Vol.2016-HPC-155 No.42
2016/8/10

GPGPU向け省メモリ指向行列格納方式の提案とGMRESへの適用

河村 知記[1,a)]　米田 一徳[3]　山崎 崇史[3]　岩村 尚[3]　渡邉 正宏[3]　井口 寧[2]

概要：近年，工学や医療等の分野で使用されるミュレーション中で必要となる連立一次方程式の求解の高速化が望まれている．また，シミュレーションの需要は病院や民間企業においても増加しており，高速なだけでなく低コストな連立一次方程式の求解が求められている．近年のシミュレーションの大規模化により連立一次方程式の規模も増大しており，この大規模な連立一次方程式の求解のために GPU を用いた高速化が盛んに行なわれている．GPU は非常に多くの演算コアを備えており，高い演算性能を有する．また，演算単位あたりの演算コストが CPU に比べ安価であることから，低コストな連立一次方程式の求解が可能である．しかし，GPU のメモリは CPU に比べ少量であり，連立一次方程式で用いる係数行列をいかに効率良く格納するかが重要である．そこで本稿では既存格納方式である ELLPACK(ELL) の列番号を保存する行列に対して圧縮する方法を提案している．この方法を用いることで，行列 pwtk において最大の 25%のメモリ使用量の削減に成功している．

キーワード：GPGPU，GMRES，SpMV，Sparse linar system

Tomoki KAWAMURA[1,a)]　Kazunori Yoneda[3]　Takashi Yamazaki[3]　Takashi Iwamura[3]
Masahiro Watanabe[3]　Yasushi Inoguchi[2]

1. はじめに

近年, 工学，医療など様々な分野にて活用されているシミュレーションは, コンピュータの演算性能の向上に伴い, 大規模化の傾向にある. 多くのシミュレーションにおいて, 差分法や有限要素法といった手法が用いられ, これら手法では連立一次方程式の求解が必要とされる. シミュレーションの規模の増大に伴い, 求解を必要とする連立一次方程式を構成する式数も同様に大規模化している. 例えば心臓内の狭窄検査に用いる心臓シミュレーションでは 2000 万を超える式からなる連立一次方程式の解が必要となる. 連立一次方程式の求解に要する演算時間はシミュレーションの大きなボトルネックとなることが知られている．そのため，現在までに多くの連立一次方程式の高速な求解方法が提案されてきた．従来のシミュレーションでは, クラスタ, スーパーコンピュータ等な大規模な演算装置が多く用いられた. しかし近年, 病院，民間企業等でもシミュレーションの需要が増大しており, 低コストで高速な連立一次方程式の解法が望まれている.

一方，Gradhical Processing Units（GPU）は，近年，様々な分野のシミュレーションを高速化するために用いられている [1]．元々描画処理専用の演算装置である GPU を汎用的な処理に用いる技術を General Proposed GPU（GPGPU) を呼ぶ．GPU は非常に多くの演算コアをチップ上に有しており，並列度の高いアプリケーションに対して顕著な高速化率を得ることが可能である．また単位演算あたりのコストが CPU に比べ，安価であること，省電量であることが GPU の特徴としてあげられる．そのため，GPU を用いた GMRES の実装を行うことで，低コストな連立一次方程式の解法を検討する．

巨大な連立一次方程式を GPU を用いて求解する際の問題として GPU メモリの容量がある．GPU で処理を行う際には，CPU メモリにあるデータを GPU メモリへ転送

1 北陸先端科学技術大学院大学　情報科学研究科
Information Science, JAIST
2 北陸先端科学技術大学院大学 情報社会基盤研究センター
情報環境研究開発部門
Research Center for Advanced Computing Infrastructure (RCACI), JAIST
3 富士通株式会社
Fujitsu LTD.
a) t-kawamura@jaist.ac.jp

し，演算を行う必要がある．しかし，GPU メモリは CPU メモリに比べ少量であることから，巨大な連立一次方程式の求解で使用する行列が GPU メモリに乗り切らない可能性がある．その場合，複数の GPU を使用し，行列を分割してそれぞれの GPU 上で処理する必要がある．しかし，低コスト化という面から見ると，使用する GPU の数が増えることは好ましくなく，可能な限り少ない GPU で処理を行う必要がある．そのため，行列をいかに効率良く格納し，一つの GPU の担当する処理を増やすことが重要となる．また，行列の格納方式は GMRES 内の処理である疎行列-ベクトル積の演算性能に大きく影響してくる．

本稿では，GPU に対して適した行列格納方式と呼ばれる ELLPack（ELL）系統の行列格納方式に対して，適用可能な行列圧縮方式を提案する．ELL 系統の GPU 上での高い演算性能を保持しつつ，メモリ使用量を削減することが目的である．提案行列圧縮方式では，ELL の列番号を格納する行列に対する圧縮を施し，省メモリ化を実現している．提案圧縮方式の評価実験において，特定の行列において，既存の手法に比べ演算性能の低下が見られるが，多くの行列においてメモリ使用量の削減に成功し，最大で 25%の削減に成功している．

第 2 章では GMRES のアルゴリズムについて説明し，GMRES で用いられる Sparse Matrix-vector Multiplication (SpMV) の説明を行う．第 3 章では，GPU を用いた GMRES の実装，行列格納方式の関連研究について述べる．第 4 章にて，提案行列圧縮方式を説明し，第 5 章にて提案圧縮方式の性能評価を行う．第 6 章では，複数の行列格納方式を GMRES に適応し，その演算性能の評価を行う．第 7 章にてまとめを行う．

