Vol.2020-MPS-127 No.5
2020/3/2

オンサイトでの高精度数値シミュレーション実施のためのGPU向き疎行列圧縮スキーム

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

概要：
近年，計算機の高性能化に伴い数値シミュレーションをオンサイトでリアルタイムに実行し，様々な産業に応用することが期待されている．このようなシミュレーションは，GPGPU を利用することによって実用的な計算時間での実行が期待できるが，メモリ容量の制約が大きな問題点である．そこで本稿では数値シミュレーションの代表的手法である Finite Element Method(FEM) で現れる疎行列のメモリ使用量削減手法を提案する．提案手法では，疎行列の列番号を表す値をパッキングし，メモリに格納する値の数を削減する．複数の疎行列による評価では，12 個中 10 個において従来手法に対しメモリ使用量を削減し，最大で 26.3%の削減率となった．また，オンサイト実施が期待される分野の一例として心臓シミュレーション用の疎行列にも適用したところ、メモリ使用量が 20.6%削減された．

キーワード：GPGPU, SpMV, FEM, 疎行列格納方式，圧縮率向上手法

1. はじめに

近年，様々な分野において高精度な数値シミュレーションの需要が増大している．通常，高精度な数値シミュレーションには大規模な計算機を使用する場合が多い．大規模な計算機は導入や管理のコストが高いため，外部組織が所有するものを借りてシミュレーションを行うケースも増えている．例えば医療分野における数値シミュレーション応用が考えられる。この分野では Heartflow 社が $\mathrm{FFR}_{\mathbf{CT}}$[1] の商用サービスを始めているが，これは Amazon Web Services(AWS) を活用したクラウドサービスである [2]．すなわち，狭心症が疑われる患者の CT 画像と診断情報が病院から AWS 上に構築されたサービス基盤に送信され，Heartflow の技術者が有限要素法による 3 次元の流体解析を実行，Fractional Flow Reserve(FFR) 値を算出し，病院にレポートを返すというものである．一方で，情報管理の観点や医師がシミュレーションをコントロールしたいという要望もあり，オンサイトのシミュレーション実行が望まれるケースがある．これに対しては，院内に配置可能なサイズのワークステーションで現実的な時間内に解析できるよう，次元削減した数理モデルを用いて FFR 値を算出する研究が行われており [3][4]，数十分での解析が実現されている．しかし，次元削減により流速や血圧の空間分布あるいはずり応力といった様々な詳細な情報が抜けているという欠点があり，高精度かつオンサイトでの実行が可能なシミュレーションが求められている．

上記のようなオンサイト環境での高精度な数値シミュレーション実現のために，General-Purpose computing on Graphic Processing Unit (GPGPU) の活用が考えられる．従来高精度な数値シミュレーションを実用的な時間で実行するためには，大規模計算機が必要であったが，GPU を汎用的な計算に適用する技術の発展により，数値シミュレーションを省スペースかつ高速に実行可能となった．しかし巨大な CPU メモリと異なり，GPU メモリは高々数十 GB しかないという大きな制約がある．数値シミュレーションの代表的な手法である Finite Element Method (FEM) を用いた場合，シミュレーションの高精度化により，FEM で扱う疎行列のサイズが増大する．この疎行列が GPU メモリへ格納しきれない場合，CPU と GPU 間の転送が頻発し，演算性能が大きく低下する．

そこで本稿では，FEM を用いた数値シミュレーションで扱う疎行列を対象にした疎行列圧縮手法を提案し，GPGPU の小容量のメモリに格納する．評価としていくつかの FEM の行列と，医療への応用例として心臓シミュレーション

[1] 北陸先端科学技術大学院大学
JAIST, Nomi, Ishikawa, 923–1292, Japan
[2] 富士通株式会社
Fujitsu Limited.
[a)] t-kawamura@jaist.ac.jp

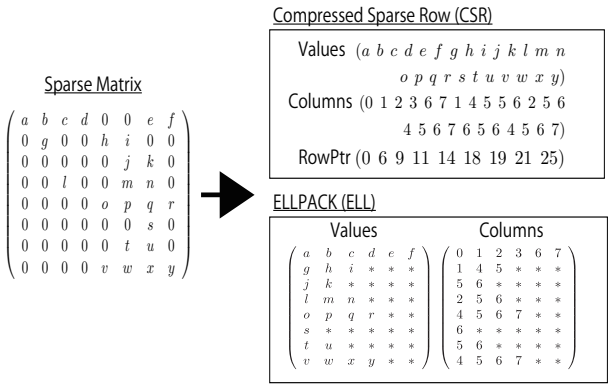

図 1 Conventional storage formats for sparse matrices

(UT-Heart)[5] の行列を用いた．GPU はメモリ参照の特性や並列演算器の動作が CPU とは異なるため，圧縮した疎行列の演算速度が重要なポイントになるが，提案手法では GPGPU による演算速度は従来手法と遜色無いながらも，圧縮率を最大 26.3%向上できる．

