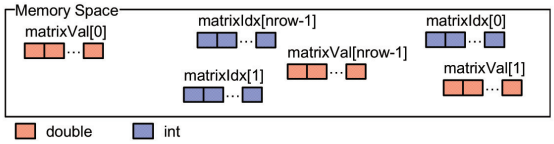

図3　メモリ空間の不連続性
各行ごとの情報が連続していない（matrixValは非ゼロ値、matrixIdxは非ゼロ値の列番号を保持する）

また、「京」ではメモリからのデータ転送は128Byteのキャッシュライン単位で行われるが、このような不連続なメモリ配置ではメモリアクセスが不連続になるという問題以外にも、各行の非ゼロ値の情報を保持する配列のサイズが128Byteの整数倍でない際には余計なデータ転送が発生してメモリバンド幅を消費するという問題も生ずる。そこで我々は行列データの格納に必要なメモリを一括して確

保するよう実装を変更した。これによりメモリ配置は連続となった。

メモリ配置改善前後のCG実行時間の比較を図4に示す。ここでコンパイル条件はウィークスケーリング測定時と同様である。実行条件は1プロセス当たりの問題サイズは同じ128^3とし、実行プロセス数は8プロセスとした。また、2節で述べたようにCGの実行回数は初めに1回だけCGを実行した際の時間から、全実行時間がパラメータファイルで指定された実行時間を超えるように実行回数を動的に決定している。そのためチューニングに伴い1回当たりのCG実行時間が短縮されると、CG実行回数が増加してしまい、チューニング前後での比較が難しくなる。よってこれ以降CGの実行回数は5回に固定している。

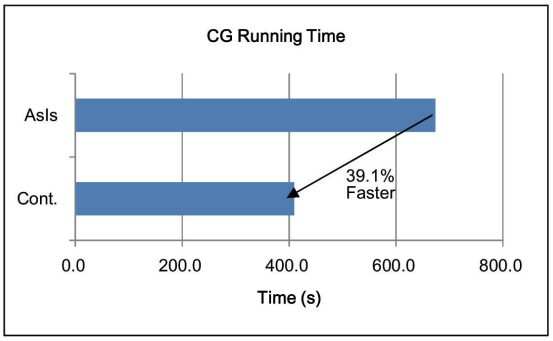

図4　メモリ連続化の効果

メモリ配置の連続化によりCG5回分の実行時間は39.1%短縮された。最終的なGFLOPS値は2.32GFLOPS（8CPU利用時の演算性能1024GFLOPSの0.226%）から3.82GFLOPS（同0.373%）へと向上した。

このときのメモリスループットはSPMVで19.02GB/s、SYMGSでは2.29GB/sであったが、STREAMベンチマークで測定した「京」の実効メモリスループットは46.6GB/sであることから、性能向上の余地をまだかなり残していると言える。

6．データアクセス順序の改善

マルチグリッド法のスムーサーとして用いる対称ガウス・ザイデル法の処理であるSYMGSには前進ループと後退ループの2つのループが含まれる。図5にSYMGSの後退ループのソース概要を示す。

Essene of SYMGS Backward
```
for(int i=nrow-1; i>=0; i--){
   double* Val = A.matrixVal[i];
   int*    Idx = A.matrixIdx[i];
   int     nz  = A.nonzerosInRow[i];
   double  Dia = *(A.matrixDiagonal[i]);

   double sum = rv[i];
   for(int j=0; j<nz; j++){
      sum -= Val[j]* xv[Idx[j]];
   }
   sum += xv[i] * Dia;
   xv[i] = sum / Dia;
}
```

図5　SYMGSの後退ループのソース例

この実装中では後退ループであるため外側ループiの回転方向は逆方向であるのに対して、内側ループjの回転方向は順方向となっている。そのため外側ループiが次の回転に移った際に、内側ループjで最初にアクセスするメモリアドレスはキャッシュに乗っていない可能性が高い。図6は外側ループiの回転がi=100からi=99に移った際のメモリ空間の模式図であるが、内側ループがj=0でアクセスする位置は図中Aの位置であるためキャッシュに乗っている可能性が低いことが判る。しかし内側ループjの実行順序に制約が無いため内側ループjの回転方向を外側ループiと同じにしてやることで、外側ループiが次の回転に移った際に、内側ループjが最初に触るアドレスは、直前にアクセスしたアドレスの近隣となり、キャッシュに乗っている可能性が高くなる。

