Vol.2013-HPC-138 No.31
2013/2/22

不揮発性メモリを用いたGraph500ベンチマークの大規模実行へ向けた予備評価

岩渕 圭太[1,2,a)]　佐藤 仁[1,2]　安井 雄一郎[3]　藤澤 克樹[3,2]　松岡 聡[1]

概要：近年大規模グラフはさまざまな分野で出現しており，DRAM の容量を増設することによる消費電力増加の問題やそもそもシングルノード上の DRAM 容量を超えるグラフも出現している．
本研究では Graph 500 ベンチマークに対して不揮発性メモリを補助的に利用することで性能低下を最小限に押さえながらシングルノード上でできる限り大容量のグラフを扱えるようにすることを目指している．そこでまず本論文では DRAM に乗りきらない問題サイズを実行するための手法を提案し，DRAM と不揮発性メモリの容量の比率が実行性能にどのような影響を与えるかについての予備評価を行った．

1. 背景

近年，SNS 解析，道路ネットワークの経路探索，スマートグリッド，創薬，遺伝子解析等の応用で，大規模グラフが出現しており，数百万頂点〜数兆頂点，数億エッジ〜数百兆エッジからなる超大規模なグラフに対する高速処理が求められている．例えば，現存する SNS における交友関係は 8 億頂点，1000 億エッジからなる大規模グラフによって表現される．単一の計算ノードでは保持できない規模のサイズのグラフも登場しはじめている一方で，一般に，グラフ処理は，参照の局所性が低く，また，メモリアクセスは非構造なものとなるため，全てのデータを DRAM へロードして実行しなければ，妥当な性能で実行することは難しい．しかし，大容量の DRAM の導入は，非常に価格の面でのコストが高く，また，消費電力も増大してしまうため望ましくない．

一方で，近年，フラッシュデバイスなどのような，DRAM と比較するとアクセスレイテンシやスループットなどの性能面で劣るものの，容量あたりの価格，消費電力の点で優れたデバイスが登場し，普及している．このようなフラッシュデバイスを DRAM に対して補助的に利用しすれば，単一の計算ノード上の DRAM には収まらない容量のグラフを実行できる可能性があるものの，その具体的手法やどの程度の性能低下が起きるのかなどの定量的な指標は明らかではない．

本論文では DRAM 容量を超えるグラフを扱う手法を提案し，不揮発性メモリと DRAM の容量の比率が性能にどのような影響を及ぼすのかについて予備評価を行った．その結果，forward graph を不揮発性メモリに退避することで Top-down の性能が低下しても，探索方向を切り替えるパラメータを調整することで Top-down の性能が低下しても本手法が有効である事が示唆された．

2. Graph 500 ベンチマーク

ここではまず，Graph 500 ベンチマークの概要を示し，次に，我々が対象とする BFS アルゴリズムについて述べる．

2.1 Graph 500 ベンチマークの概要

Graph 500 ベンチマーク [1] とは，データインテンシブなワークロードに対するスーパーコンピュータの性能ランキングである．従来の性能ランキングである Top500 は計算インテンシブなワークロードに対して競われてきたのに対し，Graph 500 ではデータインテンシブなワークロードの一つであるグラフ探索の性能が競われる．具体的には，クロネッカーグラフと呼ばれる，実グラフと似た性質を持つグラフを生成し，そのグラフに対して，ランダムに選ばれた探索開始点から繰り返し幅優先探索 (BFS) を行い，それぞれの BFS に対して結果の検証を行う．

Graph 500 で生成される，クロネッカーグラフには以下のような性質がある．

(1) スケールフリー性 (次数分布のべき乗則)
　各頂点の保持するエッジ数（次数）の分布がべき乗則

[1] 東京工業大学
Tokyo Institute of Technology
[2] JST CREST
JST CREST
[3] 中央大学
Chuo University
[a)] iwabuchi.k.ab@m.titech.ac.jp

に従っている性質である．一部の頂点が他の多くの頂点とエッジで繋がっており，大きな次数を持っている一方で，他の多くの頂点はごくわずかな頂点としか繋がっておらず，次数は小さい．この特性はロード・インバランスを引き起こし，分散メモリ環境での性能劣化を引き起こす原因となり得る．

(2) スモールワールド性
任意の頂点間の距離が頂点数が多くても極めて小さい性質である．Graph 500 では頂点数は数億個であったとしても任意の頂点間は 6 ホップ程度で探索できる．

次にベンチマークの流れについて説明する．ベンチマークは以下 4 つのステップからなる．

(1) エッジリストの生成
無向グラフのエッジリストを生成する．グラフサイズは頂点数が 2^{SCALE}，エッジ数は頂点数*$Edgefacter$ で表される．

