Vol.2018-HPC-167 No.7
2018/12/17

GraphCNN向けの疎行列積計算Batch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 Network(GCN) 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 GCN の処理では，疎行列計算（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，GCN 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 SpMM は GPU の並列性の活用が困難であるために，疎行列計算が GCN の学習や推論の性能のボトルネックとなっている．GCN の処理性能向上のために，複数のデータに対する SpMM 計算を一つのカーネルで行うことで GPU の高い並列性と演算能力を活用可能にする Batched SpMM と，GPU のメモリ階層を活用した Batched SpMM Dynamic 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，GCN アプリケーションに Batched 手法を適用することによって学習と推論の双方において高速化を実現し，学習性能は最大 1.64 倍，推論性能は最大 1.38 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 CNN (Convolutional Neural Network) 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 Network (GCN) 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，GCN 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 GCN の適用も行われている [7].

通常の CNN の処理と同様に，GCN 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，GCN 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 GCN 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 Matrix Multiplication, SpMM）の最適化も重要となる．しかしながら，GCN で扱われるデータのグラフ構造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．また，小疎行列に関する SpMM 計算を効果的に処理するための機構や研究がこれまでなされておらず，小疎行列を並列に処理するための方法は明らかではないために，GPU の高い演算能力を十分に活用出来ずに GCN の計算処理を進めていくことを余儀なくさせられている．結果として，疎行列計算に関する箇所が GCN の性能のボトルネックとなっている．

GCN の処理性能向上のために，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．GCN 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．また，Batch 最適化した際の SpMM 計算における効果的な GPU の shared memory の活用方法として Batched SpMM Dynamic を提案する．小疎行列の場合だけでなくサイズの大きい疎行列の場合においても，キャッシュブロッキングを行うことによって shared memory の恩恵を享受することが可能であり，様々なサイズの疎行列がバッチ内に混在する場合においても，dynamic parallelism によって高い実行効率を実現する．

[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

NVIDIA Tesla P100 GPU が搭載された TSUBAME3.0 を用いて，はじめに Batched 手法の予備評価を行った結果，従来の Non-batched 手法から最大 7.6 倍，サイズの大きい入力行列についても 2.3 倍の性能向上を達成した．また，予備評価では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各最適化手法の有効性を明らかにした．GCN アプリケーションへの Batched 手法の適用によって，学習と推論の双方において高速化を図ることに成功しており，学習性能は Non-batched と比較して最大 1.64 倍，推論性能は最大 1.38 倍の性能向上を達成した．

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 $G = \{G, E\}$，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．

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

u から v に向かうエッジがある場合には，$a_{uv} = 1$，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考えた場合には

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

となり，疎な特性を持つ隣接行列 A と密行列の積演算となる．このような演算を行う Graph Convolution 層を重ねていくことによって，Graph Convolutional Network を実現している．

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 の TF が示すように SparseTensor は COO と類似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．

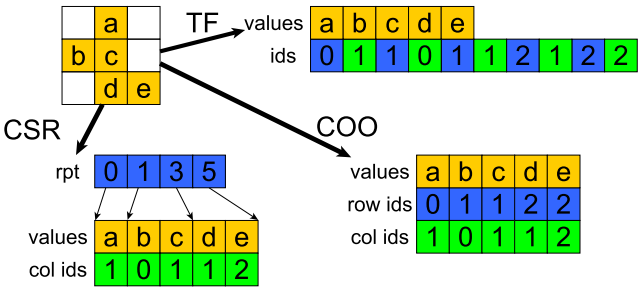

図 1: 疎行列フォーマットの例

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．各非ゼロ要素について順次計算が行われる．CUDA のコードの場合には，2 つある for ループを並列化し $nnz_A * n_B$ のスレッドを起動し，各スレッドで独立にひとつの multiply-add 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，正しく処理するために atomicAdd を用いている．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，グローバルメモリへの 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．

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 での SpMM 計算

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) と SpMM 計算手法が提案された [10]．疎行列を非ゼロ要素が偏っている箇所とそれ以外の 2 つに分け，それぞれを別々のカーネルで処理する．これによって，データの再利用性が向上し，GPU のグローバルメモリ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 GPU 向けの高速な SpMM 手法が提案された [12]．ロードバランス改善のために SpMV の手法として提案されている Merge-based [13] を 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 つを提案している．更に，Heuristic を用いて，行列によって二つのカーネルを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計算も提案されており [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]．