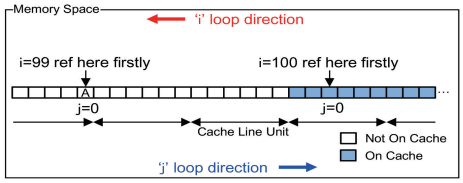

図６　SYMGSの後退ループでのアクセス順序

SYMGSの後退ループにおいて最内ループjの回転方向を反転した後のCG実行時間の比較を図７に示す。反転によりCG５回分の実行時間は２％ほどしか改善しなかった。最終的なGFLOPS値は、3.82GFLOPS（0.373%）から3.89GFLOPS（0.380%）にとどまった。

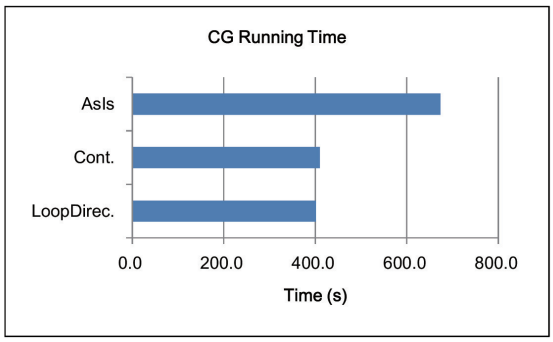

図７　ループ回転方向反転の効果

しかしループ方向反転には全く改善の効果が無いかと言えばそうではなく、メモリスループットの観点ではSYMGSの後退ループは2.29GB/sから2.60GB/sへと向上した。これは外側ループiの回転による参照メモリアクセスの移動が連続的になったためプリフェッチがうまく働くようになったためと思われる。この改変は、原理的に副作用がなく、ある程度は確実に効果があることから、CG実行時間の削減という観点では大きな効果がなかったもののこれ以降も採用している。

７．マルチスレッド化

図５を見るとSYMGSの後退ループでは、内側ループj中では参照され、外側ループiでは更新されるベクトルxvについてのデータ依存性があるため、外側ループiはコンパイラ指示子の挿入などにより簡単にマルチスレッド化することができない。一方で内側ループjにはデータ依存性がないためマルチスレッド化できるが、扱っている行列が疎行列であるため、多くの場合内側ループjの回転数は小さく、そのためマルチスレッド化の効果はない。これはSYMGSの前進ループについても同様である。

SYMGSのループにマルチスレッド化を適用するためにはデータ依存性を除去するしか無く、ここではデータ依存性除去のため、単純なカラーリングと、データをブロック化した後カラーリングする手法を検討した。

7.1. 単純なカラーリング

ここで言う単純なカラーリングとは、データ依存性のある計算対象、すなわちここでは行列の各行に対しそれぞれ異なるカラーを割り当て、同じカラーを割り当てられた行にはデータ依存性がないようにする手法である。同じカラーの行にはデータ依存性が無いため、マルチスレッドで処理することが出来る。ここでは図８に示すように行列の行間の隣接関係を、行は節点、隣接関係は辺として表すメッシュ構造として捉え、隣接する節点に異なるカラーを付している。