本稿は，第 2 章で FEM における疎行列の説明と行列圧縮に関する関連研究を俯瞰する．第 3 章で提案する疎行列格納方式に対する圧縮方法を詳しく述べる．第 4 章にて提案圧縮方法の評価し，第 5 章にてまとめを行う．

2. FEM における疎行列とその圧縮法

2.1 FEM における疎行列

物理値を持つ節点が配置される．要素内の物理値は各節点の物理値と節点ごとの補間関数により近似される．解析領域全体の積分方程式は，要素ごとの方程式に分割される．これら要素ごとに作成された方程式を重ね合わせることで，解析領域全体の連立 1 次方程式が得られ，最終的にはこの方程式の求解に帰着する．連立 1 次方程式に現れる行列の各行は節点の物理値に対応し，各列は接続関係にある節点の物理値に対応する．すなわち接続関係にある場合は非ゼロ，その他はゼロとなるため，この行列はその大部分がゼロとなる疎行列である．高精度な解析のためには有限要素をより細かくする必要があるが，それに伴い疎行列のサイズも増大する．

大規模な疎行列の連立一次方程式を解く方法として，Generalized minimum residual (GMRES) 等の反復法がよく用いられる．FEM によるシミュレーションの高速化につながることから，反復法の高速化に関する研究が数多く行われている．

FEM で生成される疎行列は，反復法内の支配的な処理である Sparse matrix-vector multiplication (SpMV) で使用される．SpMV は多くの演算時間を消費するだけでなく，疎行列格納のため多くのメモリを消費する．このことから，GPGPU における効率的な SpMV 実行，省メモリ化のため，メモリへの疎行列格納方法が，多く検討されてきた [6].

2.2 疎行列の格納方式

疎行列を格納する方式により，疎行列を格納するためのメモリ使用量，SpMV の演算性能は大きく変化する．現在までに提案された様々な疎行列格納方式は，大きく分けると Compressed Sparse Row(CSR) と ELLPACK(ELL)[7] の二つが元となっている．図 1 に CSR，ELL の格納方法を示す．CSR の長所は，無駄なゼロ要素を一切格納しないため，少ないメモリ使用量で疎行列を格納できる点が挙げられる．CSR では，3 つの配列を用いて疎行列を表す．一つ目の配列（図中 CSR の四角内，$Values$）は，疎行列内の各非ゼロ要素の値を格納する．二つ目の配列は（図中 CSR の四角内，$Columns$），疎行列内の各非ゼロ要素の列番号を格納する．三つ目の配列（図中 CSR の四角内，$RowPtr$）は，$Values$ 配列と $Columns$ 配列における，疎行列の各行の最初の要素を示すインデックスを格納する．式 (1) に CSR を用いて疎行列を格納した際のメモリ使用量を示す．メモリ使用量の単位は byte である．

$$MemUsage_{CSR} = 8N_{\mathrm{z}} + 4N_{\mathrm{z}} + 4(N+1) \qquad (1)$$

本稿において，N は疎行列の行数を示し，N_z は疎行列内の非ゼロ要素数を表す．また，本稿では，疎行列内の 1 つの値の格納に 8byte(64bit) の倍精度浮動小数点数を用い，列番号等の 1 つのインデックスを格納するために 4byte(32bit) の非負整数を使用する．

CSR は非常にメモリ使用量が少ない一方で，GPU 上で SpMV の演算性能が低くなる傾向にある [8].

一方，ELL では，2 つの行列を用いて，疎行列を表す．一つ目の行列（図 1 中 ELL の四角内，$Values$）では，疎行列内の各非ゼロ要素の値を $Values$ 行列内の対応する行へ格納する．二つ目の行列（図 1 中 ELL の四角内，$Columns$）では，疎行列内の各非ゼロ要素の列番号を $Colmuns$ 行列内の対応する行へ格納する．これら二つの行列には，左詰めで要素を格納していくため，疎行列内のゼロ要素を削減して格納することが可能である．しかし，二つの行列の列数は，疎行列の行あたりの最大非ゼロ要素数とする必要がある．そのため，疎行列内の各行の非ゼロ要素数のばらつきが大きい場合，計算に使用されない無駄な要素をパディング (図 1 中，行列内の $*$) として多く格納する必要があり，メモリ使用量が増加する．一般的に，パディングには “0” を使用するため，本稿においてもパディングには “0” を使用する．CSR では非ゼロ要素のみの情報を保持していたが，ELL ではパディングによって 1 行あたりの要素数を均等にしているため，同一 warp(GPU 上での演算単位) 内のスレッドが常に連続したアドレスへのアクセスが可能である．したがって，パディングされた無駄な要素はメモリ使用量の増加を起こすが，メモリアクセスの観点から見れば非常に効果的である [8]．式 (2) は，ELL 用いて疎行列を格納した時に必要となるメモリ使用量の計算式である．

