Vol.2020-HPC-173 No.17
2020/3/17

大容量メモリデバイスへのデータ割り当て手法に伴うグラフ解析アルゴリズムの性能評価

金刺 宏樹[1,a)]　鈴村 豊太郎[2]　高野 了成[1]　Mohamed Wahib[1]　広渕 崇宏[1]

概要：従来の DRAM で保持しきれないような大規模データをリアルタイムに処理する需要が増えるに伴い，大容量な次世代メモリデバイスを使用したインメモリ処理に関する研究が盛んである．一方でソーシャルネットワークや金融取引ネットワークなど，大規模なグラフ構造で表現されている実社会データは多岐に渡り，このようなメモリデバイス上での研究も進みつつある．しかし，複数のメモリモジュールへデータを分散させた場合，モジュール間の通信コストの増大およびデータサイズの不均衡による性能低下が発生し，特に上記のようなスケールフリーなグラフデータにおいては，頂点の次数分布に大きな偏りが存在することから，解析アルゴリズムの違いにおいても性能への影響が顕著となる．本研究ではこれらの問題に対処するための，メモリモジュールへの大規模グラフデータの効率的な割り当て手法を提案する．また，人工的に生成した Kronecker グラフをそれぞれの提案手法によって Intel Optane DC Persistent Memory Module (DCPMM) 上で割り当てた後，複数の代表的な解析アルゴリズムを使用して性能評価を行い，BFS，SSSP，PageRank アルゴリズムでそれぞれ最大 29%，17%，46%の性能向上を達成した．

1. はじめに

近年，より高密度で低消費電力なメモリデバイスが開発され，主記憶装置として DRAM を使用することが現実的でない大規模なデータにおいても，インメモリ上で高速に処理することができるようになった．例として，Intel Optane DC Persistent Memory Module (DCPMM) は従来の DRAM と比較して 10 倍程度の記憶密度を持ち，次世代メモリ技術として期待されている．DCPMM は，SSD などの NAND 型フラッシュメモリと比較して読み書き速度は 1 ～ 2 桁高速であるが [21]，DRAM と比較した場合の読み書き速度はいずれも 4 倍程度遅い [6]．これはリアルタイムなデータ処理を実現するためには，DRAM と DCPMM といった性能特性の異なるメモリモジュールの効率的な使い分けを考慮しなければならいことを意味する．特にメモリアクセスパターンが複雑になるグラフデータとその解析アルゴリズムを扱う場合は，頻繁なモジュール間の頂点，エッジデータへのアクセスによるオーバーヘッドが顕著となるため，これらを低減するように各頂点，エッジを各メモリモジュールへ割り当てることが重要となる．

本研究では，数千万から十億エッジを持つスケールフリー性のあるグラフを，それぞれの DRAM と DCPMM に紐づいている CPU コアに適切に割り当てるための手法を提案し，DCPMM を複数搭載した サーバ上で性能評価を行うことで，グラフデータとグラフ解析アルゴリズムにおける効果について考察を行う．

本稿の構成は以下の通りである．第 2 章では，大規模グラフ処理アルゴリズムの重要性と性能上の限界について説明し，第 3 章ではその問題を踏まえて大容量メモリデバイスの特徴と必要性について説明する．第 4 章では，それらの課題を踏まえて，DCPMM 上での各グラフアルゴリズムを高速化するためのグラフデータの割当手法について説明する．第 5 章では，提案手法を適用させる前後で実行時間，DRAM のキャッシュミス回数，DCPMM へのアクセス回数などの性能の変化を評価する．第 6 章では DCPMM を使用した性能評価の関連研究について説明し，第 7 章で本研究のまとめと今後の課題について述べる．

2. 大規模グラフ処理

特にソーシャルネットワークや金融取引ネットワークなど，数億頂点・エッジ以上のスケールフリーグラフを効率よく解析することが挑戦的な課題となっており，これらを模した大規模グラフによる BFS（幅優先探索）アルゴリズムの実行性能を測る Graph500 ベンチマーク [17] などが数多く提案されている．しかしながら，上記のような規模のグラフデータを全てインメモリ上に載せるためには，少な

[1] 国立研究開発法人 産業技術総合研究所
[2] IBM T.J. Watson Research Center, New York, US
[a)] kanezashi.h@aist.go.jp

くとも数百 GB から数 TB のメモリ容量が必要であり，従来の計算ノード単体上に確保することは，DRAM デバイス自体のコスト，容量および消費電力の問題があり実用化が困難である．

このような大規模グラフ処理の限界に対処するために，多数の計算ノードを高速インターコネクトで接続した分散メモリシステムや，DRAM に加えてフラッシュストレージを活用する手法が提案されている．[16], [18], [19] では，最大 160 億頂点，2560 億エッジのグラフを 82,944 ノードの分散メモリ環境で上記の Graph500 ベンチマークを実行した．また，[20] では，グラフデータを DRAM だけでなく複数のフラッシュストレージデバイスにも割り当てることで，DRAM のみで処理できるグラフデータの上限の 4 倍のサイズのグラフについて Graph500 ベンチマークを実行した．いずれもインメモリ上で数十億頂点以上のグラフデータを扱えているが，前者の場合は多数の DRAM デバイスによる電力消費量の増大が，後者の場合はフラッシュストレージによる実行速度の低下が問題となっている．

