Vol.2019-ARC-240 No.5
Vol.2019-SLDM-191 No.5
Vol.2019-EMB-53 No.5
2019/2/27

グラフニューラルネットワーク処理向けのキャッシュアーキテクチャの検討

冨田　健[1,a)]　胡　思已[1]　近藤 正章[1]

概要：近年，グラフ構造のデータを対象にした機械学習手法の一つとしてグラフニューラルネットワーク（GNN）の研究が活発に行われている．本稿では GNN の一種であり，関係グラフに対して畳み込み演算を定義した関係グラフ畳み込みネットワーク（Relational Graph Convolutional Network: R-GCN）を対象に，主としてメモリアクセスの効率化に着目して高速化を検討する．R-GCN はエッジにラベルを定義したグラフを入力とし，主に知識グラフの補完タスクなどに応用されている．R-GCN の計算処理では，その大部分を疎行列である隣接行列と密行列である特徴ベクトルを集めた行列の行列積計算に費やしている．密行列のデータに対して不規則なアクセスとなることから，R-GCN の処理はメモリアクセスがボトルネックとなりやすい．本稿ではこのメモリアクセスの高効率化のため，まず前処理として時間的局所性を高めるための頂点ノードの並べ替え手法を検討する．それでもアクセスのランダム性から再利用性が高いデータがキャッシュから追い出されてしまうことを防ぐべく，各グラフノードのエッジ数を考慮したキャッシュ置換アルゴリズムを提案する．トレースベースのキャッシュシミュレータを使用して提案手法を評価したところ，LRU によるキャッシュ置換ポリシーに対してキャッシュサイズがある程度大きい場合に提案手法は有効であり，最大で 30%程度 LLC ミス数を削減できることがわかった．

1. はじめに

近年，画像認識などにおいて深層ニューラルネットワークは必要不可欠な技術となっており，その中でも畳み込みニューラルネットワーク（Convolutional Neural Network: CNN）は中核的な技術となっている．CNN は畳み込み層とプーリング層などを組み合わせることで，画像などの特徴を効果的に抽出することができる．CNN は画像認識以外にも音声認識 [1] や自然言語処理 [2] にも応用されているが，一般的に入力のデータが規則正しい格子状のデータになっている必要がある．一方で，これら機械学習技術の発展にともない，より汎用性の高いデータ構造であるグラフ構造のデータに対しのニューラルネットワークモデルの適用が要望されるようになり，多くの研究がなされるようになってきた．これはグラフニューラルネットワーク（Graph Neural Network: GNN）と呼ばれている．

GNN の中でも特に画像認識での CNN と同様に，畳み込み演算を定義したグラフ畳み込みネットワーク（Graph Convolutional Network: GCN）が注目されている．GCN は，特に知識グラフの補完タスクや化合物の物性推定などで応用が期待されている．これらアプリケーションは大規模なグラフに対し，高度な GNN モデルが用いられることが多く，その計算量増大が予想される．そこで，本稿では GNN の一種であり，グラフのエッジにラベルがついている関係グラフを学習モデルの入力にしていて，知識ベースにおける欠損データの予測などに利用されることの多い「関係グラフ畳み込みネットワーク（Relational Graph Convolutional Network: R-GCN）」を対象にその高速化を検討することを目的とする．なお，本稿で述べる手法は他の GCN にも容易に拡張可能である．

R-GCN を含めた多くの GNN 計算では，グラフ上のノードの繋がりを隣接行列により表現する．通常，各ノードはたかだか数個のノードとしかエッジで接続されないことから，この隣接行列は疎行列の形で表すことが多い．R-GCN ではこの疎行列と，ノードの特徴量を表す密行列の行列積の計算が支配的となる．疎行列を用いた計算はデータアクセスの不規則性などから，メモリアクセスがボトルネックとなることが知られている．特に R-GCN では，比較的サイズの大きい密行列の各行が不規則にアクセスされるため，この部分のメモリアクセスを最適化することが重要である．この密行列へのアクセスの傾向として，密行列の一部の行データは頻繁にアクセスされるものの，大部分は数回以下しかアクセスされない点がある．これは，実世界におけるグラフはべき乗則の分布にしたがい，ほとんどのノードは接続しているエッジの数が少なく，一部のノードに接

[1] 東京大学 大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
[a)] tomita@hal.ipc.i.u-tokyo.ac.jp

Vol.2019-ARC-240 No.5
Vol.2019-SLDM-191 No.5
Vol.2019-EMB-53 No.5
2019/2/27

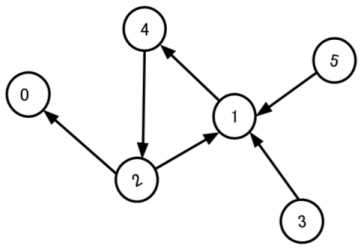

図 1 有向グラフの例

続されるエッジが偏っていることに由来する．この再利用性の高い行データと低い行データが不規則にアクセスされるため，再利用性の高いデータがキャッシュから追い出されやすく，従来のキャッシュが有効に機能しない．

