Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

メモリアクセス時データ精度変換機構による SpMV 処理の高速化の検討

胡 思已[1,2,a)]　伊藤 真紀子[3]　吉川 隆英[3]　近藤 正章[2,4]

概要：疎行列ベクトル積（SpMV）の演算カーネルは計算科学のアプリケーションをはじめ，グラフ処理や機械学習などで利用される重要な計算カーネルである．SpMV カーネルはデータアクセスに対して演算数が少なく，また疎行列データの再利用性がないため，特にメモリアクセスが計算処理のボトルネックになることが知られている．メモリとのデータ転送量を削減するためにデータ圧縮を利用することも考えられるが，疎行列データは通常倍精度浮動小数点データであり，可逆圧縮では圧縮効率低く，また復号化のレイテンシも問題となる．本稿では，メモリアクセスインタフェースにデータ精度変換機構を設け，高精度が必要ない計算の前半ではメモリアクセスの際に倍精度浮動小数点データをより低精度なフォーマットに変換することでデータ転送量を削減する手法を検討する．さらに，疎行列データを管理するための配列データも圧縮して転送することで，より高いメモリアクセス効率の達成を目指す．CG 法のカーネルを用いて評価した結果，精度変換・圧縮機構によりデータ転送量を最大で 1/3 倍程度削減でき，性能を 1.9 倍程度向上できることがわかった．また，いくつかの行列データにおいて，収束速度は倍精度浮動小数点を利用した際とあまり変わらないことがわかった．

1. はじめに

疎行列ベクトル積（SpMV）は，計算科学のアプリケーションをはじめ，グラフ処理や機械学習などでも利用される重要な計算カーネルである．疎行列は通常の行列と異なり，非零要素に関する情報のみを保存・演算するため，データ容量や演算回数の面で高効率な処理を行うことができる．一方で，疎行列は大規模になることが多くキャッシュにデータセットが収まりきらない，データアクセスに対して演算処理が少ない，またベクトルへのアクセスがランダムになる，などの理由から特にメモリアクセスが処理のボトルネックになることが知られている．そのため，メモリバンド幅が性能をほぼ律速し，システムのピーク性能に対し実効性能が非常に小さくなる．例えば，スーパーコンピュータ富岳で SpMV 処理が主となる HPCG ベンチマークを実行した際の性能は 16.0 PFLOPS 程度であり，ピーク性能に対して約 3.0%ほどの実効性能しか出せていない [1]．そのため，いかにメモリバンド幅ボトルネックを緩和するかが性能向上において重要となる．

これまでにも，プロセッサ・主記憶間のデータ転送ボトルネックを改善するための手法が検討されてきた．例えば，データ転送量を削減するために可逆圧縮技術を用いることも考えられる [2]．しかし，疎行列データは通常浮動小数点データであり，可逆圧縮では圧縮効率が低く，また復号化の際のレイテンシも問題となる．近年では，低精度のデータフォーマットを積極的に使用する混合精度演算の利用も検討されている [3,4]．例えば，倍精度浮動小数点で各データが表現されている疎行列を，単精度浮動小数点データに変換してアクセスすることで，SpMV 計算時にデータ転送のほとんどを占める疎行列データアクセスのデータ量が半分になることから効果も大きい．一方で，演算の精度が低下して収束性が悪化するため，倍精度浮動小数点による演算も併用する必要があり，データ変換のオーバーヘッドや，データ精度切り替え時の処理が複雑化するなどの問題がある．

本稿では，メモリアクセス時にメモリインタフェース上でデータ精度の変換を行う専用の機構を設け，高精度が必要ない計算の前半ではメモリアクセスの際に倍精度浮動小数点データをより低精度なフォーマットに自動的に変換しデータ転送量を削減する手法を検討する．メモリ上には倍精度のデータを保存することで，高精度が必要になる処理の後半では変換せずにそのままデータを利用して混合精度演算を簡便に利用することが可能となる．さらに，疎行列

[1] 東京大学
[2] 慶應義塾大学
[3] 富士通株式会社富士通研究所
[4] 理化学研究所
[a)] hu@hal.ipc.i.u-tokyo.ac.jp

Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

データのデータ転送量が削減されると，疎行列データを管理するための配列データのバンド幅ボトルネックが顕在化するため，当該配列も圧縮して転送することで，より高いメモリアクセス効率の達成を目指す．

提案手法を CG カーネルを用いて評価した結果，演算に対するメモリアクセスの比率である Byte-per-Flop（B/F）比を最大で 3 倍程度改善することができることがわかった．また，多くの行列データにおいて，収束速度は倍精度浮動小数点を利用した際とそれほど変わらないことがわかった．さらに，サイクルレベルのプロセッサシミュレータである鬼斬弐を用いて CG 内の SpMV カーネルの実行性能を評価した結果，精度変換・圧縮手法を用いない場合に対して最大で 1.92 倍の高性能が得られることもわかった．

