Vol.2016-HPC-154 No.5
2016/4/25

OpenPOWER クラスターにおける HPCG ベンチマークの最適化手法に関する考察

土井淳[†1]　奥野伸吾[†2]

概要：HPCG ベンチマークは，スーパーコンピューターの性能を評価する新しい標準的なベンチマークプログラムとして近年注目されつつあり今後重要なベンチマークの一つとなると考えられる．HPCG ベンチマークでは，マルチグリッド法を前処理に用いた CG 法によって線形方程式を解く問題を扱っており，マルチグリッド法の部分が処理時間の大半を占める．マルチグリッド法としてガウス・ザイデル法が採用されているため，GPU のようなメニーコア環境において大規模並列処理をどのように行うかが問題となる．本報告では，主にマルチグリッド法の最適化について，いくつかの最適化手法を，OpenPOWER クラスターの上で試し性能とベンチマークスコアについての考察を行う．

Optimization Study of HPCG Benchmark on OpenPOWER Clusters

JUN DOI[†1]　SHINGO OKUNO[†2]

Keywords: GPU, GPGPU, CUDA, HPCG, CGM, Gauss-Seidel, POWER

1. はじめに

近年のスーパーコンピューターの進化は，消費電力の増大と動作周波数の限界から，マルチコアやメニーコアによる計算能力の増強にシフトしつつある．特に，GPU のようなアクセラレーターを CPU とは別に搭載したハイブリッドなクラスターが多く見受けられるようになってきた．OpenPOWER クラスターのハイパフォーマンス・コンピューティング向け製品も，POWER プロセッサーに NVIDIA の GPU を搭載した構成となっており，これまでの GPU クラスターと同様にシミュレーションプログラムを加速することができる．

HPCG ベンチマークは，スーパーコンピューターの性能を評価するために提唱された新しいベンチマークで，HPL ベンチマークが計算機の計算能力そのものを評価しているのに対して，HPCG ベンチマークは，メモリアクセス等を考慮したより実践的なベンチマークとなっていて，重要なベンチマークの一つとなりうる．また，HPL ベンチマークが大規模な密行列を扱うのに対して，HPCG ベンチマークは大規模な疎行列を扱い，こちらもまた実際のアプリケーションに近いベンチマークであるといえる．

本報告では，IBM のハイパフォーマンス・コンピューティング向け OpenPOWER 製品において，HPCG ベンチマークの最適化についての考察を行い，性能を評価する．

2. OpenPOWER 概要

OpenPOWER は，POWER プロセッサーを中心とした計算機システムの仕様をオープン化することで，様々な用途に適応した新しい計算機システムを開発することを目的に，多くの企業，研究機関，大学等が参加する，OpenPOWER Foundation[1]によって推進されている．IBM では，IBM Power System として，汎用計算機クラスター製品や，NVIDIA の科学計算向け GPU を搭載したハイパフォーマンス・コンピューティング向けのクラスター製品などを展開している．

IBM のハイパフォーマンス・コンピューティング向けの OpenPOWER クラスターは，2016 年前半時点では，IBM Power System S822LC[2]であり，ノードあたり 2 台の POWER8 プロセッサーと，2 台の NVIDIA Tesla K80 を搭載する．リトルエンディアン対応の Linux をサポートしており，ほぼ従来の一般的な GPU クラスター環境と同等のハイパフォーマンス・コンピューティング環境を構築できるようになっている．2016 年後半以降には，NVIDIA の次世代 GPU である，Pascal アーキテクチャーの GPU を搭載したクラスター製品が登場する予定で，POWER8 プロセッサーと GPU は，NVLink[3]と呼ばれる新しい高速インターコネクトネットワークで結合されるようになる．NVLink によって GPU への処理のオフロードが効率良く行えるようになると期待される[4].

3. HPCG ベンチマーク概要

HPCG（High Performance Conjugate Gradients）ベンチマーク[5]は，Dongarra らによって提唱された新しいベンチマークプログラム[6]で，現在 Top500 で使用される HPL（High Performance LINPACK）ベンチマーク同様に，今後ハイパフォーマンス・コンピューティングのための重要な指標となりうるベンチマークの一つである．HPCG ベンチマークではその名の示すとおり，共役勾配法（CG 法）を用いて線型

†1 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research – Tokyo
†2 京都大学 大学院 情報学研究科
Graduate School of Informatics, Kyoto University

方程式の解を求めるプログラムで，有限要素法や差分法等を用いたシミュレーションプログラムで広く使用される計算手法であり，実践的なベンチマークテストと言える．

HPCG ベンチマークでは，3 次元構造格子点上の問題を 27 点ステンシル計算で解くために，隣接点情報を疎行列として保持し，疎行列ベクトル積を用いて CG 法の収束計算を行う実装になっている．ただし，3 次元構造格子点上のステンシル計算であるということは，そのように問題を生成しているというだけで，最適化をする上で 27 点ステンシル計算（から生成される疎行列パターン）に限定することは許されておらず，一般的な疎行列に対応した実装である必要がある[6].

