Vol.2015-HPC-151 No.8
2015/9/30

疎行列ベクトル積計算を対象とした GPU向けメモリアクセス削減手法

長坂 侑亮[1,a)]　額田 彰[1]　松岡 聡[1]

概要：科学技術計算において巨大で疎な問題行列を持つ連立一次方程式を解く際，疎行列ベクトル積計算が実行時間の大部分を占めている．疎行列ベクトル積計算の GPU 向けの高速化も数多く行われてきているものの，疎行列ベクトル積計算がメモリバウンドなカーネルであることや入力ベクトルへのランダムアクセスによって発生する局所性低下等が要因となって性能向上が妨げられている．我々は GPU での疎行列ベクトル積計算時のメモリアクセス量とアクセス頻度を効果的に削減する疎行列フォーマットである AMB（Adaptive Multi-level Blocking）フォーマットを提案する．16bit integer の利用と種々のブロッキング手法によって，列インデックスの圧縮を行い，メモリアクセス量の削減を図っている．Florida 大学の疎行列データセットから選出した 40 個の行列に対して，既存手法との比較を行い，cuSparse と比較して最大で 2.81 倍，平均で 1.77 倍の性能向上を果たし，また，近年提案された高速な疎行列ベクトル積ライブラリである yaSpMV と比較して最大で 1.38 倍，平均で 1.13 倍の性能向上を果たした．

1. はじめに

シミュレーションなどの数値計算分野やグラフ処理の分野において疎行列計算が行われている．疎行列を扱う際，メモリ使用量と計算量の削減という観点から，処理に必要となる非ゼロ要素に関する情報のみを保持するように圧縮を行っている．しかし，疎行列に関する処理を含む CG（Conjugate Gradient）法やページランクアルゴリズムなどで実行時間の大部分を占める疎行列ベクトル積計算では，圧縮によって入力ベクトルへのランダムアクセスが発生するため，行列ベクトル積計算の持つメモリバウンドであるという問題に加え，局所性が低下するという問題も生じている．疎行列ベクトル積計算の高速化を目的として，これまで多くの圧縮手法が提案されてきており，昨今では科学技術計算において広く用いられている GPU などのメニーコアプロセッサへの適用も行われている．GPU は CPU と比較して高い演算性能とメモリバンド幅を有しており，それを活かした疎行列ベクトル積計算の高速化が行われてきた．しかし，同時に多くのスレッドが動作することによってロードインバランシングが生じることや CPU と比較して小さいキャッシュであるがために入力ベクトルへのランダムアクセスに関してキャッシュミスが増大するという問題がある．また，メモリバウンドであるという疎行列ベクトル積計算の持つ問題は依然として残っており，GPU の持つ高い演算性能を阻害する要因となっている．

本研究では，GPU での疎行列ベクトル積計算高速化のために，メモリアクセス量とアクセス頻度を削減することを目的とした AMB（Adaptive Multi-level Blocking）フォーマットとそれを用いた疎行列ベクトル積計算手法を提案する．AMB フォーマットでは複数段階のブロッキングを行う．行列に対してキャッシュサイズを考慮した列方向分割を行うことで，入力ベクトルのキャッシュでの再利用性を高め，メモリへのアクセス頻度を抑えており，更に各非ゼロ要素を表す際に用いる列インデックスの圧縮によってメモリアクセス量の削減を可能にした．Florida 大学の Sparse Matrix Collection [1] から選出した 40 個の行列データに対して，cuSparse と比較して最大で 2.81 倍，平均で 1.77 倍の性能向上を果たし，また，近年提案された GPU 向けの高速な疎行列ベクトル積計算ライブラリである yaSpMV と比較して最大で 1.38 倍，平均で 1.13 倍の性能向上を果たした．

2. 疎行列フォーマット

数値計算等において生成される問題行列のサイズが巨大になっていくものの，その要素の多くが処理に必要でないゼロ要素である．このような行列に対して圧縮を行うことによって計算量とメモリ使用量の削減を行うというのが疎行列フォーマットの基本的な考えである．疎行列フォー