2. SpMV 処理の概要と関連研究

2.1 SpMV カーネル

疎行列ベクトル積 (SpMV) は現在最も重要な演算カーネルの一つであり，構造解析のシミュレーションや機械学習手法など幅広いアプリケーションで利用されている．SpMV カーネルは，連立一次方程式 ($Ax = b$) の反復解法ソルバーに用いられることが多く，解の収束には多数の実効イテレーションが必要になる．

密行列を表現するデータ構造では，例えば 2 次元配列で行列データを格納し，配列の各要素を行列のインデックス a_i と a_j で指定することが一般的である．このフォーマットでは，行列要素を全て格納する必要があるため，行列 A のサイズが $m \times n$ の場合，データ保存のメモリ容量やメモリとプロセッサ間のデータ転送量も $m \times n$ に比例する．一方，疎行列ではデータ要素の値のほとんどがゼロであるため，非ゼロ要素のみを保存することで，使用するメモリ容量を大幅に削減することができる．

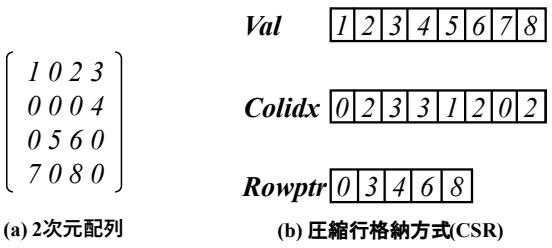

図 1: 疎行列データの保存形式

疎行列を表現するデータ格納方式としては，圧縮行格納方式（Compressed Sparse Row: CSR）や圧縮列格納方式（Compressed Sparse Column: CSC），座標形式（Coordinate: COO）など，様々な形式が存在する．本稿では，最も一般的な系式である CSR 形式を前提に説明する．CSR 形式の例を図 1 に示す．

図 1(a) の密行列の表現形式である 2 次元配列形式を CSR 形式で表現した場合を図 1(b) に示している．2 次元配列形式とは異なり，CSR 形式では行列中のゼロ要素の値を保存せず，非ゼロ要素のみを Val 配列に連続的に格納する．各非ゼロ要素の元の 2 次元配列上での位置を示すため，それぞれの要素の各行内の列インデックスを保持する配列である $Colidx$，および Val 中の各行の非ゼロ要素の先頭のインデックスを示す $Rowptr$ を用いる．第 i 行の非ゼロ要素を参照する場合，$Rowptr$ から $Row_{Start} = Row_i$ と $Row_{End} = Row_{i+1} - 1$ を算出し，第 i 行目にある $Colidx[Row_{Start}] \sim Colidx[Row_{End}]$ 列にある非ゼロデータを，$Val[Row_{Start}] \sim Val[Row_{End}]$ にアクセスすることで参照する．疎行列ベクトル積では，ベクトルデータの $Colidx[Row_{Start}] \sim Colidx[Row_{End}]$ 番目にあるデータに間接参照を行なうことでアクセスし，非ゼロデータとの積算を行う．ゼロ値である要素の比率が高い疎行列の場合，CSR 形式によりメモリやストレージの使用量を大幅に削減することができる．一方で，間接参照となるため，疎行列・ベクトル積などではベクトルデータ配列への参照がランダムになり，また非ゼロ要素へのアクセスに Val と $Colidx$ の両配列へのアクセスが必要なことから，メモリアクセスの効率が悪くなる．

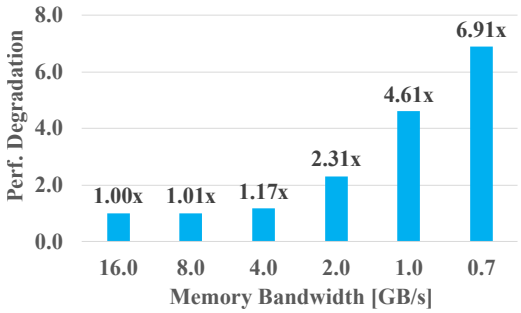

図 2: SpMV のメモリバンド幅の性能への影響

2.2 SpMV カーネルの性能特性

前節で述べたように，疎行列形式でデータを格納する方式ではメモリ使用量を削減することができるが，メモリアクセスの効率が悪くなる．疎行列・ベクトル積である SpMV カーネルでは，非ゼロ要素が倍精度浮動小数点データとすると，1 行列要素を用いた乗加算に，Val の 1 要素（8B データ）と $Colidx$ の 1 要素（4B データ）の合計 12B をメモリからアクセスする必要がある．なお，ベクトル配列はキャッシュからアクセスでき，$Rowptr$ へのアクセスは頻度が少ないため無視できると想定している．この場合，SpMV カーネルの B/F (Byte-per-FLOP) 比は 6 B/F となる．一方で，例えばスーパーコンピュータ富岳に用いられている A64FX [13] プロセッサのハードウェアが提供できる B/F 比は約 0.37 B/F であり，SpMV カーネルが要求する B/F 比とは大きな乖離がある．したがって，SpMV 処理ではメモリバンド幅が性能の最も厳しいボトルネックとなり，B/F 比から計算すると A64FX では $0.37/6 = 6.17\%$程度の実効性能しか出せないことになる．

Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

実際，4.1 節で述べるベンチンマークカーネルをサイクルレベルのプロセッサシミュレータでメモリバンド幅を変更しながら評価したものを図 2 に示す．図は 2.0 GHz のクロック周波数のプロセッサを仮定し，16 GB/s のメモリバンド幅を基準とした際にハードウェアが提供するメモリバンド幅を約半分ずつに縮小した場合の性能低下率を示している．4 GB/s 程度までは性能低下は見られないが，その後はメモリバンド幅がボトルネックとなり，バンド幅縮小の比率とほぼ一致して性能が低下している．このことからも，SpMV カーネルでは，メモリバンド幅が十分でない場合，ほぼメモリバンド幅のみで性能が律速されることがわかる．

2.3 関連研究

SpMV は非常に重要なカーネルであるため，SpMV 処理の高速化の研究は多く行われている．例えば，GPU やマルチコアシステム向けの SpMV カーネルの最適化が多く提案されている [5–7]．また，GPU 向けの高効率なブロッキングによる性能向上手法も提案されている [8]．

Grigoras らは，CSR 形式の非ゼロ要素リストに対する圧縮アルゴルズムを考案し，FPGA ベースのアクセラレータにそれを実装している [9]．値が同じ非ゼロ要素データをハッシュテーブルを用いてメモリ上で圧縮して保存することで，メモリからのデータ転送量の削減を狙ったものである．FPGA によりデータ圧縮を管理しつつ計算処理を加速することで，オーバーヘッドの削減と性能向上が得られている．

疎行列の格納データ方式の研究も行われている．Liu らは CSR 形式に対する改善手法として CSR5 フォーマットを提案し [10]，また SIMD アーキテクチャ向けの SpMV 計算処理アルゴルズムを用いることで，3 倍程度の性能向上が得られると報告している．Bian らは CSR5 を改良した CSR2 を提案し，CSR5 に比べて 1.5 倍の性能向上を達成している [11]．

これらの研究など，SpMV 処理の計算処理高速化や格納データ形式に関する研究は多く行われているものの，混合精度演算をサポートするためのアーキテクチャ手法に関する研究はまだ多くない．本研究はその点を狙っている点で新規性があると考えられる．

3. メモリアクセス時データ精度変換機構の提案

本章では，疎行列演算カーネルでの非ゼロ要素を保持する行列データ，および CSR 形式のデータ構造で列インデックスを保持 *Colidx* を対象に，メモリインタフェース上で精度変換・圧縮を行う機構を提案する．なお，*Rowptr* にも圧縮を適用することも可能であるが，もともと *Rowptr* のデータサイズは大きくないことが多いため本稿では対象としない．

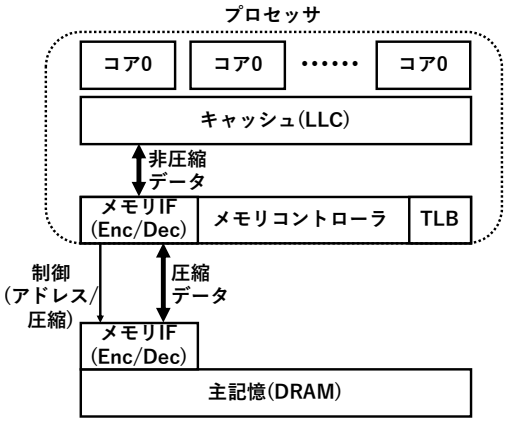

図 3: メモリ IF 上での精度変換・圧縮機構の概要

3.1 メモリインタフェース上の精度変換機構

メモリバンド幅のボトルネックを緩和させるため，データ転送量を削減するべくメモリインタフェースに精度変換・圧縮機構を追加する．提案するアーキテクチャの概要を図 3 に示す．

まず，主記憶である DRAM 側のインタフェースと，プロセッサ側のインタフェース（IF）に精度変換・圧縮のエンコード・デコードを行う機構を新たに追加する（図中の Enc/Dec）．プロセッサ側から精度変換・圧縮の方式を指定し，メモリ側にアクセスするアドレスとともにその指示を伝達する．Read 要求であれば，メモリ側 IF で読み出したオリジナルのデータに対して精度変換・圧縮のエンコードを行い，プロセッサ側 IF でそれをデコードする．Write 要求であれば，プロセッサ側 IF で書き込み対象データの精度変換・圧縮を行い，データがメモリ IF に到達した際にメモリ側でデコードし書き込みが行われる．また，プロセッサコアやキャッシュでは主記憶上の元データと同じビット幅で演算やデータ保存を行う．このように，メモリ IF 上で精度変換を行うことで，通常 CPU の命令で実行する必要のあるフォーマット変換などが必要なくなり，軽量に精度変換を利用した混合精度演算が実現できると期待されるほか，アドレッシングは主記憶データと一貫したものになるため，キャッシュやプロセッサコア部に対して特別な拡張は必要ない．

