Vol.2019-HPC-168 No.19
2019/3/7

小疎行列積計算のGPU最適化

長坂 侑亮[1,a)]　額田 彰[1]　小島 諒介[2]　松岡 聡[3,1]

概要：バイオインフォマティクス等における深層学習的手法の適用として，高い認識精度を得ることが可能である Graph Convolutional Networks (GCNs) が近年注目を集めている．グラフ構造を持つデータに対する畳込み演算が可能である GCNs の処理では，疎行列積計算（SpMM）を含む膨大な演算を処理するために GPU が用いられている．しかしながら，GCNs で扱われるデータのグラフ構造にはノード数が数十程度の小さいものが含まれており，小行列に対する疎行列積計算は GPU の並列性の活用が困難であるために，疎行列積計算が GCN の学習や推論の処理におけるボトルネックとなっている．GCNs アプリケーションの処理性能向上のために，特に小疎行列に対して効果的な新たな疎行列積計算カーネルである Sub-Warp-Assigned (SWA) SpMM と，複数の疎行列積計算を一つのカーネルで行うことによって GPU の高い並列性と演算能力を活用可能にする Batched SpMM を提案する．NVIDIA Tesla P100 GPU を搭載する TSUBAME3.0 にて評価実験を行い，Non-batched の手法から最大 9.27 倍の性能向上を達成した．

1. はじめに

近年，画像認識などにおいて高い認識精度を達成することを可能とする深層学習の研究開発が盛んに行われている．画像認識においては，画像の畳込みやプーリングを行う層を重ねた Convolutional Neural Network (CNN) によって，特徴抽出が行われている．一方で，画像などの規則正しい格子構造を持つデータだけでなく，ノードとエッジによって関係性が表されるグラフ構造を入力として扱うことが可能な Graph Convolutional Networks (GCNs) が脚光を浴び始めている [1]．化合物やタンパク質の特性を推定するバイオアッセイにおいて，GCNs では物質の持つ構造をグラフとして扱うことによって高い認識精度と高速な推論性能を実現している [2–6]．また，知識をグラフ構造として表した知識グラフに対する GCNs の適用も行われている [7]．

通常の CNNs の処理と同様に，GCNs の処理に要する演算量は膨大であり，高い演算能力を持つ GPU の活用が重要となる．また，GCNs では入力データのグラフ構造を考慮する必要があり，グラフ構造は隣接リストや隣接行列として表現される．グラフの各要素間の結合は多くの場合において疎であり，隣接行列で表現される場合には疎行列となる．高速な GCNs の学習や推論の実現においては，疎行列に関する計算，特に疎行列積計算（Sparse-Dense Matrix Multiplication, SpMM）の最適化も重要となる．しかしながら，GCNs で扱われるグラフにはしばしばノード数が数十程度の小さいものが含まれており，GPU の高い並列性の活用が困難となっている．結果として，疎行列計算に関する箇所が GCNs の性能のボトルネックとなっている．

本論文では，小疎行列積計算の高速化を目的とした新たな SpMM アルゴリズムと大量の疎行列積計算を効率的に処理するための Batched 手法を提案する．はじめに CSR や TensorFlow における SparseTensor などのデータ構造に対する新たな SpMM アルゴリズムとして Sub-Warp-Assigned (SWA) SpMM を提案し，更に SWA SpMM において GPU のシェアードメモリを活用するためのキャッシュブロッキング最適化手法を示す．次に，GPU の高い並列性と演算能力を活用可能とする Batched SpMM を提案する．Batched SpMM は，キャッシュブロッキングを適用するか，スレッドやシェアードメモリを各 SpMM にどれだけ割り当てるのかを，バッチ内の行列のサイズに基づいて適切に判断する．GCNs アプリケーションにおけるミニバッチサイズ相当の繰り返し実行される SpMM 演算を一回の CUDA カーネル呼び出しで並列処理することによって，高い並列性の達成とカーネル起動コストの削減を実現可能とする．

NVIDIA Tesla P100 GPU が搭載された TSUBAME3.0 を用いて，Batched SpMM の性能評価を行った結果，従来の Non-batched 手法から最大 9.27 倍，サイズの大きい入

[1] 東京工業大学
Tokyo Institute of Technology
[2] 京都大学
Kyoto University
[3] (国立研究開発法人) 理化学研究所　計算科学研究センター
RIKEN Center for Computational Science (R-CCS)
[a)] nagasaka.y.aa@m.titech.ac.jp

力密行列に対しても 6.09 倍の性能向上を達成した．また，入力疎行列を密行列として扱った場合における cuBLAS の Batched GEMM の評価も行っており，我々の Batched SpMM は Batched GEMM に対しても優位な性能を示した．また，バッチサイズや行列サイズなどのパラメータを変更した際の Batched 手法の性能変化の傾向を明らかにした．

2. 背景

2.1 Graph Convolution

Graph convolution ではグラフ構造を持つデータに対する畳込み操作が行われる．入力データとしてグラフ構造と特徴量が与えられ，対象とするノードの隣接ノードに対してフィルタを掛け，足し合わせるという操作を行う．グラフを $G = \{V, E\}$，グラフ上のノード $v \in V$ における特徴量を x_v，フィルタの集合を行列で表し W としたとき，以下のように定式化される．

$$y_i = \sum_{j \in V} a_{ij} x_j^T W \tag{1}$$

$a_{uu} = 1$ とした上で，u から v に向かうエッジがある場合には，$a_{vu} = 1$，なければ 0 となる．各ノードごとの処理をグラフ全体として行うとした場合には

$$Y = AXW \tag{2}$$

となり，疎な特性を持つ隣接行列 A と密行列として表される特徴ベクトル列やフィルタの集合である重み行列との積演算となる．

2.2 疎行列フォーマット

行列内の要素の多くが行列の計算において影響を与えないゼロ要素であるような疎行列の場合には，行列内のゼロ要素を削除し，処理に必要となる非ゼロ要素に関する情報のみを保管するように圧縮を行う．圧縮によって疎行列に関する処理の計算量とメモリ使用量を削減することが基本的な目的であり，処理に合わせて様々なフォーマットが提案されている．疎行列フォーマットとして広く用いられているものとして Coordinated（COO）と Compressed Sparse Row（CSR）フォーマットがある．図 1 に各疎行列フォーマットの例を示す．COO は行列の各非ゼロ要素に関して，値，行インデックス，列インデックスを組として保持する．CSR では同じ行インデックスを持つ非ゼロ要素をまとめ，各行の開始インデックス row pointer（配列 rpt）を保持した上で，各非ゼロ要素の列インデックスと値を保持する．CSR では COO と比較してメモリ使用量が削減される．また，深層学習フレームワークである TensorFlow [8] では，SparseTensor として疎テンソルは扱われる．図 1 が示すように，SparseTensor は COO と類似

