社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Sparse Neural Network における SpMM の並列/ベクトル化による高速化

田處　雄大†　木村　啓二†　笠原　博徳†

† 早稲田大学　基幹理工学部　情報理工学科
E-mail: †y_tado@kasahara.cs.waseda.jp, ††{keiji,kasahara}@waseda.jp

あらまし　Deep Learning（深層学習）におけるモデル圧縮手法の一つとしてプルーニングが知られている．プルーニングにより重要度の低い重みを削除することにより，高い認識精度を維持しつつモデルのサイズを削減することができる．またその結果，重み行列は疎行列として表現されることになる．しかし，プルーニングによって得られる疎行列は科学技術計算などに用いられる疎行列と異なり，ランダム性の高いものとなっており，非零要素の局所性を活かした高速化は困難である．本稿では，ランダム性の高い疎行列を対象とした SpMM(疎行列密行列積) の高速化手法を報告する．本提案手法を ResNet50 に対して適用し，NEC SX-Aurora TSUBASA 上で評価を行った．ベンダ提供の BLAS ライブラリ使用時に対して提案手法を適用した層では 1 コアで最大 2.78 倍の速度向上，モデル全体では 8 コアで 1.98 倍の速度向上がそれぞれ得られた．

キーワード　SpMM(疎行列密行列積)，Sparse Neural Network，ベクトルプロセッサ

Parallelization and Vectorization of SpMM for Sparse Neural Network

Yuta TADOKORO†, Keiji KIMURA†, and Hironori KASAHARA†

† Department of Computer Science and Engineering, Waseda University
E-mail: †y_tado@kasahara.cs.waseda.jp, ††{keiji,kasahara}@waseda.jp

Abstract　Pruning is one of the well-known model compression techniques in Deep Learning. Eliminating less important weights in the model provides a smaller model size than the original one while keeping high accuracy. As a result of the pruning, the weight matrices are represented as sparse matrices. However, the sparse matrices obtained by pruning are highly randomized, unlike the sparse matrices used in scientific applications. Thus it is difficult to employ acceleration techniques for them relying on the locality of non-zero elements. This paper proposes a method to accelerate SpMM (Sparse Matrix - Dense Matrix Multiplication) for sparse matrices with high randomness. The proposed method is applied to ResNet50 and evaluated on NEC SX-Aurora TSUBASA. The speed-ups were 2.78 times with one processor core for the layer to which the proposed method was used and 1.98 times with eight processor cores for the whole model.

Key words　SpMM, Sparse Neural Network, Vector Processor

1. はじめに

Deep Learning（深層学習）は画像認識や自然言語処理といった様々なタスクにおいて，驚くべき速さで発展を遂げ目覚ましい成果を挙げている．自然言語処理では BERT [1] に代表される Transformer モデルが，また画像認識では CNN といったモデルがそれぞれ使われている．しかし，これらのモデルは高い性能と引き換えに，大量のパラメータを持つためメモリの使用量が多く，実行時間もかかってしまう．自然言語処理のモデルである GPT-3 [2] では 1750 億ものパラーメタが存在するなど，巨大なモデルも登場している．そのため，モデルのサイズを小さくするための手法が研究されている．そのなかでも代表的なものがプルーニングや量子化などである．プルーニングはパラメータの中から重要ではない重みをゼロにする手法で，精度をほとんど落とすことなくパラメータを 1/10 にした研究もある [3]．量子化はパラメータをより少ないビット数で表現することでモデルを圧縮する手法である．Han らの研究 [4] では，プルーニング・量子化・ハフマン符号化などを組み合わせることにより，VGG-16 [5] において 1/49 にモデルを圧縮することに成功している．

また，Deep Learning の学習と推論の両方の実行時間短縮に対する要求も依然高く，Deep Learning を高速に実行できる SIMD 型のアクセラレータ（GPU・ベクトルプロセッサ）や SIMD 命令拡張 (Intel AVX) をもつハードウェアが広く使われている．Deep Learning には様々なモデルが存在するが，これらの主要計算はどれも行列積である．そのため，行列積が高速に行える