本提案では，非ゼロ要素配列データや *Colidx* など，対象のデータ群のみに特定の精度変換・圧縮を適用する．どのデータにどのエンコード手法を用いるかを指定するため，メモリのページ単位に圧縮属性を持たせることにする．ページテーブルおよび TLB に圧縮属性フィールドを追加し，メモリアクセスのアドレス変換時にメモリコントローラがその情報を基に精度変換・圧縮の指示を行うことを想定する．圧縮属性はシステムコールによりセットされる．このため，行列データや *Colidx* はページごとにアライン

Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

して格納する必要があり，フラグメンテーションなどメモリ領域利用の非効率性が生じる可能性もあるが，もともと非ゼロ要素配列や $Colidx$ は大規模データになることが多いため，大きな問題とはならないと考えられる．

3.2 データの精度変換・圧縮手法

前節で述べた精度変換・圧縮機構を利用するためには，圧縮属性を指定する必要がある．浮動小数点データの場合は通常の圧縮ではあまり圧縮効率が高くないこともあり，精度変換によるデータ精度を落として実行する非可逆圧縮を用いることが有効であると考えられる．一方で，$Colidx$ は整数データであり，正しく計算を実行するためには可逆圧縮が必要となる．そこで本稿では，例えばキャッシュラインなどのデータ転送単位ごとにメモリ IF 間での圧縮方式として以下のパターンを検討する．

(1) 倍精度浮動小数点 → 単精度浮動小数点

(2) 倍精度浮動小数点 → 半精度浮動小数点

(3) 倍浮動小数点 → 指数部：共通上位ビット＋個別オフセット圧縮，仮数部：固定長切り捨て

(4) 32 ビット整数 → 共通上位ビット＋個別オフセット圧縮

(1) の精度変換は，多くの場合数値の精度が劣化するが単純な変換で実装可能である．一方で (2) の場合は，指数部のビット長が 11 ビットから 5 ビットに減少するため，指数部のビット長が不足して大きな誤差が生じる可能性がある．また，bfloat16 では仮数部のビット長が不足する懸念がある．そこで，より汎用的な精度変換を行うべく，(3) として指数部は，共通上位ビット＋個別オフセットによる圧縮を行い，仮数部は固定長で切り捨てを行う方式を検討する．これにより指数部は可逆圧縮となる．また，$Colidx$ の圧縮のために，(4) の共通上位ビット＋個別オフセット圧縮方式を 32 ビット整数データに適用する．なお，これらの圧縮方式自体は単純であり，他の数値フォーマット（例えば単精度を圧縮するなど）にも拡張可能である．本稿では (3) と (4) を主として評価する．以下，それらを詳述する．

3.2.1 疎行列データ向け浮動小数点数の圧縮

非ゼロ要素を保持する行列データは SpMV カーネルでも最も大規模なデータであり，倍精度浮動小数点型で定義されている場合が多い．IEEE754 で定義されている倍精度浮動小数点型の内部表現を図 4 に示す．64 ビットのデータ中，符号部（S，1 ビット），指数部（Exponent，11 ビット），および仮数部（Significand，52 ビット）から構成される．この表現形式から，以下の式で実際の小数点数を表すことができる．

$$(-1)^{Sign} \times 2^{Exponent-1023} \times \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \quad (1)$$