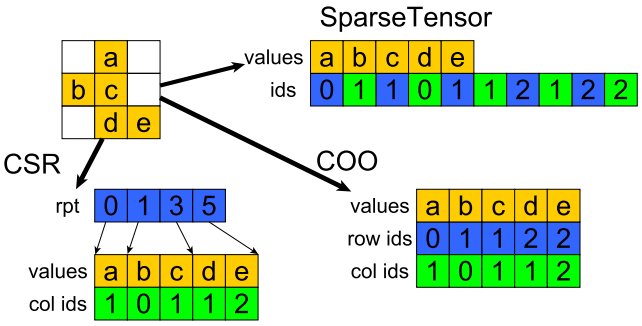


図 1: 疎行列フォーマットの例

SparseTensorDenseMatMul(C, A, B)

```
1  // set matrix C to O
2  for i ← 0 to nnz_A
3    do for j ← 0 to n_B
4          do rid ← ids_A[i * 2]
5             cid ← ids_A[i * 2 + 1]
6             val ← values_A[i]
7             C[rid][j] ← C[rid][j] + val * B[cid][j]
```

図 2: TensorFlow における疎テンソルと密行列の積計算の擬似コード．A は SparseTensor データ構造として保管されている．

したデータ構造であり，各非ゼロ要素のインデックスは行と列のインデックスの組の配列として扱われる．

2.3 疎行列積計算

A, B を入力行列，ただし A は疎行列，B は密行列であるとした場合，疎行列積計算では $C = AB$ を求める．本論文ではこれ以降，行列 X の非ゼロ要素数を nnz_X，行数を m_X，列数を n_X と表記する．図 2 に TensorFlow において SpMM 計算を行うことが可能な SparseTensorDenseMatMul の擬似コードを示す．SparseTensorDenseMatMul では各非ゼロ要素について順次計算が行われ，CUDA でのコードの場合には，2 つある “**for**” ループを並列化し $nnz_A * n_B$ のスレッドを起動することで各スレッドごとに独立して一回の積和計算を行う．最終的に出力行列への足し合わせを行う際には，各スレッドごとのメモリアクセスについて競合が発生する可能性があるため，atomic に加算処理を行っている．Atomic 処理による影響が十分に小さければ，各スレッドの処理量は等しく，高い並列性を持つ GPU においても良いロードバランスを達成することが可能となる．しかしながら，本手法は行列同士の積計算でありながら局所性をほとんど活用できていないことに加え，グローバルメモリ上での atomic 処理によって性能が律速している．また，スレッド数は $nnz_A * n_B$ となるため，小行列の場合には GPU の高い並列性の活用が困難となる．

3. 関連研究

3.1 SpMM for GPU

Vázquez らによって，ELLR-T フォーマットを用いた GPU 向け SpMM 計算の高速化手法が提案された [9]．GPU のメモリアクセスに適した ELL 系の疎行列フォーマットを用いており，高い SpMM 性能を達成している．しかしながら，COO や CSR などのフォーマットからの変換が必要となり，各疎行列について一度しか計算を行わないような処理においては，フォーマット変換のコストが性能低下の要因となりうる．

Hong らによって，GPU での SpMM 計算向けの新たなフォーマットである RS-SpMM (Row-Segmented SpMM) が提案された [10]．疎行列を非ゼロ要素が偏っている箇所とそれ以外とに分け，それぞれを別々のカーネルで処理する．これによって，非ゼロ要素の偏っている密な箇所に対応するデータの再利用性が向上し，GPU のグローバルメモリへのアクセスを削減することに成功している．また，偏っている箇所を判定するために性能モデルを導入しており，最適な場合と比較しても十分に高い性能を達成している．フォーマットの変換にはパラメータの設定と DCSR フォーマット [11] への変換が含まれるが，SpMM 計算数回分の変換コストを伴うものである．

Yang らによって，CSR での高速な GPU 向け SpMM 手法が提案された [12]．SpMV の手法として提案されている Merge-based [13] をロードバランス改善のために SpMM に取り入れた手法と，coalesced なメモリアクセスを実現するための Row-splitting 手法の 2 つを提案しており，Heuristic を用いることで二つのカーネルを行列によって適切に使い分けている．なお，評価対象としている行列のサイズは大きく，小行列において有効かは明らかではない．

3.2 Batched BLAS

密行列を小ブロックに分割して行われる計算 [14, 15] やブロック構造を持つ疎行列に関する計算 [16–18] においては，各ブロックに関する計算は一つの密行列計算として扱われ，大量の小密行列計算を処理する必要がある．GPU にて小行列を扱う場合，高い並列性やメモリバンド幅を活用することが困難であることに加え，カーネルの起動オーバーヘッドが全体の性能を低下させるという問題がある．これに対して新たな計算ルーチンとして，ひとまとまりのデータ集合をまとめて処理する Batched BLAS が提案された [19–21]．バッチ内で処理するデータのサイズが異なる場合においても，発生する処理の負荷不均衡の解消を図れている．また，小行列での GEMM 計算を効率よく行うための手法も提案されており [22]，大量の小行列計算を高いスループットをもって行うことが可能となった．また，小疎行列を対象とした疎行列ベクトル積計算である Batched SpMV も提案されている [23]．Batched SpMV では全ての行列が共通の非ゼロ配置を有する等の仮定を含んでおり，極めてアプリ特化なものであると言える．そのため，GPU における Batched SpMV でのバッチ化手法は単純にバッチ方向の並列度を追加しただけであり，バッチ内の疎行列のサイズや非ゼロ配置が異なる際に発生しうる負荷不均衡の問題が考慮されていない．