3.3 GCN Application

化学や生物分野での深層学習の活用の一端として GCN の適用が近年盛んに行われており，研究開発を支えるためのソフトウェア開発も進められている．

創薬や材料化学など幅広い分野において深層学習の手法を適用することを目的とした DeepChem がオープンソースとして公開されている [24]．DeepChem ではグラフ構造を隣接リスト（Adjacency lists）として保管しており，各ノードに着目しながらその近傍についての情報をリストから集め計算を行う．そのため，小さい行列の場合では GPU の高い並列性を活用することが困難である．

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 がオープンソースとして公開されている [25]．入力として化合物等の分子構造を受け取り，性質予測に深層学習的手法を適用することを可能としている．グラフ構造を持った入力データに対して GCN を適用している．疎な性質を持った隣接行列を扱っているものの，密行列として扱っているために実際の計算では多くのゼロ演算が発生している．また，グラフのサイズが巨大になった場合にはデータをメモリに載せるのが困難になるという問題がある．

4. 提案

グラフ構造を疎行列として保管した場合，GCN アプリケーションにおいて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．GCN 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，疎行列の次元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．また，GPU においては，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，大量の小疎行列の SpMM 計算のスループットを向上させる Batched 手法を提案する．また，TensorFlow で実装された GCN アプリケーションに対する効果的な Batched 手法の適用についても述べる．なお，全ての各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 などの単純なデータ構造を対象とする．本論文では，TensorFlow における SparseTensor のデータ構造を用いる．なお，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．