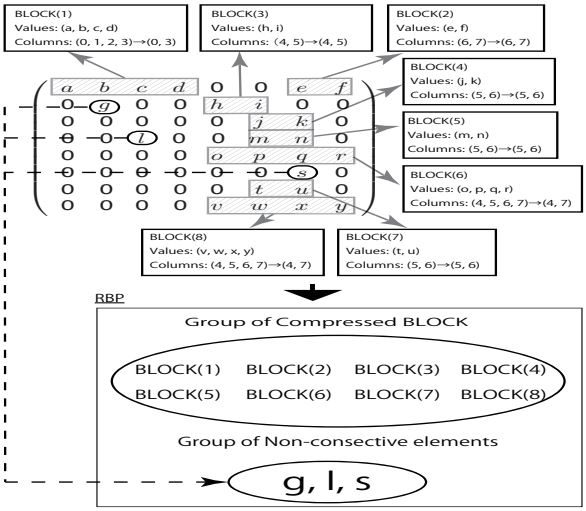

図 2 RBP 法の概略図

Algorithm 1 RBP 法の適用フロー

1: **疎行列を非ゼロ要素の連続部と不連続部に分割**
2: **連続部の要素をブロック化**
3: **各ブロックの Columns のデータの先頭と末尾のみを抽出し，格納**
4: **ブロック化部分を適応する疎行列格納方式で格納**
5: **不連続部の要素はすべて CSR 形式で格納**

$$MemUsage_{ELL} = 8NK + 4NK \quad (2)$$

ここで K は疎行列の行あたりの最大非ゼロ要素数である．$Values$ 行列と $Columns$ 行列内の要素数は NK となる．

3. ROW BLOCK PACKING METHOD (RBP 法)

3.1 RBP 法の説明

2.1 節で述べた FEM における疎行列の非ゼロ要素のパターンは，節点間の接続関係と，節点番号により決定される．1 つの節点は複数の節点と隣接していることから，疎行列内でも複数の連続した非ゼロ要素が存在すると考えられる．この疎行列内の連続する非ゼロ要素の列番号を圧縮することで，疎行列のメモリ使用量を削減可能である．

SpMV は，疎行列内に非ゼロ要素が連続して存在する場合，連続した非ゼロ要素の最初と最後の要素の列番号のみで，計算可能である．しかしながら，既存の格納方式である CSR や ELL では，連続した非ゼロ要素の全ての列番号を記憶する．RBP 法では，疎行列内に存在する連続した非ゼロ要素の列番号の最初と最後のみを格納することで，列番号を格納する配列に要する記憶領域を削減する．少量の GPU メモリを効率的に使用することで，オンサイトにおけるシミュレーションの高精度化，大規模化，同時に実行できるシミュレーション数の増加につながる．

図 2 に RBP 法の概略図，Algorithm1 に RBP 法を既存の疎行列格納方式に適用する手順を示す．

Algorithm1 の「**処理 1．疎行列を非ゼロ要素の連続部と不連続部に分割**」では，図 2 の疎行列のように，連続した非ゼロ要素（図 2 中，斜線の四角で囲まれた要素）と不連続な非ゼロ要素（図 2 中，丸で囲んだ要素）に分割する．

「**処理 2．連続部の要素をブロック化**」では，連続した非ゼロ要素の値と列番号を，図 2 中の BLOCK(1) から BLOCK(8) のように，ブロックの $Values$ と $Columns$ にそれぞれ格納する．RBP 法では，疎行列内の列番号が連続した非ゼロ要素（図中 “a, b, c, d” 等）を 1 つの塊として扱う．

「**処理 3．各ブロックの $Columns$ のデータの先頭と末尾のみを抽出し，格納**」 は RBP 法の最も重要な処理である．各ブロックの $Columns$ のデータ数を削減するため，SpMV の計算に必要な先頭と末尾の要素のみを取り出し，格納する（図 2 中，各ブロックの $Columns$ の矢印左側がデータ数を削除する前，矢印右側がデータを削除した後の $Columns$ の配列である）．この方法により，どれだけ長く連続した非ゼロ要素を BLOCK 化した場合でも，列番号を表すために必要となる要素数は 2 となる．よって，疎行列内に多くの連続した非ゼロ要素が存在，または 1 つの連続した非ゼロ要素の連続数が長い場合，大きなメモリ使用量の削減が可能である．

「**処理 4．ブロック化部分を RBP 法を適応する疎行列格納方式で格納**」は，既存の疎行列格納方式に対して，ブロック化した非ゼロ要素を格納していく処理である．格納する際には，各ブロックを本来の 1 つ非ゼロ要素と同等に扱い格納する（$Values$ と $Columns$ の複数の要素を一つの塊として考える）．この過程は，CSR や ELL などの既存の格納方式にて値や列番号などを格納する方法をブロック化された非ゼロ要素に対して適用することを意味する．例えば CSR であれば，ブロック化された非ゼロ要素の複数の値をそのまま値配列に，圧縮された 2 つの列番号は列番号の配列に，1 つの値を扱う場合と同様にブロック番号が小さいブロックから順に格納していく．ただし圧縮により，既存格納方式と同様の処理では格納できない場合も存在する．その際には配列の追加等，正しく SpMV の計算が行えるように変更を加える必要がある．本稿では代表的な疎行列格納方式である CSR,ELL の格納方式に則ったブロックの格納方法を図 3 を用い，次の節から説明を行う．