4. 提案

グラフ構造を疎行列として保管した場合，グラフ構造に則った畳み込みを行う際には SpMM 計算が行われる．GCNs の学習では Graph convolution 層を通じて大量のデータを処理するため，SpMM 計算が繰り返し行われる．しかしながら，疎行列の行数（列数）が数十から数百など非常に小さい場合には GPU の高い演算性能を活用することが困難となる．さらに，各 SpMM 計算毎に起動される CUDA カーネルの起動オーバヘッドが性能に対して顕著になる．これらの問題に対して，小疎行列に対する大量の SpMM 計算のスループットを向上させる Batched SpMM を提案する．はじめに，CSR や TensorFlow における SparseTensor 等のデータ構造を対象とした新たな SpMM アルゴリズムである Sub-Warp-Assigned (SWA) SpMM を提案する．次に，SWA SpMM におけるシェアードメモリを活用するためのキャッシュブロッキング最適化手法を示す．最後に，Batched SpMM におけるバッチ内の SpMM 計算に対するキャッシュブロッキング最適化適用の判断や，スレッド等の計算リソース割当の戦略を示す．なお，全ての疎行列データについてフォーマット変換を行うのは高い変換コストを発生させるため，COO や CSR 等の単純なデータ構造を対象とし，本論文では TensorFlow における SparseTensor と CSR を用いる．なお，SparseTensor では各非ゼロ要素について行や列インデックスに基づいたソートなどは行われていないことを仮定する．SpMM 計算における入力密行列の列数は GCNs アプリケーションにおいてはモデルのサイズを表し，バッチ内の各 SpMM 計算での入力密行列の列数は共通であるとする．

4.1 Sub-Warp-Assigned SpMM

TensorFlow の SparseTensorDenseMatMul カーネルでは $nnz_A * n_B$ のスレッドを起動し，各スレッドが一回の積和計算を行う．つまり，n_B のスレッドが各非ゼロ要素に割り当てられるという形になる．しかしながら，実際の CUDA の実装では，各スレッドがどの非ゼロ要素や列を担当するのかを見つける上で多くの命令を要しており，並列効率を低下させる要因となっている．また，SparseTensorDenseMatMul はメモリアクセスについても性能低下の要因がある．n_B が 32 の倍数でない場合には，密行列へのア

クセスはコアレスアクセスの条件を満たさないという問題があり，n_B が増加した場合，特に 32 を超えた場合には，多くのスレッドが同じ非ゼロ要素へのメモリアクセスを行うため，メモリアクセス要求や命令数が不必要に増加する．本論文では各非ゼロ要素もしくは各行の計算に割り当てるスレッドを最大 32 スレッド，つまり 1 warp に制限した Sub-Warp-Assigned (SWA) SpMM を提案する．SWA SpMM において重要となる $subWarp$ は入力密行列の列サイズ n_B をもとに以下のように決定する．

$$subWarp = \begin{cases} 32 & (n_B > 16) \\ \min 2^p \ s.t.\ n_B \leq 2^p & (n_B \leq 16) \end{cases}$$

$subWarp$ は 2 のべき乗としており，処理中に必要となる除算や剰余計算を低コストなビット計算のみで行うことが可能となる．また，各要素に割り当てるスレッド数の上限を 32 スレッドと制限することによって，SWA SpMM では過剰なメモリアクセス要求や命令を削減している．はじめに SparseTensor データ構造向けの SWA SpMM アルゴリズムを，次に CSR 向けのものを示す．なお，SparseTensor 向けのアルゴリズムは簡易に COO に適用可能である．

図 3 に SparseTensor 向け SWA SpMM の擬似コードを示す．各非ゼロ要素には $subWarp$ のスレッドが割り当てられ，$subWarp$ 内のスレッドによる入力密行列や出力行列へのメモリアクセスは連続となる．SparseTensorDenseMatMul と同様に，SWA SpMM は非ゼロ要素数に基づいて並列化が行われているため，スレッド間の負荷均衡が達成されている．なお，他の非ゼロ要素に割り当てられた $subWarp$ 内のスレッドが同じ出力行列要素へ同時にアクセスする可能性があるため，加算は atomic 処理を用いて行われる．

図 4 に CSR 向け SWA SpMM の擬似コードを示す．非ゼロ要素が行単位で管理されているという CSR の特性を活用し，CSR 向け SWA SpMM では $subWarp$ を各行に割り当てる．SparseTensor 向け SWA SpMM 同様，同一 $subWarp$ 内のスレッドは同じ非ゼロ要素へメモリアクセスを行い，入力密行列や出力行列へのメモリアクセスは連続となる．なお，SparseTensor 向け SWA SpMM とは異なり，CSR 向け SWA SpMM では出力行列へのアクセスに競合が発生しないため，計算された積は atomic 処理を用いずに加算することが可能となる．

SWA_SpMM_ST($C, A, B, subWarp$)

```
1  // A is stored as SparseTensor data structure
2  // set matrix C to O
3  i ← threadid
4  nzid ← i / subWarp
5  rid ← ids_A[nzid * 2]
6  cid ← ids_A[nzid * 2 + 1]
7  val ← values_A[nzid]
8  for j ← (i %subWarp) to n_B by subWarp
9     do Atomic(C[rid][j] ← C[rid][j] + val * B[cid][j])
```

図 3: SparseTensor 向け SWA SpMM の疑似コード

SWA_SpMM_CSR($C, A, B, subWarp$)

```
1  // A is stored as CSR
2  // set matrix C to O
3  i ← threadid
4  rid ← i / subWarp
5  for nzid ← rpt_A[rid] to rpt_A[rid + 1]
6     do cid ← colids_A[nzid]
7        val ← values_A[nzid]
8        for j ← (i %subWarp) to n_B by subWarp
9           do C[rid][j] ← C[rid][j] + val * B[cid][j]
```