(2) データ構造の変換
生成されたエッジリストを BFS の実行に適切なデータ構造に変換する．

(3) BFS の実行
変換されたグラフデータに対して BFS を実行する．この時の実行時間とグラフが持つ全エッジ数から性能が決まる．性能は単位時間に処理されたエッジ数である TEPS（Traversed Edges Per Second）値として表される．

(4) 探索結果の検証
BFS によって求めた BFS 木に対して探索結果の検証を行う．

なお，エッジリストの生成とデータ構造の変換時に探索開始点として 64 個の点が選ばれ，BFS と結果の検証は 64 回行われる．

2.2 Hybrid-BFS アルゴリズム

我々の提案手法は，Hybrid-BFS [2] アルゴリズムを対象とする．Hybrid-BFS アルゴリズムは，探索する頂点数を削減するために，従来の Top-down で頂点を探索するアプローチに加えて，Bottom-up で頂点を探索するアプローチを補助的に利用して，この両者のアプローチをハイブリッドに切り替えながら，BFS を行う手法である．この手法は，現在の Graph500 ランキング上のカスタマイズされた実装で主流となっているものである．ここでは，Hybrid-BFS のアルゴリズムの詳細について述べる．

2.2.1 Top-down アプローチ

Top-down アプローチは，従来よりよく知られた古典的な BFS のアルゴリズムであり，Graph 500 ベンチマークのリファレンス実装などでも使用されている手法である．**図 4** にその概要と擬似コードを記す．各レベル毎の探索の際に，現在，訪問している頂点の集合を frontier とし，frontier に属している各頂点に対しそれぞれ隣接している全ての頂点を neighbor とする．このとき neighbor をチェックし，未訪問であれば訪問済とし、さらにその頂点を次のレベルの探索での frontier に追加する．訪問済であれば特に操作は行わない．このような探索を繰り返し行い，始点となる頂点から最初の隣接頂点を探索するレベルを第 0 レベルとすれば，第 n レベルで訪問した頂点は始点の頂点から n ステップで隣接していることを表す．

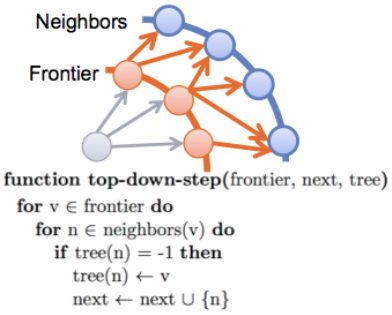

図 1 Top-down アプローチの概要と擬似コード

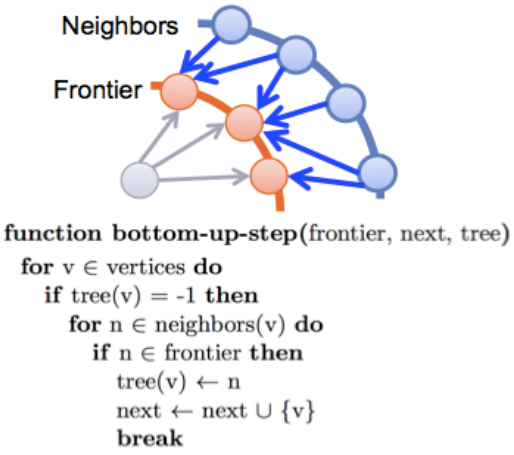

図 2 Bottom-up アプローチの概要と擬似コード

Top-down アプローチの欠点としては，frontier の数が多い場合に冗長な探索が増大していまい探索効率が悪くなってしまう，という点が挙げられる．これは，Top-down アプローチでは frontier に所属する頂点は自分が持つ全ての頂点に対して訪問済みかどうかの探索を行う．しかし，このアプローチでは，既に他の頂点によって訪問済みの頂点に対しても再度探索を行ってしまうため，冗長が探索が増えてしまう．また，頂点に対して訪問済かの確認を行う際にはアトミックな操作が必要になり，排他制御が必要となるため性能が低下してしまうという問題もある．

2.2.2 Bottom-up アプローチ

Bottom-up アプローチは，Top-down アプローチとは逆の方向に探索を行う．**図 2** にその概要と擬似コードを記す．Top-down アプローチが訪問済みの頂点を始点として，隣接

している未訪問の頂点を目指し探索を行うが，Bottom-upアプローチは全ての未訪問の頂点を始点とし，隣接する頂点の中に１つでも frontier に属する頂点があればその頂点を始点とした未訪問頂点を訪問済とし，親の頂点を辺の存在した frontier に属する頂点とする．この時，frontier に所属する頂点の一覧はビットマップを使用して保存されていると効率が良い．Bottom-up アプローチでは，未訪問の頂点が 1 つでも frontier に属する頂点への辺を見つければその時点で探索は終了するため冗長な探索を削減することができる．

