Vol.2019-ARC-237 No.20
2019/7/25

エッジセントリックなグラフ構造向けのキャッシュ性能改善手法の検討

胡 思已[1,a)]　坂本 龍一[1]　近藤 正章[1,2]

概要：
IoT デバイスで生成される多様なデータを蓄積・解析してリアルタイムなサービスを提供する上では，データ構造の設計は重要である．様々な種類のデータストアが存在する中，グラフデータ構造は，ヒトやモノ，位置情報といったデータ間の関連や依存関係を表現可能であり，今後最重要データ基盤の一つになると考えられる．グラフ処理は，広範囲のメモリ空間にランダムなメモリアクセスを発生させることから，従来のキャッシュが効率的に機能せず，メモリアクセスがボトルネックになりやすい．そのため，従来からメモリアクセスボトルネックの緩和を狙ったソフトウェアの最適化や，ハードウェアアクセラレータに関する研究が行われてきた．本稿では，特に動的グラフを効率的に扱うことのできる Edge-centric なグラフを対象に，グラフ処理を行いつつ並行して Edge リストの並び替えを行うことで，頂点データへのアクセスをスケジューリングし，メモリアクセスの時間的局所性を高める手法，およびそれを実装するためのアーキテクチャを提案する．トレースベースのキャッシュシミュレータによる評価の結果，L2 ミス回数を平均で 72.3%と大幅に削減できる可能性があることがわかった．

1. はじめに

近年，エッジ側に位置する IoT デバイス上で AI 処理や最適化処理を行う「エッジコンピューティング」が注目されている．クラウド上にデータを集約して計算を行うのに比べ，データの転送量が少なくて済み，またリアルタイム性やプライバシーなどの点に優れるという特徴がある．IoT デバイスで生成される多様なデータを蓄積・解析してリアルタイムなサービスを提供する上でデータ構造の設計は重要である．様々な種類のデータストアが存在する中，強力なデータ操作・解析能力を有するグラフデータ構造は，ヒトやモノ，位置情報といったデータ間の関連や依存関係を表現可能であり，最重要データ基盤の一つになると考えられる．現在もサプライチェーンをはじめ，交通網や電力供給網など，様々な実社会データを扱うためにグラフデータベースが利用されている．また，地理空間上の各点をノード，経路をノード間リンクに対応させたグラフ構造で，ヒトやモノの動きを表現し，その追跡をリアルタイムで行いながら行動予測や避難誘導計画の策定等に結び付ける CPS モビリティ最適化の試みも実施されている．

グラフ処理は，広範囲のメモリ空間にランダムなメモリアクセスを発生させることから従来のキャッシュが効率的に機能せず，メモリアクセスあたりの計算処理も少ないことから命令レベルの並列性活用やメモリアクセスレイテンシの隠蔽も難しいなど，特にメモリアクセスがボトルネックになりやすい．従来から，大規模グラフデータに対し，マルチコアの活用や分散計算環境でのデータ分割などを最適化したグラフ処理向けのソフトウェア環境も多く開発されてきた [1]．また，特に近年では，上述のメモリアクセスボトルネックの緩和や並列性の抽出，電力制約の厳しい組み込みシステム向けに省電力化を意識したグラフ処理専用のアクセラレータも多く提案されている [2]．

通常，大規模なグラフデータには対しては，メモリの利用効率の向上や演算処理の高速化のために，そのデータ構造として頂点 (Vertex) データの配列と，頂点データの接続関係を隣接行列の形で表現し，それを疎行列形式で保持する Vertex-centric なグラフデータ構造を採用することが多い．従来のグラフ処理専用のアクセラレータの多くは，この Vertex-centric グラフを対象として，演算対象データのスケジューリングやメモリアクセスの最適化を行ってきた．例えば，制限付きの深さ優先探索で頂点データへのアクセスを行うことで，データの時間的局所性が活用できるなどの利点を持つ．一方で，Vertex-centric グラフは頂点や辺の追加・削除を行う上で隣接行列の大幅な再構築が必