図 4: CSR 向け SWA SpMM の疑似コード

4.2 シェアードメモリの活用とキャッシュブロッキング

GPU での行列積計算の性能向上において，シェアードメモリの活用とメモリアクセスの局所性改善は非常に重要である．シェアードメモリは GPU の各 SM に実装されており，高速なメモリアクセスを提供するものである．また，シェアードメモリでは atomic 処理についてハードウェアレベルでのサポートがなされているため，atomic 処理を含む計算の性能向上を期待することが可能となる．SWA SpMM においては出力行列の途中計算結果を保管する目的でシェアードメモリを用いる．

出力行列が十分に小さい場合には，SparseTensor 向け SWA SpMM は図 5-(a) に示すようにシェアードメモリを使用する．図 3 が示すように，出力行列は実際の SpMM 計算を始める前に初期化されている必要があり，特に小行列に対する SpMM 計算においては，初期化用の CUDA カーネル起動のオーバーヘッドは無視できないものとなるが，出力行列に対してシェアードメモリを活用することによって出力行列初期化用 CUDA カーネルの起動コストを削減することが可能となる．一方，n_B もしくは入力疎行列が大きい場合，一つのスレッドブロックが管理するシェアードメモリ上に出力行列全体を載せることが困難となる．この場合には，図 5-(b) に示すように，出力行列を列方向に分割するキャッシュブロッキング最適化を適用する．本キャッシュブロッキング最適化によって，入力密行列に対するメモリアクセスの局所性についても改善されており，大きい入力行列に対してもシェアードメモリを活用した高性能な SpMM 計算を実現することが可能となる．

SparseTensor 向け SWA SpMM では，非ゼロ要素のインデックス情報が無い状態で各スレッドが特定できるのは出力行列の担当する列インデックスのみであったのに対して，CSR 向け SWA SpMM では各スレッドが担当する出力行列要素の行と列のインデックスを容易に特定することが可能である．そのため，シェアードメモリは出力行列全体を

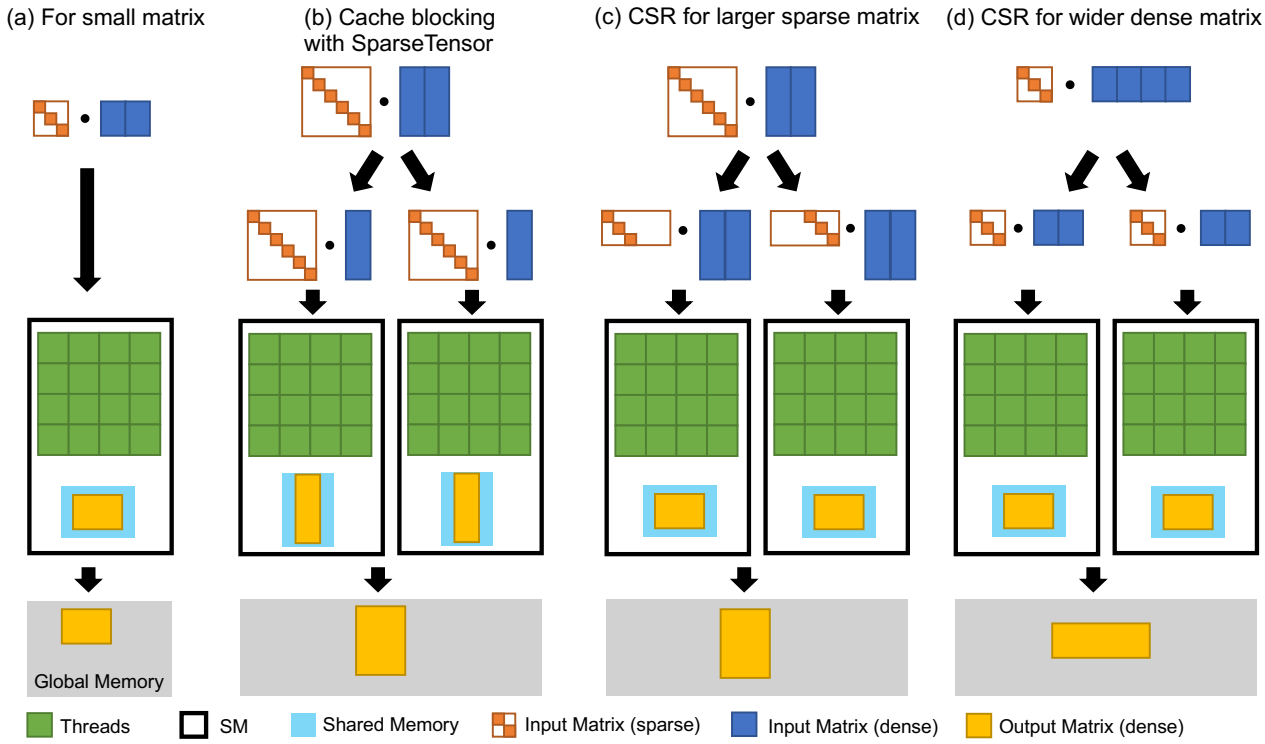


図 5: SWA SpMM でのシェアードメモリの活用とキャッシュブロッキング最適化

保持する必要はなく，各 $subWarp$ ごとに n_B，つまり出力行列の列数分のシェアードメモリ容量のみで十分となる．TB をスレッドブロックサイズとし，$TB/subWarp * n_B$ がシェアードメモリの容量を超えた場合には，図 5-(d) に示すようにキャッシュブロッキング最適化を適用する．SparseTensor 向けのキャッシュブロッキング最適化と同様，出力行列や入力密行列は列方向に分割される．

4.3 SpMM 向け Batched アルゴリズム

Batched SpMM では，数十から数百の SpMM 計算を一つの CUDA カーネルによって行う．Batched SpMM は，入力行列のサイズに基づいてキャッシュブロッキングを行うかの判断と各 SpMM 計算へ割り当てるスレッド数の決定を行う．

SparseTensor 向け Batched SpMM では，バッチ内の出力行列の最大サイズ，つまり $\max_{i \in batch} m_{A_i} * n_B$ に基づいてキャッシュブロッキングの適用を判断する．入力行列のサイズによって以下の 3 つのケースが考えられる．