RBP 法は複数の値をブロック化するというシンプルな方法を取っているため，他の疎行列格納方式に対しても，多少の変更のみで同様に提案圧縮方法を適用可能であると考えられる．例えば，ELL の派生系である ELL-R は ELL と同様の方法で RBP を適用可能である．

最後の「**処理 5．不連続部の要素はすべて CSR 形式で格納**」 では，図中の “g, l, s” のような不連続な非ゼロ要素を CSR 形式で格納する．この時，ELL などの CSR とは異

なる格納方式においても，不連続な非ゼロ要素は連続した非ゼロ要素とは別の空間に CSR 形式で格納する．連続部と不連続部を分離して格納する理由は，メモリアクセス効率を向上させるためである．RBP 法のように，不連続な非ゼロ要素については CSR 方式で格納することで，連続した非ゼロ要素の列番号を格納している配列または行列は必ず連続した非ゼロ要素に対しアクセスすることとなる．これによりスレッドが 1 回の反復にて，必ず 2 つの値を参照する．1 スレッドのアクセス数が固定であれば，warp 内のスレッドが一定のインデックスでスライドし，アクセスすることが可能である．そのため，warp 内の各スレッドは常にコアレスドアクセスを行うことが可能である．

3.2 CSR への RBP 法の適用 (RBP-CSR)

図 3 に RBP 法を施した CSR(RBP-CSR) の概要を示す．Algorithm1 の処理 1 から 5 は，図 3 中の①から⑤に対応する．RBP 法の適用のための処理に則り，図 3 の疎行列に示すよう，連続部（斜線四角）と不連続な非ゼロ要素（楕円）を分割する（Algorithm1 の処理 1）．次に各連続部の各連続した非ゼロ要素をブロック化する（Algorithm1 の処理 2）．図 3 の例では，疎行列内に 8 個の連続した非ゼロ要素が存在するため，8 個のブロックが作成される．その後，各ブロック内の $Columns$ 内のデータの先頭と末尾を抽出し，その他の値を削除する（Algorithm1 の処理 3）．図中に $Columns$ の圧縮前と後の値を示す．Algorithm1 の処理 4 では，適用する疎行列格納方式に則り，ブロックを格納する．CSR への適応では，インデックスが小さいブロックから，$Values$ と $Columns$ の値を一つの塊として，CSR の $Values$ と $Columns$(図 1 の CSR に対応) に対応する配列へ順に格納する．図中の $Comp\_Values$, $Comp\_Columns$ がブロック内の $Values$ と $Columns$ が格納される配列である．

図 3 の疎行列を CSR で格納する場合, 値と列番号は 1 対 1 で対応することから，1 行目の要素は $Values$ が “a, b, c, d”, “e, f”, Columns が “$0, 1, 2, 3$”, “$6, 7$” のようにそれぞれの配列で要素数が一致し，Ptr 配列は $RowPtr$ の 1 つのみ使用する．しかし RBP 法の圧縮により，値と列番号の数に差異が生じるため，RBP-CSR では 2 行目の要素が始まるインデックスは $Comp\_values$ で 6 番目，$Comp\_Columns$ では 4 番目となる．そのため RBP 法では $Comp\_Values$, $Comp\_Columns$ の何番目のインデックスで行が変わるのかを表す値を格納する Val_ptr, Col_ptr の 2 つの配列を用意し，並列計算に必要な各行の先頭の要素のインデックスを判別する．また，連続部と不連続部の分割により，5 行目（最初の行を 0 行目とする）のような連続した非ゼロ要素が存在しない行も出現する．その場合には Val_ptr, Col_ptr に同じインデックスを続けて格納することで，行中に要素が存在しないことを示す．図 3 の行列では 1 行目には不連続な非ゼロ要素が存在せず，2 行目から不連続な非ゼロ要素 “g” が出現する．そのため，$RowPtr$ の 1 番目と 2 番目に “$0, 0$”(図 3 中，赤い波線) を格納する．2 行目は $RowPtr$ 内 2 番目と 3 番目 “$0, 1$”(図 3 中，青い直線) のように非ゼロ要素 “g” が 1 つ存在することを示す．最後の Algorithm1 の手順 5 では，図中の “q, l, s” の不連続な非ゼロ要素を CSR 方式で格納している．ここで，不連続な非ゼロ要素の値を $ValuesCSR$ に，列番号は $ColumnsCSR$ に格納される．また，各行の最初の非ゼロ要素の情報が $ValuesCSR$ と $ColumnsCSR$ のどのインデックスに格納されているかを，$RowPtr$ に格納する．計算の際, 各スレッドは隣り合う二つの要素を $RowPtr$ から読み込むことで，$ValuesCSR$, $ColumnsCSR$ の要素を並列して計算可能である．RBP-CSR では追加の配列を必要とするため，全く連続した非ゼロ要素が存在しない疎行列においては CSR よりメモリ使用量が増える可能性がある．