[1] 東京大学大学院情報理工学系研究科
[2] 理化学研究所計算科学研究センター
[a)] hu@hal.ipc.i.u-tokyo.ac.jp

要となることから，グラフ構造の変化が少ない静的グラフには向いているものの，頂点や辺の追加・削除が頻繁に発生する動的なグラフを扱うにはオーバーヘッドが大きい．

グラフ処理をエッジコンピューティングとして行うことを考えた場合，物理世界からの情報は絶えず変化する可能性があり，対応するグラフ構造もそれに応じて変化する，すなわち頂点や辺の追加・削除が頻繁に発生することが考えられる．動的グラフを効率的に扱うためにはVertex-centric グラフではなく，辺（Edge）のランダムなリストによりグラフデータを表現する Edge-centric なグラフが適する．しかし，Edge-centric なグラフ構造は動的なグラフを容易に扱える半面，そのままでは頂点データがランダムにアクセスされるため，データアクセスの局所性を活用できずにキャッシュが有効活用できないという問題がある．これに対し，Edge リストを前処理として予めソートすることで頂点データのキャッシュヒット率を向上させることも可能であるが，動的グラフでは処理を行う都度に前処理を行う必要があるため，そのオーバーヘッドが非常に大きくなるという問題がある [2].

本稿では，Edge-centric グラフを対象に，グラフ処理を行いつつ並行して Edge リストの並び替えを行うことで，頂点データへのアクセスをスケジューリングし，メモリアクセスの時間的局所性を高める手法，およびそれを実装するためのアーキテクチャを提案する．

2. グラフ処理の基礎と関連研究

2.1 グラフ処理とデータ構造

グラフは頂点と頂点間を結ぶ辺で構成されるデータ構造である．V を頂点 (Vertex) の集合，E を辺 (Edge) の集合とすると，グラフ G は $G = (V, E)$ で表現される．各頂点，あるいは辺は 1 つあるいは複数の属性を持つ場合もあり，グラフ処理ではその属性を用いて計算などの処理を行う．

グラフデータの特徴として，ある頂点から出る辺の数はそれほど多くはなく，すなわち疎に結合されていることがあげられる．そのため，以下に容量効率よくグラフデータを表現するかが重要となる．通常グラフデータの表現形式として，頂点間の接続関係を表す（圧縮された）隣接行列を用いるか，辺のリストを配列として保持する Edge リストを用いることが多い．

圧縮隣接行列を用いる形式の場合，頂点データを保持する Vertex Data 配列，各頂点への入力元あるいは出力先の頂点番号を保持する Neighbor 配列，各頂点に対応する Neighbor 配列の最初の要素のインデックスを保持する Offset 配列を利用することが一般的である．また，入力元，あるいは出力先の頂点番号を保持するかに違いにより Compressed Sparse Row (CSR)，あるいは Compressed Sparse Column (CSC) フォーマットが定義されている．Vertex-centric なグラフ処理を行う場合は，これらのデータ表現形式が通常用いられる．

Edge リストに基づく表現形式では，Edge リスト配列の各要素はソースとデスティネーションの頂点番号の組で構成される．この辺に相当する情報に加えて，その属性を各要素に持つ Coordinate List (COO) フォーマットもよく利用される．Edge-centric なグラフ処理を行う場合は本データ表現形式が用いられることが多い．Edge リストは先頭要素から連続的にアクセスを行うことでグラフ処理を行うものであり，プログラムが簡便になるほか，単純に Edge リストの配列を分けることで並列処理が行えるといった利点もあり，比較的良く用いられる形式である．さらに，圧縮隣接行列形式では動的グラフを扱い難いが，Edge リストに基づく表現形式では簡単に動的グラフを扱えるという特徴もある．

2.2 グラフ処理アルゴリズムの例

従来から多くのアプリケーションにおいて様々なグラフアルゴリズムが用いられてきた．その中でも，全ての頂点と辺を 1 回ずつアクセスするグラフトラバーサルは多くのグラフ処理で最も共通的な操作であり，PageRank や Breadth-First Search (BFS)，Triangle Counting (TC)，Graph Coloring (GC) など多くのグラフアプリケーションでも基本操作として利用されている．

PageRank は最も基本的なグラフアルゴリズムの一つであり，グラフトラバーサルを用いて各頂点として表されるインターネット上の Web サイトの重要度を，当該 Web サイトをリンクしている他の Web サイトの重要度をもとに計算するものである．PageRank のアルゴリズムを Algorithm 1 に示す．収束するまで数回のトラバーサルが行われ，各イテレーションでは全頂点をアクセスする必要がある．PageRank はシンプルかつ基本的なグラフ処理であることから，グラフ処理フレームワークやアクセラレータなどにおいて，ベンチマークとしても多く用いられている．