2. GMRES and SpMV

2.1 GMRES

本稿で，高速化の対象としている Generalized Minimal Residual method（GMRES）は非定常反復法の一種であり，頑強な解法として知られている．GMRES は反復回数に比例し，演算量とメモリ使用量が増加するが，反復ごとに残差が単純に減少することが特徴である [2]．また，反復ごとに演算量と使用メモリ量が増加する問題を解決するために，任意の反復回数で打ち切り，リスタートするリスタート付き GMRES が一般的に使用される．

Algorithm1 に右前処理付き GMRES(m) のアルゴリズムを示す．GMRES のアルゴリズム中の主な処理として，sparse matrix-vector Multiplication（SpMV）（$lines1, 5$），の scalar-vector products（$lines3, 11$），dot products（$line7$），Euclidean norms（$lines2, 10$），AXPY operations（$line8$）が挙げられる．

主な処理の中でも SpMV は GMRES をはじめとする多くの反復法の最も支配的な処理として知られている．

Algorithm 1 GMRES with Right preconditioner

1: $r_0 = b - \boxed{\text{A}} x_0$
2: $\beta = ||r_0||_2$
3: $v_1 = r_0/\beta$
4: **for** $j = 1$ to m **do**
5: Compute $w := \boxed{\text{A}} M^{-1} v_j$
6: **for** $i = 1$ to j **do**
7: $h_{i,j} := (w, v_i)$
8: $w := w - h_{i,j} v_i$
9: **end for**
10: $h_{j+1,j} = ||w||_2$
11: $v_{j+1} = w/h_{j+1,j}$
12: Define $V_m := [v_1, ..., v_m], \hat{H} = \{h_{i,j}\}_{1 \le i \le j+1, 1 \le j \le m}$
13: **end for**
14: Compute $y_m = argmin_y ||\beta e_1 - \hat{H}_m y||_2$, and $x_m = x_0 + M^{-1} V_m y_m$.
15: If satisfied Stop, else set $x_0 := x_m$ and GoTo 1.

Algorithm1 中の波線で示す部分が SpMV の処理である．GMRES 中で解くこととなる連立一次方程式の係数行列はこの SpMV にて使用され，SpMV の演算性能が GMRES の演算性能に大きく影響する．また，この SpMV で用いられる行列 A（Alogrithm1 中の四角で囲まれた A）を圧縮することで，GMRES 中で使用するメモリ使用量の削減が可能である．そこで，本稿では，行列に対する圧縮方式を提案し，省メモリかつ ELL を用いた場合の演算性能低下の少ない行列格納方式を提案する．

また反復法において，SpMV と同様に前処理が大きなボトルネックとされている．本稿では，高い並列化が可能だが弱い前処理である対角スケーリングを前処理に使用する．高速かつ強力な前処理の提案を今後の課題としている．

2.2 Sparse Matrix Vector multiplication (SpMV)

疎行列-ベクトル積 (SpMV) は，反復法による連立 1 次方程式の求解，固有値問題等の様々な問題で使用される重要な処理の一つである．疎行列は SpMV の計算に不必要な零要素が多く存在する．非零要素のみを効率良く取り出し SpMV 処理を行うことがメモリ使用効率，演算性能の向上の重要な課題である．式 1 に本稿で使用する SpMV の定義を記載する．ここで，y, x はベクトル，A は疎行列を表す．

$$y = Ax \tag{1}$$

SpMV の演算性能は，疎行列を格納する形式により大きく変化する．行列や SpMV を処理する計算機が異なるとまた最適な疎行列格納方式も異なってくる．GPU では，メモリアクセスの最適化，分岐処理の削減等が演算性能に大きな影響を与え，CPU に対し，最適な行列格納方式と GPU に対して最適な行列格納方式は異なる．

図 1 に代表的な行列格納方式を示す．Compressed Sparse

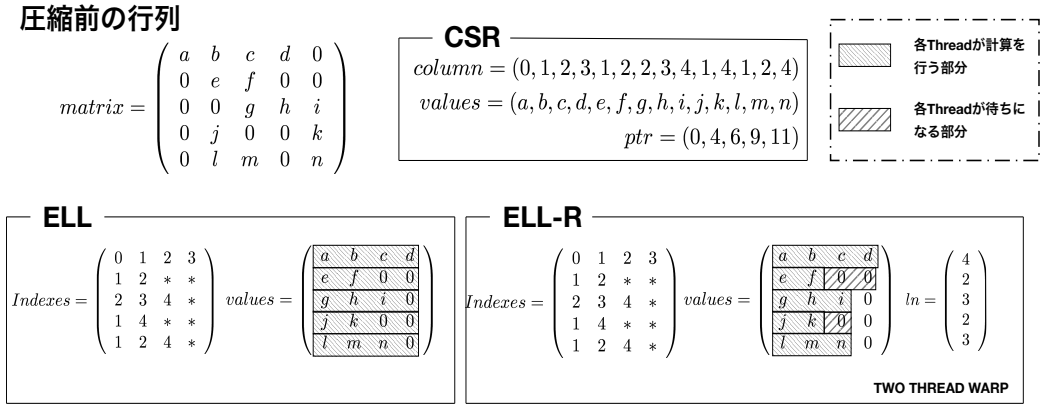


図 1　代表的な行列格納方式

Row（CSR）は，元の行列の非零要素のみを values の配列に格納する．そして，各値が格納されていた列番号を Column の配列に格納する．最後に配列の何番目の要素から次の行の値になるかを示すポインタを ptr に保持する．CSR の特徴として，非零要素のみを保持するため，無駄のない格納方式として知られている．しかし，ポインタを保持する必要があるため，巨大な行列になると行のポインタの配列も大きくなる傾向になる．また，GPU においては CSR を用いた SpMV では演算性能の向上が得にくいという問題点もある．