さらにメモリアクセスが連続となるように同じカラー内で行の番号が連続となるよう並べ替え、最終的にはメモリ配置が連続となるよう並べ替える必要がある。

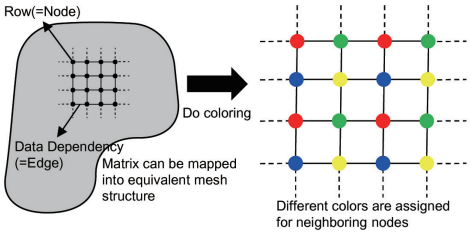

図８　単純なカラーリング模式図

図９はカラーリングに対応したメモリ配置の、並べ替えを示している。図中Row １、Row ２、… Row Nと示されている矩形が行列のある１行分の情報を保持する連続メモリ領域を示しており、行列中で最も多くの非ゼロを持つ行での非ゼロ個数をMとすると、全ての行で非ゼロ値の情報を保持する配列はM個のdouble、非ゼロ列番号はM個のintからなる。図の中段に示すようにカラーリングにより各行にそれぞれのカラーがつけられた状態を考える。データ依存性のない集合である赤について処理を行うと、処理する行はRow １、Row ３、Row ６、…となり、行の番号すなわちメモリ配置が不連続となる。そこでメモリ空間上で同じカラーの行が連続するようにソートする。図ではRow １をRow １の位置に、Row ３をRow ２の位置に、Row ６をRow ３の位置になるようなソートをして図の下段に示すように各カラーの行が連続したメモリ配置が得ている。これらの処理を２節で述べたユーザ定義の行列データ最適化に実装した。

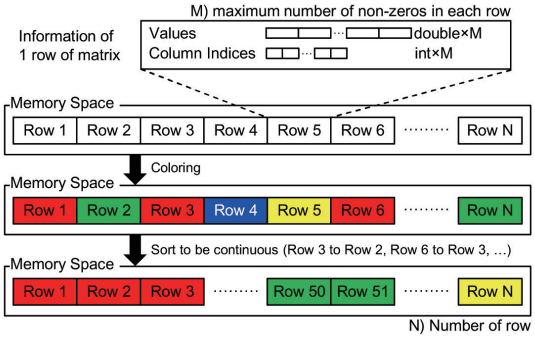

図９　カラーリングに対応したメモリ並べ替え

これに対応して、ソースコードにはこれまで外側ループだったループiの外側にさらにカラーを赤、緑、青、黄などの順番で動かすループicを追加し、ループiの回転範囲をカラーに応じたものとなるよう変更する。これによりループiは同じカラー中の行、すなわちデータ依存性のない行を処理するループとなるためコンパイラ指示子の挿入によりマルチスレッド化することができる。図10にはSYMGSの後退ループの例を示しているが、SYMGSの前進ループも同様である。

単純なカラーリングを行った際のCG実行時間の比較を図11に示す。カラーリングによりSYMGSがマルチスレッド実行可能となったため、CG５回分の実行時間はループ方向の反転から57.3%短縮された。最終的なGFLOPS値は3.89GFLOPS（0.380%）から8.62GFLOPS（0.84%）へと向上した。時間の短縮と、GFLOPS値の向上とが一致していないが、これはHPCGがGFLOPS値を計算する方法に起因する差異である。