HPCG ベンチマークでは，CG 法による収束回数を減らすための前処理として，マルチグリッド法が用いられている．マルチグリッド法は，図 1 に示すように，元の格子サイズから，何段階かの粗い格子を生成し，それぞれの格子上で行列のスムージングを細かい格子から粗い格子へ往復的に行う手法である．HPCG ベンチマークではスムージングとしてガウス・ザイデル法が用いられており，4 段階までの粗い格子が使用される．

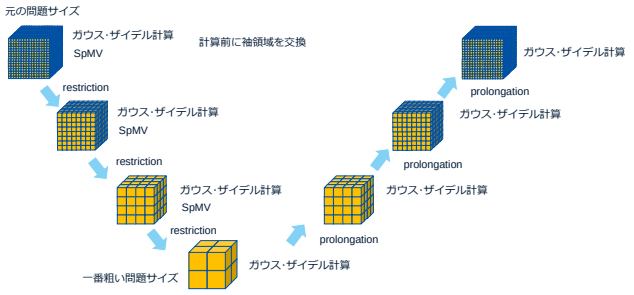


図 1　HPCG ベンチマークにおけるマルチグリッド法による前処理．ガウス･ザイデル法を使用する．

ガウス・ザイデル法には計算順序に依存関係があるため単純に並列計算をすることは難しい．ガウス・ザイデル計算を GPU のようなメニーコア環境で並列化を行うためには，カラーリングによって疎行列を並び替えることで，異なるスレッド間の値の書き換えによる不整合を防ぐ手法が広く用いられている．しかしながら，この手法は厳密には元のガウス・ザイデル計算と同じ計算をしていることにはならず，HPCG ベンチマークの実装として様々な議論があるようだ．また，並列化していない場合よりも収束回数が増えることが知られており，HPCG ベンチマークでは，収束回数の増加率によって，ベンチマークスコアを減点する仕組みとなっている．

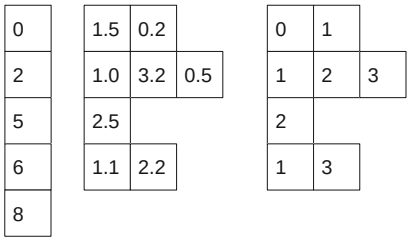
HPCG ベンチマークでは，マルチグリッド計算と，疎行列ベクトル積の 2 つの処理が実行時間の大半を占め，特にマルチグリッド計算の割合が大きい．よって，これら 2 つの処理を最適化することでベンチマークスコアをあげることができる．しかしながら，先に述べたように並列化によって生じる収束回数の増加分と，最適化の準備にかかる時間によって，ベンチマークスコアを減じる仕組みがあり，これらの要素も考慮した最適化が必要となる．また，HPCG version 3 より，問題を生成する時間によっても，ベンチマークスコアが減じられるようになり，大きな問題を効率良く生成できる工夫も必要となった．

4. CUDA による HPCG の最適化

4.1 疎行列のデータ構造の変更

元の HPCG ベンチマークでは, 疎行列は CSR（Compressed Sparse Row）形式が使用されている．CSR 形式は，式(1)に示すような 4x4 の正方行列を例にすると，図 2 のように，それぞれの行について，行列の非ゼロ要素とその列のインデックスを格納する方式である．また，図 2(a)のような各行の先頭を示すインデックスも必要となる．また，行ごとの非ゼロ要素の数の偏りがそれほど多くないのであれば，ELL（ELL-Pack）形式を用いることで比較的効率良く処理ができることが知られている．ELL 形式は，図 3 のように各行の非ゼロ要素数が一定になるように配列を割り当て，非ゼロ要素数が少ない場合は 0 で埋めることで，各行の非ゼロ要素数を個別に扱うことなく，条件分岐を省いた効率的な処理が可能となる．

$$\begin{pmatrix} 1.5 & 0.2 & 0 & 0 \\ 0 & 1.0 & 3.2 & 0.5 \\ 0 & 0 & 2.5 & 0 \\ 0 & 1.1 & 0 & 2.2 \end{pmatrix} \quad (1)$$