4.1 Batched SpMM

図 3 に Batched SpMM のアルゴリズムを記す．Batched SpMM では一つのカーネルで複数の SpMM 計算を並列に行

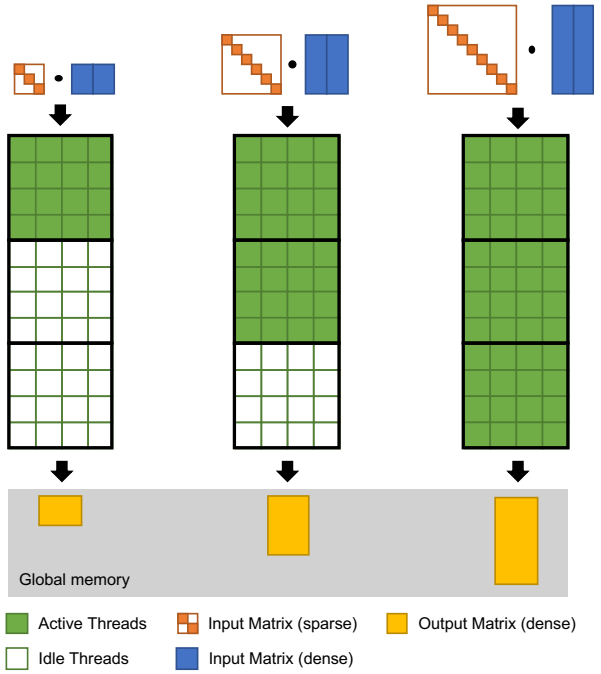

図 3: Batched SpMM の概略

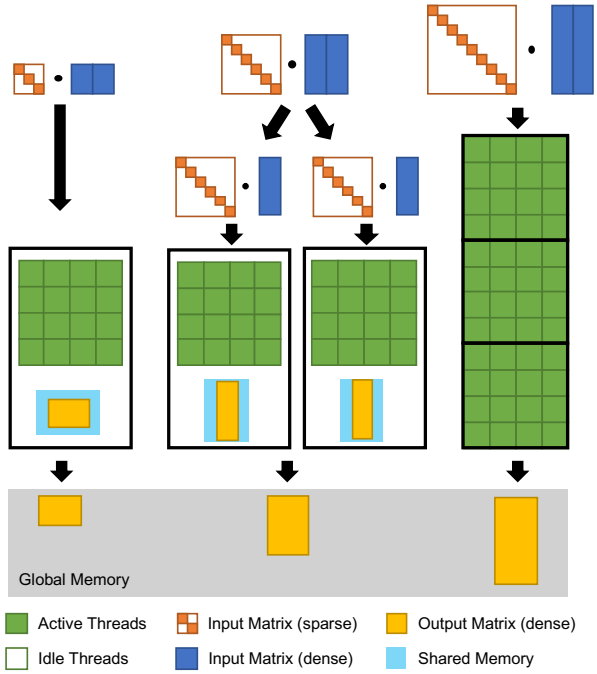

図 4: Batched SpMM Dynamic の概略

うことが可能であり，CUDA カーネルの起動オーバーヘッドを大幅に削減することが可能となる．各データに関する SpMM 計算は TensorFlow の SparseTensorDenseMatMul に準拠しており，1 CUDA thread にて 1 つの multiply-add 計算を行い，atomicAdd を用いて出力行列への加算を行う．最も計算量が多い SpMM 計算に合わせてスレッド数を決定するため，各 SpMM を処理するために起動されるスレッド数は $\max_i nnz_{A_i} * n_{B_i}$ となる．しかしながら，図 3 に示すように各 SpMM 計算同士で計算量に大きな偏りがある場合には大量のアイドルスレッドが発生するため，期待した性能向上が得られない可能性がある．

4.2 Batched SpMM Dynamic

Batched SpMM では元の TensorFlow 実装と同様 global memory への atomic 処理を行う．しかしながら，global memory への atomic 処理はコストが高く，性能低下の一因となりやすい．各スレッドブロックで共有される shared memory ではハードウェアレベルで atomic 処理がサポートされているため，高速に atomic 処理を高速に行うことが期待できる．キャッシュブロッキングと dynamic parallelism を組み合わせることによって，行列のサイズが異なる場合においても柔軟に shared memory を活用することが可能である Batched SpMM Dynamic を提案する．図 4 に Batched SpMM Dynamic の動作例を示す．

出力行列が十分に小さい場合（図 4 左），1 スレッドブロックを用いて 1 つの SpMM 計算を行う．Batched SpMM では各スレッドで一つの multiply-add 計算を行っていたが，Batched SpMM Dynamic では各スレッドは必要に応じて複数回の multiply-add 計算を行う．一つの SpMM 計算を一つのスレッドブロックで行うことによって，出力行列の局所性だけでなく，入力の密行列に関しても L1 キャッシュでの再利用性が望める．

サイズの大きい行列での SpMM 計算では，図 4 中央のように出力行列が shared memory に収まらない場合が発生する．この場合には，出力行列と入力密行列に対して列方向に分割を行うキャッシュブロッキングを施す．各部分出力行列が shared memory に収まるサイズで分割サイズを調整する．各部分行列の計算はそれぞれ独立しているため，並列に処理することが可能であり，一つのスレッドブロックを用いて一つの部分行列に関する SpMM 計算を行う．このキャッシュブロッキングによって，shared memory を活用した高速な計算が可能となるだけでなく，列数の大きい入力密行列についてもアクセスの局所性を得ることが可能となる．

バッチ内のデータによっては，全てのデータに対してキャッシュブロッキングが必要になるわけではない．Batched SpMM Dynamic では，dynamic parallelism を用いて新たにカーネルを起動し，キャッシュブロッキングを施して SpMM 計算を行う場合と，そのまま 1 スレッドブロックのみで shared memory を用いた SpMM 計算を行う場合とをデータに応じて動的に選択する．Shared memory 活用の制約を取り払いキャッシュブロッキングを効果的に適用できるだけでなく，各 SpMM 計算において無駄なくスレッドを起動することが可能になるため，occupancy を高くすることが可能となる．なお，出力行列の行数，つまり入力疎行列の行数が大きい場合（図 4 右），出力行列

を列方向に分割しても shared memory に載せることが困難なため，キャッシュブロッキングを用いることは出来ない．この場合，必要な分だけスレッドを用意したカーネルを dynamic parallelism で別途起動し SpMM 計算を行う．この時，shared memory は用いない．