3. 大容量次世代メモリデバイス

大規模データを遅延なくリアルタイムに処理できるようなインメモリ実行の需要が増えているが，数 TB に及ぶグラフデータを従来の DRAM デバイス上に全て載せるためには，大量のメモリモジュールを一度に扱わなければならず，コスト面で現実的でない．一方で，DRAM よりも安価で大容量なフラッシュストレージなどの不揮発性メモリを補助的に使用することもできるが，DRAM と比較してデータのアクセス時間が長く，実行時性能に限界がある．DRAM では保持しきれないデータを一度にインメモリ上で処理するためには，このような速度とコスト，容量のトレードオフの問題に対処しなければならない．

そこで，DRAM と比較して高密度かつ低消費電力な次世代メモリデバイスに期待が集まっている．その一つに Intel Optane DC Persistent Memory Module (DCPMM) があり，他技術に先駆けてサーバ計算機向けに実用化された．DCPMM は素子の電気抵抗値を変化させることでデータを記憶し，不揮発性を持つ一方で，バイトアドレス可能なメモリデバイスである．このような特徴から，高速なストレージクラスメモリとしてのみならず，大容量メインメモリとしても利用可能である．

4. 大容量メモリ向けグラフ割当手法

DCPMM は DRAM と比較してランダムアクセスのレイテンシが読み込み，書き込みともに 4 倍程度遅い [6]．さらに，CPU ソケットやコアをまたがるデータにアクセスする場合，該当するメモリモジュールへの読み込みリクエスト処理をその都度実行しなければならないため，アクセスパターンによっては DRAM のみの実行と比較して顕著な性能差が発生する．このような性能オーバーヘッドの高いデータアクセスを削減するために，グラフデータ解析では同一の頂点に隣接する頂点リストがなるべく同じ DCPMM にストアされるよう，事前にグラフを分割し，その結果に応じて頂点とその隣接リストを各 CPU スレッドに割り当てる手法を提案する．

4.1 前提条件

本研究では，スケールフリーグラフを扱うことを前提とする．これは，頂点の次数分布に大きな偏りがあり，多数の隣接頂点とエッジを持つごくわずかな頂点 (super-hub) 同士が集中しているようなグラフであり，ソーシャルネットワークやウェブリンクによる WWW ネットワーク，口座間の金融取引ネットワークなど大規模な実社会グラフデータで同様の特徴を持つものは多い．

一方で，このような実社会に存在する大規模グラフでは頂点およびエッジの追加削除が頻繁に行われるが，今回は静的なグラフを想定し，一度グラフデータを DRAM および DCPMM に割り当てた後は，実行時にそれ以上のデータ構造の変更は行わない．後述するグラフ分割についても，BFS などグラフアルゴリズムの性能評価とは別に，前処理として事前に済ませておくものとする．

4.2 提案手法

図 1 のように，予め入力グラフをほぼ同じ個数の頂点からなるクラスタに分割しておき，その結果に応じて各頂点の隣接リストをスレッドに割り当てる．分割の際には，なるべく同一の CPU ソケットと DRAM，DCPMM 内部で近傍の頂点の隣接リストも参照されるよう，分割後のエッジカットが最小になるようにする．

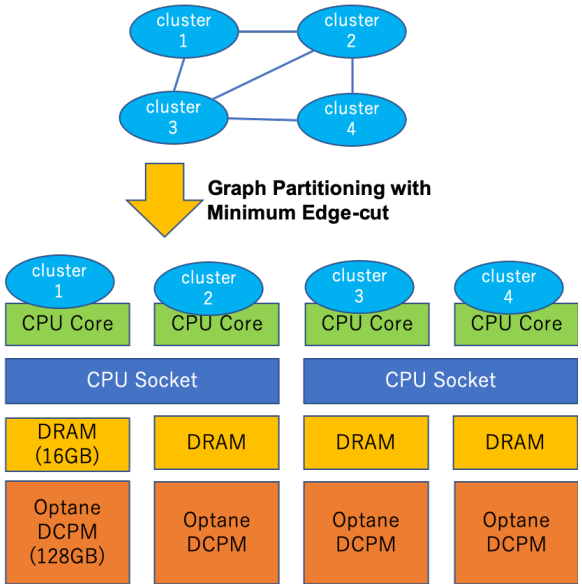

図 1 グラフデータの分割と割当

5. 性能評価

5.1 ハードウェア構成

本実験で使用した計算機サーバの諸元とそのメモリ構成は，それぞれ表 1 および図 2 の通りである．Intel Gold 6230 プロセッサの 6 つのメモリチャネルには，それぞれ DDR4 DRAM 16GB と DCPMM 128GB が搭載されている．使用した計算機サーバは 2 ソケット構成なので，DRAM の総容量は 192GB，DCPMM の総容量は 1536GB となる．また，各プロセッサは 20 個の論理コア (Core 0〜Core19, Core 20〜Core39) を有し，Hyper Threading(T0〜T79) を有効化した．

表 1　使用した計算機サーバの諸元

CPU	Intel Xeon Gold 6230 2.1GHz, 2 processors
DRAM	DDR4 DRAM 16 GB, 2666 MT/s, 12 slots
DCPMM	DDR-T 128 GB, 2666 MT/s, 12 slots
OS	Fedora 30/Linux Kernel 5.0.9-301