2.2.3 Hybrid アルゴリズム

Top-down アプローチと Bottom-up アプローチを frontier の数によって切り替えるのが Hybrid アルゴリズムである．まず，frontier 数が多くなると Top-down アプローチでは未訪問の neighbor を探すための冗長な探索，例えば，複数の頂点が同じ頂点の親になろうとする等，が行われてしまう．逆に，Bottom-up アプローチでは，frontier に含まれる頂点が１つでも見つかればよいため，探索の解となる頂点の候補の数が多くなり，探索の効率が向上する．一方，frontier に含まれる頂点の数が減少した場合，Top-down アプローチでは，frontier 上の頂点が neighbor 上の同じ頂点の親になろうとする回数は減ることになり，他方，Bottom-up アプローチでは，frontier 上にある探索の解の候補となる頂点が少なくなり，探索効率が悪くなる．

Top-down と Bottom-up のアプローチを切り替える基準としては，frontier 数，frontier から出ている辺の総数，未探索頂点の数などを使用する手法が提案されているが，今回は，frontier に含まれる頂点の数が全探索の候補となる頂点の数の中で一定の割合を超えた場合に，探索方向の切替を行うこととした．

3. DRAM 容量を超えるグラフサイズ実行の方針

Graph500 ベンチマーク [1] は，大きく分けて，以下のようなステップで実行が行われる．

(1) グラフの生成
(2) BFS 探索用グラフデータの構築
(3) BFS 探索
(4) BFS 探索の結果の検証

我々が対象にする実装では，(3) の部分に Hybrid-BFS のアルゴリズムを用いている．ここでは，まず，Hybrid-BFS アルゴリズムにおいてどのようなデータが使用され，DRAM のみを使用した実装ではどのようにグラフデータが扱われているかを示し，最後に DRAM 容量を超えるグラフサイズ実行の方針について提案する．

3.1 Hybrid-BFS アルゴリズムの実装で使用するデータ

今回対象とする Hybrid-BFS アルゴリズムを用いた実装では以下のデータを使用する．

(1) エッジリスト
　生成されたクロネッカーグラフのエッジ一覧をタプル形式で保持している．
(2) BFS 探索用グラフデータ
　Top-down アプローチと Bottom-up アプローチの探索用に，それぞれエッジリストを CSR 形式に変換して持つ．以下，Top-down アプローチ用グラフデータを forward graph，Bottom-up アプローチ用のグラフデータを backward graph とする．
(3) BFS データ
　探索結果 (BFS 木）や探索に用いる Queue などを持つ．

Graph500 ベンチマークの各ステップにおいて，グラフ生成ではエッジリスト，BFS 探索用グラフデータ構築ではエッジリストと BFS 探索のためのグラフデータ，Hybrid-BFS アルゴリズムの探索時は探索用グラフデータと BFS データ，そして，探索した結果の検証時にはエッジリストと BFS データを使用する．よって，既存実装ではこれらのデータを全て DRAM 上に配置しながら，Graph500 ベンチマークの実行を行う．

3.2 データ書き出しの局所性を高めたグラフ分割

著者の 1 人である安井らは，現在，Hybrid-BFS アルゴリズムを DRAM のみを使用した実装を行っており，2012 年 11 月の Graph 500 リストにおいて 1 台の計算ノード上で．10.5 GTEPS を記録している [1]．この実装では，NUMA ノードを考慮し，Top-down アプローチと Bottom-up アプローチにおいてそれぞれデータ書き出しの局所性を高めた探索を行うため，探索用のグラフデータについて．Top-down アプローチと Bottom-up アプローチ各々に関して探索用のグラフデータを CSR の形式で持っている．

Top-down アプローチでは，frontier にある頂点を始点とし，隣接する頂点を終点として探索を試みる．通常，各頂点は隣接する頂点の情報を全て持つが，今回，対象としている実装では，全ての探索対象の頂点は，NUMA ノード毎に分割されて管理され，例えば，ある NUMA ノードに属する始点の頂点は，同じ NUMA ノードに属する終点の頂点のみを探索し，終点の頂点が他の NUMA ノードに属するような場合は，探索を他の NUMA ノードに属する頂点に委ねる．このようなデータ配置を行うことによって，頂点が訪問済みかどうかの確認を局所的なデータアクセスにすることができる．

一方で，Bottom-up アプローチに使用する backward graph については，探索の方向が逆になり，未訪問の頂点から frontier に含まれる頂点を探す．そのため，backward graph では，各 NUMA ノードは担当する終点の頂点を分割