4.3 GCN アプリケーションへの適用

GCN アプリケーションにおいては，ミニバッチ単位で処理がまとまって行われており，ミニバッチサイズに比例した回数分の密行列積，加算，SpMM 計算が行われる．それぞれの行列サイズが小さいため，各計算をミニバッチ単位でまとめて行うことが処理性能の点において重要となる．アプリケーションへの Batched 手法の適用によって，ミニバッチの計算をまとめて処理することが可能となる．Backward 処理における SpMM 計算に対しても Batched 手法を適用することによって，カーネルの起動オーバーヘッドを削減し，GPU の高い並列性を活用することが可能となる．

5. 性能評価

Batched 手法の有効性を検証するために性能評価を行った．はじめに，Non-batched 手法と提案手法である Batched 手法について，ランダムに生成した行列データに対して SpMM 性能の予備評価を行った．次に，GraphCNN アプリケーションにおける Batched 手法の有効性を示す．

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．対象とする GraphCNN アプリケーションは TensorFlow で実装されており，TensorFlow のバージョンは 1.8.0，cuDNN は ver.7.0 である．すべての評価は単精度にて行った．

5.1 ベンチマーク評価

ランダムに生成した疎行列データ集合を用いて，SpMM の Batched 手法についてベンチマーク評価を行った．本評価では，Non-batched として cuSPARSE の SpMM 関数と TensorFlow での SparseTensorDenseMatMul を模した COO 形式での SpMM 計算を用意した．Non-Batched に対して，Batched SpMM (Batched)，Batched SpMM Dynamic (Batched_dynamic) を評価した．また，バッチ内の出力行列全てが shared memory に収まると仮定できる場合には，dynamic parallelism を用いずに 1 カーネルのみで shared memory を用いた SpMM を行う Batched_shared を評価に加えた．ランダム生成する疎行列データについては，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した．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．

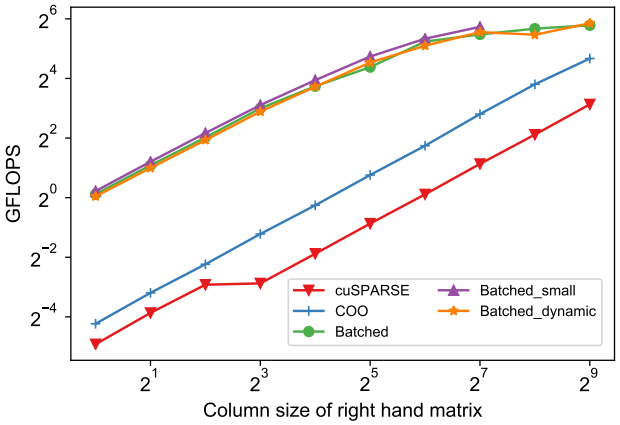

図 5: ランダム生成したデータセットに対する SpMM 性能（バッチサイズ=100, 疎行列の次元数=64, nnz/row=3）

図 5 に dim=64, nnz/row=3, batchsize=100 の場合の Non-Batched と Batched の各 SpMM 手法の性能を示す．逐次的に処理する cuSPARSE や COO では，GPU の並列性を活用できていないために各カーネルを起動する際のオーバーヘッド等により性能が低下しているのに対して，Batched の 3 手法は大幅な性能向上を達成している．全ての出力行列が shared memory に格納可能な $n_B = 128$ において，Batched_shared は Non-Batched な手法に対して 7.6 倍の性能向上を達成している．また，Batched_dynamic は単一の SpMM 計算の出力行列が shared memory に載らない場合においても shared memory を用いた SpMM 計算を行うことが可能であり，$n_B = 512$ においても 2.3 倍の性能向上を得ている．Batched 手法は多くの小疎行列積計算を処理する上で高いスループットを発揮しているが，入力密行列の列数である n_B を増加させた場合には，Batched 手法と Non-Batched 手法の性能的ギャップは減少する．単一の SpMM 計算が必要とする並列数が増加し，GPU の性能を活用することが可能となることに加え，相対的にカーネル起動のオーバーヘッドが小さくなるためである．