(1) 単一のスレッドブロックが管理するシェアードメモリ上に出力行列全体が載る (図 5-(a))
(2) キャッシュブロッキング最適化による部分出力行列が単一のスレッドブロックが管理するシェアードメモリ上に載る (Figure 5-(b))
(3) キャッシュブロッキング最適化を適用しても部分出力行列がシェアードメモリに載らない

なお，各 SpMM 計算に割り当てるシェアードメモリの最大を 32KB としたときの単精度 SpMM 計算において，(3) のケースは $m_A > 8192$ となる入力疎行列の場合のみである．十分な並列性のある SpMM 計算に対するバッチ最適化は本論文の対象外であり，Batched SpMM ではシェアードメモリを用いないカーネルを作成することによって (3) についても対応するが，本論文が対象とするのは小疎行列に関する SpMM 計算であるため，(1) と (2) について以下で詳細を述べる．Batched SpMM では各行列もしくは部分行列に関する SpMM 計算に対して，一つのスレッドブロックを割り当てる．キャッシュブロッキング最適化の適用判断について，単一スレッドブロックが管理するシェアードメモリ上に載らない出力行列がバッチ内に一つでもある場合には，バッチ内のすべての SpMM 計算に対して，キャッシュブロッキング最適化を適用する．例えば，単精度にて 100 の SpMM 計算を Batched SpMM が実行する場合を考える．各スレッドブロックに割り当てるシェアードメモリ容量は 32KB とする．バッチ内の任意の出力行列をシェアードメモリに載せることが可能，つまり $m_A * n_B *$ sizeof(float) $\leq 32 * 1024$ の場合，Batched SpMM は 100 のスレッドブロックを起動し，各 SpMM 計算に一つのスレッドブロックを割り当てる．一方，バッチ内のある SpMM 計算のみがキャッシュブロッキング最適化を必要とし，出力行列を二つの部分行列に分割する必要がある場合には，Batched SpMM では 200 のスレッドブロックを起動し，各スレッドブロックを一つの部分行列に関する SpMM 計算に割り当てる．

CSR 向け Batched SpMM では，各 SpMM 計算に必要となるスレッド数は $subWarp * m_A$ となる．図 5-(c) が示すように，起動するスレッド数は入力疎行列のサイズに応じて増加する．CSR 向け Batched SpMM では，図 5-(d) が示すように，n_B が大きい場合でのみ，キャッシュブロッキング最適化を適用する．入力密行列の列数 n_B が十分に小さい場合には，Batched SpMM は $\max m_A * subWarp * batch$ のスレッド数を全体として起動する．バッチ内の出力行列がキャッシュブロッキング最適化を必要とし，出力行列が列方向に p 個の部分行列に分解される場合には，Batched SpMM は全体として $\max m_A * subWarp * batch * p$ のスレッドを起動する．バッチ内の入力疎行列の行数 m_A に偏りがある場合，CSR 向け Batched SpMM は過剰にスレッドを起動することになるが，このような余剰スレッドは起動して直ちに終了するため，性能に与える影響は微小である．

5. 性能評価

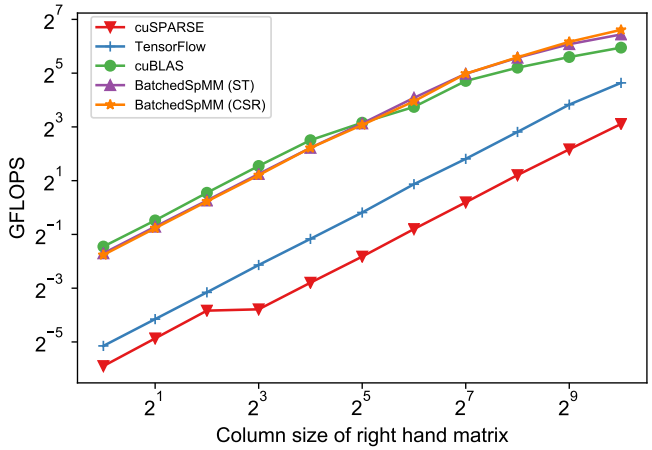
Batched SpMM の有効性を検証するために，ランダムに生成した行列データに対して SpMM 性能の評価を行った．性能評価においては東京工業大学の TSUBAME3.0 を用いた．CPU として Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz が 2 基, GPU は NVIDIA Tesla P100-SXM2 GPU が 4 基搭載されている．なお，評価では GPU を 1 基のみ用いた．Tesla P100 GPU は SM を 56，CUDA コアを計 3584 個搭載しており，理論バンド幅 732GB/sec の HBM2 を 16GB 有する．Shared memory は SM あたり 64KB，L2 キャッシュは GPU 全体で 4MB である．OS は SUSE Linux Enterprise Server 12 (x86_64) である．CUDA は ver. 9.0.176 である．