本稿では，上記の問題点への対処として，密行列へのアクセスの際の再利用性を最大限に活用することを検討する．R-GCN では，疎行列で表現される隣接行列は実行中に構成が変化しない，すなわち隣接行列を参照することで密行列のどの行データが何回アクセスされるかという情報が静的にわかることになる．そこで，本稿ではまず，前処理としてグラフ上のノードのインデックスをエッジの数，すなわち密行列上のノードの特徴量のアクセス頻度によって並べ替え，隣接行列と密行列である特徴量データを書き換える．これにより，アクセス回数の多い密行列の行要素が時間的に固まってアクセスされやすくなるため，頻繁にアクセスされる密行列データの時間的局所性が向上し，キャッシュの性能が改善すると期待される．しかし，この並べ替えによっても，高アクセス頻度行データアクセスの間に低アクセス頻度行データがアクセスされることから，LRU などの主要なキャッシュ置換アルゴリズムでは高アクセス頻度行データが追い出されことがしばしば生じ，再利用性を最大限に活用することができない．

そこで本稿では，当該密行列の各行要素が何回アクセスされるかが静的にわかり，またその密行列を用いた計算が処理の大部分を占めるという R-GCN の特徴を利用し，アクセス回数をキャッシュラインの重要度として陽に指定可能にし，それに基づいて置換するラインを決定可能なキャッシュアーキテクチャを提案する．特に，アクセス回数を直接重要度として用いる場合，およびよりハードウェア実装を考慮してアクセス回数から求めた重みを重要度として用いる場合の両者の性能を比較する．

2. R-GCN の基礎

2.1 グラフの基礎

グラフはノード（頂点）の集合 V，およびエッジ（辺）の集合 E により $G = (V, E)$ で表される．あるグラフにおいて各ノードは $v_i \in V$，各エッジは $e_{ij} \in E$ と表記される．ただし e_{ij} はノード v_i からノード v_j を結ぶエッジである．このグラフを表すデータ構造として，隣接行列がしばしば用いられる．隣接行列 A は $N \times N$ の行列（$N = |V|$: ノードの数）であり，その要素 A_{ij} は $e_{ij} \in E$ のとき $A_{ij} = 1$，$e_{ij} \notin E$ のとき $A_{ij} = 0$ とすることで，グラフの構造を表す．

例えば図 1 に示すエッジに方向ががある有向グラフの隣接行列は以下の式 (1) のように表す．

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (1)$$

実際のグラフでは，一般的に各ノードのエッジの数は多くはないため，隣接行列は 1 に対して 0 の要素が圧倒的に多くなる．そこで隣接行列を効率的にメモリに格納しつつ計算するために，隣接行列には疎行列形式のデータ構造が用いられる．疎行列形式にも様々なものがあるが，本稿では特に座標格納方式を用いる．座標格納方式では，非ゼロ要素の数だけある配列 row[i]，col[i]，val[i] を確保し，行列の $(row[i], col[i])$ にある非ゼロの要素の値として val[i] を持つように配列に格納する．例えば，式 (1) の隣接行列の場合は，row = [1, 2, 2, 3, 4, 5], col = [4, 0, 1, 1, 2, 1], val = [1, 1, 1, 1, 1, 1] となる．この例の場合，val の配列は 1 のみであるため，値を持つ必要がないかもしれないが，一般的には val の要素としてエッジの属性などを持つ場合もある．また，本稿では row の要素が昇順に並んでいる座標格納方式を前提とする．

2.2 グラフニューラルネットワーク

グラフニューラルネットワーク（Graph Neural Network: GNN）は画像や音声データといったユークリッド空間上で規則的に配されたデータだけでなく，グラフ構造のデータもニューラルネットワークモデルへの入力として扱えるようにし，またそれらに対して主として畳み込み演算を行えるように拡張されたものである．グラフのノード，およびエッジには対応する特徴ベクトルを持つ．それらの特徴ベクトルに重みをかけて異なる特徴量へと変換していくことを基本とする．エッジに沿って情報を伝搬し，各ノードの特徴の合計を計算したり，最大値を求めたりすることで次の層の特徴ベクトルを生成する．これらを複数の層に渡って行うことで，処理が進められる（図 2）．これらの計算の定義には，多くの手法が近年提案されている [3][4][5].