式 (3) に RBP-CSR を用いて疎行列を格納した場合のメモリ使用量を示す．以降，N_{col} は圧縮した後の列番号数 ($Comp\_Columns$ の要素数)，N_{val} は $Comp\_Values$ の要素数，N_{non} は疎行列内の不連続な非ゼロ要素の数を示す．

$$MemUsage_{RBP-CSR} = 12(N+1) + 4N_{\mathrm{col}} + 8N_{\mathrm{val}} + 4N_{\mathrm{non}} + 8N_{\mathrm{non}} \quad (3)$$

Algorithm2 は，RBP-CSR 形式で格納された疎行列を用いた SpMV 演算を示す．各スレッドが Algorithm2 を同時に実行し，それぞれが疎行列内の 1 行を担当する．id はカーネル関数を実行する各スレッドの番号を示している．id は，CUDA の関数を用いることで各スレッドが取得することが可能である．RBP-CSR の SpMV カーネルは連続した非ゼロ要素を計算するパートと不連続な非ゼロ要素を計算するパートに分かれている．Algorithm2 内，3 行目から 13 行目までが連続した非ゼロ要素を計算するパートである．4 行目からの for ループでは，スレッドが担当する行の非ゼロ要素の値を $Comp\_Values$ から読み込みため，Val_ptr から担当する行に対応するインデックスを読み込む．5,6 行目の $ColNow$ と $ColNext$ には連続した非ゼロ要素の最初と最後の列番号が代入される．7 行目の処理は，連続した非ゼロ要素の先頭の要素と SpMV で用いられるベクトルの要素との掛け算を計算する．8 行目の for ループにて，$ColNow$ から $ColNext$ になるまで j をインクリメントしていくことで，先頭以降の要素の列番号を毎回グローバルメモリから読み出すことなく，インクリメントのみで列番号を復元し，SpMV の計算を行う．

14 行目から 18 行目は，不連続な非ゼロ要素を計算する部分である．CSR の SpMV カーネルと同様の処理を行っている．不連続な非ゼロ要素に対し，演算を行い，連続した非ゼロ要素に対する演算結果と合算している．最後に結果を，各スレッドが担当していた行と同じベクトル中の行

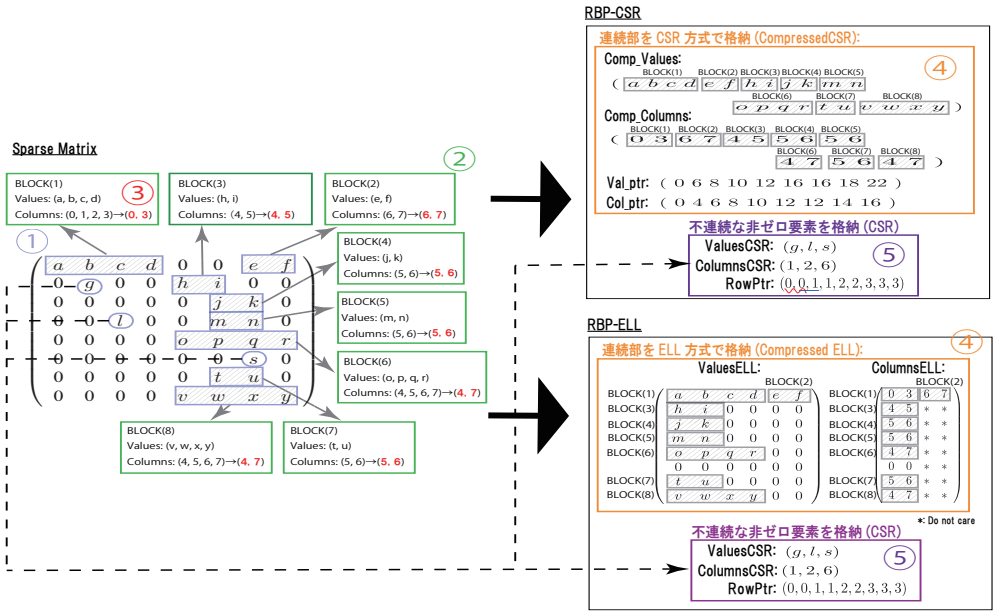

図 3 RBP-CSR and RBP-ELL

Algorithm 2 SpMV code using RBP-CSR on GPU

```
1: Let id be thread ID (id: 0 to n-1)
2: Set Count, TmpReslut = 0
   /* Calculation of compression part */
3: RowStart = col_ptr[id]
4: for jj = val_ptr[id] to val_ptr[id + 1] − 1 do
5:     ColNow = Comp_Columns[RowStart + Count]
6:     ColNext = Comp_Columns[RowStart + Count + 1]
7:     TmpResult+ = Comp_Values[jj] ∗ dVector[ColNow]
8:     for j = ColNow + 1 to ColNext do
9:         jj + +
10:        TmpResult+ = Comp_Values[jj] ∗ dVector[j]
11:    end for
12:    Count+ = 2
13: end for
   /* Calculation of non-compression part */
14: RowStart = RowPtr[id]
15: RowEnd = RowPtr[id + 1]
16: for j = RowStart to RowEnd − 1 do
17:    TmpResult+ = Values[j] ∗ dVector[Colmuns[j]]
18: end for
19: Result[id] = TmpResult
```