[1] 東京工業大学
Tokyo Institute of Technology
[a)] nagasaka.y.aa@m.titech.ac.jp

マットとして広く用いられているものとして COO と CSR がある．COO（Coordinated）フォーマットは行列の各非ゼロ要素に関して値，行インデックス，列インデックスを保持する．CSR（Compressed Sparse Row）フォーマットは各行毎に圧縮を行い，各行の開始インデックスを保持した上で，各非ゼロ要素に関して値と列インデックスを保持する．その結果，CSR フォーマットは COO フォーマットと比較してメモリ使用量の削減を図れている．また，疎行列ベクトル積計算において特定の行列に対してより高い性能を出すために，多くの疎行列ベクトル積計算手法が提案されてきた [2–5]．これに加えて，GPU などのメニーコアプロセッサ向けの最適化も数多く行われている [6, 7]．広く用いられている CSR 向けの最適化やその拡張 [8–10] も提案されているが，メニーコアプロセッサでの疎行列ベクトル積計算に適したメモリ配置である column-major のフォーマットが数多く提案されてきた．ベクトル計算機に対して最適化された JDS フォーマットへの GPU 向け最適化 [11, 12] や，ELLPACK を拡張したフォーマットが多く提案されている．また，メニーコアプロセッサでの疎行列ベクトル積計算時に問題となる要因を解決するためのフォーマットも提案されている．メニーコアプロセッサにおいて多数のスレッドが同時に動作することによって生じるロードインバランシングの解決に焦点を当てたフォーマット [13, 14] や，CPU に比べてキャッシュ容量が小さいために入力ベクトル要素へのランダムアクセス性能が悪化するため，それの改善を図った手法 [15, 16] が提案されている．

2.1 ELLPACK とその拡張

ELLPACK フォーマットでは，行方向に関して圧縮を行った後，メモリ配置を column-major にするために各行の非ゼロ要素数を揃えるためのパディングを行う．対角行列のように各行ごとの非ゼロ要素数の差が小さい場合には有用であるが，差が大きい時にはパディングの量が増大するため性能が大きく低下する．これを解決するために ELLPACK を拡張した疎行列フォーマットが提案されている．

Bell らによって，ELLPACK と COO を結合した Hybrid フォーマットが提案された [8]．基本的には ELLPACK で構築し，非ゼロ要素数の偏った部分を COO で構築することによって，パディングの量を大きく減らすことを可能にしている．Hybrid フォーマットは GPU 向けに最適化された CSR と同様に NVIDIA が提供する cuSparse ライブラリ [17] に実装されている．Hybrid フォーマットで最も重要である ELLPACK と COO を切り替えるためのパラメータの設定も含めて提供されている．

Kreutzer らによって SELL-C-σ フォーマットが提案された [18]．図 1 に例を示す．SELL-C-σ フォーマットでは

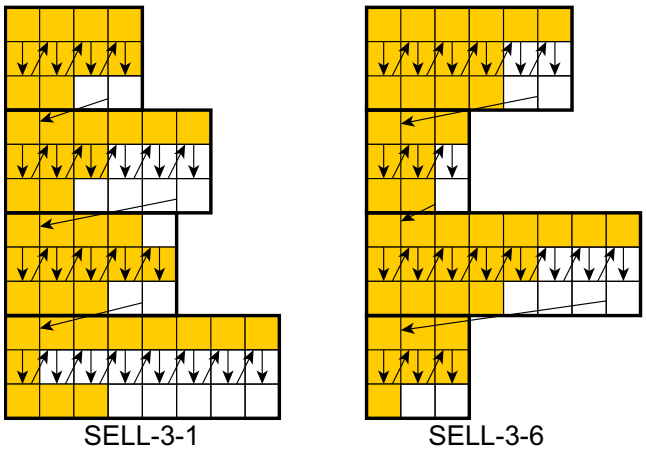

図 1　SELL-C-σ フォーマット

σ 行ごとに行あたりの非ゼロ要素数に基づく行のリオーダリングを行い，C 行ごとに ELLPACK への変換を行う．C 行ごとに変換を行うことによって，非ゼロ要素数の多い行による影響を小さくすることが出来，更にリオーダリングを行うことによってより効果的にパディングの量を減らすことを可能にしている．行方向の分割を行うことによって生成される各チャンクの行サイズを表すパラメータ C はアーキテクチャの特徴に依存しており，NVIDIA の GPU を用いる場合には WARP サイズ（=32）に合わせ，ソーティングの幅を表す σ は元の行列の特徴を失わない程度にする必要があると述べられている．

行列に対して列方向の分割を行うことによって入力ベクトルへのアクセスの局所性向上を図った Segmented フォーマットが提案された [15]．分割幅を一つ設定し，その値に基づいて均等な分割を行い，分割された各部分行列を JDS もしくは SELL-C-σ フォーマットに変換する．列方向分割によってキャッシュヒット率の向上は達成したが，分割数に比例して発生する追加のメモリアクセス量の増大という問題があった．これを緩和するために行列のリオーダリングを行った上で多段階の分割幅を設定して局所性を保ったまま分割数を減らすことを可能にした Non-Uniformly Segmented フォーマットも併せて提案されている．