本評価では，一度に処理する疎行列積計算の数を $batch$ として行列データを生成し，全ての SpMM 計算を完了するまでの時間を測定する．ランダム生成する疎行列データは，対象がグラフデータであることから正方行列とし，行 (=列) サイズ (dim) と nnz/row をパラメータとして生成する．非ゼロ配置は各行列で異なる．また，密行列の列数 (n_B) も同様にパラメータ化されている．Non-batched 手法ではバッチ内の各 SpMM 計算を逐次的に実行する．Non-batched 手法として，cuSPARSE ライブラリの CSR 向け SpMM 関数を用いており，“csrmm()” と “csrmm2()” を評価し，最適なものを “cuSPARSE” として示す．加えて，TensorFlow での SparseTensorDenseMatMul を模した SparseTensor 形式での SpMM 計算を実装し評価した．Non-batched に対して，CSR 向けと SparseTensor 向けの Batched SpMM をそれぞれ評価した．また，Chainer を使用した化学向けの深層学習ライブラリである Chainer Chemistry [24] では，疎な性質を持った隣接行列を密行列として扱うことで行列計算を行っている．本評価では，入力疎行列を密行列として扱うことによる cuBLAS ライブラリの “gemmBatched()” の性能評価を行っており，“cuBLAS” として示す．Batched GEMM 関数は多数の GEMM 計算を処理することに最適化されており，GPU の高い計算能力を活用することが可能である一方で，疎な特性を持つ行列に対して適用するために多くのゼロ乗算やゼロ加算が行われる．なお，本評価では行列計算のスループットのみを示すが，Batched GEMM 適用によって，疎行列を密行列として保持する必要があり，結果としてメモリ使用量増加を引き起こす．実際のアプリケーションシナリオを考慮した場合，密行列として保管することによるメモリ使用量の増加は GPU 上にデータを保持することを困難にさせ，結果として CPU と GPU 間のメモリ転送を増大させるという問題を生じさせる．性能は FLOPS で表しており，$2 * nnz_A * n_B / exe\_time$ で計算される．重要であるのは実際の実行時間であるため，多くのゼロ計算が行われる “gemmBatched()” についてもこの性能計算方法に従う．Batched SpMM と Batched GEMM ではデバイス上の各行列データへのアクセスのためにポインタが必要であるが，Non-batched においてはホストメモリ側に保管されているデータとなっている．性能評価には，ポインタに関するデータ配列のホスト・デバイス間のメモリ転送時間を含めている．なお，すべての評価は単精度にて行った．

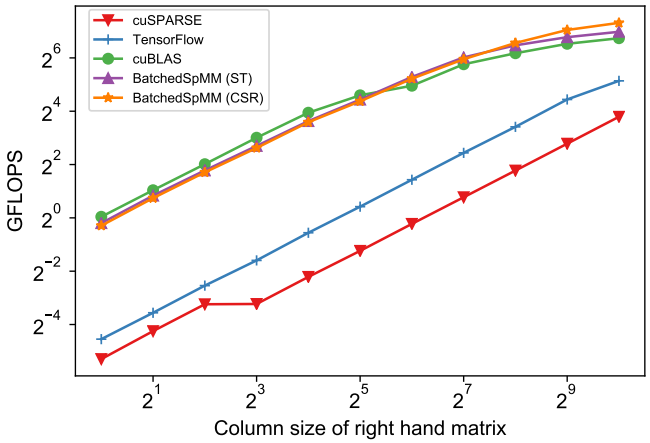
5.1 GCNs アプリケーションを模したデータセットでの評価

実際の GCNs アプリケーションで用いられているデータセット [25] に基づいて入力疎行列のパラメータを設定して生成した行列データに対して，SpMM 性能の評価を行った．図 6 に示すように，各 Batched 手法は Non-batched 手法に対して高い比率での性能向上を達成した．各 SpMM 計算を逐次的に処理する Non-batched 手法では，並列性が低く，繰り返し発生する CUDA カーネル起動によるオーバーヘッドのために高性能を達成することが困難となっている．逐次的に実行する Non-batched 手法に対して，Batched SpMM は図 6-(a) においては $n_B = 64$ で最大 9.27 倍，図 6-(b) においては $n_B = 512$ で最大 6.09 倍の性能向上を達成した．より詳細な性能分析を行うため，NVIDIA GPU 向けのプロファイラの一つである nvprof を用いて各 SM の稼働率 $sm\_efficiency$ の評価を行った．図 6-(b) の $n_B = 512$ において，Non-batched 手法である TensorFlow での SpMM の $sm\_efficiency$ は 35.51%であったのに対し，Batched SpMM の SparseTensor 向け，CSR 向けはそれぞれ 89.07%，87.87%であり，Batched SpMM が GPU の計算リソースを効果的に活用出来ていることを示している．

本評価において cuBLAS の Batched GEMM はゼロ計算を含んでいるにも関わらず高いスループットを示したが，



(a) $batch = 50, dim = 50, nnz/row = 2$



(b) $batch = 100, dim = 50, nnz/row = 3$

図 6: GCNs アプリを模したデータセットにおける疎行列積計算性能の評価．Batched SpMM (ST) は SparseTensor 向け Batched SpMM の結果を表す．

これは本評価で対象とした疎行列が小さく，ゼロ計算の割合が小さかったためである．疎な特性を持つ行列を疎行列として扱う Batched SpMM は，n_B が大きい場合において cuBLAS に対して性能面での優位性を示しており，図 6-(a) の $n_B = 64$ における cuBLAS に対する Batched SpMM の性能向上率は 1.26 倍であり，図 6-(b) の $n_B = 512$ においては 1.43 倍の性能向上を達成した．Batched SpMM によって発生する CPU と GPU 間のメモリ転送は Batched GEMM で発生するメモリ転送と比較して多く，GPU 上での SpMM 自体の実行時間が大変短い場合においては通信のオーバーヘッドが性能に対して支配的になる．その結果，入力密行列の列数 n_B が小さい場合においては，Batched GEMM が Batched SpMM に対して高い性能値を示した．

5.2 Batched 手法の比較評価

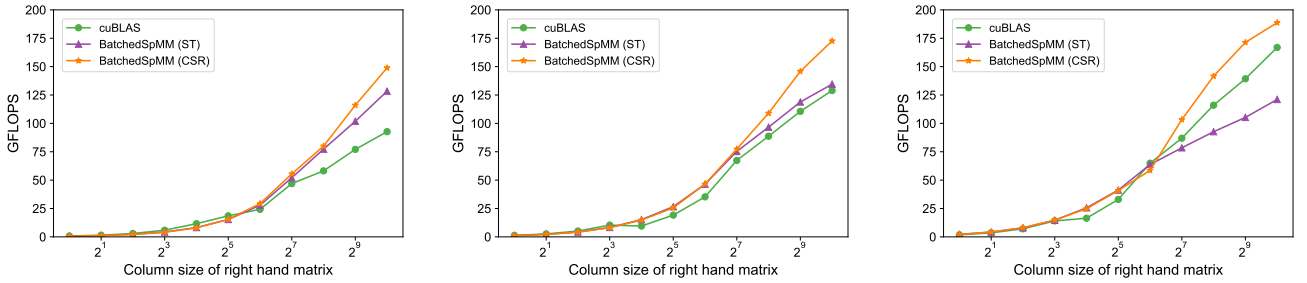
入力疎行列のパラメータやバッチサイズを変更した際の 3 つの Batched 手法，cuBLAS, Batched SpMM (ST), Batched SpMM (CSR) の性能評価結果を示す．図 7-(b) と (d) は $batch$ を変えた際の性能差を表しており，バッチサイズの増加によって全ての Batched 手法が高いスループットを得ていることを示している．$batch = 50$ の時，いずれの Batched 手法についても GPU 上のすべての SM を使えておらず，GPU の高い演算性能を活用することが出来ていない一方，より大きなバッチサイズである $batch = 100$ では十分な並列性が保障され，高い性能を達成している．