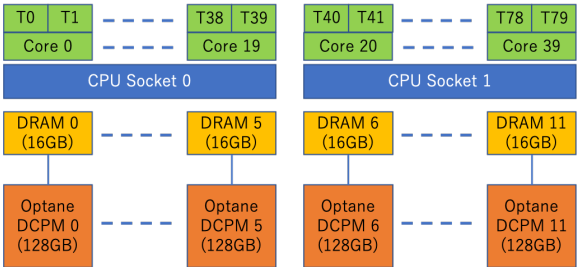

図 2　使用した DCPMM ハードウェアアーキテクチャ

本実験で扱うグラフアルゴリズムの実行では，ハードウェアで利用可能な最大限の 80 個のハイパースレッドを使用した．また，DCPMM の動作モードとして，DCPMM を大容量の揮発性メモリとして扱う Memory Mode と，DCPMM を不揮発性メモリとして扱う App Direct Mode の 2 種類が提供されているが，本実験では Memory Mode を使用した．本モードではメモリコントローラによって DRAM が DCPMM のキャッシュとして利用される．

5.2 グラフアルゴリズムと実装

本実験では，グラフ処理系としてグラフアルゴリズムのベンチマーク用として公開されている GAP Benchmark Suite (GAP-BS)[2] の実装を使用した．評価に使用したグラフアルゴリズムは GAP-BS で実装されている中の以下の 5 種類で，各アルゴリズムをそれぞれ 16 回繰り返し，その合計時間と CPU キャッシュミスの総数，および DCPMM への読み込み，書き込みアクセスリクエスト回数の総数をカウントした．

BFS(Breadth-First Search)　BFS は，指定したある一つの頂点を始点とし，隣接した頂点を順に探索する処理を繰り返すことにより，ある他の頂点へ通じる最短経路と視点からの距離を求めるアルゴリズムである．GAP-BS では Beamer によるアルゴリズムの最適化 [1] がされている．

PageRank　それぞれの頂点の重要度をスコアリングするアルゴリズムの一つであり，自身の頂点が持つスコアを，隣接する頂点のスコアを足し合わせて重み付けをした値に更新する処理を繰り返す．繰り返しの処理は指定した回数に達するか，全頂点の更新前後のスコアの変化分の合計が閾値以下に達した場合に終了となる．今回の実装では，更新回数の上限を 20，変化分の閾値を 10^{-4} としている．

SSSP (Single-Source Shortest Path)　ある一つの頂点から BFS アルゴリズムを繰り返し，全ての頂点への最短距離を求めるアルゴリズムである．始点となる頂点は，毎回ランダムに選ばれる．

BC (Betweenness Centrality)　全ての頂点のペアについて BFS アルゴリズムを実行し，その最短経路が通過した回数に応じて頂点にスコアを付けるアルゴリズムである．本実験では，BFS アルゴリズムを開始する頂点を 4 個ランダムに選び，さらに Brandes のアルゴリズム [3] による近似を行うことで，計算量をエッジ数の線形に抑えている．

CC (Connected Component)　互いにたどり着くことのできる経路が存在するような頂点のグループ (connected component) を求めるアルゴリズムである．GAP-BS では，自身の頂点が持つラベルを隣接する頂点のラベルを繰り返し参照することで更新し，最終的に同じラベルを持つ頂点のグループを近似的に connected component とする Afforest アルゴリズム [15] を採用している．

GAP-BS 内部では，グラフデータは Compressed Sparse Row (CSR) として表現されており，同じ頂点に接続している頂点の隣接リストがメモリ空間内で連続して並んでいる．そのため，特に BFS およびそれを繰り返し使用するグラフアルゴリズムにおいては，近傍の頂点同士が近いメモリアドレスに書き込まれていれば，CPU のキャッシュミスおよび異なる CPU ソケットに割り当てられている DCPMM へのデータ呼び出しリクエストのオーバーヘッドが削減されると期待できる．

また入力グラフを最小エッジカットで，かつなるべく均等に分割するために，前処理としてグラフ分割ライブラリの METIS[9] を使用した．前述の GAP-BS ではグラフデータの入力フォーマットとして連番の頂点 ID が書かれたエッジリストをサポートしているが，図 1 の通り事前に入力グラフを最小エッジカットになるように 4 個のクラスタに分割し，同じクラスタに属する頂点の ID が隣り合うように頂点 ID を振り直した．

5.3 グラフデータセットと実験条件

性能評価で使用するためのグラフデータとして，GAP-BS で用意されているグラフ生成ライブラリにより下記の表 5.3 にある Scale が 20，22，24，26 の Kronecker グラフ を生成した．生成した Kronecker グラフの頂点数は，2^{Scale}，各頂点に繋がるエッジ数の平均は 16 となっており，これらのグラフデータは Graph500 ベンチマークで扱うデータセットに準拠したものである [17]．なお，上記の METIS ライブラリの実装の都合上，Scale26 より大規模なグラフデータを扱うことはできなかった．