Algorithm 1 PageRank($G(V, E)$)

```
1: for each Edge ∈ E do
2:     Src ← Edge.SourceVertexID
3:     Dst ← Edge.DestinationVertexID
4:     VertexData[Dst].newValue +=
                    VertexData[Src].oldValue /
                    VertexData[Src].outDegree
5: end for
```

2.3 関連研究

従来から多くのグラフ処理フレームワークが開発され [3], [4], [5], [7], [8], [10], [11], [14]，また近年ではグラフ処理専用のハードウェアも開発されるようになってきた [1], [4], [7], [8], [9], [10], [11], [12], [13], [14].

また，前処理を行うことで効率的にグラフ処理を行うための手法も多く検討されている．前処理としては，グラフデータの並び替えやグラフ分割が用いられることが多い．グラフデータ並び替えは，メモリアクセスの効率化などを目的に，配列に保存されている頂点や辺の並び替えを行うものである．グラフ分割は，並列処理向け，または大規模なグラフをメモリに収まる範囲で計算するために，グラフデータを複数の相互に関連性の少ないサブグラフに分割するものである．

例えば，グラフ処理フレームワークの ForeGraph [4] や Graphicionado [8] は Edge-centric グラフ向けに頂点番号にもとづく Edge リストの並び替え手法を採用している．ソース，あるいはデスティネーションの頂点番号順に Edge リストをソートしておくことで，メモリアクセスのランダム性が緩和されキャッシュを有効に利用することが可能となる．また，FASTCF [9] や HMC を搭載する FPGA ベースのアクセラレータ [11], [12] では，頂点の次数の降順にソートする手法を採用している．次数の大きな頂点は多くアクセスされるが，それらをメモリ上の近くに配置することで空間的局所性の活用を狙っている．さらに， 並列処理を行う際のキャッシュライン上での競合の排除を意識した並び替え手法も提案されており，ForeGraph [4] や AccuGraph[10] といったフレームワークで採用されている．

グラフ分割の観点では，文献 [13] および Graphicionado [8]，また GraphP [14] といったグラフ処理フレームでは，source-oriented，または destination-oriented なグラフ分割手法が提案されている．これらの手法は元のグラフを，比較的に独立に処理ができる部分グラフに分割することを意図している．ForeGraph [4] では，FPGA 上のオンチップメモリを有効利用するために，source-oriented と destination-oriented を組み合わせた分割手法も用いられている．

時間発展していくような動的グラフにおいて，部分グラフを考慮した頂点や辺の追加手法も検討されている [6] が，比較的小規模な追加・削除が対象である．また，そのような動的グラフを効率よく処理するためのアクセラレータに関しては従来からあまり検討されてこなかった．本稿で提案するアーキテクチャは，特に動的グラフを意識したものであり，この点で本アーキテクチャの新規性は高いと考えられる．

3. 提案手法

3.1 Edge リストの並び替えによる局所性向上手法

Edge-centric グラフでは，Edge リストの配列に保存されているソースおよびデスティネーション頂点番号順に頂点データへのアクセスを行う．各頂点データは複数回アクセスされるものも多いが，Edge リストに保存されている頂点番号順がランダムであると，頂点データへのメモリアクセスがランダムとなり再利用性が活用できない．頂点番号順にソースあるいはデスティネーション側を事前にソートすることで，いずれかの再利用性を完全に活用することが可能であるが，動的なグラフを対象とする場合ソートのオーバーヘッドが問題となる．そこで，本稿ではグラフ処理の実行中に，部分的に Edge リストの頂点番号を並び替えることにより，時間的・空間的局所性を向上させる手法を提案する．提案手法の概要を図 1 に示す．なお，Edge リストは連続的にアクセスされるが，各イテレーションにはリスト全体をアクセスする必要があるため，空間的局所性は活用できるが，時間的局所性はない．