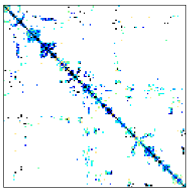

図 1　科学技術計算の疎行列の例 [12]

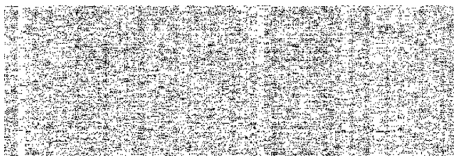

図 2　Sparse Neural Network の疎行列の例

SIMD 型アクセラレータが有効である．

プルーニングは様々な手法が研究されており，それらは Fine-Grain と Coarse-Grain に分けることができる．Fine-Grain なプルーニングには Element-Wise(EW)，Vector-Wise(VW) [6] [7]，Block-Wise(BW) [8]，Tile-Wise(TW) [9] などが存在する．Element-Wise のプルーニングは規則性のないプルーニングを行う方式で高い Sparsity の代わりにイレギュラーなスパースパターンのため前述の SIMD 型アクセラレータによる高速化が困難であるという特徴がある．EW 以外の手法は，規則性のあるプルーニングを行う手法で，密行列計算が得意な SID 型アクセラレータでも高速に実行するための方式で，EW のプルーニングと比較すると低い Sparsity であるが，Coarse Grain なプルーニングよりは高い Sparsity を得ることができる．Coarse-Grain のプルーニングにはチャンネルプルーニング [10] やフィルタープルーニング [11] といった手法がある．これらの手法はチャンネルやフィルター単位で枝刈りしていくもので，fine-grain なプルーニングと比較すると低い Sparsity ではあるが，SIMD 型アクセラレータによる高速化が行いやすい．本稿では，高い Sparsity を得ることができる Element-Wise なプルーニングによって生じる Sparse Neural Network を対象としている．

科学技術計算で用いられる疎行列は何らかの構造を扱うために帯状に非零要素が集中するという特徴を持つことが多い．一方で，Element-Wise なプルーニングによって得られる疎行列はランダム性が高い．科学技術計算の疎行列の例を図 1 に示す．これは Power Network Problem を解く際に現れる疎行列である．Element-Wise なプルーニングによって得られる疎行列の例を図 2 に示す．

本稿では，Sparse Neural Network の高速化のために，主要計算である SpMM（疎行列密行列積）の SIMD アクセラレータのアーキテクチャを意識した高速化とマルチコアによる並列化を行った。さらに，画像認識に用いられる CNN モデルである ResNet50 を用いた評価を行ったので，それらの結果について報告する．

本稿の構成は以下の通りである．まず，第 2. 節で関連研究を紹介する．第 3. 節で評価に用いた SX-Aurora TSUBASA について説明を行う．第 4. 節で疎行列密行列計算の高速化について述べ，第 5. 節で提案手法の評価結果の報告を行い，第 6. 節でまとめる．

2. 関 連 研 究

2.1 SpMM の高速化

疎行列高速計算ライブラリとして Intel MKL [13] が挙げられる．SBLAS というライブラリで SpMM をサポートしている．しかし，ランダム性の高い疎行列では高い性能が出ない．Hong ら [14] [15] は疎行列を非零要素の集中しているパートと非零要素の集中していないパートに分けることで SpMM の高速化を行う手法を提案している．Kurt ら [16] は SpMM と SDDMM のタイリングによる高速化手法および Matrix Signatures というタイルサイズの決定方法の提案を行っている．

2.2 Sparse Neural Network の高速化

Park ら [17] と大野ら [18] の研究は SparseCNN の高速化で，CNN では一般的な im2col(lowering) という畳み込み演算を行列積に変換する処理をしない Direct Convolution という方式をとっている．この方式は CNN には適用できるが，im2col の処理を持たないモデル (LSTM や BERT) に対しては適用することができない．また，Gale らの研究 [19] は本研究と同じ Sparse Neural Network における SpMM の高速化だが，対象としているハードウェアが GPU という点が異なる．