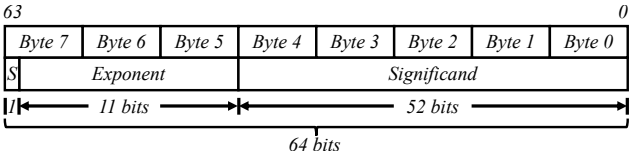

図 4: 倍精度浮動小数点型の内部表現

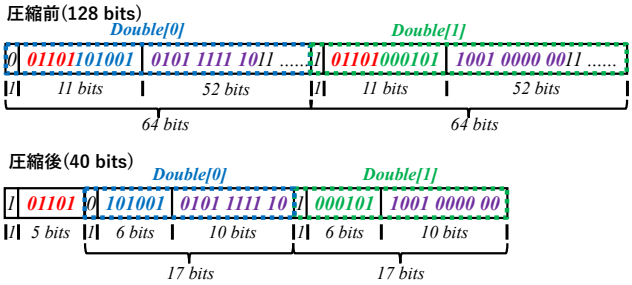

図 5: 倍精度浮動小数点データのブロック毎の精度変換の概要

指数部として最大 2^{1023} を，また $2^{-53} \approx 10^{-16}$，つまり 15 桁程度の有効数字での数値精度を表現することができる．一方で，問題によっては計算過程ではこれほどの精度を必要としない場合も多い．例えば，後で評価するように CG 法では計算途中ではより低精度なデータ型を利用しても収束性評への影響が小さいこともわかっている．そこで，精度において重要な指数部には可逆圧縮を，精度に影響が相対的に少ない仮数部には非可逆圧縮を用いる精度変換手法を提案する．提案する変換手法の概要を図 5 に示す．

まず，指数部はアドレス空間上で近いデータ間では値の絶対値が大きくは変わらないことが多いため，指数部の値は比較的近い値となる，つまり指数部の上位ビットは共通であることが多いと考えられる．そこで，メモリアクセス時の単位となるキャッシュライン毎に指数部の上位 n ビットを共通指数部，下位 $m = (11 - n)$ ビットを個別オフセット部と分け，キャッシュライン内の全てのワードで上位ビットが共通であれば，共通部は 1 回のみ転送し，残りは各ワード毎に個別オフセットとして転送する．次に仮数部については数値精度が悪化しても計算の収束性への影響が少ないことを考慮し，あるビット長 s を定義して上位 s ビットのみを仮数部保持ビットとして転送し，下位ビットは切り捨てる．これにより非可逆圧縮となるが，疎行列データは通常読み出しのみが行われ，書き戻しは発生しない．そのため，高い数値精度が必要になった場合は s のパラメータを変更して実行することで，計算への影響は少なく抑えられると期待される．

図 5 の例では，共通指数ビットが $n = 5$，個別オフセットが $m = 6$，仮数部保持ビット $s = 10$ の例を示している．また，簡単のためキャッシュラインサイズは 16B，すなわち倍精度の浮動小数点 2 ワード分と仮定した場合である．2 ワードで指数部の上位ビットが共通であるため圧縮が可能であり，データ転送の際は最初に圧縮されているかどうか

Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

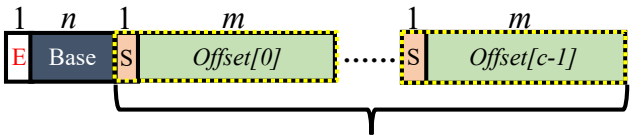

図 6: *Colidx* データに対する圧縮手法

のフラグビットがある他は，移行に共通指数ビット，そして各ワードの符号と個別オフセット，保持仮数部と続いたデータが転送されることになる．この場合，もともと 128 ビットの転送量だったものが 40 ビットまで圧縮できることがわかる．

表 1: 疎行列データ圧縮パターンに対する理論圧縮率（64B ライン）

n	m	s	合計ビット数	圧縮率
4	7	10	149	3.43
5	6	10	142	3.61
6	5	10	135	3.79
7	4	10	128	4.00
4	7	20	229	2.24
5	6	20	222	2.31
6	5	20	215	2.38
7	4	20	208	2.46

表 1 にラインサイズが 64B の場合における m, n, s のパラメータを変更した場合の非ゼロ要素配列データに対する理論圧縮率を示す．共通ビット部分 n が大きいほど，また保持仮数部ビット長 s が小さいほど圧縮率は高くなる．一方で，n が大きいと圧縮できないラインも増加することからトレードオフとなる．

3.2.2 *Colidx* 向けの整数データの圧縮

Colidx は，CSR 形式で疎行列データを保存する際に，各非ゼロ要素の列インデックスを示す整数（通常は 32 ビット）データの配列であり，非ゼロ要素配列と同じ回数アクセスされ，また再利用性も少ないため，メモリバンド幅の多くを消費するデータである．圧縮を考える上では，列のインデックスを保持するため，先に述べたように正確な値でなくてはならず，可逆圧縮手法を用いる必要がある．

ここで，列インデックスは疎行列データの非ゼロ要素の分布にはよるものの，各行では単調増加となるため，アドレス空間上で連続するデータは値が近いことが多いと考えられる．そこで，疎行列データの指数部に対して適用した，ワード毎に上位ビットと下位ビットに分割し，キャッシュライン単位で上位ビットが共通の場合には当該部分を共通して転送する方式を *Colidx* データにも適用する．図 6 にその概要を示す．