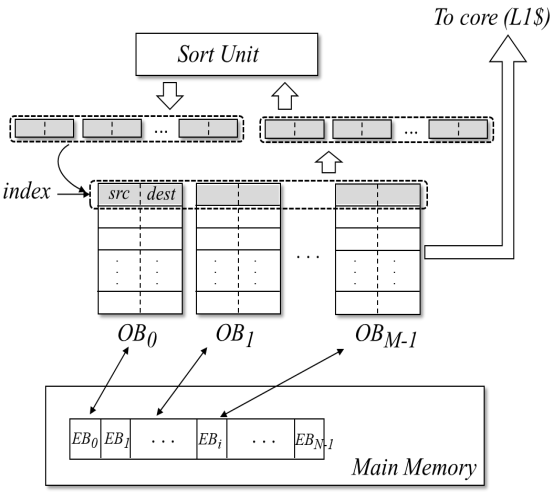

図 1 Edge リストの並び替え手法の概要

まず，Edge リストを数エントリ程度のブロックである Edge ブロックに分割する．ここで，各 Edge ブロックを EB_i と表し，合計で N 個のブロックがあると仮定する（$i = 0, 1, ..., N-1$）．メモリ中から M 個の Edge ブロック選択しアクセラレータチップ内の M 個の Edge リスト専用のオンチップバッファ (OB) にロードする．ここで，各オンチップバッファを OB_j と表す（$i = 0, 1, ..., M-1$）．チップ内にロードした Edge リストに対して以下の手順でグラフ処理，および並べ替えを行う．

(1) 各 OB から一部の頂点番号を取得し，それぞれの頂点に対するグラフ処理をする
(2) 上記 (1) のステップと同時に，取得した頂点番号をソートする
(3) ソートした頂点番号を OB の昇順に (1) で取得したエントリ部に書き込む（並べ替えが行われる）
(4) OB 内の全ての要素を処理し終えたら各 OB の内容をメモリの該当 Edge ブロックへ書き戻す

各 OB の先頭要素から 1 エントリ毎に順に処理と並べ替えを行う場合を例を図 2 に示す．なお，この例では，ソース頂点番号を基準にソートを行った場合である．上記のステップを実行することにより，並べ替え後の OB 内の頂点

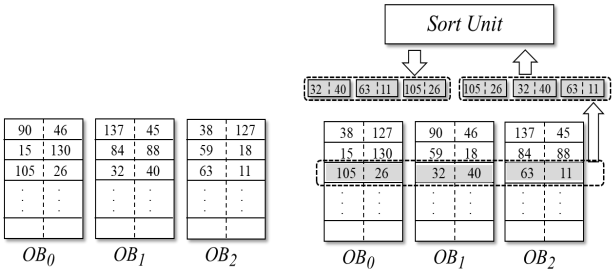

図 2 オンチップバッファ上での並び替えの例

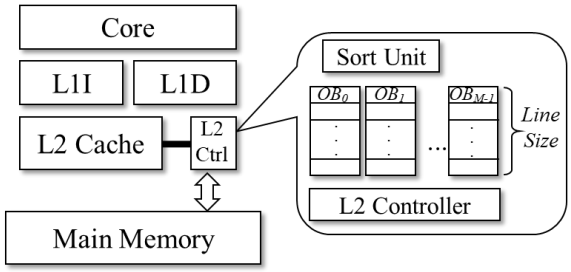

図 3 提案アーキテクチャ

番号はランダムではあるものの，ある程度近い頂点番号のものが同じ OB 内，すなわち書き戻された Edge ブロック内に集約されることが期待される．すなわち，1 つの Edge ブロックを処理する上で，並び替え後の Edge ブロックはメモリアクセスの局所性が向上していると考えられる．

ただし，上記を 1 ステップのみ行うだけではそれほど頂点番号の近いものが集約されない可能性がある．一方で，PageRank など多くのグラフ処理アルゴリズムでは，Edge リストの全エントリに対する処理を数回か繰り返すことが通常である．そこで，複数回のイテレーションに渡り，同時に OB にロードする Edge ブロックの組み合わせを変えつつ上記の処理を繰り返して適用することで，イテレーションが進むにつれてより近い頂点番号のエントリが各 Edge ブロックに集約されるようになり，徐々に局所性が向上していくことも期待できる．