(a) 行のポインタ　(b) 行列の非ゼロ要素　(c) 非ゼロ要素の位置

図 2　CSR 形式による疎行列の格納の例

1.5	0.2	0.0
1.0	3.2	0.5
2.5	0.0	0.0
1.1	2.2	0.0

0	1	1
1	2	3
2	2	2
1	3	3

(a) 行列の非ゼロ要素　(b) 非ゼロ要素の位置

図 3　ELL 形式による疎行列の格納の例

GPU において CUDA コアを用いて並列化するとき各スレッドが 1 行分の計算を行うとすると，図 4 のように，ELL 形式の配列を転置して格納することで，行列の非ゼロ要素とインデックスをコアレスアクセスを利用して効率良く扱うことができる．そこで，本報告では，図 4 に示すような, 転置 ELL 形式を使用し行列要素とインデックスを格

納した．

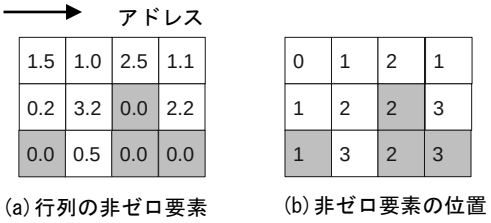


図 4　コアレスアクセスを考慮した ELL 形式の格納例

4.2 カラーリングによるガウス･ザイデル計算の並列化

ガウス・ザイデル法は，式(2)に示すように，先に計算された値を参照して次の値を書き換えていくような計算順序の依存性があるため，単純な並列処理はできない．

$$x_i^{(m+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{m}\right) \qquad (2)$$

また，単純に並列計算をしてしまうと，参照先のデータが書き換えられることによって，正しく同期処理を行わないと計算結果が不定になってしまう問題もある．

そこで，依存関係を取り除いて並列計算を行うために，カラーリングが用いられる．ベクトルの行を節点として依存関係を表すグラフを考えたとき，隣接する節点同士が異なる色になるようにグラフのカラーリングを行う．こうすることで同じ色の行同士には依存関係が無いので，並列計算することが可能となる．グラフを *Nc* 色に塗り分けた場合，*Nc* 回に分けて，それぞれの色の行の集合を並列計算すれば良い．HPCG ベンチマークでは三次元構造格子上の問題を扱うため，図 5 のように幅 2 の立方体 8 要素がすべて異なる色になるように 8 色で色分けすれば良い．（ただし，実装上は 8 色にカラーリングできることを知っていてはいけない．）

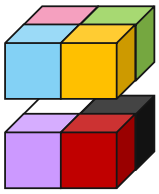


図 5　三次元構造格子における 8 色カラーリング

行のカラーリングができたら，連続的に処理ができるように，行を並び替える．つまり，ELL 形式で保存される行列要素とインデックスを色別に連続的に処理できるように並び替える．ベクトル成分へのアクセスは元々コアレスアクセスにはならないため，ベクトル成分は並び替える必要は無い．

CUDA による GPU での実装では，スレッドあたり 1 行分を処理するようにし，各色毎に逐次カーネル呼び出しを行うことでガウス・ザイデル計算を行う．なお，ガウス・ザイデル計算では，順方向，逆方向に往復的に処理を行うので，色番号の昇順に繰り返しカーネルを呼び出した後，降順に繰り返す．このとき，一番大きな色番号の集合について，順方向，逆方向で同じ計算を行うため，逆方向の処理の分は省略できる．

4.3 階層的カラーリングによるガウス･ザイデルの並列化

前述のカラーリング手法では，ベクトル成分のメモリアクセスについて考慮されていない．例えば，図 6 では，左図の A と C を処理したい場合に図 6 の右に示す数字の回数同一カーネル内でベクトル成分が参照されるが，自分自身の成分は一度しかアクセスされないなどベクトル成分の再利用があまり行われないことがわかる．