図左下に ELLPACK（ELL）の格納方式を示す．ELL は，行列を 2 つの行列（値を保持する行列と列番号を保持する行列）を用いて保持する．二つの行列のサイズは，（元の行列の行数*行の非零要素の最大数）となる．そのため，計算に不必要な要素も格納することとなるために，メモリ使用量が増加する．しかし，各行の要素数がほぼ同じであれば，CSR に比べ，行番号の情報を保持する必要がないため，メモリ使用量が少なくなる．また，ELL は GPU 等のベクトル型マシンに対して適した行列格納方式として知られている [3]．そこで，本稿では高速かつ省メモリな行列格納方式を目指すため，ELL に対する圧縮方式を検討する．

図右下に ELL の派生系である ELLPACK-R（ELL-R）の格納方式を示す．ELL-R では，各行の非零要素数を保持する．各行の非零要素数を保持することで，並列計算する際に，各行を担当するプロセッサは，各行の非零要素数分のみループを回すこととなり，ELL と比べ無駄な演算が削減される．但し，図中の青いブロックで示すように，GPU 上では同 warp 内では，全ての Thread の処理が終了するまで，次の処理を行わない．そのため，warp 内の一番非零要素数が多い行を担当する Thread の演算が終了するまで他の Thread は待たされる．

3. 関連研究

3.1 GPU を用いた GMRES の実装

現在までに，GPU を用いた GMRES を含む反復法の高速化が多く試みられてきた．Chong らの研究 [4] では GPU クラスタ上における CG 法の高速化を提案している.GPU クラスタ上では, ノード間のデータ転送時間がボトルネックとなり,CG 法の高速化の問題点となっていた. これを解決するため, 行列のバンド幅を削減することで, データ転 送が必要なデータ量が減少し, オーバーラップを行うことで, クラスタ内でのデータ転送時間をほぼ隠ぺいすることに成功している. Byron らは FGMRES のマルチコア CPU，GPU，マルチ GPU の環境での並列化を提案している [5]. OpenMP と CUDA を用いて，二つの並列粒度により並列化を行っている．疎行列-ベクトル積の部分を LocalScope として実行し，GPU 上で演算が終わるまでを LocalScope としている．Globalscope では，同期の必要としない部分を複数のスレッドに分割し，並列化を行っている．この二つの Scope を効率良く組み合わせ，高速化に成功している．しかし，Byron らの研究では，行列の格納方式には触れていないため，行列を格納するために多大なメモリを使用していると考えられる．Jacques らは GPU クラスタ上での GMRES の実装を行っている [6]．CPU-GPU, GPU-GPU のデータ転送を削減するために，各ノードが保持する SpMV の演算に必要とされるベクトルの分割方法を工夫している．しかし，行列の保持の方法として既存手法である HYB 行列格納方式を使用している．HYB 行列格納方式は ELL と COO を組み合わせた格納方式である．ELL 単体で使用するより保持する零要素が減り，効率のよい格納方式である．しかし，列番号を保持するために冗長な情報が含まれている．本稿では，この冗長な情報をさらに減らすような圧縮の方式を提案する．圧縮により一つの GPU が保持できる行列の量が増えるとノード間のデー

タ転送が減り，クラスタでの演算性能も向上すると考えられる．

本稿では，既存の ELL 格納方式に対する圧縮方式を提案し，メモリ使用量の少ない ELL 格納方式を提案する．1GPU あたりの演算量を増やすことで，巨大な連立一次方程式を解く際に必要となる GPU の数が減少し，低コスト化が可能であると考えられる．

3.2 様々な疎行列格納方式

3.2 節の GMRES に関する研究では，既存の行列格納方式を使用していたが，現在では GPU に適した行列の格納方式に関する研究が盛んに行われている．

Vzquez らは，ELL に対し，改良を施し，ELLPACK-R（ELL-R）という行列格納方式を提案している [7]．この方式では，ELL で格納されている行列の各行の要素数を保持することで，無駄な演算を削減し，演算性能を向上させている．通常 ELL では，演算に不必要な零要素を保持するため，演算量が多くなってしまう．しかし，各行の演算に必要な要素数を保持することで，演算性能を向上させることが可能である．但し，ELL-R は ELL のメモリ使用量を削減する格納方式ではなく，演算性能を高める方式である．Alexander らの研究では，slicedELL と呼ばれる格納方式を提案している [8]．slicedELL では，行列中の複数行を一つのブロックとし，各ブロックごとに ELL の格納方式を適用する．複数のブロックを使用することで，各ブロックごとの一番要素数が多い行に依存して各ブロックを格納する ELL の行列サイズが決定される．よって，各行の要素数が大きく異なっても，ブロック内でのみ無駄な零要素を保持するだけで済むので全体としてのメモリ使用量が削減される．本稿で提案する格納方式は省メモリ化に重点を置いている．提案する圧縮方式は，ELL-R,slicedELL 共に適用可能であり，多くの ELL から派生した格納方式に適用可能であると考えられる．このため，既存の ELL-R,sclicedELL よりも省メモリ化が可能である．Indexes 行列を圧縮する方法として，Wai らの研究がある [9]．Wai らはデルタエンコーディングを用いた疎行列の圧縮によるメモリ使用量の削減, そして,GPU が最適なメモリアクセスを行うことが可能なデータ配置を提案している. 前後の列の差分をとることで，列番号を表すために必要な bit 数が減り，結果的に使用するメモリ量が削減可能である．これにより SpMV の演算時間を, 古典的な手法に比べ最大 2.7 倍の短縮に成功している. 提案する圧縮方式では，連続する列番号の開始と終了のみを保持するため，デルタエンコーディングを用いる手法に比べ，連続する非零要素が多い行列では高い圧縮率を得ることが可能であると考える．