グラフデータ	頂点数	エッジ数
Scale20	1,048,575	15,699,687
Scale22	4,194,303	64,155,718
Scale24	16,777,216	260,376,709
Scale26	32,804,550	2,103,846,430

また，グラフ分割による頂点 ID の並べ替えの前処理の有無と，以下に示す Intel OpenMP でサポートされている CPU コアへのスレッド ID 割り当て手法 (NUMA affinity)[5] の組み合わせによる，合計 6 通りの条件で実験を行った．

compact 小さい ID の CPU コアから，スレッドが割り当てられるだけ順にスレッド ID を連番で割り振る．OpenMP のオプションで指定したスレッド数が全ての CPU コアで利用可能なスレッド数よりも小さい場合，スレッド ID が割り当てられない CPU コアが存在する場合もある．

balanced それぞれの CPU コアがなるべく同じ数だけスレッドを持つようにスレッド ID を割り振る．ID は CPU のコア ID の小さい方から連番で割り当てられる．

scatter balanced と同様にそれぞれの CPU コアに均等にスレッド ID を割り振るが，false sharing を避けるために同じ CPU コア内ではなるべく離れたスレッド ID が割り当てられる．

5.4 実行結果

ここでは，それぞれのグラフアルゴリズムについて，提案手法の有無と NUMA affinity の種類に応じて，実行時間および CPU キャッシュミス回数，DCPMM の読み込み，書き込みリクエスト回数を測定した結果について説明する．なお，以下の図で示した実行時間はグラフアルゴリズムベンチマークの実行によるものだけを指し，Kronecker グラフの生成，METIS によるグラフ分割および頂点 ID の並べ替えなどの前処理は含まない．METIS によるグラフ分割処理にかかる時間はは Scale20, 22, 24, 26 でそれぞれ 25 秒，149 秒，15 分，88 分とグラフ処理と比較して極めて長い．この問題については，大規模なグラフ分割は一度だけ実行し，[10], [13], [14] などのインクリメンタルな分割アルゴリズムを使用するなど，今後の課題とする．

Scale20 から Scale26 まで生成した Kronecker グラフを使用した場合の，BFS アルゴリズムの実行時間および CPU キャッシュミス回数はそれぞれ図 3 と図 4 の通りである．なお，キャッシュミス回数については，使用した 3 種類の NUMA affinity 全てにおいてほとんど差が見られなかったため，scatter の結果のみ載せている．他のアルゴリズムについても同様である．

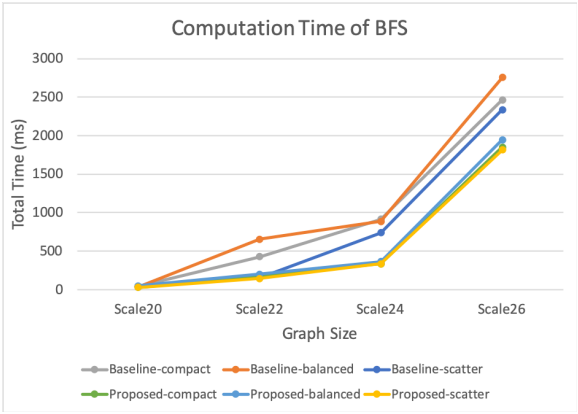

図 3 BFS アルゴリズムの実行時間

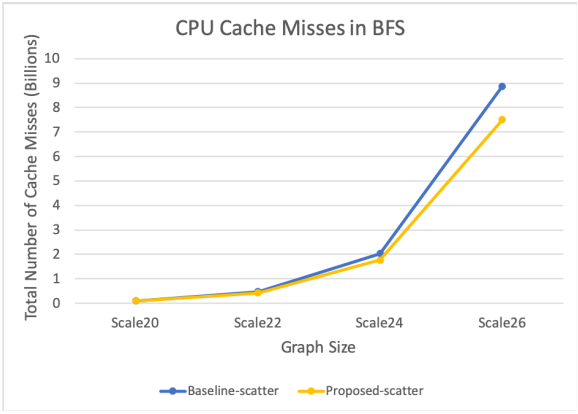

図 4 BFS アルゴリズムの CPU キャッシュミス回数

METIS を使用したグラフ分割による前処理を行った場合 (Proposed) では，行わなかった場合 (Baseline) と比較して Scale24 のデータセットで最大 63%，Scale26 で最大 29%(2.75 秒から 1.94 秒) 実行時間が短縮され，scatter の NUMA affinity オプション (Proposed-scatter) を使った場合が一貫して最も短時間 (1.81 秒) で実行できた．これは，BFS アルゴリズムによる隣接頂点の探索において，近傍の頂点同士が同一のキャッシュに入る割合が増えたためであり，実際に CPU キャッシュミス回数も Scale26 で 15%前後減少している．

また，BFS ベンチマークを実行した間に，DCPMM にあるデータを読み込む，書き込むためのアクセスリクエ

ストを実行した回数はそれぞれ表 2，表 3 の通りである．Scale26 のグラフデータを使用した場合では読み込み，書き込みともに Scale24 の場合と比較して数百倍に増加している．これは全体のグラフデータが DRAM の容量に収まり切らず，DCPMM へもグラフデータの大部分がストアされたためである．また Scale26 では，提案したグラフ分割による頂点の割り当ての有無と NUMA affinity の組み合わせに依らず，リクエスト回数に目立った変化が見られなかった．