2.3 科学技術計算の疎行列と Sparse Neural Network の疎行列の比較

定量的な調査として Gale らの大規模な調査がある [19]．この調査では，SuiteSparse Matrix Collection [12] と Sparse Neural Network の重みのデータセットを定量的に比較行なっている．この Sparse Neural Network のデータセット [20] は ResNet-50 [21] と Transformer model から構成されており，49 の異なるモデルから 3012 個の行列が抽出されている．SuiteSparse Matrix Collection からは 2833 個の行列を調査対象としている．この調査によると，Sparse Neural Network には SuiteSparse Matrix Collection と比較して平均で以下の特徴がある．

- Sparsity は 13.4 倍低い (非零要素の数が 13.4 倍多い)
- 1 行あたりに含まれる非零要素の数の平均は 2.3 倍
- 1 行あたりに含まれる非零要素の数のの分散は 1/25

3. SX-Aurora TSUBASA [22]

SX-Aurora TSUBASA は NEC によって開発されたベクトルコンピュータである．SX-Aurora TSUBASA は Vector Host（VH）と Vector Engine（VE）から構成されている．VH は x86 ノードであり，主に OS の処理を行う．VE は NEC のベクトルマルチコアプロセッサであり，アプリケーションの実行を行う．両者は PCIe で接続されている．

3.1 SX-Aurora TSUBASA の実行モデル

SX-Aurora TSUBASA のプログラムは，OS の処理を行う VH 上で起動され，VE 上で実行される．VH 上で VE の実行を制御するソフトウェアとして VEOS が提供されている。VE 上で動いているプログラムがファイルアクセスなどによりシステム

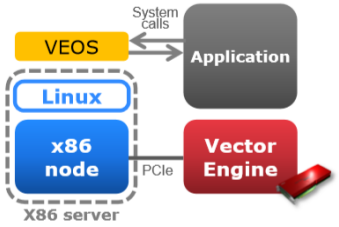

図 3　SX-Aurora TSUBASA の構成 [22]

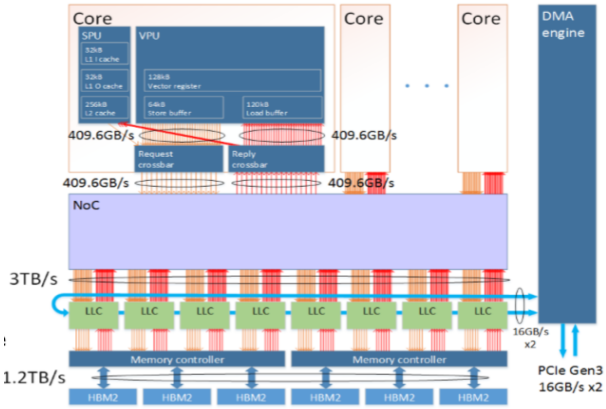

図 4　SX-Aurora TSUBASA
Vector Engine のアーキテクチャ図 [22]

コールを呼び出した際は，それらの処理を VH に委譲するため VH 上の VEOS とコミュニケーションをとる．図 3 に実行モデルを示す．図の X86 Server が VH にあたる．

3.2　Vector Engine のアーキテクチャ

Vector Engine は 8 つのベクターコアを持つプロセッサである．メモリは HBM2 であり，それによって高いバンド幅を実現している．本稿で用いた VE type 10C のプロセッサの理論演算性能（倍精度）は 2.15TFLOPS であり，メモリ帯域は 0.75TB/s である．

それぞれのコアが Scalar Processing Unit（SPU）と Vector Processing Unit（VPU）を持っている．VPU のベクトル長は最大 256 要素である．ベクトルレジスタは 64 本存在する．また，キャッシュはそれぞれの SPU 内に L1・L2 キャッシュを持っており，LLC は 8 つのコアから共有されるキャッシュである．よって，VPU は L1・L2 キャッシュは持たず，キャッシュは LLC だけである．1 つのプロセッサは 8 つの LLC スライスを持ち，一つの LLC スライスは 2MB である．また，ウェイ数は 4way で，ラインサイズは 128B である．SX-Aurora TSUBASA の VE のアーキテクチャ図を図 4 に示す。

3.3　NEC コンパイラ

NEC が提供する SX-Aurora TSUBASA 用コンパイラは Fortran/C/C++ に対応しており，自動ベクトル化・自動並列化の機能を有している．Vector Engine 上で動かすことのできるバイナリを出力する．