ワード毎に最上位ビットの符号部を除く 31 ビットの上位 n ビットを共通部，下位 $m = (32 - n - 1)$ ビットを個別オフセット部に分け，キャッシュライン内の全てのワードで上位ビットの値が共通であれば，共通部は 1 回のみ転送し，下位ビットは各ワード毎に個別オフセットとして転送する．共通部の値が異なる場合は圧縮は行わない．圧縮後のデータの最初のビット ‘E’ は圧縮ラインであるかどうかのフラグである．ここで，*Colidx* の場合は全てのデータが正の整数であるため，符号部も共通部分に含めて扱うことが可能であるが，将来的な他のデータへの拡張性も考え正負混在のデータにも圧縮が適用できるよう，本圧縮手法を用いることとした．なお，SpMV カーネル実行時のベクトルアクセスの再利用性向上のためキャッシュブロッキングを行った場合は各行で列インデックスの値が分割されるが，同じブロック内では列インデックスがある程度近い値になるため，効果は増加すると考えられる．

表 2: *Colidx* の圧縮パターンに対する理論圧縮率（64B ライン）

n	m	合計ビット数	圧縮率
15	16	272	1.88
17	14	242	2.12
19	12	212	2.42
21	10	182	2.81

表 2 にラインサイズが 64B の場合における m, n のパラメータを変更した場合の *Colidx* に対する理論圧縮率を示す．浮動小数点数である疎行列データの場合と同様に共通ビット部分 n が大きいど圧縮率は高くなる．一方で，n が大きいと圧縮できないラインも増加することが予想される．

4. 評価

本稿では，ベンチマークデータの疎行列に節で述べた精度変換・圧縮手法を適用した場合の圧縮率，CG 法の収束性能への影響，メモリアクセス時に圧縮を適用した場合の性能への影響を評価する

4.1 評価環境

圧縮率や収束性能評価には，C++版の NAS Parallel Benchmarks (NPB) 3.1 [12] 中の CG カーネルを利用する．NPB CG で作成される疎行列データの他に，疎行列データを集めた SuiteSparse Matrix Collection [14] から，データサイズが大きく，また CG 法に適用できるデータのいくつかを選択して用いた．圧縮率と収束性能評価は Intel Xeon プロセッサを搭載する実機で評価を行った．

提案手法を適用した場合の性能の評価には，サイクルレベルのプロセッサシミュレータである「鬼斬弐」を用いた．評価におけるプロセッサの仕様を表 3 に示す．シミュレーション評価では，シミュレーション時間の関係から NPB CG のデータセット中，サイズの小さな CLASS-S の SpMV カーネル部の 1 イテレーションのみを用いた．通常，SpMV カーネルの演算では，疎行列値と *Colidx* はキャッシュに収まりきらないが，各イテレーション内でも

Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

表 3: シミュレーションパラメータ

CPU Core	RISC-V single core @ 2GHz
Memory	200 cycle latency, 2.0 GB/s Bandwidth
L1 Cache	32 KB, 4-way set associative,
	64 Bytes cache line
L2 Cache	512 KB, shared, 8-way set associative,
	64 Bytes cache line
L1 Prefetcher	Stream Prefetcher
L2 Prefetcher	Stream Prefetcher

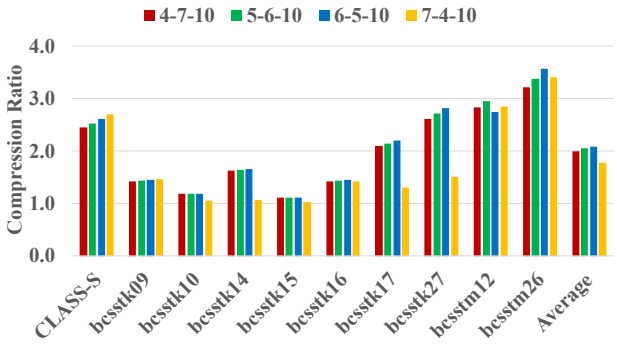

図 7: 非ゼロ要素行列の圧縮率評価結果

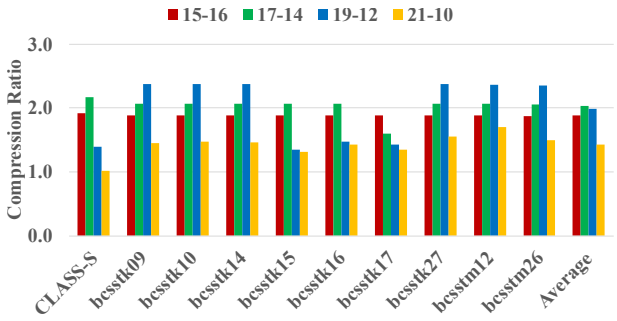

図 8: *Colidx* データの圧縮率評価結果

再利用性のあるベクトルは（ブロッキングなども利用して）キャッシュに収まることが多い．そこで，ラストレベルキャッシュである L2 キャッシュは 512KB と小さ目の容量を設定し，CLASS-S でも疎行列値と *Colidx* がキャッシュ上で再利用されず，メモリボトルネックとなる状況を想定した．また，1 コア 1 スレッドで，1 サイクルあたり倍精度浮動小数点乗加算が 1 回実行できるパラメータであるため，スーパーコンピュータ富岳の CPU である A64FX [13] の B/F 比（≈ 0.37）と近くなるよう，メモリバンド幅を 2.0GB/s と設定した．この際の B/F 比は 0.5 となる．