BFS アルゴリズムと同様のグラフデータセットを使用した場合の，PageRank アルゴリズムベンチマークの実行時間および CPU キャッシュミス回数はそれぞれ図 5 と図 6 の通りである．

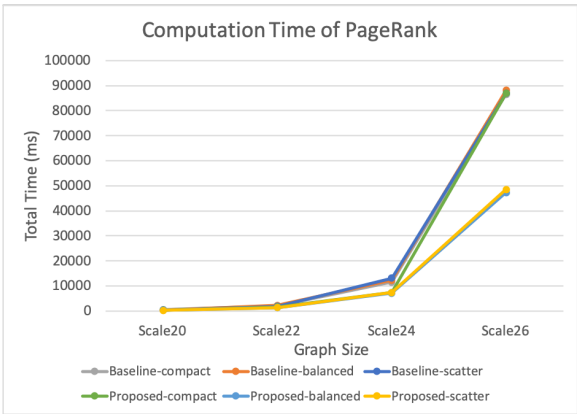

図 5 PageRank アルゴリズムの実行時間

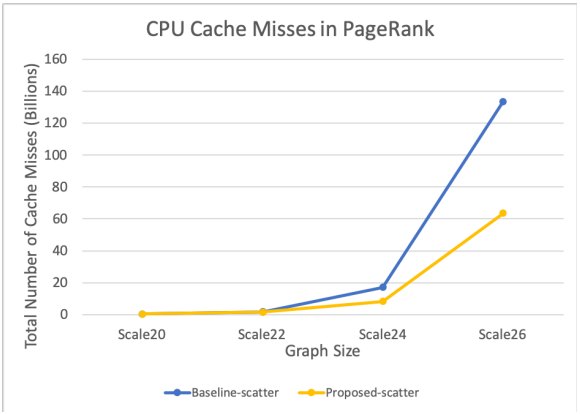

図 6 PageRank アルゴリズムの CPU キャッシュミス回数

PageRank アルゴリズムでは，使用した他のグラフアルゴリズムと比較して提案手法による性能向上が特に著しく，比較的小規模なグラフデータ (Scale20, Scale22) でも 25%以上，Scale24, Scale26 ではそれぞれ 13.3 秒から 7.4 秒，88.2 秒から 47.3 秒と，最大 44%，46%の実行時間短縮を実現している．CPU キャッシュミス回数においても Scale24, Scale26 で特に効果が現れており，50%から 52%減少している．これは，新しい頂点を探索する度にスレッドを動的に起動しなければならない BFS および BFS を使用する SSSP, Betweenness Centrality とは異なり，PageRank ベンチマークでは 1 回のイテレーションの実行中は一貫して各スレッドに割り当てられた同じ頂点の隣接リストしか参照しないので，局所性をより活用できたことによるものと考えられる．

SSSP アルゴリズムでの実行時間，CPU キャッシュミス回数はそれぞれ以下の図 7，8 の通り， DCPMM への読み込み，書き込みリクエスト回数はそれぞれ表 4，5 の通りである．

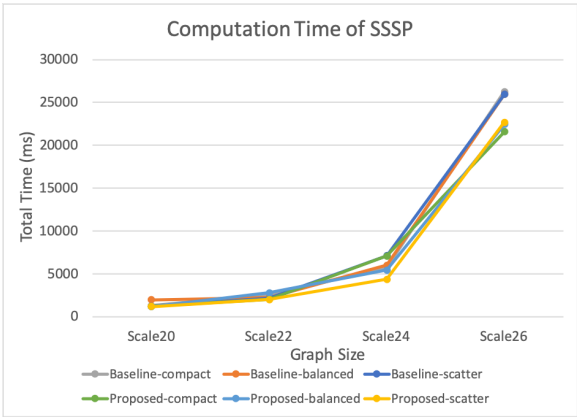

図 7 SSSP アルゴリズムの実行時間

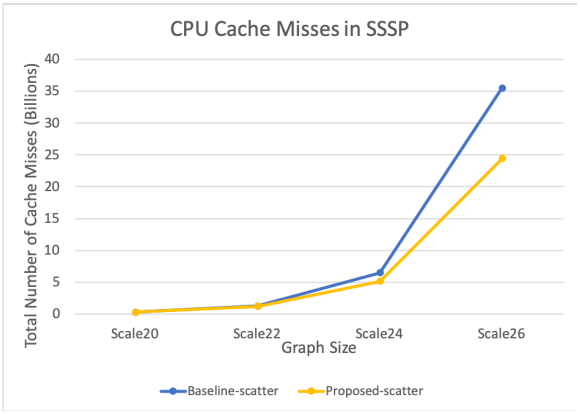

図 8 SSSP アルゴリズムの CPU キャッシュミス回数

SSSP ベンチマークを Scale26 のグラフデータで実行した場合では，NUMA affinity の違いにより多少のばらつきはあるが，グラフ分割による頂点の割り当てにより，実行時間が 12%から 17%まで削減された．CPU のキャッシュミス回数においては，いずれの NUMA affinity においても Scale24 で 21%前後，Scale26 では 31%程度削減されている．一方，DCPMM への読み込み，書き込みリクエスト回数については，BFS と同様グラフ分割による頂点の割り当てをした場合でも目立った変化は見られなかった．