に格納し，終了する．

3.3 ELLPACK への RBP 法の適用 (RBP-ELL)

図 3 に RBP 法を施した ELL 格納方式，RBP-ELL の概念図を示す．RBP 法の ELL への適用方法は CSR とほぼ同じであることから，ここでの詳細な説明は省略する．

式 (4) に RBP-ELL を用いて疎行列を格納した場合のメモリ使用量を示す．

$$MemUsage_{RBP-ELL} = 8NK_{\mathrm{v}} + 4NK_{\mathrm{c}} + 8N_{\mathrm{non}} + 4N_{\mathrm{non}} + 4(N+1) \quad (4)$$

ここで，K_{v} は Compressed ELL の $ValuesELL$ 行列の列数，K_{c} は $ColumnsELL$ 行列の列数を示す．

表 1 Environment for the experiments

	Specification
OS	Ubuntu 16.04
CPU	Intel Core i7 6700K @ 4.0 GHz
GPU	NVIDIA Tesla V100 @ 1.38 GHz
device memory	16 GB
device memory bandwidth	900 GB/s
CUDA core	5120
CUDA	CUDA 10.1
Compiler	gcc-4.4.7

表 2 Various sparse matrices for the experiments

Name of matrix	N	N_{z}	N_{non}	K	K_{v}	K_{c}
cant	62,451	4,007,383	39,097	40	40	16
rma10	46,835	2,374,001	447	145	145	40
consph	83,334	6,010,480	30,461	78	78	38
parabolic_fem	525,825	3,674,625	999,115	7	6	6
pwtk	217,918	11,524,432	199	90	90	18
thermal2	1,228,045	8,580,313	2,846,134	11	9	8
af_shell9	504,855	17,588,845	919	30	30	10
F1	343,791	26,837,113	46,738	378	378	150
nd24k	72,000	28,715,634	441,476	483	472	114
dielFilterV2real	1,157,456	48,538,952	10,631,364	99	79	48
UT-Heart1	82,047	3,423,519	0	63	63	40
UT-Heart2	130,595	6,954,413	969,595	124	111	50

4. RBP 法の性能評価実験

4.1 Experimental environment

表 1 に本稿の評価実験で用いた GPU サーバの実験環境を示す．GPU による評価を行うために，NVIDIA が提供する CUDA10.1 を使用し，各疎行列格納方式を用いた SpMV のプログラムを記述した．データ転送時間，GPU 演算時間は，CUDA のイベント変数を用いて，“cudaEventRecord()”,“cudaEventElapsedTime()” により測定を行った．また，本評価実験で使用した 12 個の疎行列を表 2 に示す．最初の 10 個は Florida Sparse Matrix Collection[10] よりダウンロードした．UT-Heart1，UT-Heart2 は，心臓

シミュレーション [5] の流体構造連成解析における，非圧縮性を扱った混合型の定式化により現れる行列である．医療分野のアプリケーション例として評価対象に入れた．また，表 2 中の N,N_z,N_{non},K，K_{v},K_{c} は，式 (1) から (4) で使用されているものに対応している．本実験にて使用する疎行列格納方式は，CSR, RBP-CSR, ELL, RBP-ELL に加え，ELL-R と ELL-R に RBP 法を適用した RBP-ELL-R である．RBP-ELL-R の説明は紙面の都合上省略したが，ELL と同様の方法で RBP 法を適用可能である．Vázquez らが提案した，ELL の演算性能を高めた ELL-R[9] に RBP 法を適用することで，RBP 法の演算性能への影響も確認する．

4.2 各疎行列格納方式のメモリ使用量の評価

図 4 に各疎行列格納方式を用いてそれぞれの疎行列を格納した際のメモリ使用量を示す．疎行列格納方式のメモリ使用量は式 (1) から (5) を用いて導かれた値である．