次に，一度に処理する量であるバッチサイズを変更した際の性能を評価した．図 6, 7 にそれぞれバッチサイズ 50, 100 の場合での Batched の 3 手法の性能を示す．図 7 から，より大きなバッチサイズにすることによって Batched_dynamic が性能向上をもたらしていることがわかる．Batched_small と Batched_dynamic では shared に収まる範囲で一つのスレッドブロックを用いており，1 つもしくは 2 つのスレッ

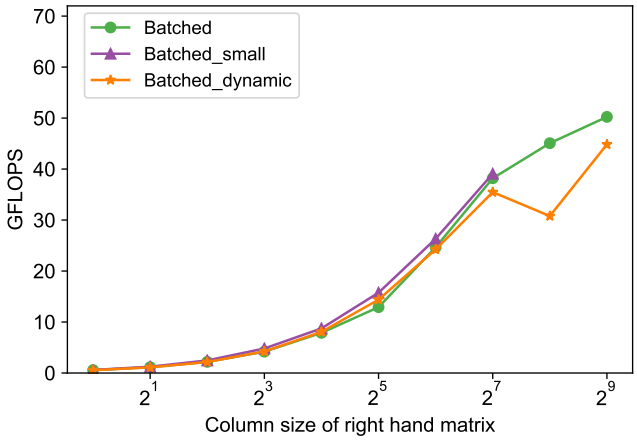

図 6: ランダム生成したデータセットに対する Batched SpMM 性能（batchsize=50, dim=64, nnz/row=3）

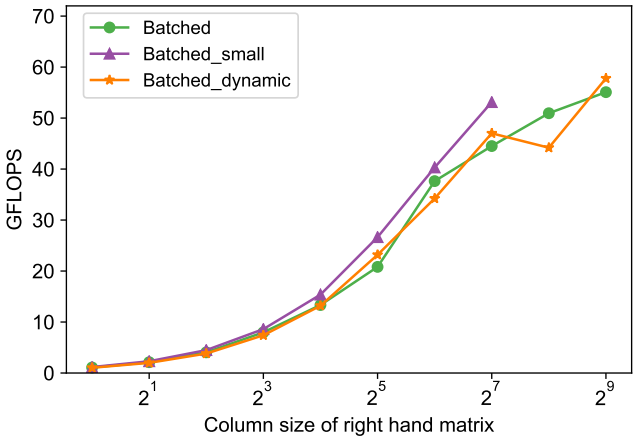

図 7: ランダム生成したデータセットに対する Batched SpMM 性能（batchsize=100, dim=64, nnz/row=3）

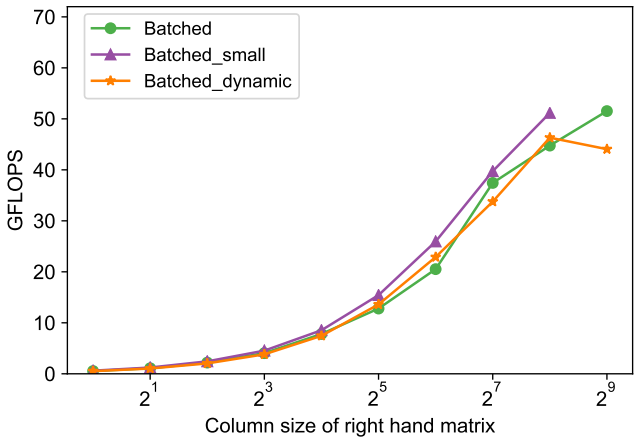

図 8: ランダム生成した疎行列データセットに対する Batched SpMM 性能（batchsize=100, dim=32, nnz/row=3）

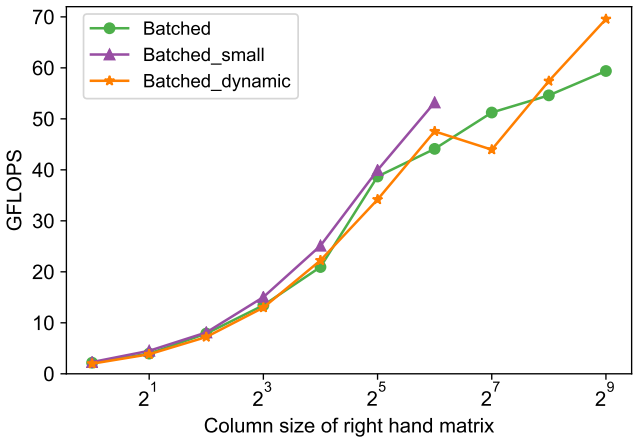

図 9: ランダム生成した疎行列データセットに対する Batched SpMM 性能（batchsize=100, dim=128, nnz/row=3）

ドブロックが各 SM に割り当てられる．バッチサイズが 50 の時には GPU 内にある SM を全て活用することは出来ない．小さいバッチサイズでは GPU の性能を完全に活用することが困難である一方，バッチサイズが 100 の場合においては，Batched_dynamic でのキャッシュブロッキングによって増加した並列数も合わさり，十分な並列数を確保することが可能となる．このとき Batched_dynamic は，shared メモリについて最適化を行っていない Batched と比較して，性能の優位性を得ることが可能となる．