3.4　NEC Numeric Library Collection

NEC Numeric Library Collection [23] は、SX-Aurora の Vector Engine 上で利用できる高速な科学技術計算のライブラリである．密行列計算ライブラリである CBLAS や疎行列計算ライブラリである SBLAS などが存在する．CBLAS は Level3 の行列-行列演算までサポートしている．一方，SBLAS の方は Level2 の行列-ベクトル演算までのサポートとなっている．

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 3 \\ 4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \end{bmatrix} \quad \begin{aligned} \text{val} \quad & [1, 2, 3, 4, 5] \\ \text{colidx} \quad & [2, 1, 3, 0, 1] \\ \text{rowptr} \quad & [0, 1, 3, 4, 5] \end{aligned}$$

図 5　CSR 形式の例

```
for(i = 0; i < M ; i++){
  for(j = row_ptr[i]; j < row_ptr[i+1] ; j++){
    for(k = 0; k < K; k++){
      output[i][k] += val[j] * input[col_idx[j]][k];
    }
  }
}
```

図 6　CSR 形式でのナイーブな SpMM の実装

4.　ベクトルアクセラレータ・SIMD 命令を意識した疎行列密行列積 (SpMM)

SX-Aurora TSUBASA の VPU は L1，L2 キャッシュを持たず直接 LLC からデータを持ってくる．この LLC は 16MB なので，本稿で評価する行列積はキャッシュに乗り切ってしまうため，ベクトルレジスタによるタイリングを行った．一方，Intel Xeon は各コアが L1，L2 キャッシュを持つため，通常のキャッシュに対するタイリングに加え，ベクトルレジスタによるタイリングを行った．

4.1　CSR 形 式

疎行列は零要素が多いため，密行列のように 2 次元配列でデータを持っていると無駄が多い．そのため，特殊なデータ構造が用いられる．最も一般的な疎行列の形式は CSR 形式であり，大抵の疎行列計算ライブラリでは CSR 形式はサポートされている．CSR 形式の例を図 5 に示す．この形式では疎行列を val，colidx，rowptr という 3 つの配列を用いて表現される．val は行列内の非零要素を保持した配列で，配列の長さは非零要素数である．colidx は val の各要素と対応していて，非零要素の列の位置を保持している配列で，配列の長さは非零要素数である．rowptr は val の何要素目で行が変わるかを保持した配列で，配列の長さは (行の数 +1) である．CSR 形式でのナイーブな SpMM のコード例を図 6 に示す．

4.2　SX-Aurora 上での SpMM の高速化

図 6 のような実装を最内側ループでベクトル化した際に，output 行列に対する Vector Load/Store が最も多くなる．そのため，本提案手法は output 行列の一部をベクトルレジスタに持ち続けることにより高速化を行う．SX-Aurora では指示節 vreg を用いることで条件を満たした配列をベクトルレジスタに割り当てることができる．

```
for(i = 0; i < M ; i++){
  for(k = 0; k < K ; k += Tk){
    reg = vector_load(&output[i][k]);
    for(j = row_ptr[i]; j < row_ptr[i+1] ; j++){
      int idx = col_idx[j];
      int value = val[j];
      input = vector_load(&input[idx][k]);
      reg = vector_fma(value, input, reg);
    }
    vector_store(&output[i][k], reg)
  }
}
```

図 7　CSR 形式でのレジスタタイリングの疑似コード

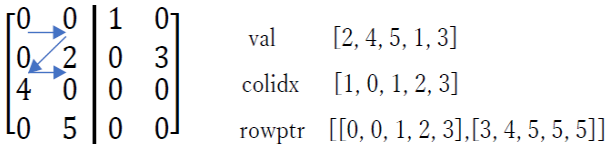

図 8　拡張した CSR 形式の例

提案手法の疑似コードを図 7 に示す．ベクトル化を効率よく行うために k ループのストライド幅 Tk はベクトル長の倍数が望ましい．また，ベクトルレジスタにロードした値を使い回すために，j ループの回転数が大きいほど速度向上が見込める．また，並列化を考えた場合，最外側ループに i ループか k ループを持っていくことが考えられるが，SX-Aurora のような長いベクトル長を持つアーキテクチャの場合，k ループは十分な回転数を得られない可能性がある。そのため，ここでは i ループを外側にインターチェンジした．