Betweenness Centrality アルゴリズムベンチマークの実行時間および CPU キャッシュミス回数はそれぞれ図 9 と

	Baseline			Proposed		
グラフデータ	compact	balanced	scatter	compact	balanced	scatter
Scale20	9.7	11.6	10.7	11.9	11.2	11.0
Scale22	41.1	101.7	49.5	68.0	111.8	64.6
Scale24	8099.4	11933.3	2229.9	1593.0	1082.5	3639.9
Scale26	537839.9	532993.4	534495.5	569399.2	568838.5	560676.6

表 2 BFS ベンチマークの DCPMM 読み込みリクエスト回数 ($\times 10^3$)

	Baseline			Proposed		
グラフデータ	compact	balanced	scatter	compact	balanced	scatter
Scale20	5.3	7.4	5.9	6.9	6.3	6.4
Scale22	30.8	57.6	39.1	44.6	73.3	44.9
Scale24	1089.3	1201.1	574.3	861.9	655.9	834.2
Scale26	288811.2	400380.9	401629.9	331370.1	398912.4	463950.1

表 3 BFS ベンチマークの DCPMM 書き込みリクエスト回数 ($\times 10^3$)

図 10 の通りである．

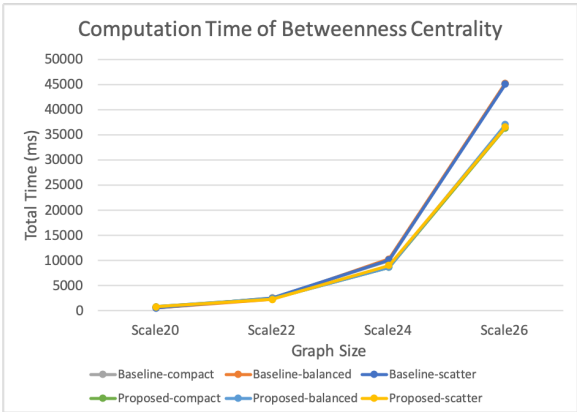

図 9 Betweenness Centrality アルゴリズムの実行時間

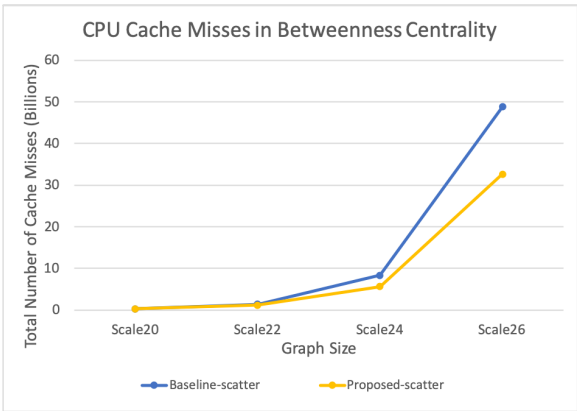

図 10 Betweenness Centrality アルゴリズムの CPU キャッシュミス回数

Betweenness Centrality アルゴリズムにおいても，Scale24, Scale26 のグラフデータにおいて提案手法の効果がはっきりと見えており，いずれもキャッシュミス回数は 32〜33%，実行時間は Scale24 で 9.9 秒から 8.6 秒 (最大 16%減少)，Scale26 で 45.2 秒から 36.3 秒 (最大 19%減少) まで短縮された．

Connected Component アルゴリズムベンチマークの実行時間および CPU キャッシュミス回数はそれぞれ図 11 と図 12 の通りである．

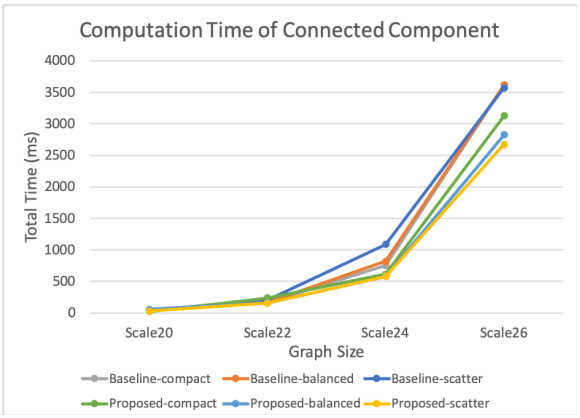

図 11 Connected Component アルゴリズムの実行時間

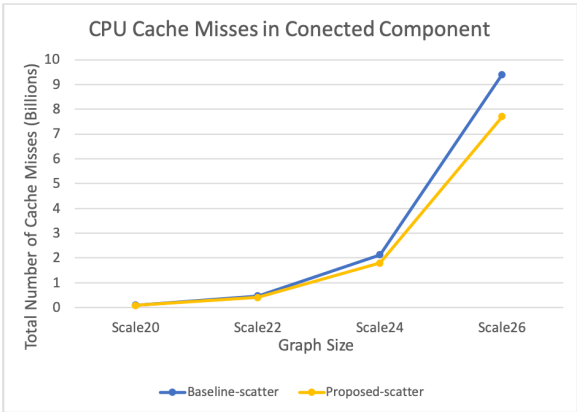

図 12 Connected Component アルゴリズムの CPU キャッシュミス回数

Connected Component アルゴリズムに関しても提案手法を適用させた場合が一貫して短時間で実行できている

	Baseline			Proposed		
グラフデータ	compact	balanced	scatter	compact	balanced	scatter
Scale20	32.1	20.3	17.4	8.1	13.5	9.4
Scale22	430.7	404.3	389.4	474.2	300.5	590.7
Scale24	6044.6	6618.4	7553.5	10376.5	2333.2	3321.1
Scale26	1455216.2	1433304.4	1452124.9	1491013.9	1474359.1	1502437.1