図 7 の上列，つまり (a), (b), (c) はそれぞれ $dim = 32, 64, 128$ での結果を表している．入力疎行列のサイズを増大させることによって，CSR 向け Batched SpMM と cuBLAS が高い性能を達成した．この処理性能向上は並列性の改善に由来するものであり，CSR 向け Batched SpMM は入力疎行列の行数に比例して起動するスレッド数が増加し，GPU での実行効率改善がなされる．cuBLAS についても入力疎行列のサイズ増大に合わせて並列数が増加するが，sparsity も同時に増加するために CSR 向け Batched SpMM と比較して性能向上率は小さい．なお，SparseTensor 向け Batched SpMM では入力疎行列のサイズ増大による性能変化は見受けられなかった．SparseTensor 向け Batched SpMM においても，入力疎行列のサイズ増大による並列数増加は発生するが，同時に，キャッシュブロッキングによる入力密行列の分割も増加し，結果として同一非ゼロ要素へのメモリアクセスが増加するためであると考えられる．

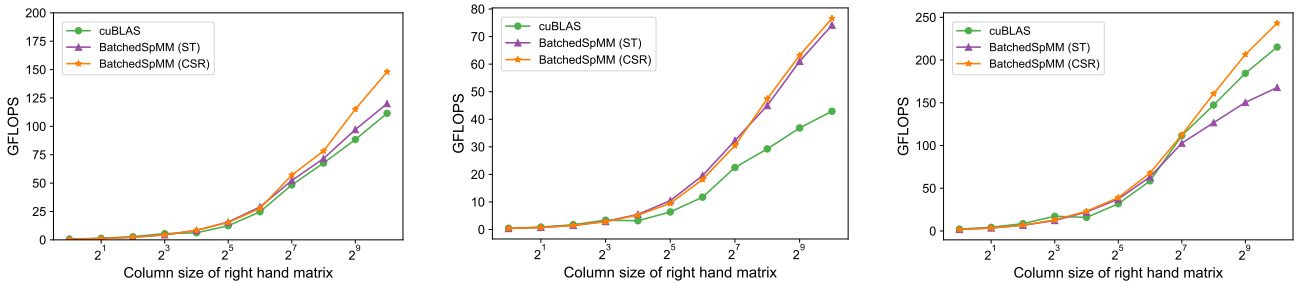
図 7-(e), (f) にそれぞれ $nnz/row = 1, 5$ での性能評価結果を示す．Batched SpMM はより疎である入力に対して，cuBLAS はより密である入力に対して性能の優位性を示している．nnz/row が小さい場合においては，SparseTensor 向け，CSR 向け双方の Batched SpMM が cuBLAS に対して性能的に優位となっている．一方，nnz/row が大きい場合において，SparseTensor 向け Batched SpMM では atomic 処理によるメモリアクセスの競合がより多く発生するため性能向上が制限される．CSR 向け Batched SpMM は atomic 処理を含まないため，密な入力疎行列に対しても最も高い性能を達成している．

5.3 行列ごとに異なるパラメータ設定での評価

バッチ内の疎行列データの次元数や nnz/row が行列毎に異なる場合での性能評価を行った．図 8 に $batch = 100$, $dim = [32, 256]$, $nnz/row = [1, 5]$ での評価結果を示す．なお，“gemmBatched()” ではバッチ内の行列積計算においてすべての行列のサイズが等しい必要があるため，本評価からは cuBLAS を除外する．バッチ内の行列データにばらつきがある場合においても Batched SpMM は Non-batched 手法からの大幅な性能向上を示しており，$n_B = 1024$ にお



(a) $batch = 100, dim = 32, nnz/row = 3$ (b) $batch = 100, dim = 64, nnz/row = 3$ (c) $batch = 100, dim = 128, nnz/row = 3$



(d) $batch = 50, dim = 64, nnz/row = 3$ (e) $batch = 100, dim = 64, nnz/row = 1$ (f) $batch = 100, dim = 64, nnz/row = 5$

図 7: ランダム生成したデータセットでの Batched 手法の性能比較評価

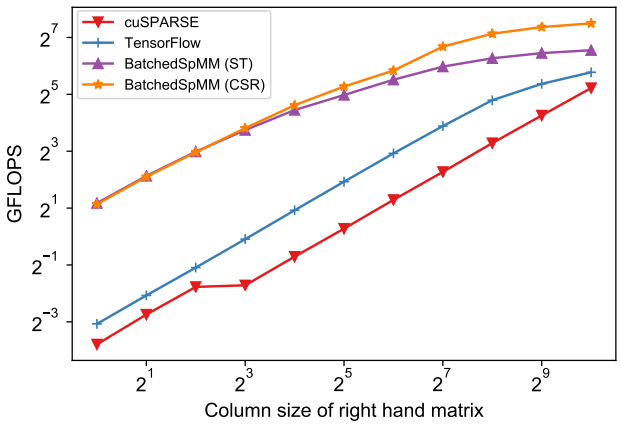


図 8: ランダム生成したデータセットでの SpMM 性能 ($batch = 100$, dim と nnz/row は行列毎に異なる)