4. 提案行列圧縮方式

本稿では，ELL の列番号を格納する行列，Indexes 行列

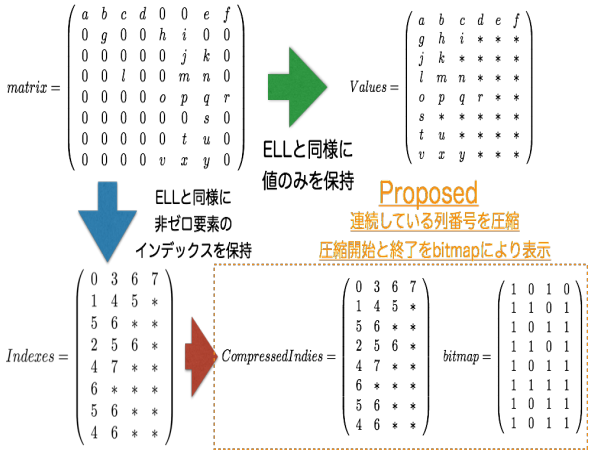


図 2 提案行列圧縮方式

に対する圧縮方式を提案する．疎行列では，連続した列に非零要素が存在することがある．このことに注目し，連続した非零要素を圧縮した形で表す．連続した非零要素の最初と最後を表すために，bitmap を導入する．図 2 に提案圧縮方式の概略を示す．上記のように ELL 方式では，matrix を Values 行列と列番号の Indexes の行列に分割し，保持する．提案圧縮方式では，図示したように，列番号が 0 から 3 まで連続している場合は，CompressedIndexes の行列に 0 と 3 を格納する．bitmap 行列には，連続した非零要素の最初に対応する要素に 1，最後に 0 を格納する．このようにすることで，列番号を圧縮後も元の行列に復元することが可能である．bitmap は原理的には 0 と 1 の二値のみ使用するため，integer 型の列番号を保持するより，少ないメモリ使用量となることが期待される．ELL の Indexes の行列を圧縮した行列を用いる方式を ELL-C（ELLPACK-Compression）とする．

Algorithm2 に ELL の Indexes 行列を圧縮するアルゴリズムを示している．ここで，$CcI[]$ は圧縮後の Indexes 行列を表し，$bitmap[]$ は圧縮された連続非零要素の最初の要素と最後の要素を表すための行列である．N は行数，k は圧縮前の ELL の列数である．$line8$ で要素が連続しているか判定し，連続している場合，$Count$ を更新せず同じ $CcI[count]$ に書き続ける．これにより連続している要素は最初と最後の要素のみ CcI の行列に格納される．圧縮に伴い，$bitmap$ の行列には 1 か 0 の値が格納される．$line12$ で前のループで読み込んだ要素が圧縮した最後の要素に当たる場合は，その要素に対応する $bitmap$ の index に 0 を格納する．以外の要素に対応する index には 1 が格納される．本稿では，SpMV の演算性能を向上させるため，行列は全て列優先で格納している．列優先で要素を格納することにより GPU のメモリアクセス効率が向上し，GPU での SpMV の演算性能が向上する．

Algorithm3 に ELL-R に対し提案圧縮方式を適応した行列を ELL-RC とし，ELL-RC を用いた SpMV のカーネ

Algorithm 2 Method of Compress Indexes

```
1: for i = 0 to N do
2:     count, seq = 0
3:     base = indies[i + N * count]
4:     bitmap[i + N * count] = 1
5:     CcI[i + N * count] = Indies[i + N * count]
6:     count = count + 1
7:     for j = 0 to k do
8:         if Indies[i + N * j] − base == 1 then
9:             base + +, seq + +
10:            CcI[i + N * count] = base
11:        else
12:            if seq >= 1 then
13:                CcI[i + N * count] = base
14:                bitmap[i + N * count] = 0
15:                count + +, seq = 0
16:            end if
17:            CcI[i + N * count] = base = Indies[i + N * j]
18:            bitmap[i + N * count] = 1
19:            count + +
20:        end if
21:    end for
22: end for
```

ルを示す．ここで，ID は自身のスレッド番号，st,end は圧縮された非連続零要素の開始位置と終了位置，len は各行の要素数，CcI は図 2 の CompressIndexes 行列に相当し，$bitmap$ は図 2 の bitmap 行列に相当する．val は図 2 の Values 行列に相当し，ELL と同様の行列である．