3.2 アーキテクチャの実装

前節で述べた Edge リストの並び替えによる局所性向上手法を実装するアーキテクチャについて述べる．本稿では，まず通常のキャッシュ階層を持つ汎用プロセッサコアに対する拡張として，提案手法を実装することを検討する．

図 3 にプロセッサコア，L1 キャッシュ，L2 キャッシュを持つ通常のプロセッサに対する拡張として提案手法を実装する場合のアーキテクチャのブロック図を示す．Edge リスト以外のデータは通常の L2 キャッシュを経由してアクセスする一方，Edge リストデータは L2 キャッシュコントローラ内の OB に保持され，L1 キャッシュにデータを供給する．この場合，各 OB のサイズは通常のキャッシュアクセスとの互換性から，ラインサイズ分の容量とする．L2 コントローラ内には，アクセスされたアドレスが Edge リストのアドレスであるかを識別するために，Edge リストの配列の先頭，および終了アドレスを保持するレジスタを搭載する．本アドレスは，メモリマップ I/O などで設定することを想定する．

コアは通常の load 命令により，Edge リストから OB サイズにアラインされた M 個のアドレスを連続でアクセスすることで各 OB に Edge リストデータをロードする．なお，図には示していないが，メモリアクセスレイテンシ削減のために，M 個の OB を 2 セット設け，ダブルバッファリングを行うことも可能である．EB のロード後は，前節のステップにもとづいて各エントリ毎に並び替えを行う．全エントリの並び替えが終了次第，主記憶へのライトバックが行われる．

Edge リストデータはグラフ処理中はリードオンリーデータであるため，主記憶へのライトバックは本来必要ないが，提案手法では発生する．ライトバックはクリティカルな処理ではないため，メモリバスのアイドル時に実行することで性能への影響は限定的と考えられるが，既存の処理に比べて余分にバンド幅を消費してしまう．ただし，並び替えは最初の数イテレーション行えば十分に頂点データの局所性を向上できると考えられるため，バンド幅消費の影響も小さいと考えられる．

4. 評価

本稿では，主にキャッシュヒット率に着目して提案アーキテクチャの有効性を評価する．以下に，評価方法やデータセット，評価の仮定について述べる．

4.1 評価手法

一般的に，プロセッサアーキテクチャやアクセラレータを評価するにあたっては，サイクルレベルのシミュレーション評価ツールを利用することが多い．一方で，グラフ処理は処理するデータも大規模であり，また本稿の提案手法を評価するためには，トラバーサルを複数回のイテレーションに渡って評価する必要があり，サイクルレベルのシミュレーションでは評価のための時間がかかり過ぎる懸念がある．そこで，提案手法の初期評価を効率的に行うために，本稿ではトレースレベルのキャッシュシミュレーションを用いる．

実際には，グラフアプリケーション自体のソースコード中で，対象とすべきメモリアクセスをフックし，アクセスしたアドレス情報をキャッシュシミュレータに渡しつつ実行を進める自作のツールを開発して評価を行った．対象としたメモリアクセスは，ソース・デスティネーションの頂点データ，Edge リストのみであり，その他のローカル変数などは対象としない．Edge-centric なグラフ処理では

表 1　評価対象キャッシュの仮定

L1 caches	32 KB, private per-core, 4-way set associative, 64 Bytes cache line
L2 caches	8 MB, shared, 8-way set associative, 128 Bytes cache line

表 2　グラフデータセット

Graph	Vertices (M)	Edges (M)
uk-2002	19	298
it-2004	41	1150
sk-2005	51	1949
twitter-2010	42	1468
webbase-2001	118	1020

頂点データと Edge リストがほとんどを占めるため，ある程度性能に比例するキャッシュヒット・ミスの振る舞いを解析できると考えられる．ただし，キャッシュレベルのシミュレーションであるため，メモリバンド幅の影響は考慮できない．この点も含めて評価を行うことは今後の課題であり，将来的にサイクルレベルの評価も行う予定である．

4.2　評価の仮定

評価に用いたキャッシュの仮定を表 1 に示す．図 3 にあるような，2 レベルのキャッシュ階層を前提に，提案手法は L2 キャッシュコントローラに OB があるモデルを想定する．