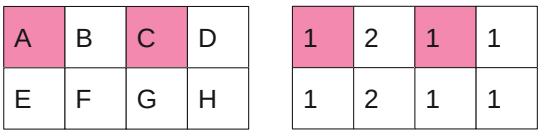


図 6　カラーリングによるベクトル成分へのアクセス回数

これに対して，二色分の処理を同一のカーネルで計算できるとすると，図 7 のように，ベクトル成分を再利用できる機会が増える．

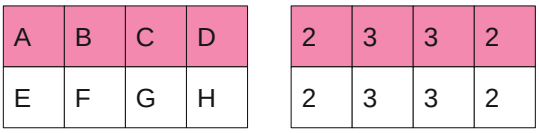


図 7　隣接する二色分をまとめて処理する場合のベクトル成分へのアクセス回数

これらのベクトル成分を，CUDA の共有メモリを利用して，スレッド間で共有すれば，ベクトル成分の再利用が効率良く行える．また，自分自身の行のベクトル成分へのメモリアクセスはコアレスアクセスになるため，各行に対応するスレッドが自分自身のベクトル成分を共有メモリに読み込むようにする．このように複数色分の処理を一度のカーネル呼び出しで実現するために，本報告では階層的カラーリング手法を提案する．

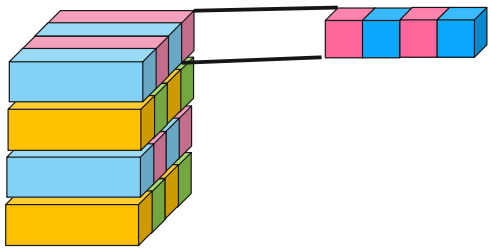


図 8　階層的カラーリングの例．三次元構造格子を階層的に 4 色と 2 色にカラーリング

図 8 は，三次元構造格子上の問題の場合の，階層的なカラーリング例で，連続する格子点でグループを作った上で，グループにカラーリングを適用する．同時に，グループ内の格子点についてもカラーリングを行うことで，階層的カラーリングを行う．この例では，X 軸方向に連続する格子点を 1 つのグループとすることで，グループ内は 2 色でカ

ラーリングし，グループは 4 色にカラーリングしている．他にも，グループ内を 4 色でカラーリングし，グループを 2 色または 4 色でカラーリングすることも考えられる．このように階層的にカラーリングすることで，グループ間に依存関係が生じないため，同じ色のグループは同時に計算でき，グループ内では依存関係はローカルなカラーリングで記述されているため，依存関係を考慮した並列化が可能である．階層的ではない実装同様に，グループの色数分，逐次に処理を行う．CUDA カーネルの呼び出し時は，グループ内の点数分のスレッドを用い，グループ数分のスレッドブロックを使用する．

HPCG ベンチマークの最適化実装では，三次元構造格子であることは意識してはいけないため，実際の実装では，(i)グループ毎の最大色数，(ii)グループ内の最大点数，の 2 つの値を与えることで，連続する点でグループを形成しながら隣接点間で同じ色にならないようにカラーリングを行い，最後にグループのカラーリングを行う実装になっている．(ii)はスレッドブロックあたりのスレッド数を超えず共有メモリ容量が足りる範囲で最適な値を選択して指定する．問題サイズによってこれらのパラメーターをどのように設定するかを検討する必要がある．

次の擬似コードは，複数の色のガウス・ザイデル計算を一度のカーネル呼び出しで処理するための CUDA カーネルである．

```
共有メモリ: buf
buf[i] = 自分のベクトル成分の読み込み
グループ外を参照するガウス・ザイデル計算
do グループあたりの色数
  スレッド間同期（__syncthreads）
  if 自分が処理対象の色 then
    buf を参照したグループ内のガウス・ザイデル計算
    buf[i]に計算結果を保存
  endif
enddo
buf[i]をグローバルメモリに書き出し
```

まず，グループの外部を参照する部分については各点に割り当てられた色に無関係に計算できるので先に計算をしておく．続いて，グループ内部を参照する部分は，計算に依存関係があるので，色順に処理を行う．グループ内の参照は，ベクトル成分が共有メモリ上にあるのでメモリアクセスを減らすことができる．計算が終わったら結果を共有メモリに保存し，他のスレッドが更新された値を参照できるようにする．色を切り替えるときにスレッド間同期を取ることで，共有メモリ上のベクトル成分が正しく書き換えられていることを保証する．

また，ELL 形式の疎行列データのインデックスには，非ゼロ要素の列番号が入るが，共有メモリを参照するためにグループ内の非ゼロ要素のインデックスには，グループの先頭からの番号を保存する．よって，グループ内の最大点数を 256 に限定すれば，この部分の配列を 1 バイト整数にすることが可能であり，メモリアクセス量を減らすことができる．

階層的カラーリング手法は，ガウス・ザイデル計算だけではなく疎行列ベクトル積についても，共有メモリを利用してグループ内部の参照を効率良く行うことが可能である．

5. OpenPOWER クラスターにおける性能評価

5.1 評価環境

HPCG ベンチマークの評価には次の 2 種類の OpenPOWER クラスターを利用した．GPU の単体性能を評価するにあたり，Tesla K80 はやや特殊な GPU であり評価が難しいため, GPU 単体性能の評価は Tesla K40 を用いた．