$line4$ では ELL-R を参考に，各行の非零要素数を len に格納し，無駄なループを削減している．$line6$ から $line12$ までが ELL-R のカーネルと大きく異なり，圧縮された Indexes 行列を展開する処理を記述している．st,end には，圧縮された非零要素の開始位置と終了位置が格納される．もし，圧縮されている場合には，$bitmap$ は 1 と 0 が並んで格納されるため，$bitmap[j]$ と $bitmap[j+1]$ を読み込み，1 と 0 と並んでいる場合，st をループごとにインクリメントしていき，$st == end$ になるまで次の index を読み込まないように処理を行う．$CcI[]$ と $bitmap[]$ は $val[j]$ とは異なるインデックスを使用する．$counter$ を使用し，圧縮されている部分を処理している間は $CcI[counter]$,$bitmap[counter]$ として，インデックスがインクリメントされないように制御する．また，GPU ではカーネル内の分岐処理が増加すると，演算時間も増加する傾向にある．そのため，$line7,8,9$ のようび $bitmap$ が 0 か 1 しか保持しない特性を用いて，極力 if 分を使用せず，ループごとの処理を制御し，演算時間の削減に努めている．x は式 1 の x に対応するベクトルである．

5. SpMV 性能評価実験

複数の行列を univesity of Florida collection（UF collection）[10] から入手し，各行列格納方式ごとの SpMV の演算性能を測定する．実験に使用する環境を表 1 に，使用する行列の情報を表 2 に示す．表 2 中の N は行列の一辺の長さを示し，nnz は行列中の非零要素数，k は行列を ELL で格納した場合の Values 行列と Indexes 行列の列数を示している．ck は ELL の Indexes 行列に提案圧縮方式を適応した後で Indexes 行列の列数を示す．本実験では，各行列格納方式ごとのメモリ使用量，SpMV の処理時間（GPU 演算時間+CPU-GPU データ転送時間）を測定する．式 1 に対応するベクトル x は乱数を用いて生成した．

Algorithm 3 SpMV Code on GPU

```
1: counter ←0
2: ID ← Thread′sid
3: if ID < N then
4:     len ← ln[ID]
5:     for j = 1 to len do
6:         bitval = bitmap[ID + N * counter]
7:         st = CcI[ID+N*counter]*bitval+(1−bitval)*(st+1)
8:         counter = counter + (bitval − 1)
9:         end = CcI[ID + (N * (counter + 1))]
10:        if st == end then
11:            counter = counter + 1
12:        end if
13:        temp = temp + val[ID + N * j] * x[st]
14:        counter = counter + 1
15:    end for
16:    y[ID] = temp
17: end if
```

表 1 SpMV 評価実験環境
Table 1 Enviroment.

仕様	Tesla K20
OS	CentOS 6.2
CPU	Intel Xeon E5-2687 @ 3.1GHz
GPU	NVIDIA Tesla K20 @ 0.71GHz
GPU Memory	480MB
CUDA	CUDA 7.5
Compiler	gcc-4.4.7

表 2 実験に使用した行列
Table 2 Information of Matrix.

仕様	N	nnz	k	ck
gemat12	4,929	33,111	28	22
dw8192	8,192	41,746	8	7
mac_econ	206,500	1,273,389	47	47
cop20k_A	121,192	1,362,087	75	53
cant	62,451	2,034,917	40	18
mc2depi	525,825	2,100,225	4	4
rma10	46,835	2,374,001	145	40
consph	83,334	3,046,907	78	38
pwtk	217,918	11,634,424	90	18
af_shell9	504,855	17,588,845	30	10
F1	343,791	26,837,113	378	150
nd24k	72,000	28,715,634	483	128