次に，dim や nnz/row を変更した際の性能を示す．図 8 と 9 にそれぞれ dim=32，128 であり，他のパラメータは図 7 と同様 nnz/row=3, batchsize=100 である場合での評価結果を示す．疎行列の次元数が大きい場合において，Batched_dynamic は高い優位性を示している．これはバッチサイズを増加させた時と同様で，dim の増加によって Batched_dynamic の並列数が向上することが理由である．NVIDIA GPU における Profiler ツールである nvprof を用いることで，GPU 内の各 SM の稼働率 sm_efficiency を評価した．Batched_dynamic の sm_efficiencty は，dim=32, 64, 128 に対してそれぞれ 75.97%, 81.32%, 87.11%となり，次元数を増加させることで Efficiency が向上することが確認された．

図 10 と 11 に batchsize=100，dim=64，nnz/row がそれぞれ 1 と 5 の場合の性能評価結果を示す．Sparsity が異なる場合，Batched_small と Batched_dynamic は密な行列において優位になる．nnz/row=1 の場合，shared memory へのアクセスが複数回行われる可能性が低い．そのため，shared memory を用いることによる atomic 処理の高速化と局所性改善の効果が小さい．一方，nnz/row が大きい場合には shared memory へのアクセスが増加するため，shared memory を活用する効果が性能に大きく表れる．

最後に，密行列の列数 n_B 以外の各パラメータを同時に変えた場合，つまりバッチ内の疎行列データの次元数や nz/row が全て異なる場合での性能評価を行った．図 12 に batchsize=100，dim=[32, 256]，nz/row=[1, 5] での評価結果を示す．なお，全ての SpMM 計算の出力行列が