CSR，ELL と RBP-CSR，RBP-ELL のメモリ使用量を比較すると，それぞれ表 2 の 12 個中 10 個の行列において，RBP 法を適用した格納方式の方が少ない結果となった．RBP-CSR では nd24k において，最大の 26%のメモリ使用量の削減に成功した．10 個の疎行列における平均のメモリ使用量の削減率は，13.2%であった．この結果から，FEM で生成される行列には多くの連続した非ゼロ要素が存在することが確認できる．RBP-ELL では，pwtk において，メモリ使用量の削減率が高く，ELL に比べ，26.3%の削減に成功している．また，RBP-ELL の 10 個の疎行列の平均メモリ使用量削減率は 15.3%となった．結果として RBP 法の適用により，既存の疎行列格納方式のメモリ使用量を 6 個以上の疎行列において，20%以上の削減に成功した．大きな削減率となった疎行列の特徴として，連続した非ゼロ要素が各行に多く存在することがあげられる．また医療分野のアプリケーションレイとして評価に使用した UT-Heart1，UT-Heart2 においても，RBP 法を適用することでそれぞれ 12.4%，20.6%，20.6%のメモリ使用量の削減を達成した．

しかしながら，parabolic_fem と thermal2 の 2 つの行列では，RBP 法を用いても既存の疎行列格納方式のメモリ使用量を削減できていない．これは，parabolic_fem や thermal2 の行列内には，連続した非ゼロ要素が少ないことと，連続した場合でも連続数が少ないことが原因である．parabolic_fem や thermal2 では，最大連続数が 2 であり，RBP 法による効果がない．それどころか，不連続な非ゼロ要素を格納するのに CSR を使用するため，この分のメモリ使用量が増大した．

4.3 GPU 上での SpMV 演算時間の評価

本節では，RBP 法を各疎行列格納方式に適応することによる SpMV 演算時間への影響を評価する．図 5 に各疎行列格納方式を用いた SpMV の演算時間を示す．図に示した SpMV 演算時間は，GPU 上で実行される SpMV のカーネル関数の実行時間であり，データ転送時間は含まない．それぞれの疎行列で SpMV の演算時間を 10 回測定し，算術平均で求めた．使用メモリ量の削減に成功した疎行列において，既存の疎行列格納方式 CSR，ELL を用いた SpMV の演算時間よりも，RBP-CSR，RBP-ELL の演算時間がそれぞれ短くなった．RBP-CSR が最も演算時間の削減に成功した nd24k では，CSR の SpMV 演算時間に比べ 2.07 倍の高速化を達成した．また平均では，1.45 倍の高速化となった．RBP-ELL は，pwtk において最大の 2.89 倍の高速化に成功している．全体の平均高速化率は 1.49 倍である．圧縮率が高い疎行列においては RBP 法は，SpMV 演算性能向上のための最適化技術の一つとして使用可能である．

UT-Heart1, UT-Heart2 では，RBP-ELL を使用することで．ELL-R と同等の演算性能となり，メモリ使用量は ELL-R 以下となったことから，さらなる高精度なシミュレーションをオンサイトで行うことが可能となった．RBP 法は，メモリ使用量の少ない CSR のメモリ使用量をさらに削減したり，演算性能が高い ELL や ELL-R のメモリ使用量を削減することで，使用する GPU のメモリ量や演算性能を考慮し，疎行列格納方式に対する選択の幅を広げる．

5. おわりに

本研究では，高精度な数値シミュレーションを GPGPU によりオンサイトで実現するための課題の一つとして，連立一次方程式の求解の省メモリ化に取り組み，FEM で生成される疎行列の特殊性を考慮した，疎行列格納方式に対する圧縮方式を提案した．本提案圧縮方法である RBP 法は，FEM が生成する疎行列内に多くの連続した非ゼロ要素が存在することに着目し，連続した非ゼロ要素の列番号を圧縮し，格納する方式を採用している．メモリ使用量の評価実験では，RBP 法を用いることで CSR のメモリ使用量を平均 13.2%，ELL のメモリ使用量を平均 15.3%削減した．SpMV 演算性能も従来手法に比べてほぼ同等か，場合によっては性能が向上した．

参考文献

[1] Min, J.K., Taylor, C.A., Achenbach, S.K., Bon Kwon Leipsic, J., Nørgaard, B.L., Pijls, N.J., De Bruyne, B. : Noninvasive Fractional Flow Reserve Derived From Coronary CT Angiography Clinical Data and Scientific Principles, *JACC: Cardiovascular Imaging*, Vol.8, No.10, pp.1209-1222 (2015).

[2] Modi, B. N., Sankaran, S., Kim, H. J., Ellis, H., Rogers, C., Taylor, C. A., Rajani, R., Perera, D.:Predicting the physiological effect of revascularization in serially diseased coronary arteries: Clinical validation of a novel CT coronary angiography-based technique, *Circ Cardiovasc Interv.*, Vol.12, No.2, pp.1-8 (2019).