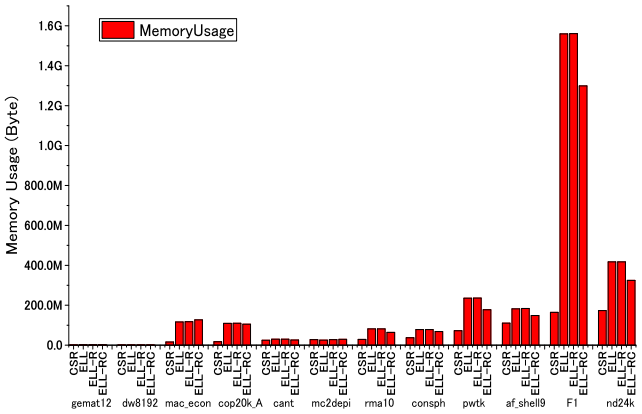


図 3　MemoryUsage

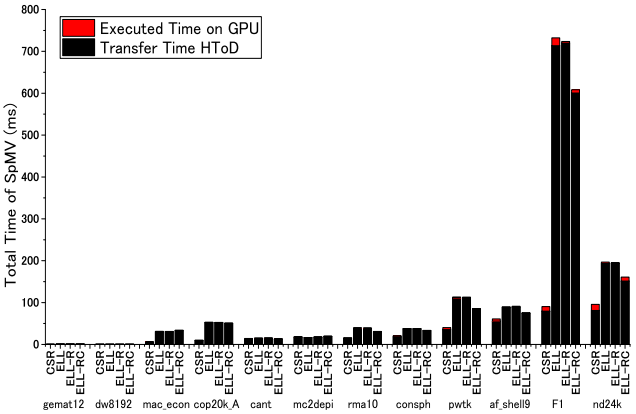


図 4　Total Time of SpMV

5.1　メモリ使用量の評価

以下に各既存行列格納方式が必要とするメモリ使用量を求める式を示す．ここで，N, nnz, k, ck は表 2 の値と対応している．

$$MemUsage_{CSR} = 8nnz + 4nnz + 4(N+1) \quad (2)$$

$$MemUsage_{ELL} = 8(N*k) + 4(N*k) \quad (3)$$

$$MemUsage_{ELL-R} = 8(N*k) + 4(N*k) + 4N \quad (4)$$

提案行列格納方式のメモリ使用量を式 5 に示す．ここで $ck =$ は Indexes 行列を提案圧縮方式で圧縮後の列数を表す．

$$MemUsage_{ELL-RC} = 8(N*k)+4(N*ck)+(N*ck)+4N \quad (5)$$

図 3 に各行列格納方式で行列を格納した場合の使用メモリ量を示す．ここで，図 3 の縦軸は使用メモリ量 (Byte) を示し，横軸は使用した行列名と行列を格納した方式名である．格納方式は既存手法として CSR，ELL，ELL-R，提案手法として ELL-RC である．提案圧縮方式が最も圧縮に成功している行列は pwtk であり，25 %の削減となった．pwtk では $k = 90$ から $ck = 18$ と大幅に列数が減少した．また，他の行列において，0.5%から 22.2%のメモリ使用量削減に成功した．しかし，mac_econ，dw8192,mc2depi の 3 つの行列において 3%から 8.33%，メモリ使用量が増加した．これは，連続した非零要素が少なく，効率良く圧縮できなかったためである．このことにより，bitmap を追加した分のメモリ使用量が増加した．しかし，全体的に ELL 系統の行列格納方式に対して，提案圧縮方式の高いメモリ使用量の削減効果を得ることに成功した．

5.2　SpMV 演算時間の評価

図 4 に各行列格納方式ごとの SpMV 処理時間を示す．図 4 の縦軸は CPU-GPU 間のデータ転送時間と GPU 上での

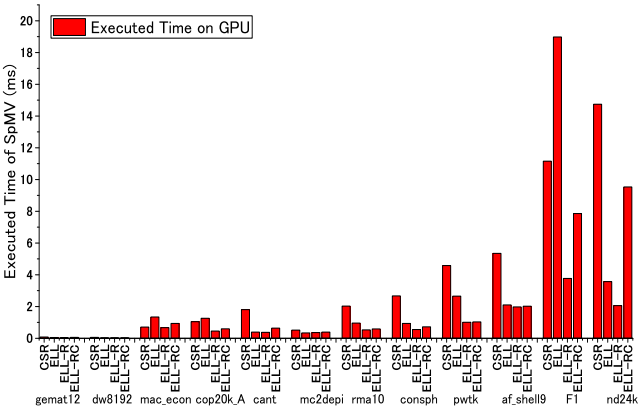


図 5　Executed Time on GPU

演算時間と足した SpMV の合計時間（ms）であり，横軸は使用した行列名とその行列の格納方式名である．CSR は ELL に比べ，データ転送時間が短く，1 回あたりの SpMV 処理時間も最短の行列が多い．これは，ELL が無駄な 0 要素を保持していることが問題である．ELL 系統の圧縮方式では，ELL-R,ELL-RC が ELL に比べ処理時間が短くなっている．ELL 系統の圧縮方式の中では，データ転送時間は ELL-RC が最短の行列が多く，提案圧縮方式によるメモリ使用量の削減が効果的であることが明らかである．

SpMV1 回の演算時間では，CSR が最短である行列が多く存在した．しかし反復法では，最初にデータを転送し，収束するまで GPU 上で処理を行うことが可能なため，データ転送時間に比べ，SpMV の演算時間が支配的になる．このため，SpMV のデータ転送時間だけでなく，純粋な GPU 上での演算時間についても重要である．図 5 に GPU 上での SpMV の演算時間を示す．縦軸は GPU 上での SpMV の演算時間（ms）であり，横軸は図 4 と同様である．SpMV の演算時間のみでは，CSR に比べ，ELL-R, ELL-RC が高速な行列が多く見て取れる．pwtk や af_shell9 の行列では，ELL-RC は ELL-R と同等もしくは微小な演算時間の増加にとどまっており，ELL-RC はメモリ使用量の少なく，演算性能の低下が少ない格納方式と言える．しかし，圧縮効率

表 3　GMRES 性能評価実験に使用した行列
Table 3　Information of Matrix.

行列名	N	nnz	k	ck
pwtk	217,918	11,634,424	90	18
af_shell9	504,855	17,588,845	30	10
F1	343,791	26,837,113	378	150
nd24k	72,000	28,715,634	483	128

が悪い行列（mac_econ）においては，演算時間が ELL-RC が増加が顕著である．また，削減率が高い F1，nd24k の行列についても ELL-RC の演算時間が増加している．これは，F1,nd24k 共に圧縮後も不連続な値が多く存在し，カーネル内の st, end を計算する処理に多く時間が取られ，演算時間の増加が起こったと考えられる．複数の行列において，ELL-RC と ELL-R の演算時間は同等であり，連続した値が多い行列においては優れた行列格納方式であると言える．

6. GMRES 性能評価実験

本稿では，GPU 上での GMRES(m) の性能を評価するために，CUDA を用いた右前処理付き GMRES(m) プログラムを作成した．GMRES 中の SpMV に CSR 方式で格納した行列と ELL-R に提案圧縮方式を施した ELL-RK で格納した行列を用い，GMRES の処理時間の比較を行う．その他の GMRES の処理には NVIDIA が提供している cuBLAS（Dot Products : cublasDdot()，AXPY operations : cublasDaxpy()，Euclidean norms : cublasDnrm2()）を使用した．リスタートサイクル m : 150，最大反復数 : 1500 とし，残差 ϵ が 10^{-1} になるまで反復を行う．実験環境は SpMV 評価実験で用いた環境（表 1）と同じ環境である．GMRES に用いる係数行列を表 3 に示す．今回，SpMV 評価実験で使用した行列の中で，GMRES を用いて収束した行列のうち 4 つ選択した．また，右辺ベクトルは乱数，乱数を用いて生成したベクトルと係数行列を乗算し，生成した．この時，乱数を用いたベクトルを真の解とする．