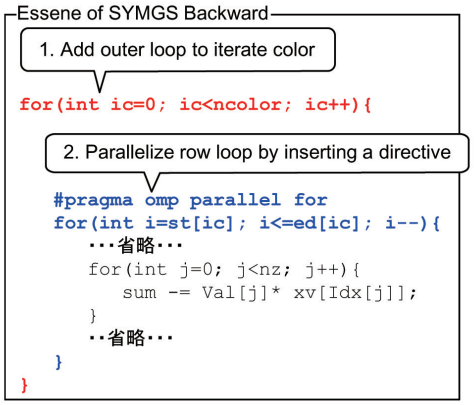

図10　単純なカラーリングに対応したソースコード例

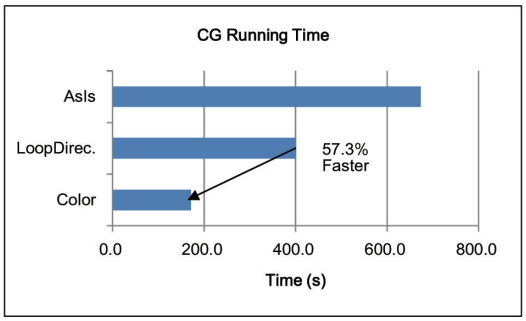

図11　単純なカラーリングの効果

HPCGでは２つの方法でGFLOPS値を計算している。一つは問題サイズから理論的に

導かれる演算量を実際のCG実行時間で割ったものであり、もう一つは理論的に導かれる演算量を、CG実行時間と、ユーザ定義の行列データ最適化に要した時間に重み係数0.1とCG実行回数を乗じた時間との合計で割って算出したものであり、最終的に提出するGFLOPS値としてはこちらが採用される。これまではユーザ定義の行列データ最適化は何も実装していなかったためその時間は0秒であったが、ここではカラーリングとそれに付随する並べ替え処理を実装したため、最適化時間に約19秒を要し、結果、CG実行時間の短縮量と、GFLOPS値の向上量が一致しなくなった。CGの実行時間だけから算出したGFLOPS値は9.09GFLOPS（0.88%）である。

SYMGSの後退ループのメモリスループットは2.60GB/sから9.34GB/sへと向上したが、「京」の実効メモリスループット46.6GB/sと比べるとまだ相当に低く、また1スレッド実行から8スレッド実行になったにも関わらず実行時間は約155秒から約52秒と、約66%しか向上していない。またSPMVは当初よりマルチスレッド化されているためこの段階ではソースには手を加えていないにもかかわらず、カラーリング及び並び替えの副作用でキャッシュミス率がL1データキャッシュで14.85%から26.33%へ大きく悪化し、その結果実行時間が10.97秒から24.71秒に劣化した。これはオリジナルコードの18.12秒よりも遅く問題である。

7.2. ブロック化カラーリング

この副作用を避けるために我々は図8に示したメッシュ構造で近隣の節点をある程度の大きさのブロックへと分割し、ブロックをカラーリングする手法［3］［4］を採用した。この手法は我々が過去に行った有限要素法による汎用流体シミュレーションコードに対するチューニングにおいて、SYMGSと同様にデータ依存性があるため単純には並列化できないループの並列化に効果があった。

この手法でもまず、7.2節のカラーリング手法と同様に行列と等価なメッシュ構造を想定する。次いで図12に示すように、メッシュを複数のブロックへと分割する。ここで各ブロックには可能な限り等しい数の節点、すなわち行が含まれるようにしておく。カラーリングは各節点ではなくこのブロックに対して行う。当然ここでもメモリ配置を連続的にするため、前述の手法と同様に、メモリ上で行データの並べ替えを行い、行データを各ブロック中で連続にする。従って、同じカラーに属するブロック間にはデータ依存性がないので、図13に示すように、各ブロックをマルチスレッドで処理することができる。一方で、あるブロック内の行群にはデータ依存性があるため行をマルチスレッドで処理することはできない。

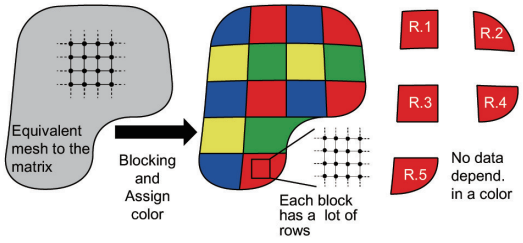

図12　ブロック化カラー模式図

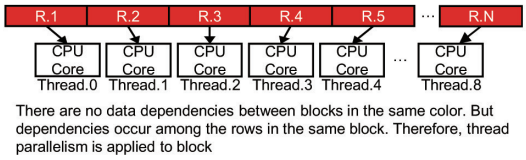

図13　ブロックに対するスレッド割り当て

これに対応して、ソースコードにはこれまで外側ループだったループicの内側に、さらに、同じカラー内のブロックを動かすループibを追加し、スレッド並列化のためのコンパイラ指示子はループibに挿入する。図14には