表 4 SSSP ベンチマークの DCPMM 読み込みリクエスト回数 ($\times 10^3$)

	Baseline			Proposed		
グラフデータ	compact	balanced	scatter	compact	balanced	scatter
Scale20	19.1	12.8	12.8	4.1	4.4	5.2
Scale22	291.1	277.9	273.2	320.1	182.9	353.6
Scale24	1842.1	1812.0	1960.7	1609.2	1186.8	1674.2
Scale26	782366.9	783229.4	796226.5	779371.8	779708.3	818672.2

表 5 SSSP ベンチマークの DCPMM 書き込みリクエスト回数 ($\times 10^3$)

が，NUMA affinity の違いにより，若干実行時間にばらつきが出ている．例えば Scale26 においては，最も短時間で実行できた scatter では 2.67 秒かかったが，compact では 3.13 秒かかっており，提案手法を適用させる前の Baseline との実行時間減少率もそれぞれ 25.2%, 13.6%と開きがある．compact で比較的性能が低下した原因としては，あるスレッドが処理する頂点のラベルが全て更新され収束した一方で，別のスレッドが処理する頂点のラベルの更新が終わらないためにスレッド間のの不均衡が目立ったことが考えられるが，より詳細な分析は今後の課題とする．

6. 関連研究

DCPMM が新しいメモリデバイスとして注目されている中，様々なベンチマークを用いた性能評価の研究が行われている．

[12] では，HPC 環境で用いられる科学技術計算用アプリケーションをベンチマークとした性能評価を行い，DRAM と従来の不揮発性メモリを使用した場合にデータアクセス速度の乖離がある問題点を踏まえて，比較的アクセス速度が DRAM に近い DCPMM を用いた場合の効果について検証している．

また [4] では，DCPMM を使用してグラフアルゴリズムの性能評価を行っている．本研究で使用した GAP-BS も含めた 3 種類のグラフ処理フレームワークの実行性能を比較している他，DRAM では処理しきれない大規模なグラフデータに対して従来の分散メモリシステムとの実行時間を比較し， DCPMM に向いているグラフアルゴリズムについての考察も行っている．

7. まとめと今後の課題

本研究では，次世代不揮発性メモリデバイスである DCPMM による大規模グラフアルゴリズムのインメモリ実行と，グラフ分割による頂点の再割り当て，NUMA affinity の適切な設定による性能最適化手法を提案した．人工的に生成した数千万〜数億エッジのスケールフリーグラフとグラフアルゴリズムベンチマークによる性能評価の結果，再割り当てをする前と比較して Scale26 のグラフにおいて BFS，SSSP，PageRank アルゴリズムでそれぞれ最大 29%，17%，46%の性能向上を達成した．

今後の課題として，より実社会データとアプリケーションに即した手法を提案するために，以下が挙げられる．

まず本研究では 4.1 で述べた通り，一度グラフデータが DRAM や DCPMM 上にストアされた後は，アルゴリズム実行中に頂点やエッジが追加削除されない，つまり DRAM や DCPMM 上のデータが更新されない静的なグラフデータを前提としている．しかしながら，ソーシャルネットワークや金融取引ネットワークなど，実社会のグラフデータは秒単位で数百万の頂点やエッジの追加削除が常に発生している．そこで，本研究で提案した手法を拡張し，以上のような動的グラフデータでも適用できるようにする．そのためには，動的な DRAM と DCPMM へのデータの書き込み処理のオーバーヘッド，実行時の CPU 同士のワークロードバランス，動的なグラフに対応するグラフ解析アルゴリズム等についても考慮する必要がある．

また，グラフ分割手法そのものについても検討が必要である．本研究では分割後の各サブグラフについて頂点数の均等化とサブグラフ間の最小エッジカットを実現するために METIS ライブラリを用いたが，実装上の問題から 1 億頂点以上の大規模なグラフを METIS で分割することができず，分割可能な場合でも 5.4 の冒頭で述べた通り一度の分割で数分から数十分規模の時間を要してしまい，頻繁に変化する大規模グラフに対して前処理として実行するのはまだ現実的ではない．このような問題に対処するために，特に動的なグラフデータに対してはインクリメンタルな分割と頂点の再割り当てアルゴリズムを採用することも検討する．また，最小エッジカットを考慮したグラフ分割の

代わりに，[7] ではそれぞれの頂点の次数に応じて高速な DRAM にストアするか大容量のフラッシュストレージにストアするか判断する手法を取っている．今後は前述の動的なグラフデータへの対応も含めて，より計算コストの小さい効率的な頂点のスレッド割り当て手法についても考慮する必要がある．さらに，DCPMM を一様なメモリ空間として扱う Memory Mode だけでなく，App Direct Mode による DRAM と DCPMM の明示的な使い分けによる最適化技術の開発も検討する．