表 1　性能評価に用いた OpenPOWER クラスター

	(1) IBM Power System S824L	(2) IBM Power System S822LC
GPU	2*Tesla K40	2*Tesla K80
CPU	2*POWER8	2*POWER8
# of cores	2*12	2*10
CPU peak	289.92 GFlops /cpu	233.6 GFlops / cpu
Memory bw	192 GB/s	192 GB/s

どちらのクラスターでも，NVIDIA CUDA Toolkit version 7.5 を使用し，いずれの GPU も ECC ありの状態で実行した. また, 使用したコンパイラーは, IBM XL C/C++ for Linux V13.1.3 で，MPI 並列化には，IBM Parallel Environment for Linux on Power V2.3 を使用した．

本報告では，(i)単純なカラーリングを用いた場合，(ii)階層的カラーリングを用いた場合，の 2 つの実装について，マルチグリッド法，疎行列ベクトル積を中心に，性能を評価する．また，HPCG ベンチマークスコアに影響する最適化に要する時間と反復回数の増分についても評価する．

5.2 Tesla K40 による GPU 単体性能評価

まず，GPU 単体での性能を比較するため，Tesla K40 を 1 台使い HPCG ベンチマークを実行した．このときの，三次元格子は 256x128x128 とした. 図 9 は, HPCG の出力する，疎行列ベクトル積（SpMV），マルチグリッド計算（MG），性能補正前の全体（Raw Total）の性能を，単純なカラーリング（Naïve coloring）と，提案手法である階層的カラーリング（Hierarchical coloring）を比較したものである．階層的カラーリングを使用することで，SpMV で 2 割，MG で 3 割程度，Raw Total で 2 割程度，性能が向上しているのが分かる．

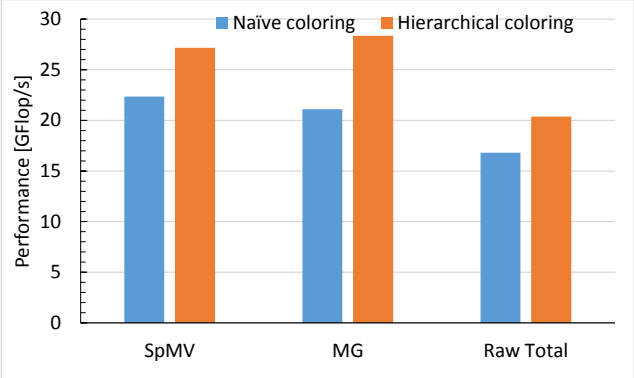


図 9　Tesla K40 単体による HPCG ベンチマークの SpMV，MG，Raw Total の性能比較

一方，図 10 は，HPCG のスコアとして出力される値を比較したものである．これらの値は Raw Total の値から，反復回数の増加率による減点と最適化のための準備にかかる時間による減点，さらには v3.0 からは問題生成にかかる時間による減点を引いたものが出力される．表 2 にそれぞれの実装について，最適化に要した時間とリファレンス実装で 50 回反復するのと同等な残差を得るための反復回数をまとめる．図 10 の結果を比較すると，階層的カラーリングによる性能向上は 1 割程度と低下してしまっている．これは，最適化のための準備に要する時間が増加してしまった分の減点が大きいためで，この部分の実装を見直し，準備時間の短縮をする必要がある．反復回数はどちらの実装でも同じとなった．なお，問題生成に要した時間は，どちらの実装でも共通の実装であり，おおよそ 17 秒程度かかった．この部分についても原点が大きいので，最適化が必要と思われる．