SYMGSの後退ループの例を示しているが、SYMGSの前進ループも同様である。

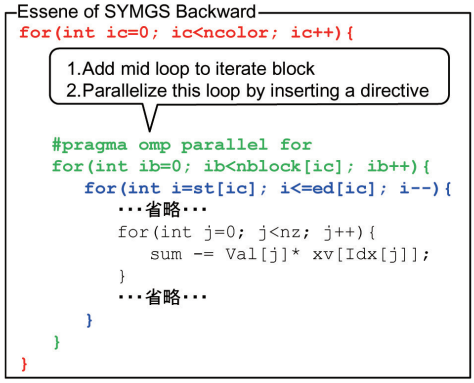

図14　ブロック化カラーリングに対応したソースコード例

このブロック化カラーリングでは、ある一つのブロックに含まれる行は、ブロックの境界面にあるものを除いて、同じブロック中の行とだけデータ依存性があり、かつ同じブロック中ではメモリ上で連続的に配置しているため、ベクトルxvの各成分へのアクセスが時間的にも空間的にも局所的になり、キャッシュの利用効率が良くなるという長所がある。

ブロック化カラーリング適用前後のCG実行時間の比較を図15に示す。

ここでは１ブロックが含む行数はおよそ1000とした。ブロック化カラーリングによりCG５回分の実行時間は72.1%短縮され、最終的なGFLOPS値は8.62GFLOPS（0.84%）から31.37GFLOPS　（3.06%）に向上した。CGの実行時間のみから算出したGFLOPS値は33.29GFLOPS（3.25%）である。

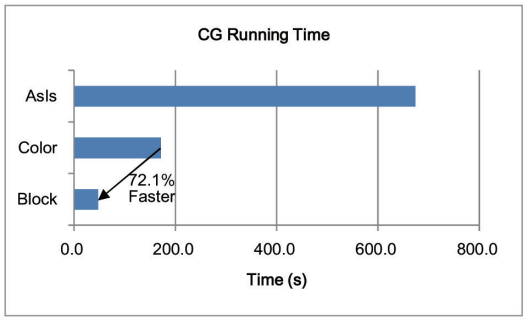

図15　ブロック化カラーリングの効果

単純なカラーリング適用時には副作用により性能劣化してしまったSPMVについては、ブロック化カラーリングによりＬ１データキャッシュミス率が26.33%から4.77%に低減した。実行時間は24.71秒から5.89秒へと短縮した。

SYMGSの後退ループの実行時間は単純カラーリングの51.75秒から13.49秒へと短縮した。これはカラーリング適用前の155.47秒のおよそ１/11以下であり、実行スレッド数が１から８になった以上の効果が出ているが、カラーリング適用前と比してＬ１データキャッシュミス率は28.34%から5.34%に低減しており、先に述べたようなブロック化カラーリングが持つメモリアクセスの局所化の効果によるものである。

メモリスループットの観点ではSPMVは36.69GB/s、SYMGS後退ループは31.96GB/sとなっており、性能向上の余地があると考えられる。

８．ループ最適化

SPMV、SYMGSともにメモリスループットの観点ではまだ性能向上の余地があるため、さらなる最適化を試みた。図16はSPMVのソースの概要であるが、二つの問題、非ゼロ値への参照経路の複雑さと、内側ループの短さがあることが判る。

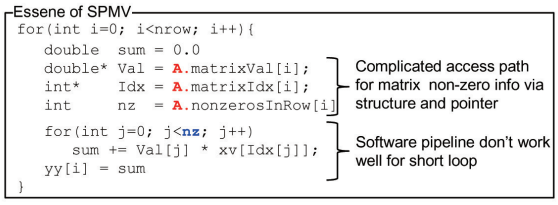

図16　SPMVソース

まず、非ゼロ値への参照経路の複雑さであるが、この実装では、ある行の非ゼロ値の情報（値と列番号）を得るには、行列の構造体Aのメンバであるポインタ配列から各行の非ゼロ値の情報を保持する配列の先頭ポインタを取得し、そのポインタから非ゼロ値の情報を得るという流れになり複雑である。なぜ一旦ポインタ配列を介すのかというと、オリジナルのHPCGでは5節で述べたように、各行の非ゼロ値の情報を保持する配列は行ごとにメモリ空間上で不連続に配置されているためである(図17上段)。しかし5節のチューニングにより、各行の非ゼロ値の情報を保持する配列はメモリ空間上で連続的に配置されるようになっており、また行列の保持形式がELL形式であること、つまり各行の非ゼロ値の情報を保持する配列の長さは全ての行で一定であるため、ポインタ配列を介さずとも各行の非ゼロ情報の配列にアクセスすることができる（図17下段)。