4.3　Intel Xeon 上での SpMM の高速化

Intel プロセッサは SIMD 命令拡張である AVX を持っている．AVX512 で 512bit 長のベクトルレジスタを持ち，float 型の場合 16 要素格納できる．これは，SX-Aurora のような長いベクトル長ではないが，有効に活用すれば性能向上が見込める．Intel Xeon に対しては SX-Aurora に適用した出力行列の一部をベクトルレジスタに保持し続けるレジスタタイリングに加えて，キャッシュでのタイリングを行った．

タイリングを行うために CSR 形式の拡張を行った．拡張した CSR 形式の例を図 8 に示す．通常の CSR 形式を比較した際に，row_ptr 配列が 2 次元配列となり，サイズが大きくなる．しかし，基本的に CSR 形式の配列のうち val 配列と col_idx 配列が主にメモリを使用するため，拡張した CSR 形式における row_ptr 配列のオーバーヘッドは問題にならない．AVX512 のイントリンシック関数を用いた，拡張した CSR 形式での提案手法のコードイメージを図 9 に示す．

```
for(kk = 0; kk < K ; kk += 32){
  for(jj = 0; jj < N  T_j ; jj++){
    for(i = 0; i < M ; i++){
      int nnz_begin = row_ptr[jj][i];
      int nnz_end = row_ptr[jj][i+1];
      __m512 reg1 = _mm512_load_ps(&output[i][kk]);
      __m512 reg2 = _mm512_load_ps(&output[i][kk+16]);
      for(j = nnz_begin ; j < nnz_end ; j++){
        int idx = col_idx[j];
        __m512 value = _mm512_set1_ps(val[j]);
        __m512 input1 = _mm512_load_ps(&input[idx][kk]);
        __m512 input2 = _mm512_load_ps(&input[idx][kk+16]);
        reg1 = _mm512_fmadd_ps(value, input1, reg1);
        reg2 = _mm512_fmadd_ps(value, input2, reg2;
      }
      _mm512_store_ps(&output[i][kk],reg1);
      _mm512_store_ps(&output[i][kk+16],reg2);
    }
  }
}
```

図 9　CSR 形式でのタイリングのコードイメージ

5.　評 価 結 果

5.1　評 価 環 境

5.1.1　SX-Aurora TSUBASA

提案手法は第 3. 節で述べた SX-Aurora TSUBASA 上で評価を行った．使用した VE のモデルは Type 10C である．VE Type 10C の諸元を表 1 に示す。Type 10C の動作周波数は 1.4GHz で，8 つのベクトルプロセッサコアを持ち，LLC のサイズは 16MB である．また，使用したコンパイラのバージョンは 3.0.7 である．評価対象として用いた，NEC Numeric Library Collection のバージョンは 2.1.0 である．

表 1　Vector Engine Type 10C の諸元

Vector Core	8 cores
動作周波数	1.4 GHz
L1 I Cache	32KB
L1 D Cache	32KB
L2 Cache	256KB
LLC	16MB (プロセッサあたり)

表 2　Intel Xeon W-2145 の諸元

Core	8 cores
動作周波数	3.7 GHz
L1 I Cache	32KB （コアあたり）
L1 D Cache	32KB （コアあたり）
L2 Cache	1024KB （コアあたり）
L3 Cache	11MB （プロセッサあたり）

5.1.2　Intel Xeon

評価に用いた Intel Xeon W-2145 の諸元を表 2 に示す．このマシンは SIMD 命令拡張は AVX512 までサポートされている．Intel C コンパイラ（Parallel Studio XE 2018）を用いてコンパイルを行い，提案手法およびナイーブな CSR の実装に対するコンパイルオプションは-O3 -xCORE-AVX512 -qopt-zmm-usage=high である．提案手法はイントリンシック関数を用いてベクトル化を行った．