評価では 2.2 節にも述べた PageRank アルゴリズムを用い，3 イテレーション分を評価する．加えて，並び替えを行うイテレーション数を変化させた場合も評価する．入力とするグラフデータは 2 に示すように実際の Web やソーシャルグラフのベンチマークデータセットを用いる．ランダムな Edge リストで評価を行うために，フィッシャーイェーツのシャッフルアルゴリズムを利用して Edge リストを作成した．

5.　評価結果

5.1　オンチップバッファ数の影響

図 4 に，オンチップバッファ数 M を 4, 16, 32, 64, 128 と変化させた場合の L1 キャッシュ，および L2 キャッシュのヒット・ミス回数を示す．データセットには *uk*-2002 と *sk*-2005 を用いた．PageRank を 3 イテレーション目まで実行し，1 イテレーション目は Edge リストを M 等分して OB_i には EB_i の領域のデータを順番に割り当て，2 イテレーション目は Edge リストを M^2 等分し，OB_i には $EB_{(i \bmod M)}$ の領域のデータを順番に割り当てることにした．3 イテレーション目は，2 イテレーション目までの並べ替えの効果を検証するために並び替えは行わずにトラバーサルのみを行った．図中の例えば “4-OB (Iter-1)” は，OB 数 4 で 1 イテレーション目のキャッシュヒット・ミス回数を示すことを意味している．また，Random は並べ替えを行わないオリジナルの Edge リストでの結果を，また，Sorted は事前に完全に Edge リストをソース頂点番号でソートした場合の結果を示している．

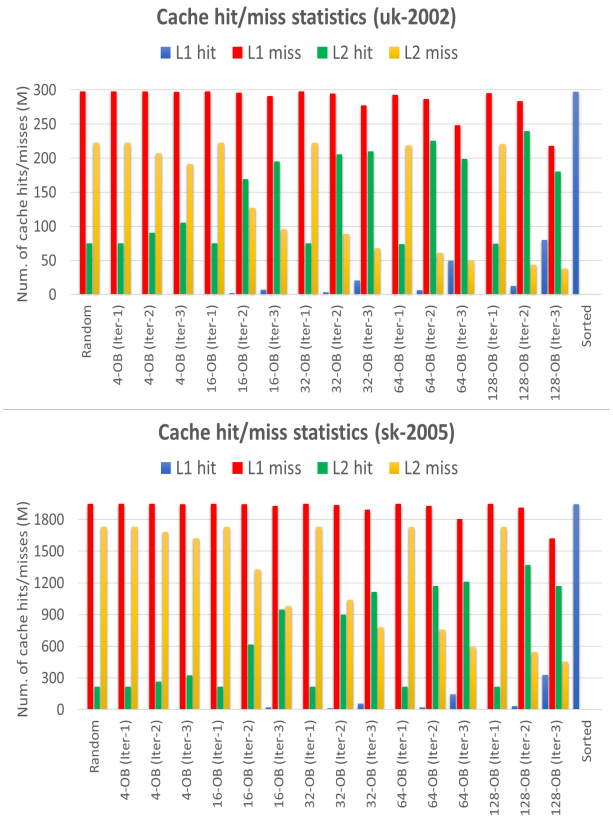

図 4　オンチップバッファ数を変化させた場合の評価結果

図より，両データセットともに提案手法を用いることで，並べ替え手法を行っていないオリジナルの Random の場合に比べてキャッシュミス回数が削減できていることがわかる．また，2 イテレーション目より 3 イテレーション目の方が，また OB 数が多い方がキャッシュミス削減効果が大きい．イテレーションを繰り返す毎に頂点番号の近いものが集約され局所性が向上すること，また，より多くの OB を持つ方が，多数の候補の中から近い頂点番号のものを集約化できるためである．規模の大きな *sk*-2005 データセットでは 4-OB の場合にあまり効果がないことから，グラフデータの規模が大きくなると，より多数の OB が必要になることもわかる．

全データセットに対する 3 イテレーション目の結果を図 5 に示す．ここではスペースの関係で OB 数 M が 4, 32, 128 の場合のみ示している．図より，全てのデータセットでキャッシュミス削減効果があることが確認できる．Random に比べ，L1 ヒット回数を平均で約 167 倍に向上でき，また L2 ミス回数を平均 72.3%削減できることがわかった．