3. 関連研究

疎行列ベクトル積計算における，GPU 向けのメモリアクセス量削減を目的とした既存の研究について記す．

各非ゼロ要素の列インデックスを表すのに，32bit integer ではなく 16bit，もしくは 8bit で表せるようにする CoAd-ELL フォーマットが Maggioni らによって提案された [19]．行内の列インデックスの差分を取り，それらを 16bit もしくは 8bit で表すことでメモリアクセス量の削減を図っている．WARP サイズレベルで圧縮を行うかを判定しているが，差分に関してビット圧縮を行っているため，差分が大きい場合には圧縮が行えないため，圧縮が十分に行えてい

るとはいえない．

行列の各行の列インデックスの差分を取り，ビットレベルで圧縮を行う BRO フォーマットが Tang らによって提案された [20]．CPU 向けに行われていたビットレベルでの圧縮手法では，GPU が苦手とする逐次処理や分岐処理が多く含まれていたが，BRO フォーマットではこれを解消し GPU 向けに最適化を図ったものである．結果として，列インデックスに用いるメモリアクセス量を大きく減らすことに成功しているが，デコード等の追加の処理が多く発生している．

Yan らによって，ブロッキングによってメモリアクセス量の削減とロードバランスの改善を図った BCCOO フォーマットが提案された [21]．BCCOO フォーマットでは，行列を細かい単位でブロック化し，それらを改良した COO フォーマットで保持したものである．従来の COO フォーマットと異なり，行インデックスの差分を圧縮したものを保持することによって，メモリ使用量を抑えている．また，各ブロックの計算結果を足し合わせる際には効果的に shared memory などのローカルメモリを使用することでメモリへのアクセスを避けている．BCCOO フォーマットでは行列に合わせてブロックサイズや疎行列ベクトル積計算時の戦略など多くのパラメータ設定が必要となっており，BCCOO フォーマットが実装されている yaSpMV ライブラリは最適なパラメータの自動探索を行う機構も提供している．しかし，BCCOO フォーマットの持つパラメータ探索空間は巨大であるため，パラメータチューニングに要する時間が大きいという問題がある．

4. 提案

疎行列ベクトル積計算の高速化のために多くの疎行列フォーマットが提案されてきており，特にメモリアクセス量の削減を目的とした高速化は大きく成功していると言える．しかし，その削減は未だ十分であるとは言えない．また，行列データに関するメモリアクセスだけではなく，入力ベクトルへのランダムアクセスによる性能低下に関しても同時に考慮する必要がある．本研究では行列データに関するメモリアクセス量と入力ベクトルへのメモリアクセス頻度の双方を大きく削減することを可能とした AMB（Adaptive Multi-level Blocking）フォーマットを提案する．

4.1 AMB フォーマット

AMB フォーマットは 3 段階のブロッキングを行うことによって，疎行列ベクトル積計算時のメモリアクセス量とアクセス頻度の削減を図っている．図 2 に AMB フォーマットの各段階のブロッキングの流れを記す．

第一段階のブロッキングは行列の列方向分割である．既存手法である Segmented フォーマットと同様に分割幅を一つ設定して，その値に基いた列方向分割を行う．各分割