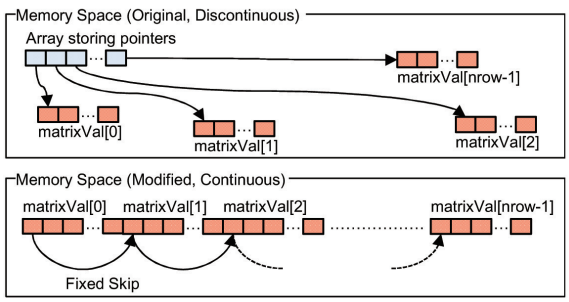

図17　複雑なアクセス経路とその改善

次に内側ループの短さであるが、内側ループjはnz回転するループであり、コンパイラにより自動的にソフトウェアパイプラインが適用されている。ループjの回転数nzは行によって異なるが、HPCG中で測定のために生成される行列は最大でも27回転と小さい。HPCGに限らず有限要素法などの係数行列は疎行列であり行が持つ非ゼロ値の数は小さいことが多く、このようにループ回転数が小さい場合のソフトウェアパイプラインは効果が小さい。この場合の改善策としてはループjの回転数を27に固定し、ループjをアンロールすれば良い。ただし回転数をHPCGで生成される行列に特化した最大27回転に固定することは、2節で述べた「特定の行列データに特化したチューニングをしてはならない」というルールに抵触する可能性がある。この問題を解決するために我々は、さまざまな回転数に特化した複数のSPMVループを用意し、行列に応じてどのSPMVループを通るかを実行時に動的に切り替える方式を採用した。この手法は我々が過去に行った有限要素法による汎用流体シミュレーションコードに対するチューニングにおいて採用した手法であり、汎用コードのように様々な入力データを扱う必要がある場合に有用である。この改善よりループjがフルアンロールされ、結果ソフトウェアパイプラインは充分に回転数がある上位ループiに適用されるようになった。ここで我々はループiのアンロールについても検討し2展開することとした。

図18にSPMVループについて上記の改善を適用したソースコード例を示すがSYMGSに関しても同様である。

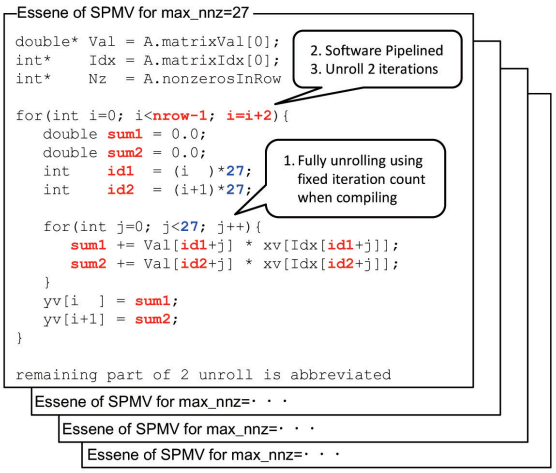

図18　改善後ループ例

ループ最適化を行う前後のCG実行時間の比較を図19に示す。ループ最適化により、CG 5 回当たりの実行時間はおよそ29.9%改善された。最終的なGFLOPS値は31.37GFLOPS（3.06%）から44.01GFLOPS（4.29%）まで向上した。

CGの実行時間のみから算出したGFLOPS値は47.49GFLOPS（4.63%）である。メモリスループットの観点では、SPMVはSTREAMベンチマークによる測定値46.6GB/sを超える47.85GB/s、SYMGSの後退ループは44.38GB/sと充分な値が出た。