5.2　イテレーション回数の影響評価

図 6 は *uk*-2002 データセットを用い，9 回の連続するイ

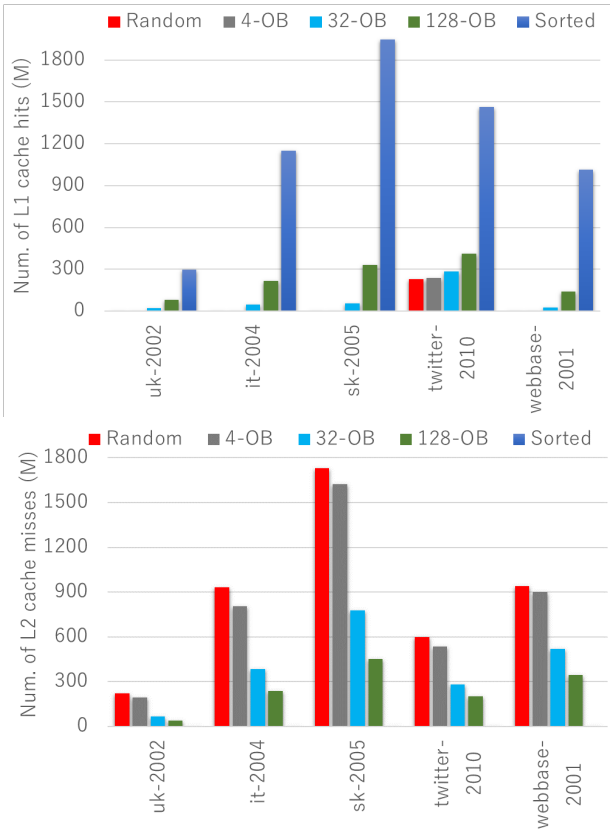

図 5　全データセットに対する評価結果 (3 イテレーション目)

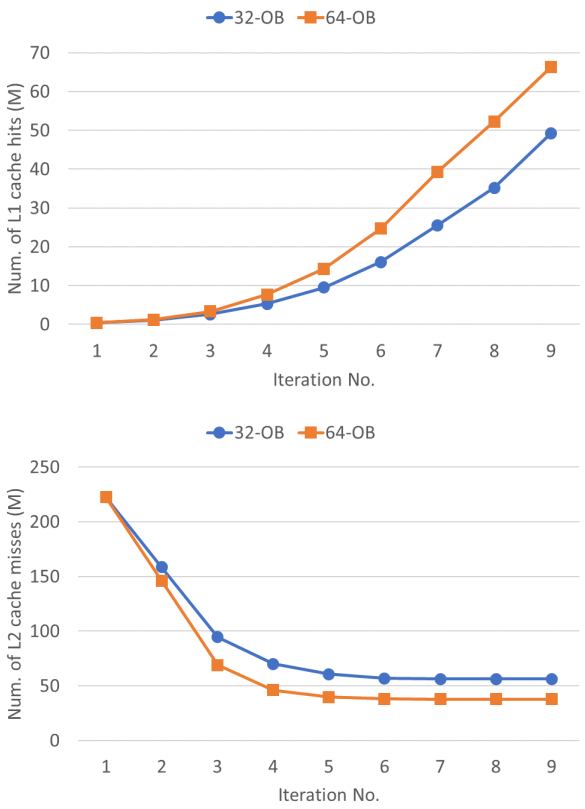

図 6　イテレーション回数の影響評価

テレーションで並べ替えを継続的に行なった場合の各イテレーションの L1 ヒット回数，および L2 ミス回数の評価結果を示している．本評価でも OB 数 M が 32 と 64 の場合のみ示す．なお，i 番目のイテレーションでは，EB 数は $M * 4^{(i-1)}$ とし，$EB_{(k \bmod M)}$ を OB_k に割り当てるものとした．2 番目のイテレーションから EB 数が OB 数を超えるので，Edge リストを 4^{i-1} 部分に分けて更に細かく調整することができる．

図の結果より，L1 ヒット回数はイテレーションが繰り返される毎に徐々に向上していくことがわかる．徐々に細粒度に近い頂点番号のものが時間的に近くでアクセスされるように集約化され，L1 キャッシュサイズ内でも局所性が活用できるためである．一方で，L2 キャッシュミス数の削減効果は，4 イテレーション目あたりで頭打ちになる．本データセットでは，3 回程度イテレーションを繰り返せば L2 キャッシュ上では十分に再利用性が活用できるほどに，集約化できたためと考えられる．