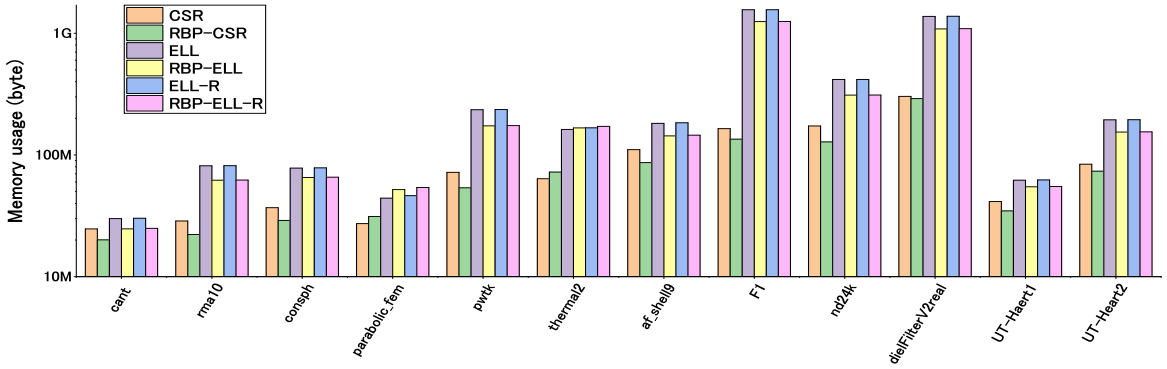

図 4 Memory usage of each storage formats

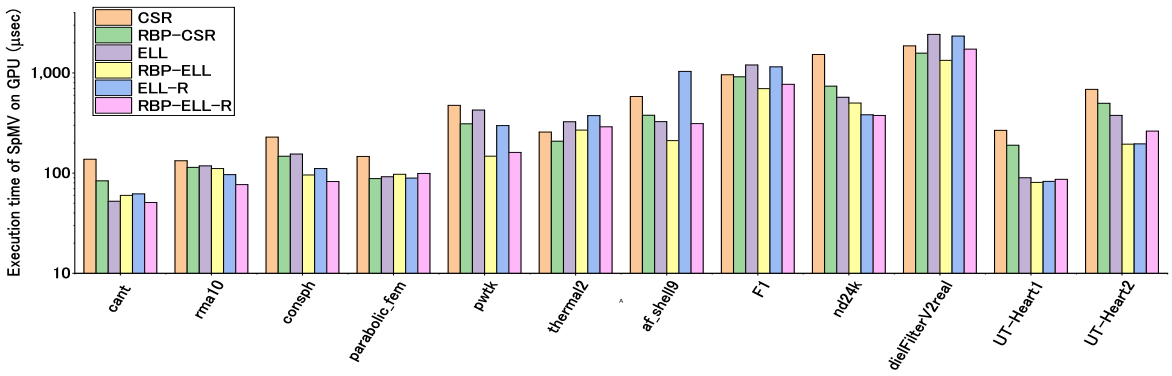

図 5 Execution time of SpMV on GPU

[3] Kato, M., Hirohata, K., Kano, A., Higashi, S., Goryu, A., Hongo, T., Kaminaga, S., Fujisawa, Y. :Fast CT-FFR Analysis Method for the Coronary Artery Based on 4D-CT Image Analysis and Structural and Fluid Analysis, Proc. *International Mechanical Engineering Congress and Exposition* (IMECE2015), ASME, p.1-10 (2015).

[4] Coenen, A., Lubbers, M.M., Kurata, A., Kono, A., Dedic, A., Chelu, R.G., Dijkshoorn, M.L., Gijsen, F.J., Ouhlous, M., van Geuns, .M., Nieman, K. : Fractional Flow Reserve Computed from Noninvasive CT Angiography Data: Diagnostic Performance of an On-Site Clinician-operated Computational Fluid Dynamics Algorithm, *Radiol.*, Vol.274, No.3, pp.674-683 (2015).

[5] Sugiura, S., Washio, T., Hatano, A., Okada, J., Watanabe, H., Hisada, T.: Multi-scale simulations of cardiac electrophysiology and mechanics using the University of Tokyo heart simulator, Progress in Biophysics and Molecular Biology, Vol.110, pp.380-389(2012).

[6] Salvatore Filippone,Valeria Cardellini, Davide Barbieri, et al. “Sparse Matrix-Vector Multiplication on GPGPUs”, ACM Transactions on Mathematical Software (TOMS), Volume 43 Issue 4, March 2017

[7] David R. Kincaid, Thomas C. Oppe, and David M. Young, “IT-PACKV 2D User’ s Guide, CNA-232”, https://www.ma.utexas.edu/CNA/ITPACK/manuals/userv2d/node3.html January 2018 Available

[8] Nathan Bell, Michael Garland: “Efficient sparse matrix-vector multiplication on CUDA”, NVIDIA Technical Report NVR-2008-004, NVIDIA, http://www.nvidia.com/docs/IO/66889/nvr-2008-004.pdf, December 2008

[9] F.Vázquez, J. J. Fernández, E. M. Garzón, “A new approach for sparse matrix vector product on NVIDIA GPUs”, Concurrency and Computation Practice and Experience, vol.23, no.8, pp.815-826, 2011.

[10] Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. ACM Trans. Math. Softw. 38, 1, Article 1 (December 2011), 25 pages. DOI: https://doi.org/10.1145/2049662.2049663