本研究では性能評価のためのグラフアルゴリズムとして BFS や SSSP，PageRank などベンチマークで用いられる基本的なアルゴリズムを採用したが，サイクルパスの検出 [8] や Graph Convolution Network (GCN)[11] など，より実践的で計算量が頂点数・エッジ数の線形を超えるアルゴリズムでの性能評価と最適化も重要である．例えばサイクルパスの検出は，前述した金融取引ネットワークでは不正取引検出のアルゴリズムの一つとして用いられ，また GCN はソーシャルネットワークの頂点に相当するユーザの属性を推定するために用いられるなど実用面での需要が高い．一方で，いずれのアルゴリズムも計算コストが依然として高く，また実行中により多くのメモリ空間を必要とするが，その分本提案手法を適用させた効果がさらに顕著になると期待できる．そこで，以上のようなアルゴリズムを DCPMM 上で実行できるように最適化手法を拡張し，さらなる性能評価を行うつもりである．

参考文献

[1] Beamer, S., Asanović, K. and Patterson, D.: Direction-Optimizing Breadth-First Search, *SC12, November*, pp. 10–16 (2012).
[2] Beamer, S., Asanović, K. and Patterson, D.: The GAP benchmark suite, *arXiv preprint arXiv:1508.03619* (2015).
[3] Brandes, U.: A faster algorithm for betweenness centrality, *Journal of mathematical sociology*, Vol. 25, No. 2, pp. 163–177 (2001).
[4] Gill, G., Dathathri, R., Hoang, L., Peri, R. and Pingali, K.: Single machine graph analytics on massive datasets using Intel Optane DC Persistent Memory, *arXiv preprint arXiv:1904.07162* (2019).
[5] Gregg S.: Process and Thread Affinity for Intel® Xeon Phi™ Processors — Intel® Software, https://software.intel.com/en-us/articles/process-and-thread-affinity-for-intel-xeon-phi-processors-x200 (2016).
[6] Hirofuchi, T. and Takano, R.: A Prompt Report on the Performance of Intel Optane DC Persistent Memory Module, *IEICE Transactions on Information and Systems*, Vol. E103.D, No. 5 (2020).
[7] Iwabuchi, K., Sato, H., Yasui, Y., Fujisawa, K. and Matsuoka, S.: NVM-based Hybrid BFS with memory efficient data structure, *2014 IEEE International Conference on Big Data (Big Data)*, pp. 529–538 (2014).
[8] Johnson, D. B.: Finding all the elementary circuits of a directed graph, *SIAM Journal on Computing*, Vol. 4, No. 1, pp. 77–84 (1975).
[9] Karypis, G. and Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on scientific Computing*, Vol. 20, No. 1, pp. 359–392 (1998).
[10] Kim, M. and Candan, K. S.: SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices, *Data & Knowledge Engineering*, Vol. 72, pp. 285–303 (2012).
[11] Kipf, T. N. and Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks, *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, (online), available from ⟨https://openreview.net/forum?id=SJU4ayYgl⟩ (2017).
[12] Patil, O., Ionkov, L., Lee, J., Mueller, F. and Lang, M.: Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel Optane DC Persistent Memory Modules, *Proceedings of the International Symposium on Memory Systems*, pp. 288–303 (2019).
[13] Petroni, F., Querzoni, L., Daudjee, K., Kamali, S. and Iacoboni, G.: Hdrf: Stream-based partitioning for power-law graphs, *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 243–252 (2015).
[14] Stanton, I. and Kliot, G.: Streaming graph partitioning for large distributed graphs, *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1222–1230 (2012).
[15] Sutton, M., Ben-Nun, T. and Barak, A.: Optimizing Parallel Graph Connectivity Computation via Subgraph Sampling, *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 12–21 (online), DOI: 10.1109/IPDPS.2018.00012 (2018).
[16] Suzumura, T., Ueno, K., Sato, H., Fujisawa, K. and Matsuoka, S.: Performance characteristics of Graph500 on large-scale distributed environment, *2011 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, pp. 149–158 (2011).
[17] The Graph 500 List: Benchmark Specification — Graph 500 (2017).
[18] Ueno, K. and Suzumura, T.: Highly scalable graph search for the Graph500 benchmark, *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pp. 149–160 (2012).
[19] Ueno, K., Suzumura, T., Maruyama, N., Fujisawa, K. and Matsuoka, S.: Efficient breadth-first search on massively parallel and distributed-memory machines, *Data Science and Engineering*, Vol. 2, No. 1, pp. 22–35 (2017).
[20] 岩渕圭太， 佐藤仁， 溝手竜，安井雄一郎，藤澤克樹，松岡聡：不揮発性メモリを用いた Hybrid-BFS アルゴリズムの最適化と性能解析，情報処理学会第 141 回 HPC 研究会報告，Vol. 2014-HPC-141, No. 18 (2014).
[21] 田邊 昇，遠藤敏夫：中遅延大容量メモリ階層出現のインパクトと新たな対応に関する初期検討，技術報告 11，東京工業大学 (2016).