2.3 関係グラフ畳み込みネットワーク

関係グラフ畳み込みネットワーク（Relational Graph Convolutional Network : R-GCN）は知識グラフなどの関係グラフ上で畳み込み演算を定義したグラフニューラルネットワークである．関係グラフはグラフのエッジに向き

Vol.2019-ARC-240 No.5
Vol.2019-SLDM-191 No.5
Vol.2019-EMB-53 No.5
2019/2/27

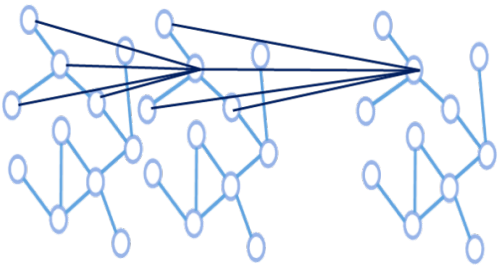

図 2　グラフニューラルネットワークの伝搬

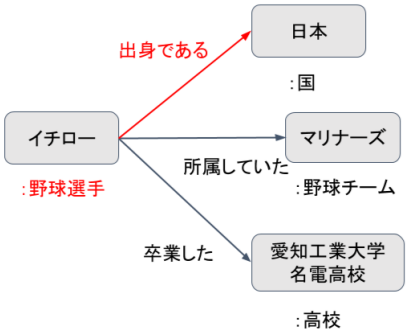

図 3　知識グラフの一部：ノードはエンティティ，エッジはリレーションと呼ぶ．エンティティとリレーションはそれぞれラベル付されている．赤色で示されたエンティティのラベルとエッジは補完タスクにおいて予想されるべきものである．

とラベルとがあるものを指し，R-GCN は主に知識グラフでの補完タスクへの応用として研究されている．図 3 は知識グラフの一例である．エンティティをノード，リレーションをラベル付きのエッジに対応させることで知識グラフをラベル付きの有向グラフとして表現している．

知識グラフは，質問応答や情報検索を含む幅広い分野で用いられる．一般に，大規模な知識グラフ（DBPedia や Wikidata，Yago）はそのメンテナンスに多大な労力がかかるため，情報が不完全なままの部分が多い．この知識グラフにおける欠落情報の予測は重要であり，R-GCN を利用してその欠落情報の補完を行うことが可能である．先の図 3 の例では，他のノード同士の関係性をヒントに，赤字で書かれたリレーションを予測することが R-GCN の代表的なタスクとなる．

R-GCN はグラフと，グラフのそれぞれのノードに対応する特徴ベクトルを入力とする．関係性毎に特徴ベクトルに重みを掛け，隣接したノードの特徴ベクトルを集約することで新しいノードの特徴ベクトルを計算することが主な計算処理である．具体的には以下のように定式化される．

$$x_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}x_j^{(l)} + W_0^{(l)}x_i^{(l)}\right) \quad (2)$$

ここで $\mathcal{N}_i^r$ は $r \in \mathcal{R}$ (r はエッジのラベル，R はラベルの集合) でのノード i の隣接ノードのインデックスの集合を表す．$c_{i,r}$ は問題に固有の正規化定数（例えば $c_{i,j} = |\mathcal{N}_i^r|$）である．$W_r$ はそれぞれのエッジのラベルごとのパラメータの行列である．なお，σ は活性化関数である．

R-GCN の演算は隣接行列 A を用いることで次のように表現することができる．ただし，活性化関数は省略している．

$$X^{(l+1)} = \begin{bmatrix} A^{(1)}X^{(l)} & A^{(1)\mathsf{T}}X^{(l)} & \dots & EX^{(l)} \end{bmatrix} \begin{bmatrix} W_{r1in} \\ W_{r1out} \\ \vdots \\ W_{self} \end{bmatrix} \quad (3)$$

ここで A はそれぞれのエッジのラベルごとの隣接行列（サイズはノード数 × ノード数であり，疎行列形式で保存），また X はノードの特徴ベクトルを集めた行列（サイズはノード数 × 特徴ベクトルの次元数で密行列）であり，W_r はそれぞれのエッジのラベル毎のパラメータ（サイズは特徴ベクトルの次元数 × 次層の特徴ベクトルの次元数で密行列）となる．

3. R-GCN 向けキャッシュアーキテクチャ提案

ここでは，R-GCN 処理において各ノードの特徴ベクトルを保持する行列アクセスの再利用性を活用するための手法を述べる．本手法は大きく分けて 2 つのの要素から構成される．1 つ目は前処理として行うデータの変換で，もう 1 つはキャッシュアーキテクチャに関するものである．データ変換は，頻繁にアクセスされる再利用性の高いデータの時間的局所性を高めることを目的とし，キャッシュアーキテクチャはデータの大域的な再利用性に応じてライン毎に優先度を設定し，それに基づいて追い出すラインを決めるものである．