いて最大 3.29 倍の性能向上を達成した．

6. 結論

化学や生物分野を含めた様々な分野への機械学習的手法の適用において，GCNs が高い認識精度を実現している．しかしながら，処理性能の点においては未だに十分な研究がなされておらず，小さい疎行列に関する計算を大量に行う必要のある GCNs では GPU などのアクセラレータを活用できていないという問題がある．本論文では，次元数が数十程度の小さい疎行列を含む多数の疎行列積演算を GPU にて一括して効率的に行う Batched SpMM を提案した．また，より効果的に各 SpMM の計算を行うための新たな SpMM 計算手法である Sub-Warp-Assigned (SWA) SpMM を提案し，キャッシュブロッキング最適化による局所性の改善をもって，効果的なシェアードメモリの活用を実現した．ランダム生成した行列データを用いて Batched 手法と Non-batched 手法についてベンチマーク評価を行った結果，Batched 手法は Non-batched 手法から最大 9.27 倍の性能向上を達成した．

謝辞 本研究の一部は，JST CREST(JPMJCR1303，JPMJCR1687) の支援を受けたものである．本研究の一部は，産総研・東工大実社会ビッグデータ活用オープンイノベーションラボラトリ (RWBC-OIL) の活動として実施しました．本研究の一部は，株式会社デンソーアイティーラボラトリとの共同研究として実施しました．また，本研究について多くのご助言を下さった成瀬彰氏（NVIDIA）に感謝いたします．

参考文献

[1] Kipf, T. N. and Welling, M.: Semi-supervised classification with graph convolutional networks, *arXiv preprint arXiv:1609.02907* (2016).
[2] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. and Adams, R. P.: Convolutional networks on graphs for learning molecular fingerprints, *Advances in neural information processing systems*, pp. 2224–2232 (2015).
[3] Li, Y., Tarlow, D., Brockschmidt, M. and Zemel, R.: Gated graph sequence neural networks, *arXiv preprint arXiv:1511.05493* (2015).
[4] Kearnes, S., McCloskey, K., Berndl, M., Pande, V. and Riley, P.: Molecular graph convolutions: moving beyond fingerprints, *Journal of computer-aided molecular design*, Vol. 30, No. 8, pp. 595–608 (2016).
[5] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O.

and Dahl, G. E.: Neural message passing for quantum chemistry, *arXiv preprint arXiv:1704.01212* (2017).

[6] Faber, F. A., Hutchison, L., Huang, B., Gilmer, J., Schoenholz, S. S., Dahl, G. E., Vinyals, O., Kearnes, S., Riley, P. F. and von Lilienfeld, O. A.: Machine learning prediction errors better than DFT accuracy, *arXiv preprint arXiv:1702.05532* (2017).

[7] Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I. and Welling, M.: Modeling relational data with graph convolutional networks, *European Semantic Web Conference*, Springer, pp. 593–607 (2018).

[8] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, http://tensorflow.org/ (2015).

[9] V'zquez, F., Ortega, G., Fern'ndez, J., García, I. and Garzón, E. M.: Fast sparse matrix matrix product based on ELLR-T and gpu computing, *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, IEEE, pp. 669–674 (2012).

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

[11] Buluc, A. and Gilbert, J. R.: On the representation and multiplication of hypersparse matrices, *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008.*, IEEE, pp. 1–11 (2008).

[12] Yang, C., Buluc, A. and Owens, J. D.: Design Principles for Sparse Matrix Multiplication on the GPU, *arXiv preprint*, No. arXiv:1803.08601 (2018).

[13] Merrill, D. and Garland, M.: Merge-based parallel sparse matrix-vector multiplication, *SC '16 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press, p. 58 (2016).

[14] Dong, T., Haidar, A., Luszczek, P., Harris, J. A., Tomov, S. and Dongarra, J.: LU Factorization of Small Matrices: Accelerating Batched DGETRF on the GPU., *HPCC/CSS/ICESS*, pp. 157–160 (2014).

[15] Haidar, A., Dong, T. T., Tomov, S., Luszczek, P. and Dongarra, J.: A framework for batched and GPU-resident factorization algorithms applied to block householder transformations, *International Conference on High Performance Computing*, Springer, pp. 31–47 (2015).

[16] Zheng, R., Wang, W., Jin, H., Wu, S., Chen, Y. and Jiang, H.: GPU-based multifrontal optimizing method in sparse Cholesky factorization, *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, IEEE, pp. 90–97 (2015).

[17] Venkat, A., Mohammadi, M. S., Park, J., Rong, H., Barik, R., Strout, M. M. and Hall, M.: Automating wavefront parallelization for sparse matrix computations, *SC '16 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press, p. 41 (2016).

[18] Li, X., Li, F. and Clark, J. M.: Exploration of multifrontal method with GPU in power flow computation, *2013 IEEE Power and Energy Society General Meeting (PES)*, IEEE, pp. 1–5 (2013).

[19] Abdelfattah, A., Haidar, A., Tomov, S. and Dongarra, J.: Performance, design, and autotuning of batched GEMM for GPUs, *International Conference on High Performance Computing*, Springer, pp. 21–38 (2016).

[20] Nath, R., Tomov, S. and Dongarra, J.: An improved MAGMA GEMM for Fermi graphics processing units, *The International Journal of High Performance Computing Applications*, Vol. 24, No. 4, pp. 511–515 (2010).

[21] Haidar, A., Dong, T., Luszczek, P., Tomov, S. and Dongarra, J.: Batched matrix computations on hardware accelerators based on GPUs, *The International Journal of High Performance Computing Applications*, Vol. 29, No. 2, pp. 193–208 (2015).

[22] Masliah, I., Abdelfattah, A., Haidar, A., Tomov, S., Baboulin, M., Falcou, J. and Dongarra, J.: High-performance matrix-matrix multiplications of very small matrices, *European Conference on Parallel Processing*, Springer, pp. 659–671 (2016).

[23] Anzt, H., Collins, G., Dongarra, J., Flegar, G. and Quintana-Ortí, E. S.: Flexible batched sparse matrix-vector product on GPUs, *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ACM, p. 3 (2017).

[24] pfnet research: Chainer Chemistry, https://github.com/pfnet-research/chainer-chemistry.

[25] 長坂侑亮， 額田彰，小島諒介， 松岡聡：GraphCNN 向けの疎行列積計算 Batch 最適化，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2018-HPC-167, No. 7, pp. 1–9 (2018).