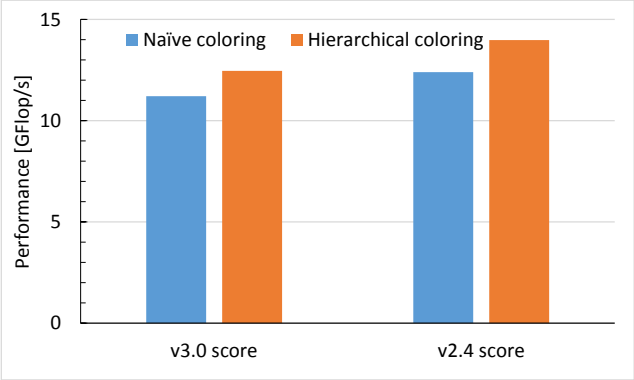


図 10　Tesla K40 単体による HPCG ベンチマークスコアの比較

表 2　Tesla K40 単体による HPCG の最適化のための時間と反復回数（リファレンス=50）の比較

	Naïve coloring	Hierarchical coloring
最適化[sec]	4.40	7.48
反復回数	63	63

5.3 POWER8 プロセッサーによる性能評価

次に，GPU を使用せずに POWER8 プロセッサーのみを計算に使用した場合の性能を測定した．使用するクラスターは表 1 の(2)である．CPU 用のマルチグリッド計算の実装方法は，GPU 向けの単純なカラーリング方法と同じであり，OpenMP を用いて各行をスレッド並列化した．このとき，疎行列は転置を行わない通常の ELL 形式で保存した．また，ノードあたり，2 つの POWER8 プロセッサーが搭載されるため，ノードあたり 2 プロセスで実行し，最大で 8 ノードを使用した．プロセスあたりの三次元格子は，単一 GPU 実行と同じく，256x128x128 とした．このときの，SpMV，MG，Raw Total の性能の測定結果を図 11 に示す．いずれの性能値も，良好なスケーラビリティが得られているのが分かる．（HPCG ベンチマークは weak scaling 実行．）図 12 は，HPCG ベンチマークスコアを表し，GPU 実装同様に，反復回数 63 回，最適化の準備時間平均 2.8 秒，問題生成時間おおよそ 17 秒の分が減じられている．最適化の準備にかかる時間は，GPU 実装に比べて，GPU 側のデータを用意しない分短くなっている．

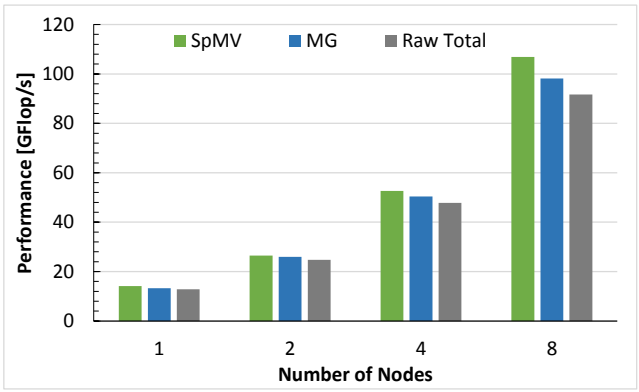


図 11　POWER8 による HPCG ベンチマークのマルチノード実行の性能測定

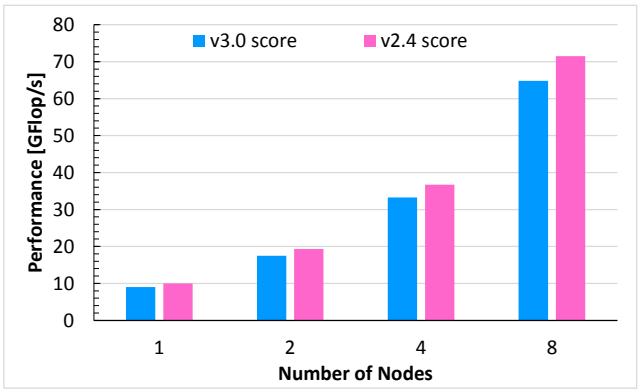


図 12　POWER8 によるマルチノード実行の HPCG ベンチマークスコア

5.4 Tesla K80 によるマルチ GPU，マルチノード性能評価

最後に，表 1 の(2)のクラスターを使用して，GPU を使って HPCG ベンチマークを実行した場合の性能測定を行った．ノードあたり 1 枚の Tesla K80 が搭載されており，Tesla K80 は 1 枚あたり 2 つの GPU があるように見えるため，ノードあたり 4 プロセスで実行した．プロセスあたりの三次元格子は同じく 256x128x128 とした．図 13 に SpMV の