3.1 隣接行列の前処理による高速化

一般的に GCN の実行（学習あるいは推論）フェーズでは，頂点の情報や頂点間の関係性（エッジの結ばれ方）は変化しないため，前処理として頂点ノードのインデックスを変換することができる．ノードの特徴ベクトルを保持する密行列部分の行データへのアクセス回数（次節で定義する q_i）が多いものほどインデックスが小さくなるように変換することで，アクセス数の多い，すなわちメモリアクセスの観点でより重要性の高いノードのデータへのアクセスが時間的に集中し，データの再利用性が活用できると期待される．

このノードのインデックス変換では，隣接行列 A と頂点の特徴ベクトルを密行列として保持する x を書き換える必要がある．これは前処理として行うため，実行時のオーバーヘッドはないが，この変換時間が長すぎると全体として処理の効率が落ちてしまう可能性がある．しかし，GCN の計算は複数の層にわたって何回かの疎行列・密行列の積の計算が行われるため，前処理の時間は大きな問題にならない．実際，いくつかのベンチマークで前処理の時間とそ

の後の R-GCN の計算時間を比較したところ，前処理時間はおよそ 1 層で構成される GCN の計算の 1 エポック程度であった．層の数が多い場合や同じデータに対して様々なパラメータで R-GCN を実行する場合にはこのオーバーヘッドは大きな問題とならないと考えられる．

3.2 提案手法

前節で説明したインデックス変換の前処理を行っても，複数回アクセスされる再利用性の高い密行列の行データが，再利用性の低い行データアクセスにより追い出されてしまうことがあり，LRU などの通常のキャッシュ置換アルゴリズムでは，再利用性を最大限に活用することができないことがある．そこで本稿では，特徴ベクトルを格納する行列の各行要素が何回アクセスされるかが静的にわかるという GCN の特徴を利用し，キャッシュラインの重要度としてアクセス回数，あるいはそれに基づく重みを明示的に指定可能にし，それを指標として置換するキャッシュラインを決定可能なキャッシュアーキテクチャを提案する．

まず，i 番目のノードの特徴ベクトルの優先度 p_i をそのアクセス回数 q_i から決定する．ここでグラフのノード数を N とし，隣接行列をラベルごとに $A^{(1)}, A^{(2)}, \ldots, A^{(R)}$ で表す．i 番目ノードに対応する特徴ベクトルを x_i とし，i 行目に x_i を並べた行列を $X \in \mathbb{R}^{N \times D}$ で表す．ただし，D は特徴ベクトルの次元数である．ここで，R-GCN で必要となる疎行列積計算は $A^{(1)} \times X, A^{(1)\mathsf{T}} \times X, A^{(2)} \times X, A^{(2)\mathsf{T}} \times X, \ldots, A^{(R)\mathsf{T}} \times X, E \times X$ となる．A^{T} は A の転置行列であり，有向エッジのソース側とデスティネーション側のそれぞれに対して特徴の集約を行うために転置した隣接行列に関しても計算を行う．なお，E は $N \times N$ の単位行列である．このとき，これら全ての行列積計算における i 番目ノードの特徴ベクトルのアクセス回数 q_i は，a_{ij} を隣接行列 A の i 行 j 列目の要素として以下の式 (4) となる．

$$q_i = \sum_{j=1}^{N} a_{ji}^{(1)} + \sum_{j=1}^{N} a_{ji}^{(1)\mathsf{T}} + \sum_{j=1}^{N} a_{ji}^{(2)} + \sum_{j=1}^{N} a_{ji}^{(2)\mathsf{T}} + \ldots + \sum_{j=1}^{N} a_{ji}^{(R)\mathsf{T}} + \sum_{j=1}^{N} e_{ji} \quad (4)$$

この q_i をもとに，i 番目ノードの特徴ベクトルが格納されるキャッシュラインの優先度 p_i を決定する．提案するキャッシュアーキテクチャでは，キャッシュ上のラインに優先度情報を保持し，実行中にそれを適宜更新しつつ，優先度をもとに追い出すラインを決定することを基本とする．なお，各キャッシュラインに対応する p_i は 1 つだけとなるようにデータの配置を行う必要がある．これは，各ノードの特徴ベクトルのサイズをキャッシュラインのサイズの整数倍にし，特徴ベクトルを格納する行列の先頭要素のアドレスをキャッシュラインサイズにアラインすれば良い．特徴ベクトルのサイズはある程度柔軟に決められるパラメータであるため，これは現実的である．

本稿では優先度の定義と実行時におけるその更新方法に関して *access count base*，および *priority base* の 2 種類を検討する．まず，それらを説明する．