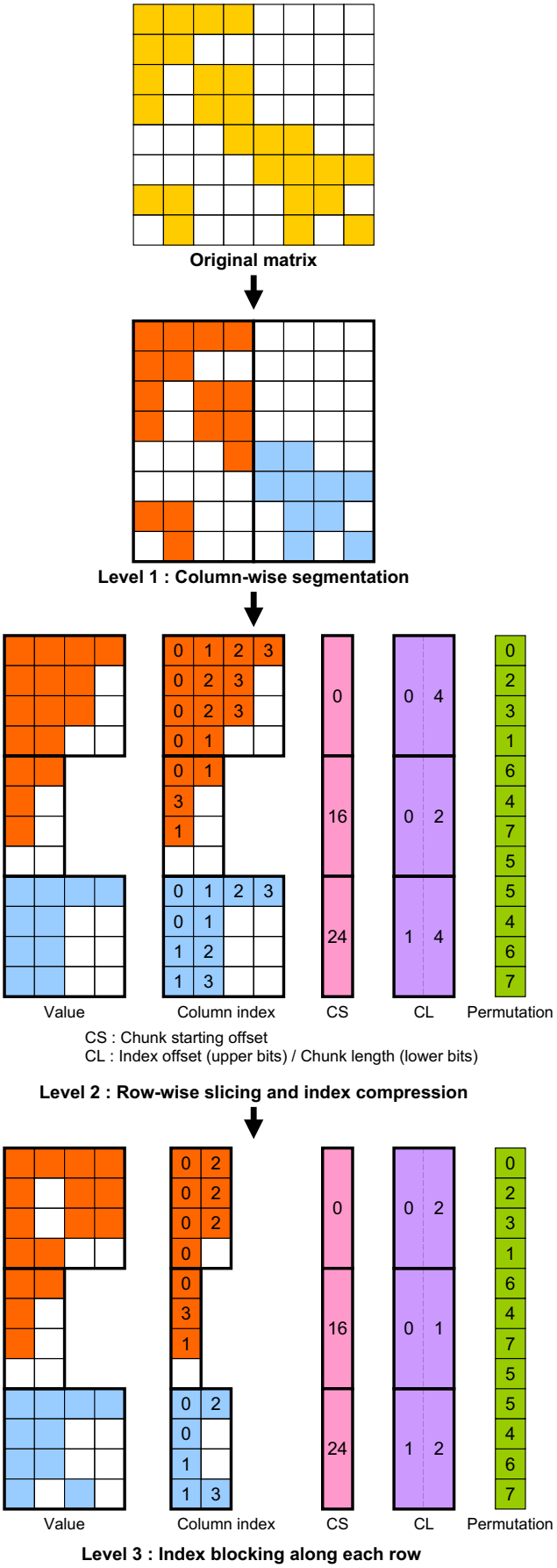

図 2 AMB フォーマット

毎に疎行列ベクトル積計算を行うことによって，入力ベクトルへのアクセスの局所性が向上し，メモリアクセスの頻度を抑えることが可能となる．なお，第二段階で行う列インデックスの圧縮に備え，AMB フォーマットでは分割幅は 65536 要素以下に制限する．

次に，第二段階のブロッキングは行方向の分割である．第一段階のブロッキングによって生成された部分行列を SELL-C-σ に変換する．行あたりの非ゼロ要素数に応じた行のソーティングと行方向の分割を行った上で各チャンクを ELLPACK に変換することによって，パディング等の不必要なメモリ使用を削減している．SELL-C-σ のパラメータはチャンクサイズは WARP サイズに合わせて C=32 で設定し，ソーティングを行う行数は，パディングを抑えられかつ，出力ベクトルへのアクセスの局所性を維持できるサイズとして σ=32k とした．また，元の SELL-C-σ から二つの改良を施している．一つ目は，行方向分割によって発生する非ゼロ要素を含まないチャンクが発生するが，このチャンクに関する情報を持つ配列要素を削除することである．二つ目は各列インデックスに対して 32bit integer から 16bit の unsigned short への圧縮を行ったことである．列インデックスとしてそれぞれを分割幅で割った余りを保存し，列インデックスのオフセットとして各チャンクの先頭の非ゼロ要素の列インデックスを分割幅で割った値を各チャンクの幅を保持している配列（図 2 の CL 配列）の上位 16bit に格納する．なお，各チャンクの幅は 1 以上 64k 以下であるため，CL 配列の下位 16bit で表すことが可能である．この列インデックスの圧縮によって，メモリアクセス量の大幅な削減が可能となっている．

第三段階が行方向に沿った列インデックスのブロッキングである．第二段階で行単位での並び替えを行っているので，各行内でのブロッキングとした．ブロッキングを行うことによって，連続する列インデックスに関して圧縮を行うことができる．ブロックサイズを大きくすることによって列インデックスの圧縮率を高めることができるが，行列の値に関して余分なゼロフィリングが増加する場合もあるため，各行列に適したブロックサイズの選択が重要である．

4.2 疎行列ベクトル積計算カーネル

AMB フォーマットでの疎行列ベクトル積計算は基本的に Segmented フォーマットのものに準じており，第一，第二段階で生じたチャンクを順次処理していく形となっている．疎行列ベクトル積計算は二つの CUDA カーネルから成り立っている．一つ目のカーネルでは出力ベクトルの初期化を行うカーネルを行う．二つ目のカーネルでは，各行に 1 スレッドを割り当て，順次各行の計算を行う．各行の計算結果は atomic を用いて出力ベクトルのそれぞれ所定の行に足しあわせていくことで，最終的な答えを得る．また，第三段階のブロッキングでブロックサイズを 2 以上に設定した場合には，ブロック内の処理をアンローリングすることで反復回数を削減している．