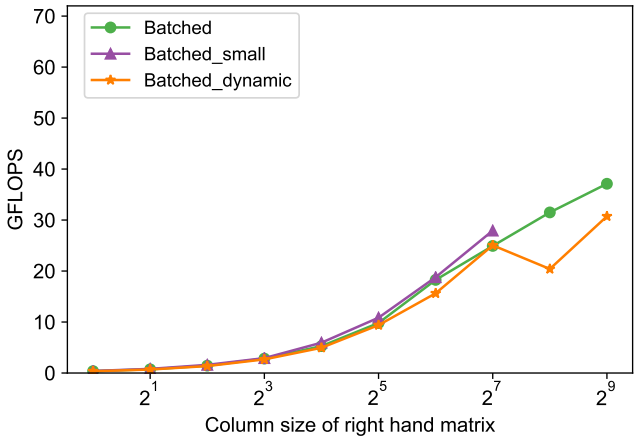

図 10: ランダム生成した疎行列データセットに対する Batched SpMM 性能（batchsize=100, dim=64, nnz/row=1）

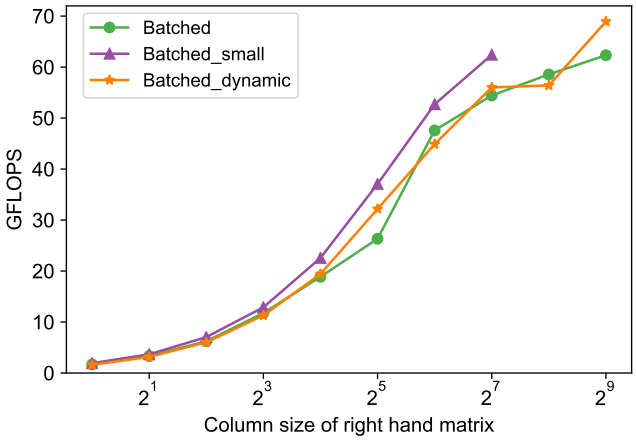

図 11: ランダム生成した疎行列データセットに対する Batched SpMM 性能（batchsize=100, dim=64, nnz/row=5）

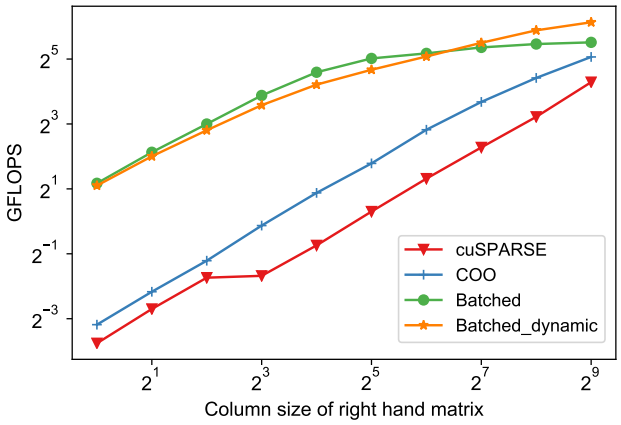

図 12: ランダム生成したデータセットに対する SpMM 性能（batchsize=100, dim と nnz/row は固定せず）

shared memory に収まるという仮定は成立しないため，Batched_small は除外する．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 手法によって Non-batched 手法からの大幅な性能向上を達成した．また，1 つの SM に一つの SpMM 計算を割り当てるやり方では負荷不均衡が発生してしまうことに加え，密行列側のサイズが大きくなってきた場合には高速な shared memory の活用等が重要となってくる．Batched_dynamic ではキャッシュブロッキングと dynamic parallelism の活用により，良いロードバランスを達成しつつメモリアクセスの局所性を得ることが可能である．

5.2 GCN アプリでの性能

TensorFlow で実装された GCN アプリケーションに対して，Batched 手法を適用をした場合の性能評価を行った．表 1 に今回用いたデータセットと各データセットでのアプリケーションの設定を記す．データセットには，Tox21 [26] に加えて，Reaxys データベース [27] から代表的な 100 種類の反応を選び作成した，化合物構造式のグラフ表現からそれらの反応を予測をするデータセットの二種類を用いた．行列数は隣接行列を表す疎行列と feature を表す密行列のペア数である．学習は k-分割交差検証の手法を用いて行っており，今回の評価ではどちらのデータセットにおいても k=5 である．推論では学習によって生成されたモデルを用いて，全データについて推論を行うのに要した時間を評価した．学習，推論共に 5 回の実行の平均実行時間を結果として記す．なお，Batched 手法は Batched_dynamic を用いた．

表 1: GCN アプリケーション評価でのデータセットと設定

	行列数	最大次元数	Epoch	バッチサイズ (Training/Inference)
Tox21	7,862	50	50	50 / 200
Reaxys	75,477	50	20	100 / 200

表 2: GraphCNN での学習性能の評価 [秒]

	CPU	GPU		
		Non-Batched	Batched	**Speedup**
Tox21	757.97	807.12	632.27	**1.20x**
Reaxys	16712.41	2845.93	1732.77	**1.64x**

表 3: GraphCNN での推論性能の評価 [秒]

	CPU	GPU		
		Non-Batched	Batched	**Speedup**
Tox21	2.44	2.24	1.71	**1.31x**
Reaxys	45.67	22.02	15.94	**1.38x**

表 2, 3 にそれぞれ GCN アプリケーションの学習性能と推論性能を示す．Tox21 データセットにおける学習では，CPU のみを用いた学習が Non-Batched での GPU 実行よりも高速であった．これは，Tox21 はデータセットとして小さく，行列データや特徴ベクトルを CPU の Last-level キャッシュに載せることが可能であるためである．一方，