図 6 に各行列ごとの GMRES 処理時間を示す．縦軸に GMRES の合計処理時間（ms）を示し，横軸に使用した行列名と，その行列の格納方式名を記載している．図 6 のグラフは GMRES の主要な処理ごとの演算時間を示しており，SpMV とデータ転送時間が支配的になっていることがわかる．pwtk，af_shell9 において，ELL-R と比べ最大で 5%と微小ではあるが，ELL-RC の処理時間が短い．これは，ELL の演算性能を保ちながら，行列のメモリ使用量を減らし，データ転送にかかる時間の削減に成功したためと考えられる．しかし，F1 では CSR，nd24k では ELL，ELL-R の処理時間が ELL-RC の処理時間よりも短くなっている．F1 では CSR のデータ転送時間が非常に短く，ELL 系統の方式が無駄な零要素を多く保持しているこ

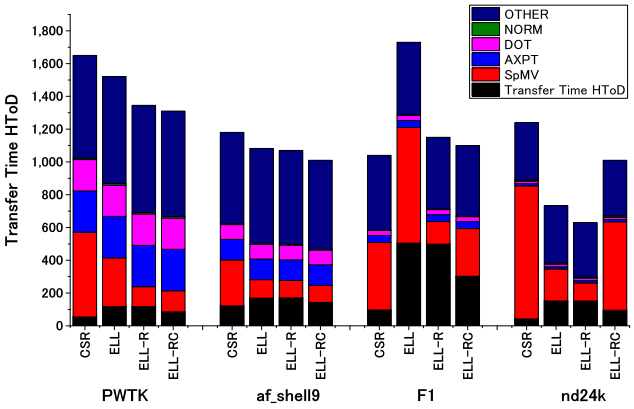


図 6　GMRES の処理時間

とがわかる．このことにより SpMV の演算時間は CSR に比べ ELL-R，ELL-RC 共に短いが，データ転送時間が支配的となってしまい，CSR が最も高速な結果となった．n24k では ELL と ELL-R が顕著に高速であるという結果となった．図 5 の SpMV の演算時間の結果からもわかるように ELL-R は nd24k において最も高速であり，CSR，ELL-RC は低速な演算性能であった．このため，GMRES の反復回数に伴い SpMV の演算時間が支配的なり，CSR．ELL-RC 共に全体の処理時間が増加した．そのため，今後の課題として，nd24k や F1 の行列のような演算性能の低下が見られる行列に対しても，オーバヘッドの少ない ELL-RC のカーネルを検討する必要がある．

7. おわりに

本稿では，行列格納方式 ELL に対する列番号を保持する行列に対する圧縮方式を提案した．ELL-R に提案圧縮方式を適応した ELL-RC のメモリ使用量は ELL-R に比べ，行列 pwtk において最大 25%の削減に成功している．また，複数の行列において SpMV の演算時間は ELL-R と ELL-RC は大きく変わらず，圧縮による演算性能の低下は微小であることが明らかになった．しかし，圧縮方式により圧縮が困難な行列においては，使用メモリ量の微量の増加が見られ，このことに伴い，演算時間の増加も見られた．今後の課題として，圧縮が有効な行列の判別方法の提案，各スレッドのロードバランスの平滑化が挙げられる．そして，今回列番号を保持する行列のみを圧縮したが，メモリ使用量が多い values 行列の圧縮方式の検討を今後の課題とする．

ELL-RC を適応した GMRES の演算性能の評価を行った．結果として，二つの行列において提案手法である ELL-RC が最速となったが，他の二つの行列においては，CSR，ELL-R が最短となった．原因として，提案圧縮方式を適用した場合に ELL-R の演算性能を大幅に低下させる行列が存在することが挙げられる．圧縮した行列を復元する際の処理がボトルネックとなっているため，今後この処理を