表 1 実験に用いた疎行列

Matrix	Row size	Non-zeroes (nz)	nz / row
af_{0,1,2,3,4,5}_k101	503,625	17,550,675	34.85
af_shell{3,4,7,8}	504,855	17,588,875	34.84
apache2	715,176	4,817,870	6.74
audikw_1	943,695	77,651,847	82.28
BenElechi1	245,874	13,150,496	53.48
bmw7st_1	141,347	7,339,667	51.93
bmwcra_1	148,770	10,644,002	71.55
bone010	986,703	71,666,325	72.63
boneS10	914,898	55,468,422	60.63
Dubcova3	146,689	3,636,649	24.79
ecology2	999,999	4,995,991	5.00
Emilia_923	923,136	41,005,206	44.42
Fault_639	638,802	28,614,564	44.79
Flan_1565	1,564,794	117,406,044	75.03
G2_circuit	150,102	726,674	4.84
G3_circuit	1,585,478	7,660,826	4.83
Geo_1438	1,437,960	63,156,690	43.92
hood	220,542	10,768,436	48.83
Hook_1498	1,498,023	60,917,445	40.67
inline_1	503,712	36,816,342	73.09
ldoor	952,203	46,522,475	48.86
msdoor	415,863	20,240,935	48.67
offshore	259,789	4,242,673	16.33
parabolic_fem	525,825	3,674,625	6.99
pwtk	217,918	11,634,424	53.39
Serena	1,391,349	64,531,701	46.38
shipsec1	140,874	7,813,404	55.46
shipsec5	179,860	10,113,096	56.23
StocF-1465	1,465,137	21,005,389	14.34
thermal2	1,228,045	8,580,313	6.99
thermomech_dM	204,316	1,423,116	6.97
tmt_sym	726,713	5,080,961	6.99

5. 性能評価

AMB フォーマットと既存の疎行列フォーマットとの疎行列ベクトル積計算性能の比較を行った．比較は Flops 値性能を元に行い，すべての測定は単精度で行った．実験に用いた行列は University of Florida の Sparse Matrix Collection [1] から選出した．表 1 に用いた行列データを記す．行サイズが 131072 以上の正定値対称行列を疎行列コレクションからすべて選択した結果，総計 40 個となった．なお，af_*_k101 と af_shell*の 2 種類はそれぞれ複数のパターンを持つが，行列のサイズや非ゼロ配置が同一で値のみが異なる．