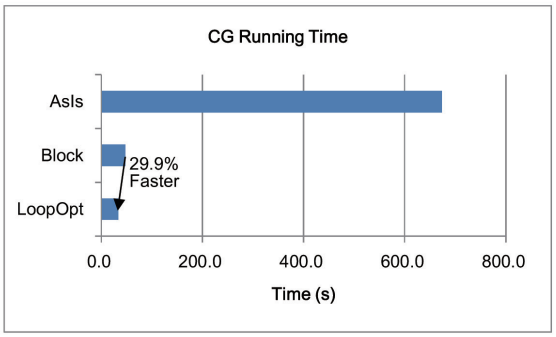

図19　ループ最適化の効果

9．その他最適化と最終測定結果

この時点で「京」フルノード82,944ノードを用いた測定を行った結果、0.427PFLOPS（4.02%）を得た。この値は2014年 6 月のISC14で発表されたHPCGのプレランキングでは中国のTianhe- 2 の0.58PFLOPSに続いて 2 位にランクされた。

SC14に向けて、我々はさらにパラメータ最適化と、ユーザ定義の行列データ最適化部の改善を行った。

パラメータ最適化として 7 節で述べたブロック化カラーリングの 1 ブロック当たりの行数を調節して、最終的にブロック当たり約2625行となるように調節した。

次いでユーザ定義の行列データの最適化部の改善であるが、 7 節で述べたようにHPCGでは最終的に提出するGFLOPS値は、CG実行時間と、ユーザ定義の行列データ最適化に要した時間に重み係数0.1とCG実行回数を乗じた時間との合計で割って算出したものであるため、ユーザ定義の行列データ最適化時間を短縮することが重要である。我々は最適化部分の実装をリファインすることで最適化に要する時間を3.88秒から1.39秒へと短縮した。

この時点で「京」フルノード82,944ノードを用いた測定を行った結果、0.461PFLOPS（4.34%）を得た。表 1 に2014年11月のSC14で発表されたHPCGランキングの上位 5 つのスーパーコンピュータとそのHPCG性能を示す。このランキングでは「京」はISC14でのランキングと同様に 2 位にランクされた。

表 1　SC14でのHPCGランキング上位 5 件

Rank	Computer	HPL PFLOPS	HPCG PFLOPS	Ratio to HPL %
1	Tianhe-2	33.86	0.623	1.8%
2	K computer	10.51	0.461	4.4%
3	Titan	17.59	0.322	1.8%
4	Mira	8.59	0.167	1.9%
5	Piz Daint	6.27	0.105	1.7%

10. むすび

「京」の結果はHigh Performance LINPACK（HPL）の性能では「京」よりも高いアメリカのTitanの0.322PFLOPSよりも上位であり、また、HPLによるFLOPS値に対するHPCGによるFLOPS値の比率が4.4%と他のスーパーコンピュータよりも高いことから「京」の性能をよく引き出せていると言えるであろう。

本稿で述べたチューニングによって「京」上でのHPCGの性能はおよそ19.5倍に向上した。コードの要求メモリバンド幅から推定する方法［5］によって、主要ループであるSPMVとSYMGSの「京」上での理論的な性能上限を推察すると5.72%および6.87%となるが、我々のチューニングでは5.42%および6.13%が得られており、ほぼ性能上限に達していると考えられる。

参考文献

［1］ Dongarra J, Luszczek P, Heroux M, Toward a new (another) metric for ranking high performance computing systems, SC' 13 Top500 BoF, (2013).

［2］ 建部修見，小柳義夫，マルチグリッド前処理付き共役勾配法，情報処理学会研究報告ハイパフォーマンスコンピューティング，Vol.66，pp.9-16，(1992)．

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

［4］ Kumahata K, Inoue S, Minami K, Kernel performance improvement for the FEM-based fluid analysis code on the K computer, Procedia Computer Science 18, pp. 2496-2499, (2013).
DOI: 10.1016/j.procs.2013.05.427.

［5］ 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，2012年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講演論文集，pp.23-31，(2012)．