3.2.1 access count base

本手法では，それぞれのノードに対してのアクセス回数をそのままノードの優先度にする．つまり，$p_i = q_i$ となる．ここで，優先度はライン毎に保持することを仮定し，当該ノードの特徴ベクトルを保持するラインの優先度として，この p_i を用いる．このように，当該データ（ライン）に何回アクセスされるかの情報を直接優先度として用いることで，実行時には当該ラインがあと何回アクセスされ得るかを追跡し，そのラインをキャッシュしておく必要ないかどうかを明示的に判別することができる．そこで，各ラインの先頭要素，すなわちラインオフセットが 0 のデータにアクセスされた際に，当該ラインの優先度を 1 減らす．これによりノードの優先度 p_i は i 番目ノードの残りのアクセス回数を示すことになる．

本手法では各ラインの優先度を保持するために，各ノードの最大のアクセス回数を保存可能なビット数分だけのフィールドが必要となる．

3.2.2 priority base

access count base では各ラインの優先度を保持するために，各ノードの最大のアクセス回数を保存可能なビット数分だけのフィールドが必要となるため，ハードウェア的なコストがかかり，また上限がアプリケーション依存である点が問題となる．そこで，優先度の最大値を比較的小ビット数に固定したものとして，priority base 手法を提案する．

本手法では，i 番目のノードの優先度 p_i を式 5 のように定義する．

$$p_i = \lfloor \{(max\_p + 1) \sum_{m=i}^{n} q_m\} / total\_edges \rfloor \quad (5)$$

ここで，$total\_edges$ は $A^{(1)}, A^{(1)\mathsf{T}}, A^{(2)}, A^{(2)\mathsf{T}}, \ldots, A^{(R)\mathsf{T}}, E$ の非ゼロ要素の総和である．また，$\lfloor \cdot \rfloor$ は $\cdot$ の整数部分を表す．この優先度の値は 0 から max_p までの整数となる．

この場合，アクセス回数と優先度は直接対応しないため，キャッシュラインの優先度を定期的に減らし，キャッシュラインのデータにアクセスがあるたびに優先度を p_i にセットし直すものとする．この周期は，各キャッシュセット毎にアクセスされた回数が一定回数に達した際に 1 周期とする．これは，直近にアクセスされたキャッシュラインの優先度を高くし，一方で最近アクセスされていないデータの優先度を低くすることに相当するため，LRU 置き換えポリシーと近いが，その際に優先度も加味して置き換えるラインが決定される点が違いとなる．

3.3 提案手法のハードウェア構成

本節では前節で提案したキャッシュ置換アルゴリズムのハードウェア構成について述べる．基本となるハードウェア構成は access count base，および priority base 手法で共

Vol.2019-ARC-240 No.5
Vol.2019-SLDM-191 No.5
Vol.2019-EMB-53 No.5
2019/2/27

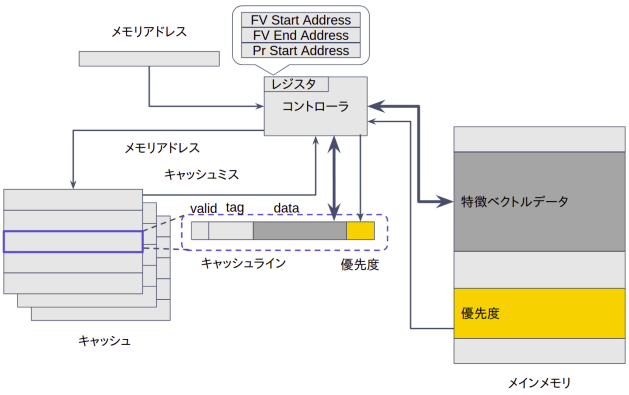

図 4　提案手法のハードウェア構成

通であり，優先度フィールドの更新手法が異なる

ハードウェア構成の概要を図 4 に示す．キャッシュの各ラインに優先度を保持するフィールドが追加されている．図中 FV_Start_Address，および FV_End_Address は，キャッシュコントローラ内に設ける制御レジスタであり，論理アドレス空間での特徴ベクトル領域の先頭と末尾のアドレスを保持する（これらはプログラム実行時に設定される）．キャッシュミスが発生した際には，当該アクセスが特徴ベクトルに対するものであるかを，そのアドレスと FV_Start_Address と FV_End_Address とを比較することで判定する．

もし，キャッシュミスの対象データが特徴ベクトルに対するものであった場合は，通常のキャッシュのようにデータをキャッシュに転送する他，主記憶中に保存した優先度の値（初期値）をキャッシュラインの優先度フィールドに保存する．この際に，キャッシュコントローラ中の Pr_Start_Address レジスタを先頭アドレスとする領域に，特徴ベクトルをラインサイズ毎に区切った場合の優先度の初期値を予め保存しておく必要がある．なお，特徴ベクトル領域以外のアドレスであった場合は，デフォルトの値を優先度に設定する．