実験には NVIDIA の Tesla K20X が搭載された TSUBAME-KFC を用いた．GPU は最大で 250GB/sec の

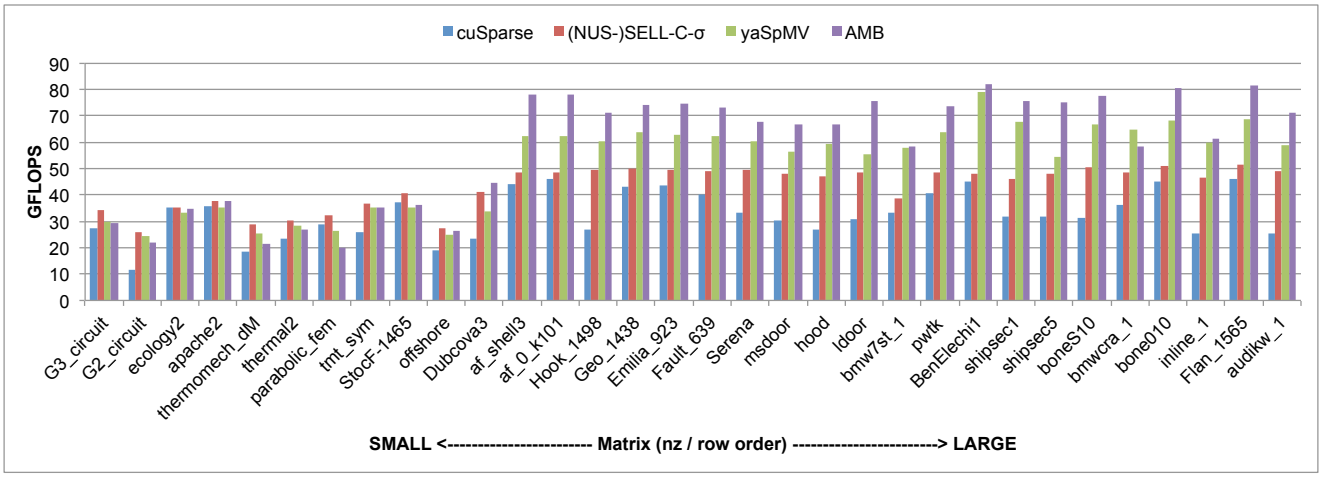

図 3　疎行列ベクトル積計算性能

帯域幅である容量 6GByte のメモリを持つ．これに加えて，Kepler 世代の GPU では 14 の SMX で 1.5MByte の L2 キャッシュを共有し，各 SMX に 48KByte の read-only キャッシュを有する．実際には read-only キャッシュは 4 つに別れており，それぞれのサイズは 12KB である．Read-only キャッシュにはランダムアクセスが発生する入力ベクトル要素に加えて，SELL-C-σ や AMB フォーマットでは各チャンクのオフセットとサイズを表す配列（CL，CS 配列）要素にも適用している．コードは CUDA7.0 で実装されており，CentOS Linux 6.4 OS 64bit のもと実行された．

5.1　疎行列ベクトル積計算性能

疎行列ベクトル積計算の性能比較として，cuSparse と yaSpMV という二種類の疎行列計算ライブラリに加え，既存手法である SELL-C-σ フォーマットと分割フォーマットである NUS-SELL-C-σ フォーマットの評価も併せて行った．NVIDIA が提供する cuSparse ライブラリから CSR，HYBRID，BSR（Blocked CSR）を用い，3 種類のフォーマットで最適なものを cuSparse の性能とした．なお，BSR フォーマットでの設定パラメータであるブロックサイズは 1〜4 とした．yaSpMV ライブラリでは各行列に対して最適なパラメータを探索した上で，最適なパラメータでの疎行列ベクトル積計算性能を結果とした．AMB フォーマットでの設定パラメータについては，第一段階の列方向分割時に用いる分割幅は 64k で固定し，第三段階でのブロックサイズは 1〜8 とし，実行時のスレッドブロックサイズは 32，64，128，256，512 とした．これらの設定パラメータのもと最適なものを AMB の性能とした．

図 3 に疎行列ベクトル積計算の性能を示す．結果の各行列は行あたりの平均非ゼロ要素数（nz / row）を昇順に並べたものである．また，af_shell*と af_*_k101 に関しては各パターンでの性能が同一とみなせたことから，代表としてそれぞれ一つの行列のみを載せた．（nz / row）が小さい場合，出力ベクトルへの書き込みに用いる atomicAdd などによるコストがメモリアクセス量削減による効果を上回るため，性能向上が果たせなかった．一方，（nz / row）が大きい場合にはメモリアクセス量の削減効果が大きくなるため，既存手法から飛躍的に性能を向上させることが可能となった．我々の提案手法である AMB フォーマットは cuSparse ライブラリから最大で 2.81 倍，平均で 1.77 倍の性能向上を，yaSpMV ライブラリからは最大で 1.38 倍，平均で 1.13 倍の性能向上を果たしており，これらのライブラリを含めた今回比較として用いた既存手法全体からは最大で 1.38 倍，平均で 1.11 倍の性能向上を果たすことに成功した．

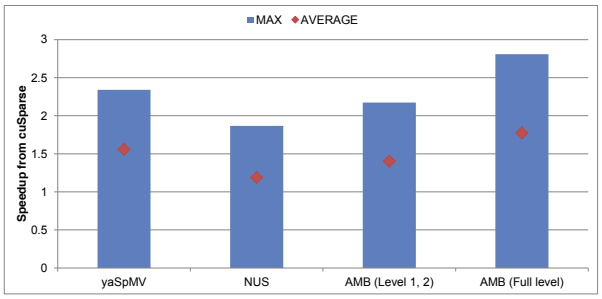

図 4　フォーマット毎の性能向上率

次に，図 4 に cuSparse からの各フォーマットの最大と平均の性能向上率を示す．Level 2 のブロッキングで行われる列インデックスの 16bit 圧縮と Level 3 のブロッキングが性能向上に大きく貢献していることがわかる．また，インデックスに関するメモリの使用を削減した yaSpMV も高い性能向上率を示していることから，メモリアクセス量削減が疎行列ベクトル積計算性能に与える影響は大きいと言える．