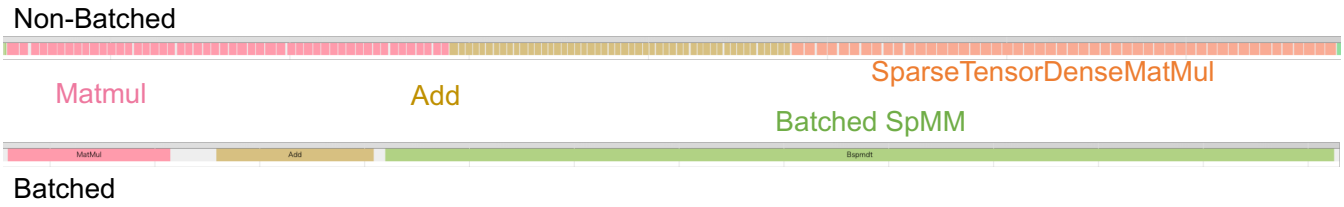

図 13: GraphCNN 性能の Timeline 可視化

表 4: GraphCNN における各カーネルの実行時間 [μsec]

	Non-Batched	Batched
Matmul	1,232	31
Add	950	30
SpMM	1,518	181

GPU での Non-Batched 手法では，並列度が不足しており GPU の演算性能を活用出来ていないだけでなく，CUDA カーネルの起動コストが高速化を阻害しているという問題点がある．これに対して，Batched 手法の適用によって GCN アプリケーションの性能を大幅に向上させることに成功した．特に，データ数やバッチサイズが大きい Reaxys においては Batched 手法の効果が顕著に現れており，学習性能については Non-Batched 手法と比較して 1.64 倍の高速化を達成している．また，推論についてはバッチサイズを大きくすることが可能であるため，Batched 手法の効果をより大きくすることが可能となる．結果として Batched 手法は最大 1.38 倍の推論性能の高速化を達成した．

TensorFlow の性能解析ツールである Timeline を用いて，各カーネルの性能を評価した．図 13 に Tox21 データセットを用いた際の Non-Batched と Batched でのそれぞれのカーネルの実行状況を示す．図 13 では Non-Batched 手法，Batched 手法どちらも等幅で示しているが，実際の各カーネルの所要時間は表 4 にそれぞれ記す．同じ区間において，Non-Batched 手法では CUDA カーネルは $batchsize * 3 = 150$ 回起動していたのに対し，Batched 手法ではカーネル起動回数は 3 回のみであり，カーネル起動によるオーバーヘッドを大幅に削減できたと言える．また，Batched 手法を用いることで Matmul については実行時間を 40 分の 1 程度にまで削減することに成功している．これは元の Non-Batched 手法では GPU の並列性を活用できていなかったことを示唆しており，Batched 最適化によって GPU の稼働率が向上していることが確認できる．

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，Graph Convolution 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 GCN 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．また，小疎行列演算，特に疎行列積演算を多数処理するシナリオはハイパフォーマンスコンピューティング領域の中においても存在せず，このような処理の高速化手法は明らかではなかった．本論文では，GCN の高速化を目的として，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，SpMM 計算を shared memory を活用して効果的に行うために，キャッシュブロッキングや dynamic parallelism の手法を導入した処理方法である Batched SpMM Dynamic を併せて提案した．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 2.3 倍の性能向上を達成した．また，GCN アプリケーションに対して Batched 手法を適用した結果，学習に要する時間を最大 1.64 倍短縮することに成功した．

今後の課題として，Batched 手法によって SpMM に関係する部分の高速化がなされたが，その結果としてアプリケーションのホットスポットが遷移した．GCN アプリケーションのさらなる高性能化のために，性能解析や最適化を行っていくことを考えている．また，Batched 手法はカーネル起動のオーバーヘッドを削減するだけでなく，各計算の局所性向上や計算機の occupancy 向上も果たしている．Batched 手法がより一般に拡張可能かどうか，CPU や Intel KNL に対して適用し検証する必要があると考えている．

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．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] : DeepChem, https://deepchem.io/.

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

[26] for Advancing Translational Sciences, N. C.: Tox21 Data Challenge 2014, https://tripod.nih.gov/tox21/challenge/about.jsp.

[27] : Reaxys, https://www.reaxys.com/.