キャッシュアクセス時には，前節で述べた手法に従い優先度フィールドを更新し，キャッシュミス時に置き換え対象のラインを選択する上では，優先度フィールドが最も低い値のキャッシュラインを置き換え対象とする．

なお，priority base 手法を実装する上では，ハードウェアのオーバーヘッドや性能オーバーヘッドを削減するために，工夫が必要である．ヒットの度に優先度フィールドを初期値に戻す必要があるが，その度にメモリからデータを取ってくるのは現実的でないため，優先度フィールドには，初期値と減算用の一時的な優先度を設けるフィールを個別に用意して，アクセス時に初期値を一時フィールドにコピーする必要がある．また，各セット毎にアクセスされた回数をカウントするカウンタも必要になる．

表 1　それぞれのデータセットでのノード数（Entities），ラベル数（Relations），合計エッジ数（Edges）

データセット	AIFB	MUTAG	BGS	AM
Entities	8,285	23,644	333,845	1,666,764
Relations	45	23	103	133
Edges	29,043	74,227	916,199	5,988,321

表 2　キャッシュシミュレータとキャッシュに格納する特徴ベクトルの詳細

L1 キャッシュサイズ	32KiB
L2 キャッシュサイズ (MiB)	0.256, 0.512, 1, 2, 4, 8, 16, 32
キャッシュラインサイズ	64byte
連想度	8
特徴ベクトルのデータ型	double(8byte)
特徴ベクトルの次元数	64
priority base の優先度の最大値	10

表 3　各データセットでの特徴ベクトルのデータサイズ

データセット	AIFB	MUTAG	BGS	AM
データサイズ	4.0MiB	11.5MiB	163.0MiB	813.8MiB

4. 評価環境と評価条件

本稿では，提案したキャッシュアーキテクチャの効果を評価する．グラフのデータセットは文献 [6] で用いられていた関係グラフのベンチマークデータセットを用いる．それぞれのデータセットのノード数，ラベル数，エッジ数を表 1 に示す．

評価は，トレースベースのシミュレータを用いることでキャッシュミス回数を測定し行う．本シミュレータのは，R-GCN の計算のプログラムに対して対象とするメモリアクセスをフックする関数を挿入し，アドレス等の情報を取得しつつ，キャッシュの動作をエミュレートするものである．実際にホスト計算機上でもとのプログラムを動作させつつ評価が行えるため，高速に大規模なプログラムのキャッシュの振る舞いを評価できる．

評価では 2 階層のキャッシュを持つアーキテクチャを想定し，L2 キャッシュのミス回数を主な指標として評価する．表 2 に評価で用いたシミュレーションのパラメータと特徴ベクトルのパラメータを示す．キャッシュに格納する全ノードの特徴ベクトルのサイズは（1 要素のデータサイズ）×（特徴ベクトルの次元数）×（ノード数）で計算でき，それぞれのデータセットで表 3 のようになる．

また，評価では隣接行列の優先度は 0，行列積結果の書き込み先の優先度は前章で定義した p_i に設定して評価を行う．priority base ではキャッシュラインの優先度を各キャッシュセットに対して 100 アクセスごとに 1 減らす．