6. まとめと今後の課題

本稿では，動的なグラフを効率的に扱うことのできる Edge-centric グラフを対象に，グラフ処理を行いながら並行して Edge リストの並び替えを行うことで，メモリアクセスの時間的局所性を高める手法，およびそれを実装するためのアーキテクチャの一つを提案した．トレースベースのキャッシュ評価ツールにより PageRank アルゴリズムについて提案手法の初期評価を行った結果，ランダムな Edge リストの場合に比べて L1 ヒット回数を平均で約 167 倍向上でき，また L2 ミス回数を平均 72.3%削減できることがわかった．

今後は，PageRank 以外のグラフアルゴリズムでの評価や，並び替えた Edge リストをメモリにライトバックするオーバーヘッドを含め，より詳細にサイクルレベルのシミュレーションにより評価を行うことが課題である．また，ハードウェアの詳細な設計を行い，ハードウェア的なオーバヘッドの解析やクロック周波数への影響などについても評価する予定である．

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」）の研究によるものである．

参考文献

[1] A Mukkara, N Beckmann, M Abeydeera, X Ma, D Sanchez, “Exploiting Locality in Graph Analytics through Hardware-Accelerated Traversal Scheduling”, In Proc. MICRO-51, 2018.
[2] C Y Gui, L Zheng, B S He, C Liu, X Y Chen, X F Liao, H Jin, “A Survey on Graph Processing Accelerators: Challenges and Opportunities”, Journal of Computer Science and Technology, Vol.34, No.2, 2019.
[3] A Kyrola, G E Blelloch, C Guestrin, “GraphChi: Large-

scale graph computation on just a PC", In Proc. the 10th USENIX Conf. Operating Systems Design and Implementation, pp.31-46, 2012.

[4] G Dai, T Huang, Y Chi, N Xu, Y Wang, H Yang, "ForeGraph: Exploring large-scale graph processing on multi-FPGA architecture", In Proc. the 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.217-226, 2017.

[5] A Roy, I Mihailovic, W Zwaenepoel, "X-Stream: Edge-centric graph processing using streaming partitions", In Proc. the 24th ACM SIGOPS Symp. Operating Systems Principles, pp.472-488, 2013.

[6] X Shi, B Cui, Y Shao, Y Tong, "Tornado: A system for realtime iterative analysis over evolving data", In Proc. the 2016 Int. Conf. Management of Data, pp.417-430, 2016.

[7] G Dai, Y Chi, Y Wang, H Yang, "FPGP: Graph processing framework on FPGA a case study of breadth-first search", In Proc. the 2006 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.105-110, 2016.

[8] T J Ham , L Wu, N Sundaram, N Satish, M Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics", In Proc. the 49th Annual IEEE/ACM Int. Symp. Microarchitecture, 2016.

[9] S Zhou, R Kannan, Y Min, V K Prasanna, "FASTCF: FPGA-based accelerator for stochastic-gradient-descentbased collaborative filtering", In Proc. the 2018 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.259-268, 2018.

[10] P Yao, L Zheng, X Liao, H Jin, B He, "An efficient graph accelerator with parallel data conflict management", In Proc. Int. Conf. Parallel Architectures and Compilation Techniques, 2018.

[11] J Zhang, J Li, "Degree-aware hybrid graph traversal on FPGA-HMC platform", In Proc. the 2018 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.229-238, 2018.

[12] S Khoram, J Zhang, M Strange, J Li, "Accelerating graph analytics by co-optimizing storage and access on an FPGA-HMC platform", In Proc. the 2018 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.239-248, 2018.

[13] S Zhou, C Chelmis, V K Prasanna, "High-throughput and energy-efficient graph processing on FPGA", In Proc. the 24th IEEE Annual Int. Symp. Field-Programmable Custom Computing Machines, pp.103-110, 2016.

[14] M Zhang, Y Zhuo, C Wang, M Gao, Y Wu, K Chen, C Kozyrakis, X Qian, "GraphP: Reducing communication for PIM-based graph processing with efficient data partition", In Proc. the 2018 IEEE Int. Symp. High Performance Computer Architecture, pp.544-557, 2018.