5.2 メモリ使用量

表 2 に各行列でのメモリ使用量を示す．行列を表すのに用いている配列のサイズの和を表している．これらの配列は一回の疎行列ベクトル積計算で一度しかアクセスされないため，実質的に行列に関するメモリアクセス量とみなせる．比較として CSR，yaSpMV，AMB フォーマットを記す．Reduction rate は CSR と yaSpMV からの AMB フォーマットのメモリ使用量削減率を示しており，

$$Reduction\ rate = \frac{min\{CSR, yaSpMV\} - AMB}{min\{CSR, yaSpMV\}} * 100$$

で与えられ，値が大きいほど AMB フォーマットにおいてメモリ使用量がより大きく削減されていたことを表す．一つの行列を除いて，他の全ての行列について行列を表すのに用いるメモリの使用量が削減されており，特に疎行列ベクトル積計算性能で最大の性能向上率を示した行列'shipsec5'はメモリ使用量の削減率に関しても最大であり，メモリ使用量の削減が疎行列ベクトル積計算性能に与える効果は大きいと言える．

6. 結論

多くの科学技術計算において性能に支配的なカーネルとなっている疎行列ベクトル積計算の性能向上を図ることは極めて重要な課題である．スーパーコンピュータで広く用いられている GPU などのメニーコアプロセッサでの疎行列ベクトル積計算の高速化が盛んに行われているが，入力ベクトル要素へのランダムアクセスによって発生するキャッシュミスの増加と疎行列ベクトル積計算の持つメモリバウンドであるという問題が GPU の持つ高い演算性能を阻害している．これに対し，我々は行列に対して複数段階のブロッキングを施す AMB フォーマットを提案した．入力ベクトルへのアクセスの局所性を向上させることで，メモリアクセス頻度を抑え，列インデックスの圧縮を行うことでメモリアクセス量の削減を可能とした．結果として，我々の提案するフォーマットは疎行列ベクトル積計算において，NVIDIA が提供しているライブラリや既存の GPU 向けの高速なフォーマットやライブラリから大幅な性能向上を果たすことに成功した．

今後の課題として，我々の提案するメモリアクセス削減手法が他のメニーコアプロセッサ，例えば AMD Radeon GPU などにおいても有用であるかを確認する必要があると考えられる．また，AMB フォーマットが持つ複数のパラメータに関して，自動最適化を行う機構を構築する必要があると考えられる．

謝辞 本研究の一部は科学研究費補助金基盤研究 (S)23220003 「１０億並列・エクサスケールスーパーコンピュータの耐故障性基盤」及び科学技術振興機構戦略的創造研究推進事業「EBD: 次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」による。

表 2 フォーマット毎のメモリ使用量

	CSR [MB]	yaSpMV [MB]	AMB [MB]	Reduction rate [%]
af_*_k101	135.82	102.39	75.80	25.97
af_shell*	136.12	102.82	76.02	26.06
apache2	39.49	48.85	30.68	22.31
audikw_1	596.04	422.48	357.89	15.29
BenElechi1	101.27	59.81	55.40	7.37
bmw7st_1	56.54	42.75	34.09	20.27
bmwcra_1	81.77	55.06	48.41	12.07
bone010	550.53	400.82	324.33	19.08
boneS10	426.68	280.12	251.57	10.19
Dubcova3	28.31	29.06	22.44	20.74
ecology2	41.93	50.65	32.78	21.83
Emilia_923	316.37	226.59	187.84	17.10
Fault_639	220.75	155.62	131.21	15.69
Flan_1565	901.71	599.88	531.43	11.41
G2_circuit	6.12	7.37	4.88	20.14
G3_circuit	64.50	63.05	53.96	14.42
Geo_1438	487.33	346.34	290.57	16.10
hood	83.00	61.82	52.26	15.46
Hook_1498	470.48	333.98	280.66	15.97
inline_1	282.81	199.52	170.04	14.78
ldoor	358.57	273.51	200.68	26.63
msdoor	156.01	123.35	101.25	17.92
offshore	33.36	33.90	25.88	22.42
parabolic_fem	30.04	30.25	30.56	-1.72
pwtk	89.59	63.82	49.58	22.32
Serena	497.65	354.18	302.07	14.71
shipsec1	60.15	35.05	32.97	5.94
shipsec5	77.84	65.83	43.29	34.24
StocF-1465	165.85	167.83	131.21	20.88
thermal2	70.15	70.62	56.27	19.79
thermomech_dM	11.64	14.01	10.06	13.56
tmt_sym	41.54	50.63	32.22	22.42