5. 評価結果

5.1 LRU との比較

まず，access count base と priority base の提案手法と，通常の LRU のキャッシュの場合を比較評価する．なお，

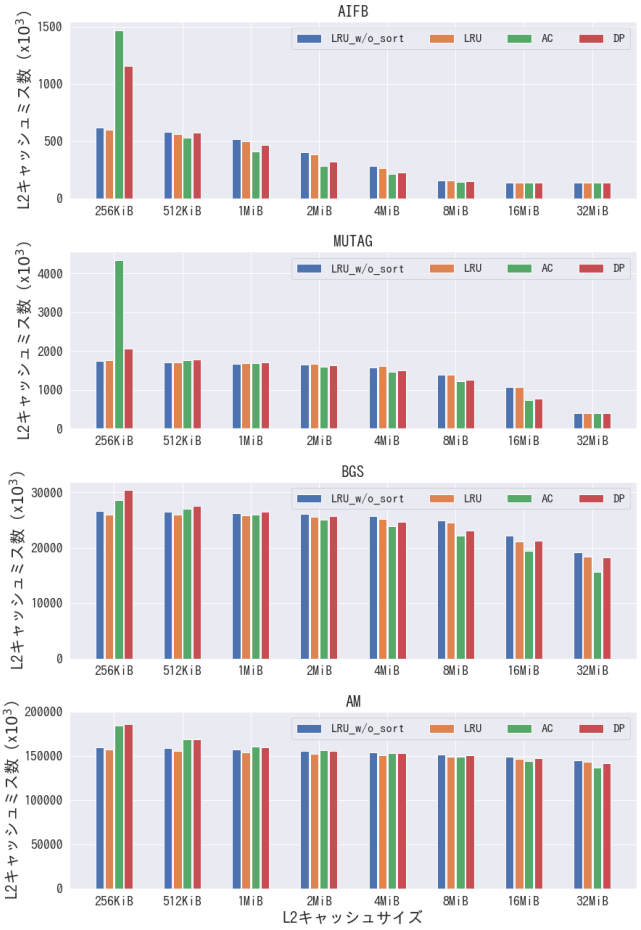

図 5 提案手法と LRU 置換アルゴリズムの比較

L1・L2 キャッシュともに提案手法の置換アルゴリズムを用いる．キャッシュサイズを変化させた場合のキャッシュミス数の評価結果を図 5 に示す．図中，“LRU_w/o_sort” が通常の LRU 置換アルゴリズムで前処理のソートをしない場合，“LRU” が通常の LRU 置換アルゴリズムで前処理としてソートを行った場合，そして “AC” および “PB” がそれぞれ access count base と priority base であり，前処理を行っているものである．

まず，“LRU_w/o_sort” と “LRU” を比較することで，前処理の効果を確認する．図より，前処理としてエッジの数をもとにノードのインデックスを並べ替えることで，多少キャッシュミス数を削減できることがわかる．頻繁にアクセスされる特徴ベクトルの要素へのアクセスをなるべく時間的に近くすることで，実際に再利用性が活用できるようになったと考えられる．一方で，効果はそれほど大きくなく，数%程度のキャッシュミス数削減にとどまっている．

次に前処理を行った条件下での，提案手法と LRU のキャッシュミス回数を比較する．図より，AC，PB ともにキャッシュサイズがある程度以上の場合には，LRU に比べてキャッシュミス削減効果があることがわかる．また，AC の方が PB に比べてミス削減効果が大きいことがわかる．これは，アクセス回数の多い密行列要素が，そうでないデータに比べてキャッシュから追い出されにくくなったため，アクセスの再利用性が最大限活用できるようになったためである．また、AC が PB に比べて効果が大きいのは，各ラインの残りの期待アクセス数を直接置換対象決定の際の指標にすることで，再利用性の高いデータがよりキャッシュに残りやすくなっているためと考えられる．

キャッシュサイズが小さい場合に，LRU に比べて提案手法のミス数が増加する理由であるが，これはキャッシュサイズが小さいとキャッシュに保存できるライン数が少ないため，時間的局所性はないが空間的局所性により近々何回かアクセスされるラインも，優先度の高いラインが存在するせいで追い出されてしまうという非効率が発生するためと考えられる．提案手法は LRU に比べ，長い時間軸で（大局的に）データの重要性を決めて，キャッシュ内に残すデータを選択していると解釈することができる．キャッシュサイズがある程度以上の場合は各データがキャッシュに読み込まれてから追い出されるまでの平均時間が長くなりやすいため提案手法が良く，逆にサイズが小さい場合はその時間が短くなりやすいため，短期的な傾向から追い出されるラインを決定する LRU が良いと考えられる．

最終的に，access count base 手法は前処理後の LRU に比べると、AIFB，MUTAG，BGS，AM それぞれのデータセットにおいて最大で 25%，32%，10%，5%ミス数を削減でき，priority base 手法では各データセットに対して最大で 16%，29%，6%，0%ミス数が削減できた．前処理なしの LRU に比べた場合，access count base 手法は最大で 29%，31%，18%，6%，また priority base 手法では 21%，28%，7%，2%ミス数を削減できた．これらより，提案手法はデータセットに依存するもの，メモリアクセスのボトルネック解消に一定の効果が見込まれると考えられる．

5.2 priority base 手法のパラメータの影響

priority base 手法では一定の周期でキャッシュライン中に保存している優先度の値を 1 減らす．そのため，周期の設定の仕方によりキャッシュのセット中のどのラインが置き換え対象になるかに影響を与える．そこで，その影響を調査すべく，周期をパラメータとして評価を行った．結果を図 6 に示す．図中の PB は周期的な優先度の更新を行わなかった場合，PB_xc は当該キャッシュセットへのアクセス数をカウントし，x カウントを周期として優先度を 1 減らす場合を示している．

評価結果より，周期による差は全体的には大きくないが，AM ベンチマーク以外では，L2 キャッシュサイズが小さい場合は比較的周期が短い方が L2 キャッシュミス効果があり，L2 キャッシュサイズが大きくなると周期が長い方が効果があるという傾向があることがわかる．周期が長い方がより長い時間軸でデータの重要度を重視することになるため，これもキャッシュに保存できるサイズに応じて，どの程度の時間軸で追い出されるラインをデータの重要性により決定すべきかが変わるためと考えられる．