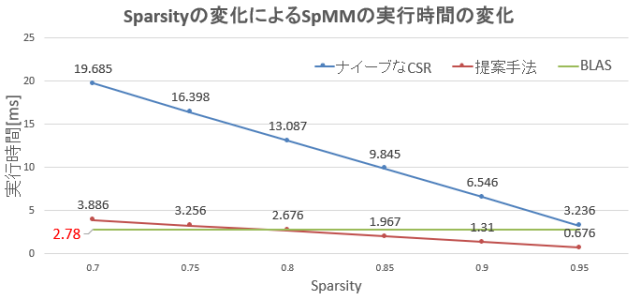

図 10　SpMM の評価結果 (1 コア)

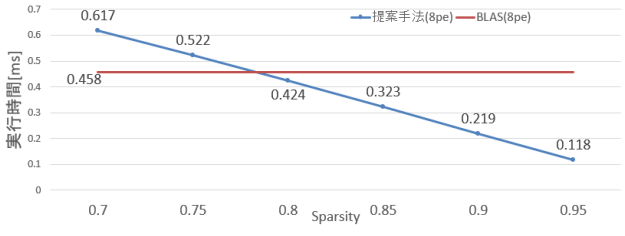

図 11　SX-Aurora 上での SpMM の評価結果 (8 コア)

5.2　SpMM の評価結果

Sparsity の変化による SpMM の実行時間の変化を調査した．SpMM は 512 × 1024 の疎行列と 1024 × 1024 の密行列の積で，データは float 型，疎行列の形式は CSR である．疎行列は分布がランダムになるように生成した行列を評価に用いた．

5.2.1　SX-Aurora 上の評価

提案手法について，1 コアでの図 6 のナイーブな CSR の実装と NEC Numeric Library Collection の BLAS による密行列積との比較を行った．また，8 コアでの提案手法の手動並列化と BLAS の並列実行の比較を行った．1 コアでの評価結果を図 10 に，8 コアでの評価結果を図 11 に示す．1 コアでの評価では，提案手法はナイーブな CSR に対して 5 倍の速度向上，BLAS の性能を Sparsity が 0.8 以上のときに上回った．8 コアでの評価でも同様に，提案手法は BLAS の性能を Sparsity が 0.8 以上のときに上回った．

5.2.2　Intel Xeon 上の評価

提案手法について，1 コアでの図 6 のナイーブな CSR の実装と MKL の BLAS による密行列積と，MKL の SBLAS による疎行列密行列積の比較を行った．SBLAS では，疎行列の解析を行うフェーズは実行時間に含まない．1 コアでの評価結果を図 12 に示す．1 コアでの評価では，提案手法は Sparsity が 0.9 のとき，ナイーブな CSR に対して 2.77 倍，BLAS の性能に対して 1.80 倍，SBLAS の性能に対して 2,56 倍の速度向上が得られた．また，提案手法は BLAS による実行に対して，Sparsity は 0.75 の時点で上回った．

5.3　ResNet50 での評価結果

プルーニングされたモデルのパラメータは GitHub 上で公開されている SkimCaffe [24] から抽出した．ResNet50 [21] は 50 層からなる CNN で，利用した重みでは 16 個の Convolution 層に対してプルーニングが行われている．各層の Sparsity を表 3 に示す．ここでは，推論にかかった時間の測定を行った．提案

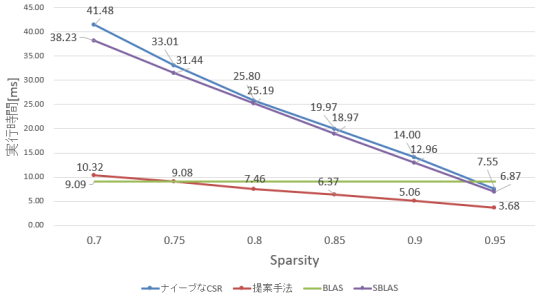

図 12　Intel Xeon 上での SpMM の評価結果 (1 コア)