4.2 精度変換・圧縮による圧縮率の評価

NPG CG CLASS-S，および SuiteSparse Matrix Collection 中の疎行列データに対して提案する精度変換・圧縮手法の圧縮率評価結果を図 7（非ゼロ要素配列）と図 7（*Colidx*）に示す．非ゼロ要素配列には 3.2.1 節で述べた圧縮パラメータの n，m を変化させ s は 10 固定とした．図 7 中では，“*n-m-s*” と表記している．*Colidx* には 3.2.2 節で述べた圧縮パラメータの n，m を変化させた．図 8 中では，“*n-m*” と表記している．

図より，非ゼロ要素配の圧縮率は疎行列データに依存して大きく異なる一方，圧縮パラメータにはそれほど依存しないことがわかった．データの値の分布により指数部の共通ビットがキャッシュラインで同じかどうかに大きく影響されるためと考えられる．圧縮率は最大で 3.6 程度の場合もあり，データによっては高い圧縮率が達成できると言える．一方，*Colidx* の圧縮率は疎行列データの他，圧縮パラメータにも大きく依存していることがわかる．非ゼロ要素の分布の仕方の他，どれだけの上位ビットを共通部分とするかにより，圧縮可能かどうかと圧縮の効率が変化するためである．総じて *Colidx* では 16-15 のパターンの圧縮率が高く，その場合の理論圧縮率に近い 1.9 程度の圧縮率が達成できている．

表 4: 精度変換による CG カーネルでの収束性の影響

疎行列データ	非ゼロ要素数	収束までのイテレーション回数			
		非圧縮	1/10 iter	1/2 iter	All iter
bcsstk10	11578	215	1096	5099	*N/A*
bcsstk14	32630	2752	2615	2356	2356
bcsstk15	60882	4412	8092	8937	7957
bcsstk16	147631	38	38	38	38

4.3 CG の収束性評価

表 4 に，いくつかの疎行列データにおける精度変換・圧縮を適用した場合の収束までの CG カーネルの反復回数を評価した結果を示す．非圧縮は精度変換手法を適用しない場合，1/10 iter は非圧縮時の反復回数の 1/10 までを圧縮したデータで実行した場合，1/2iter は非圧縮時の反復回数の半分までを圧縮したデータで実行した場合，all iter は収束まで全てのイテレーションを圧縮データで実行した場合を示す．なお，NPB CG のデータセットでは反復回数が 1 回程度でも収束してしまうため，本評価では用いなかった．

結果より，bcsstk10 や bcsstk15 では精度変換をすることで収束までのイテレーション回数が大幅に増加してしまっていることがわかる．一方で，bcsstk14 や bcsstk16 では全イテレーションを精度変換しても反復回数が変わらないか，逆に少なくなっている．少なくなっているのは，数値精度が悪かった場合に偶然にも解への到達方向が良かったためと予想される．本評価では，NPB CG を用いて収束性能の評価を行ったが，より正しくは前処理付きの共役勾配法により評価すべきであると考えられる．HPCG [15] など，前処理付き共役勾配法を用いたベンチマークプログラムで評価を行うことは今後の課題である．

4.4 実行サイクル数評価

サイクルレベルのプロセッサシミュレータである鬼斬弐を用い，NPB CLASS-S のデータにおいて，SpMV カーネ

Vol.2022-ARC-248 No.42
Vol.2022-SLDM-198 No.42
Vol.2022-EMB-59 No.42
2022/3/11

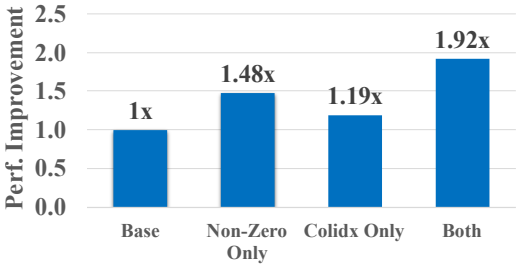

図 9: 精度変換手法による性能向上

ルの 1 イテレーション分の実行サイクル数を評価した結果を図 9 に示す．図は非圧縮の場合の性能（Base）に対する，非ゼロ要素配列にのみ圧縮を行った場合（Non-Zero only），*Colidx* にのみ圧縮を行った場合（Colidx only），両者に圧縮を行った場合（Both）の相対性能を表している．なお，非ゼロ配列の圧縮パラメータは 5-6-10，*Colidx* の圧縮パラメータは 17-14 を用いた．