測定結果，図 14 に MG の測定結果，図 15 に Raw Total の測定結果をそれぞれまとめる．

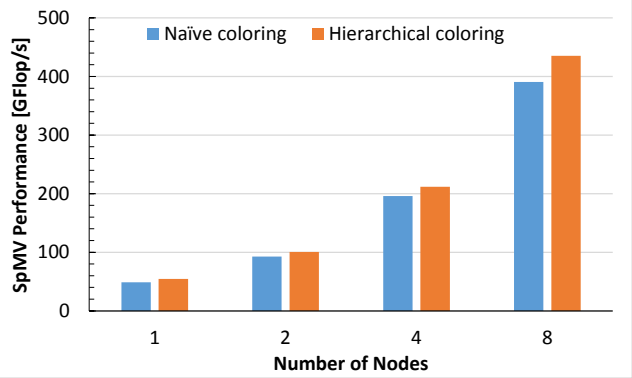


図 13　Tesla K80 による SpMV の性能比較

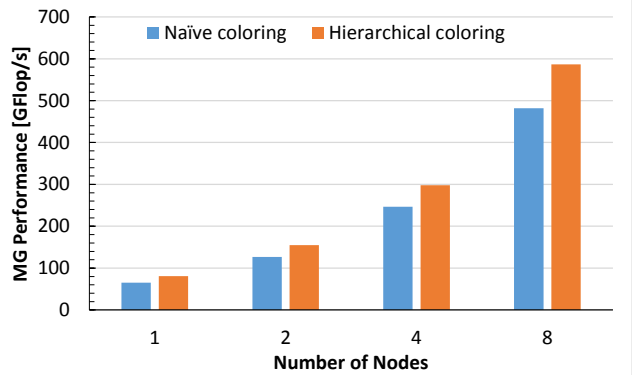


図 14　Tesla K80 による MG の性能比較

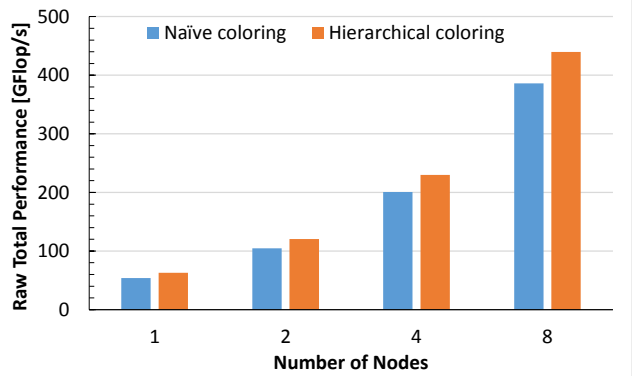


図 15　Tesla K80 による Raw Total の性能比較

こちらの結果も，良好なスケーラビリティが得られている．SpMV で 1 割程度，MG で 2 割程度，Raw Total で 1.5 割程度，階層的カラーリングで性能が向上できた．また，図 16 に，HPCG ベンチマークスコアの比較を示す．階層的カラーリングによって約 1 割ほどスコアを改善できた．表 3 には，このときの最適化に要した時間の平均値と反復回数を示す．最適化に要した時間は，使用したノード数によらず，ほぼ一定となった．また，問題生成に要した時間はどの場合でもおおよそ 22 秒であった．POWER8 のみを使用した場合はノードあたり 2 プロセスであったのに対して，ノードあたり 4 プロセスとしたためにノードあたりの問題サイズが増大したことが問題生成時間の増加につながったと考えられる．

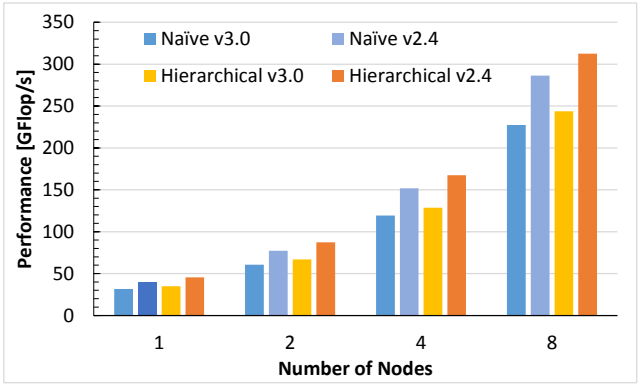


図 16　Tesla K80 による HPCG ベンチマークスコアの比較

表 3　Tesla K80 による HPCG の最適化のための平均時間と反復回数（リファレンス=50）の比較

	Naïve coloring	Hierarchical coloring
最適化[sec]	4.11	6.13
反復回数	63	63

図 17 および図 18 は，POWER8 を使用した場合と Tesla K80 を使用した場合の，SpMV および MG の性能を比較したものである．Tesla K80 の性能値は階層的カラーリングを使用したときの値である．SpMV では 4 倍程度，MG では 6 倍程度の性能差があるが，逆に POWER プロセッサーの比較的高いメモリバンド幅のおかげでこの程度の性能差になっているとも言える．POWER プロセッサーの比較的高い性能値を無駄にしないためにも，POWER プロセッサーと GPU の両方を計算に使用する実装が望まれる．