表 3　ResNet50 の各層の Sparsity

layer	Sparsity[%]	重み行列のサイズ
res2a_branch2b	88.0046	64 × 576
res2b_branch2b	83.9708	64 × 576
res2c_branch2b	80.7943	64 × 576
res3a_branch2b	91.7562	128 × 1152
res3b_branch2b	91.9101	128 × 1152
res3c_branch2b	95.5024	128 × 1152
res3d_branch2b	91.1343	128 × 1152
res4a_branch2b	96.5534	256 × 2304
res4b_branch2b	97.4677	256 × 2304
res4c_branch2b	97.1471	256 × 2304
res4d_branch2b	96.7648	256 × 2304
res4e_branch2b	96.8435	256 × 2304
res4f_branch2b	96.8374	256 × 2304
res5a_branch2b	97.7809	512 × 4608
res5b_branch2b	98.0271	512 × 4608
res5c_branch2b	97.9203	512 × 4608

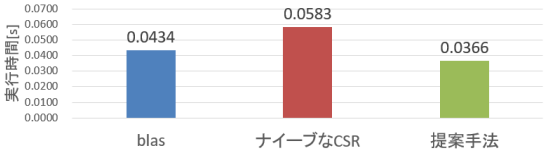

図 13　モデル全体での実行時間 (1 コア)

手法やナイーブな CSR の評価の際に，プルーニングのされていない Convolution 層での行列積には BLAS を使用した．1 コアでの評価結果を図 13 に示す．また，1 コア時の提案手法を適用した層の実行時間の比較を図 14 に示す．そして，並列化を行った提案手法を適用し，それ以外の箇所には NEC コンパイラによる並列化を適用したものと，BLAS の並列実行と NEC コンパイラによる並列化を組み合わせたものの比較を行った．10 回の測定を行った最速値をグラフにまとめたものを図 15 に示す．

1 コアでの実行時，モデル全体での評価は BLAS での実装と比較して 1.19 倍の速度向上，ナイーブな CSR に対して 1.59 倍の速度向上が得られた．各層での実行時間では BLAS に対して 0.55 倍から 2.78 倍の速度向上，ナイーブな CSR に対しては 1.00 倍から 4.72 倍の速度向上が得られた．BLAS での実行に対して提案手法では速度が鈍化している層がいくつか存在する．鈍化の原因は Sparsity が低いのも要因の一つではあるが疎行列のサイズが小さいことが挙げられる．疎行列の 1 行あたりの非

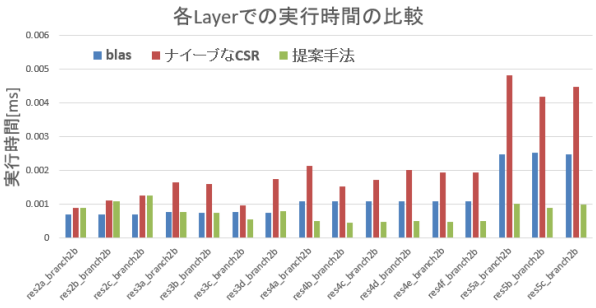

図 14　各層での実行時間の比較 (1 コア)

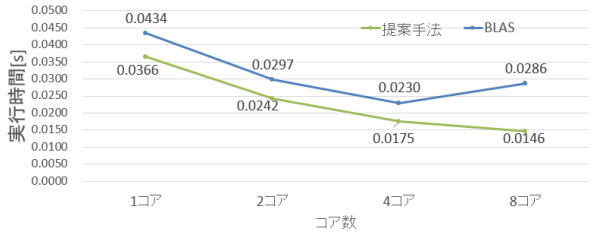

図 15　モデル全体の評価 (並列化)

ゼロ要素が少ない場合，提案手法のレジスタでタイリングしても，再利用できる回数が少ない．そのため，提案手法が BLAS に対して鈍化している層では，ナイーブな CSR での実装と実行時間がほとんど変わらない結果となっている．また，鈍化している層は重み行列のサイズが小さいので，提案手法を適用せずに BLAS を適用した場合でも，モデルの圧縮に大きな影響は与えない．並列化では，8 コアの最速値で 1 コアの最速値と比較して 2.51 倍の速度向上が得られた．これは 1 コアでのナイーブな CSR の実装と比較すると 4.00 倍の速度向上である．また，BLAS の並列実行と最速値同士で比較すると，1.19 倍（1 コア使用時）から 1.96 倍（8 コア使用時）の速度向上が得られた．