Vol.2019-ARC-240 No.5
Vol.2019-SLDM-191 No.5
Vol.2019-EMB-53 No.5
2019/2/27

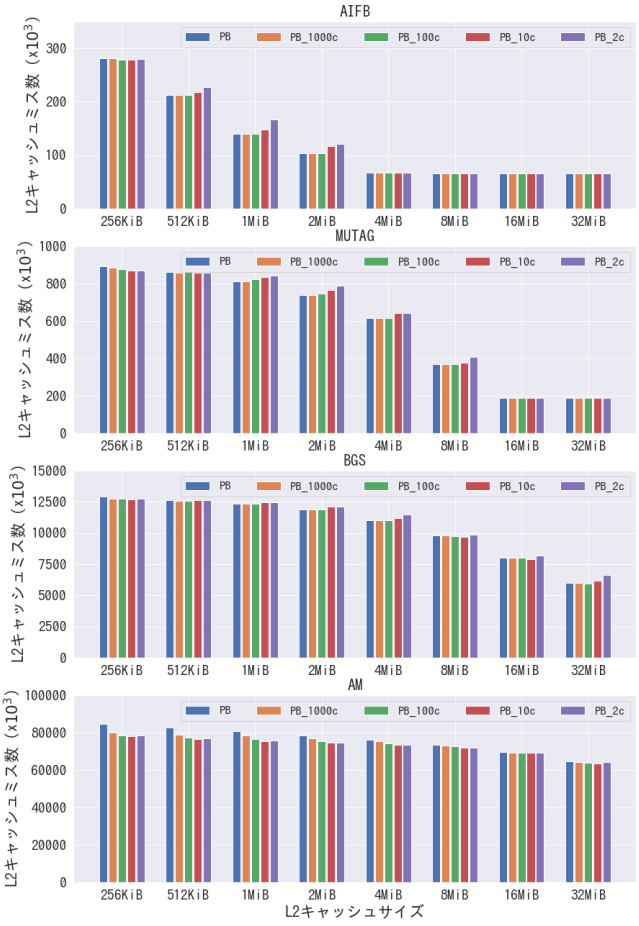

図 6 priority base において優先度を減らす周期を変化させた際のキャッシュミス数

6. 関連研究

深層学習とその画像や音声データなどにおける応用の発展により様々な深層学習のアクセラレータが開発されている [7][8][9]．また，PageRank や幅優先探索，単一始点最短経路といった一般的なグラフアルゴリズムを高速化する FPGA や ASIC で構成されたアクセラレータも研究されている [10][11]．しかし，これらのアクセラレータは GNN 向けに設計されておらず，GNN の計算を高速化できるとは限らない．

GNN の研究発展や重要なアプリケーションの登場にともない，GNN を高速化する研究も行われるようになってきている．文献 [12] では GPU 上で GNN を並列処理するフレームワークについて述べられている．また，最近では GNN の専用アクセラレータ [13] も研究されている．本研究も GNN の高速化を目指したものであるが，CPU のキャッシュの拡張として，再利用性を明示的に指定できる構成を検討している点で，他の研究とは異なる．

7. まとめと今後の課題

本稿では，特に GNN の一種である R-GCN の計算処理に着目し，処理の大部分を占める疎行列と密行列の行列積の計算を対象に高速化の検討を行った．まず，特徴ベクトルの行列へのアクセスの時間的局所性を向上させるため，データ構造の変換を行い，それでも，再利用性が高いデータがキャッシュから追い出されるのを防ぐべくエッジ数を考慮したキャッシュ置換アルゴリズムを提案した．シミュレータを用いてキャッシュミス数による評価を行い，LRU と比較して最大で 30%程度 LLC ミス数が削減できることがわかった．

今後の課題としては，本手法をサイクルレベルシミュレータに実装し，実際にどの程度高速化できるかを評価することや，さらに効率の良いハードウェア構成の検討などがある．また，隣接行列の情報を利用したキャッシュプリフェッチも検討の余地があると考えている．

謝辞 本研究の一部は，JST CREST 課題番号 JPMJCR18K1（研究課題名「エッジでの高効率なデータ解析を実現するグラフ計算基盤」）によるものである．

参考文献

[1] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, Oct 2014.

[2] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[4] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 933–941. JMLR.org, 2017.

[5] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2015.

[6] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 593–607, Cham, 2018. Springer International Publishing.

[7] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[8] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, Dec 2014.

[9] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay,

Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 1–14. IEEE Press, 2018.

[10] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martínez, and C. Guestrin. Graphgen: An fpga framework for vertex-centric graph computation. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 25–28, May 2014.

[11] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[12] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. Towards efficient large-scale graph neural network computing, 2018.

[13] Lei He. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks, 2019.