参考文献

[1] Davis, T.: The University of Florida Sparse Matrix Collection.
[2] Nishtala, R., Vuduc, R. W., Demmel, J. W. and Yelick, K. A.: When cache blocking of sparse matrix vector multiply works and why, *Applicable Algebra in Engineering, Communication and Computing*, Vol. 18, No. 3, pp. 297–311 (2007).
[3] Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K. and Demmel, J.: Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms, *Parallel Computing*, Vol. 35, No. 3, pp. 178–194 (2009).
[4] Lee, B. C., Vuduc, R. W., Demmel, J. W. and Yelick, K. A.: Performance Models for Evaluation and Automatic Tuning of Symmetric Sparse Matrix-Vector Multiply, *ICPP 2004: International Conference on Parallel Processing 2004*, IEEE, pp. 169–176 (2004).
[5] Karakasis, V., Goumas, G. and Koziris, N.: Perfomance Models for Blocked Sparse Matrix-Vector Multiplication kernels, *ICPP'09: International Conference on Parallel Processing 2009.*, IEEE, pp. 356–364 (2009).
[6] Monakov, A. and Avetisyan, A.: Implementing Blocked

Sparse Matrix-Vector Multiplication on NVIDIA GPUs, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Springer, pp. 289–297 (2009).

[7] Baskaran, M. M. and Bordawekar, R.: Optimizing Sparse Matrix-Vector Multiplication on GPUs using Compile-time and Run-time Strategies, Technical report, IBM (2008).

[8] Bell, N. and Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM, p. 18 (2009).

[9] Greathouse, J. and Daga, M.: Efficient Sparse Matrix-Vector Multiplication on GPUs Using the CSR Storage Format, *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pp. 769–780 (online), DOI: 10.1109/SC.2014.68 (2014).

[10] Ashari, A., Sedaghati, N., Eisenlohr, J., Parthasarath, S. and Sadayappan, P.: Fast Sparse Matrix-Vector Multiplication on GPUs for Graph Applications, *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pp. 781–792 (online), DOI: 10.1109/SC.2014.69 (2014).

[11] Cevahir, A., Nukada, A. and Matsuoka, S.: Fast conjugate gradients with multiple GPUs, *Computational Science–ICCS 2009*, Springer, pp. 893–903 (2009).

[12] Cevahir, A., Nukada, A. and Matsuoka, S.: High performance conjugate gradient solver on multi-GPU clusters using hypergraph partitioning, *Computer Science-Research and Development*, Vol. 25, No. 1-2, pp. 83–91 (2010).

[13] Liu, X., Smelyanskiy, M., Chow, E. and Dubey, P.: Efficient sparse matrix-vector multiplication on x86-based many-core processors, *Proceedings of the 27th international ACM conference on International conference on supercomputing*, ACM, pp. 273–282 (2013).

[14] Ashari, A., Sedaghati, N., Eisenlohr, J. and Sadayappan, P.: An Efficient Two-dimensional Blocking Strategy for Sparse Matrix-vector Multiplication on GPUs, *Proceedings of the 28th ACM International Conference on Supercomputing*, ICS '14, New York, NY, USA, ACM, pp. 273–282 (online), DOI: 10.1145/2597652.2597678 (2014).

[15] Nagasaka, Y., Nukada, A. and Matsuoka, S.: Cache-aware Sparse Matrix Formats for Kepler GPU, *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, pp. 281–288 (online), DOI: 10.1109/PADSW.2014.7097819 (2014).

[16] Anh, P. N. Q., Fan, R. and Wen, Y.: Reducing Vector I/O for Faster GPU Sparse Matrix-Vector Multiplication, *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pp. 1043–1052 (online), DOI: 10.1109/IPDPS.2015.100 (2015).

[17] NVIDIA Corporation: cuSPARSE Library.

[18] Kreutzer, M., Hager, G., Wellein, G., Fehske, H. and Bishop, A. R.: A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units, *SIAM Journal on Scientific Computing*, Vol. 36, No. 5, pp. C401–C423 (online), DOI: 10.1137/130930352 (2014).

[19] Maggioni, M. and Berger-Wolf, T.: CoAdELL: Adaptivity and Compression for Improving Sparse Matrix-Vector Multiplication on GPUs, *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, IEEE, pp. 933–940 (2014).

[20] Tang, W. T., Tan, W. J., Ray, R., Wong, Y. W., Chen, W., Kuo, S.-h., Goh, R. S. M., Turner, S. J. and Wong, W.-F.: Accelerating sparse matrix-vector multiplication on GPUs using bit-representation-optimized schemes, *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, p. 26 (2013).

[21] Yan, S., Li, C., Zhang, Y. and Zhou, H.: yaSpMV: Yet Another SpMV Framework on GPUs, *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14 (2014).