6. ま と め

本稿では，Sparse Neural Network に現れるランダム性の高い疎行列を用いた SpMM(疎行列密行列) の高速化手法を提案した．本提案手法はベクトル演算機構を持つアーキテクチャを対象としており，ベクトルレジスタでタイリングを行うことで速度を向上させる手法である．本手法を，疎行列の Sparsity を変化させて SpMM(疎行列密行列積) での評価を行ったところ，Sparsity が 0.8 以上で提案手法がベンダ提供の BLAS ライブラリを上回った．また，SX-Aurora TSUBASA 上で ResNet50 に対して提案手法を適用し評価を行った．図 15 より，ベンダ提供の BLAS ライブラリ使用時に対して提案手法を適用した層では 1 コアで最大 2.78 倍の速度向上，モデル全体では 8 コアで 1.98 倍の速度向上がそれぞれ得られたことが確認できる．

文　献

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019.

[2] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020.

[3] M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” 2017.

[4] S. Han, H. Mao, and W.J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” arXiv preprint arXiv:1510.00149, pp.••–••, 2015.

[5] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[6] Z. Yao, S. Cao, W. Xiao, C. Zhang, and L. Nie, “Balanced sparsity for efficient dnn inference on gpu,” Proceedings of the AAAI Conference on Artificial Intelligence, vol.33, pp.5676–5683, 2019.

[7] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, “Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus,” Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pp.359–371, 2019.

[8] S. Narang, E. Undersander, and G. Diamos, “Block-sparse recurrent neural networks,” 2017.

[9] C. Guo, B.Y. Hsueh, J. Leng, Y. Qiu, Y. Guan, Z. Wang, X. Jia, X. Li, M. Guo, and Y. Zhu, “Accelerating sparse dnn models without hardware-support via tile-wise sparsity,” Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp.1–15, 2020.

[10] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” Proceedings of the IEEE International Conference on Computer Vision, pp.1389–1397, 2017.

[11] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H.P. Graf, “Pruning filters for efficient convnets,” 2017.

[12] “SuiteSparse Matrix Collection,” https://sparse.tamu.edu.

[13] “Intel Math Kernel Library,” https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html.

[14] C. Hong, A. Sukumaran-Rajam, B. Bandyopadhyay, J. Kim, S.E. Kurt, I. Nisa, S. Sabhlok, Ü.V. Çatalyürek, S. Parthasarathy, and P. Sadayappan, “Efficient sparse-matrix multi-vector product on gpus,” Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, pp.66–79, 2018.

[15] C. Hong, A. Sukumaran-Rajam, I. Nisa, K. Singh, and P. Sadayappan, “Adaptive sparse tiling for sparse matrix multiplication,” Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, pp.300–314, 2019.

[16] S. Kurt, A. Sukumaran-Rajam, F. Rastello, and P. Sadayappan, “Efficient tiled sparse matrix multiplication through matrix signatures,” 2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)IEEE Computer Society, pp.1234–1247 2020.

[17] J. Park, S. Li, W. Wen, P.T.P. Tang, H. Li, Y. Chen, and P. Dubey, “Faster cnns with direct sparse convolutions and guided pruning,” 2017.

[18] 大野善之，石坂一久他，“ベクトルプロセッサを用いた direct sparse convolution の最適化,” 研究報告システム・アーキテクチャ (ARC)，vol.2017，no.16，pp.1–7，2017.

[19] T. Gale, M. Zaharia, C. Young, and E. Elsen, “Sparse gpu kernels for deep learning” 2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)IEEE Computer Society, pp.219–232 ••, .

[20] T. Gale, E. Elsen, and S. Hooker, “The state of sparsity in deep neural networks,” 2019.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[22] Y. Yamada and S. Momose, “Vector engine processor of nec’s brand-new supercomputer sx-aurora tsubasa,” Proceedings of A Symposium on High Performance Chips, Hot Chips, vol.30, pp.19–21, 2018.

[23] “NEC Numeric Library Collection,” https://www.hpc.nec/documents/sdk/SDK_NLC/UsersGuide/blas/c/ja.

[24] “SkimCaffe,” https://github.com/IntelLabs/SkimCaffe.