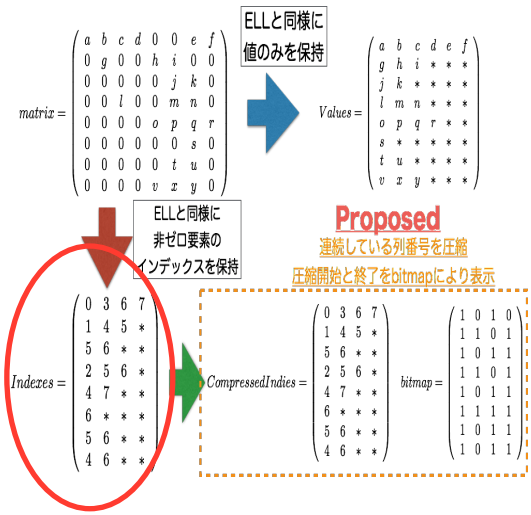
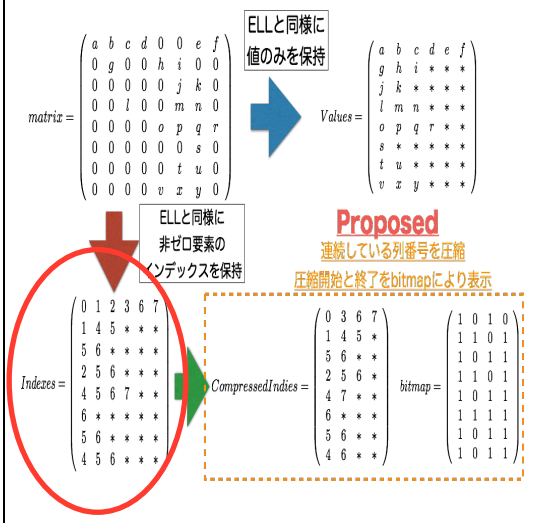
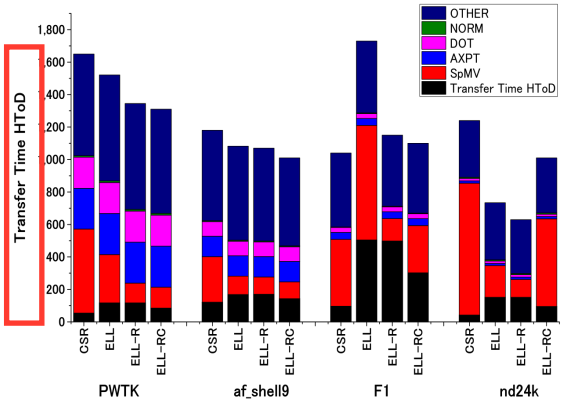
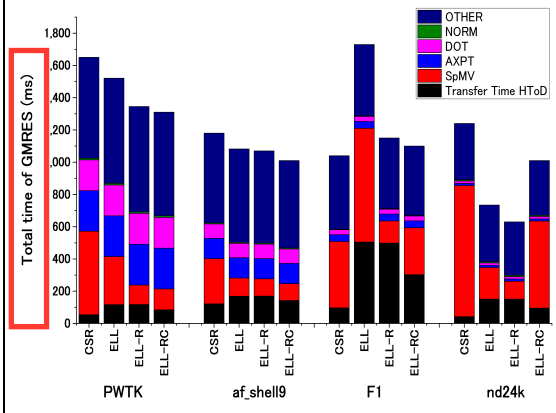
さらに効率よく行えるようにカーネルを改良する．また，CPU-GPU 間のデータ転送時間がボトルネックとなっていることから，GMRES の処理とデータ転送をオーバーラップさせ，データ転送時間を隠蔽する手法を検討する．

参考文献

[1] Nico Godel,Steffen Schomann,Tim Warburton, and Markus Clemens: GPU Accelerated AdamsBashforth Multirate Discontinuous Galerkin FEM Simulation of High-Frequency Electromagnetic Fields, IEEE Transactions on Magnetics, Volume 46, Issue 8, Pages 2735-2738, Arg,2010.
[2] Yousef Saad: Iterative Methods for Sparse Linear Systems Second Edition, Society for Industrial and Applied Mathmatics, 2003.
[3] Nathan Bell, Michael Garland: Efficient Sparse Matrix-Vector Multiplication on CUDA, "NVIDIA Technical Report NVR-2008-004", December 2008
[4] Chong Chen, Tarek M.Taha: A comunication reduction approach to iteratively solve large sparse linear systems on a GPGPU cluster, Springer US, Journal of Cluster Computing, Volume 17, Issue 2, Pages 327-337.
[5] Byron DeVries, Joe Iannelli, et.al. : Parallel implementations of FGMRES for solving large sparse non-symmetric linear systems，2013 International Conference on Computational Science，volume 18, Pages 491-500, 2013.
[6] Jacques M.Bagu, Lilia Ziane Khodja et. al. : Parallel GMRES implementation for solving sparse linear systems on GPU clusters，HPC '11 Proceedings of the 19th High Performance Computing Symposia, Pages 12-19 , 2011.
[7] F.Vzquez, J. J. Fernndez, E. M. Garzn: A new approach for sparse matrix vector product on NVIDIA GPUs, Concurrency and Computation Practice and Exprience, Volume 23, Pages 815-826, 2011.
[8] Alexander Monakov, Anton Lokhmotov, and Arutyun Avetisyan: Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures, 5th International Conference, HiPEAC 2010, Italy, January 25-27, Pages 111-125, 2010.
[9] Wai Teng Tang et.al.: A Familu of Bit-Representation-Optimized Formats for Fast Sparse Matrix-Vecter Multiplication on the GPU, IEEE TRANSACTION ON PARALLEL AND DISTRIBUTED SYSTEM, VOL26, Pages 2373-2385, 2014.
[10] Tim Davis, Yifan Hu : The University of Florida Sparse Matrix Collection，入手先 ⟨https://www.cise.ufl.edu/research/sparse/matrices/⟩（参照 2016-07-03）.

正誤表

発表原稿内の記載に誤りがあることが判明いたしました. 下記の通り訂正いたしますと共に深くお詫びを申し上げます.

訂正箇所	誤	正
p4 4. 提案行列圧縮方式　内 図 2　提案行列圧縮方式	 図中の Indexes の行列（赤丸部分）に間違いが判明いたしました.	 Indexes の行列（赤丸部分）内の値を正しく修正いたしました.
p7 6. GMRES 性能評価実験 “図 6 GMRES の処理時間”	 図中の縦軸ラベル”**Transfer Time HToD**” が誤りとなっております.	 図中の縦軸ラベルを”**Total time of GMRES (ms)**”と修正いたしました.
p7 19 行目	残差 ε が10^{-1}になるまで	残差 ε が10^{-9}になるまで