図より，非ゼロ要素配列に圧縮手法を適用することで，1.48 倍の性能向上が得られることがわかる．非ゼロ要素配列はもともとが各ワード 64 ビットの倍精度浮動小数点データであり，圧縮率も 2.5 程度と半分以上までデータ転送量を削減できることが影響している．また *Colidx* は各ワード 32 ビットの整数データであり，圧縮率も 2.1 程度と非ゼロ要素配列ほど性能への影響は大きくないが，両者を同時に適用することで，性能は 1.92 倍ほど向上している．メモリバンド幅がボトルネックとなる SpMV カーネルでは，メモリアクセスの大部分を占める両配列に圧縮を適用することの性能への影響は大きいことがわかる．

5. まとめと今後の課題

本稿では，SpMV 処理向けにメモリインタフェース上でデータ精度の変換や圧縮を行う専用の機構を設け，非ゼロ要素配列に対して高精度が必要ない計算部分ではメモリアクセス時に倍精度浮動小数点データを低精度なフォーマットに変換し，かつ疎行列表現形式で必要となる *Colidx* などのデータもその特性を利用して圧縮し，データ転送量を削減する手法を提案した．圧縮率や収束性能について予備評価を行ったほか，サイクルレベルシミュレータにより最大で 1.92 倍の性能が得られる可能性があることを示した．

今後の課題としては，HPCG などの前処理付きの CG 法のベンチマークで評価を行うことや，他の精度変換・圧縮パラメータを用いた場合の評価を行うことが考えられる．さらに，コア内部でも変換後の低精度なデータのまま計算することで，計算スループットや電力消費を削減するアーキテクチャの検討も今後の課題である．

参考文献

[1] HPCG Ranking June 2021. https://www.top500.org/lists/hpcg/2021/06/.

[2] K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris, “CSX: an extended compression format for spmv on shared memory systems,” SIGPLAN Not. 46, 8, 2011.

[3] K. Ahmad, H. Sundar, and M. Hall, “Data-driven mixed precision sparse matrix vector multiplication for GPUs,” ACM Transactions on Architecture and Code Optimization (TACO), 16(4), pp.1-24, 2019.

[4] R. Sakamoto, M. Kondo, K. Fujita, T. Ichimura, and K. Nakajima, “The Effectiveness of Low-Precision Floating Arithmetic on Numerical Codes: A Case Study on Power Consumption,” in Proc. HPCAsia2020, pp.199-206, 2020.

[5] F. Vazquez, G. Ortega, J. J. Fernandez, and E. M. Garzon, ‘‘Improving the performance of the sparse matrix vector product with GPUs,’’ in Proc. 10th IEEE ICCIT, ser. CIT, pp. 1146-1151, 2010.

[6] W. T. Tang, W. J. Tan, R. Ray, Y. W. Wong, W. Chan, S. H. Kuo, R. S. M. Goh, S. J. Turner, and W. F. Wong, ‘‘Accelerating sparse matrix-vector multiplication on GPUs using bit-representation optimized schemes,’’ in Proc. ICHPC, Netw., Storage Anal., 2013.

[7] W. Yang, K. Li, Z. Mo, and K. Li, ‘‘Performance optimization using partitioned SpMV on GPUs and multicore CPUs,’’ IEEE Trans. Comput., vol. 64, no. 9, pp. 2623-2636, 2015.

[8] A. Ashari, N. Sedaghati, J. Eisenlohr, and P. Sadayappan, “An Efficient Two-Dimensional Blocking Strategy for Sparse Matrix-vector Multiplication on GPUs,” In Proceedings of the 28th ACM International Conference on Supercomputing, ICS ’14, pages 273-282, 2014.

[9] P. Grigoras, P. Burovskiy, E. Hung, and W. Luk, ‘‘Accelerating SpMV on FPGAs by Compressing Nonzero Values,’’ In 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, pp. 64-67, 2015.

[10] W. Liu and B. Vinter, “CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication,” In Proceedings of the 29th ACM on International Conference on Supercomputing (ICS ’15), 339-350, 2015.

[11] B. Bian, J. Huang, R. Dong, L. Liu, and X. Wang, ‘‘CSR2: A New Format for SIMD-accelerated SpMV,’’ In CCGRID. 350-359, 2020.

[12] NAS Parallel Benchmarks, RNR-94-007. https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf

[13] FUJITSU Processor A64FX Datasheet. https://www.fujitsu.com/downloads/SUPER/a64fx/a64fx_datasheet_en.pdf

[14] T. A. Davis and Y. Hu, “The University of Florida Sparse Matrix Collection. ACM Transactions on Mathematical Software” Vol.38, No.1, Article 1, 25 pages, Dec. 2011.

[15] J. Dongarra, M. A. Heroux, and P. Luszczek, ‘‘HPCG Benchmark: A new metric for ranking high performance computing systems,’’ Knoxville, Tennessee, 2015.