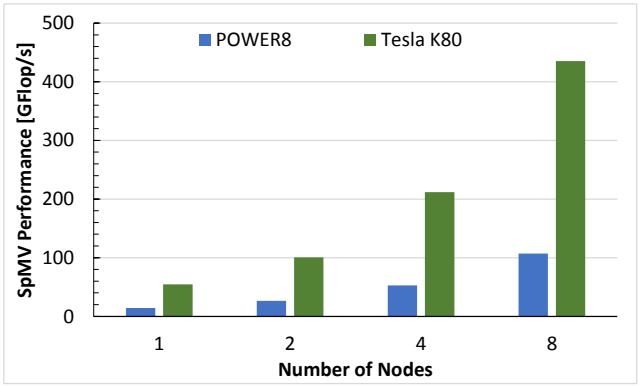


図 17　POWER8 と Tesla K80 の SpMV の性能比較

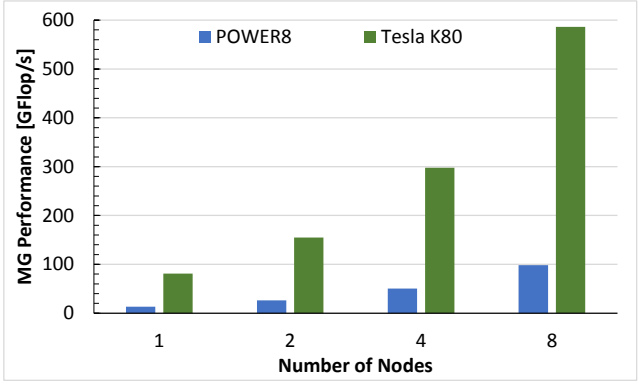


図 18　POWER8 と Tesla K80 の MG の性能比較

6. おわりに

HPCG ベンチマークの, ガウス･ザイデル法によるマルチグリッド計算を，OpenPOWER クラスター上で，GPU を用いて並列化を行う手法について，単純なカラーリングによる実装と，本報告で提案する階層的カラーリングによる実装との性能を比較した．階層的カラーリングでは，ベクトル要素を，GPU の共有メモリを使用することで効率良く読み書きし，またコアレスアクセスによって GPU のグローバルメモリへのアクセスも効率化した．その結果，マルチグリッド計算の性能をおおよそ 2〜3 割程度引き上げることができた．

階層的カラーリングによって，反復回数の増加は，単純なカラーリングと比較して差は無いが，最適化のための準備時間がやや増大することが分かった．HPCG ベンチマークスコアを評価する場合に，不利となるためこの部分の改良が求められる．また，HPCG v3.0 からは問題生成にかかる時間にもペナルティがかかるため，この部分の最適化も必要となる．

単純なカラーリングによる実装として，NVIDIA による HPCG 実装[7]があるが，Tesla K40 の単体性能で比較すると，本報告の実装による MG の性能はやや劣っている．階層的カラーリングで逆転はしているものの，原因を調査し単純なカラーリング実装についても性能向上の可能性を探る必要があると考えている．

また, 今後は GPU のみによる実行だけではなく, POWER プロセッサーの性能を上乗せできるようなハイブリッド実装について検討していきたい．このとき，互いの担当する計算部分の更新された領域を交換し合う必要が生じるため, GPU と CPU 間でデータ転送が生じることになる．OpenPOWER により高速なインターコネクトである NVLink が搭載されると，効率的にハイブリッド実装ができるようになると考えられる．OpenPOWER クラスターにおいて最適なハイブリッド実装を行いもっと大規模なクラスターで測定したい．

参考文献

[1] OpenPOWER Foundation, http://openpowerfoundation.org/
[2] IBM Power System S822LC, http://www-06.ibm.com/systems/jp/power/hardware/s822lc-high-performance/
[3] NVIDIA NVLink High-Speed Interconnect, http://www.nvidia.com/object/nvlink.html
[4] 土井淳，“次世代 Power システムにおける NVLink と GPU を利用したアプリケーションの性能予測”，第 152 回ハイパフォーマンスコンピューティング研究会, 2015.
[5] HPCG Benchmark, http://www.hpcg-benchmark.org/
[6] Jack Dongarra, Michael A. Heroux, Piotr Luszczek "HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems," Technical Report, Electrical Engineering and Computer Science Department, Knoxville, Tennessee, UT-EECS-15-736, November, 2015.
[7] Everett Phillips, Massimiliano Fatica, A CUDA implementation of the High. Performance Conjugate Gradient benchmark, High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation, Volume 8966 of the series Lecture Notes in Computer Science, pp 68-84.