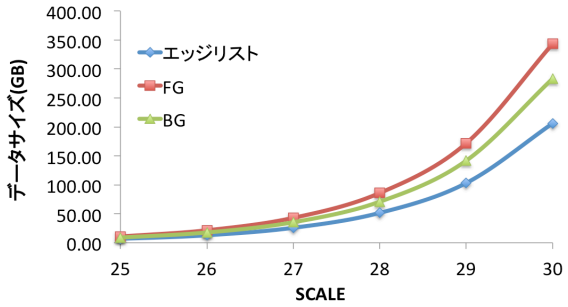

図 3 スケールを変化させた場合のデータサイズ

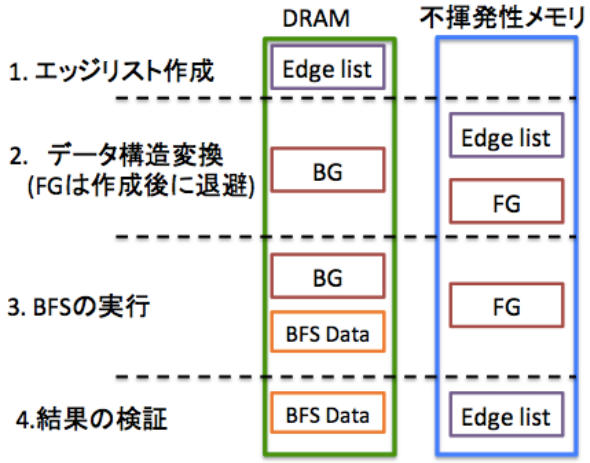

図 4 DRAM と不揮発性メモリを用いたデータ配置

して持ち，始点の候補となる頂点の情報を各 NUMA ノードで持つことにより，frontier へ探索を行った場合でも頂点が訪問済みかどうかの確認を局所的なデータアクセスにすることができる．

3.3 不揮発性メモリを用いたグラフデータの配置

不揮発性メモリを用いた場合のデータ配置方法について提案する．不揮発性メモリを使用して大規模グラフに対する BFS を実行する方法としては，文献 [3] らの指摘や，BFS データのみを DRAM 上に乗せグラフデータ自体は不揮発性メモリに載せる手法がある [4]．BFS データを DRAM 上に固定する理由としては，BFS データは探索中に使用するキューやビットマップであり，探索中に頻繁にアクセスが行われ探索の性能に大きな影響を及ぼすためである．

しかし，この方針では DRAM に全てを乗せた場合と比べてパフォーマンスが大きく低下してしまう．そこで，探索用グラフデータの一部を DRAM 上に残す手法について検討を行う．具体的には forward graph を全て不揮発性メモリに退避する手法を提案する．

各グラフサイズの違いを調べるため，SCALE を変化させた場合のデータサイズを**図 3** に示す．グラフデータは頂点数に比例するので，SCALE が上がればグラフデータも指数関数的に上昇する．SCALE が 31 になると edge list, forward graph(FG), backward graph(BG) の合計は 1.6TB になる．また，forward graph は backward graph よりもサイズが大きいことがわかる．これは，グラフ分割の仕方に起因する．forward graph ではそれぞれの頂点におけるエッジ数は少ないが全頂点についてのエッジ情報を持っており CSR 形式でデータを管理する場合はインデックスを表す配列が全頂点分の長さになる．一方で，backward graph はそれぞれの頂点におけるエッジ数は多いが，頂点数は少ないため CSR 形式でのインデックス配列は短くなる．

Hybrid-BFS アルゴリズムでは bottom up アプローチによって探索されるエッジ数が大半を占めており，探索された全エッジ数の内 forward graph が占める割合は極めて小さいことが知られている [SC]．そこで，Hybrid-BFS アルゴリズムに対して不揮発性メモリを使用して大規模グラフを扱う方針として探索に使用されるエッジ数が少なく，性能に影響の少ない forward-graph を不揮発性メモリに退避する．

さらに，Bottom-up 探索では frontier の頂点へのエッジが見つかった時点で探索を終了するため，参照されないエッジが存在する．特に，Graph 500 ベンチマークで使用するようなスケールフリー性のあるグラフでは，次数が大きい頂点において参照される可能性の低いエッジが多数あり，backward graph についても参照される可能性の低いデータを不揮発性メモリに退避することで，性能の低下抑えながら DRAM の容量を削減できる．

4. 評価

提案手法を用いた場合に不揮発性メモリと DRAM の容量の比率が実行性能にどのような影響を与えるのかについて予備評価を行う．また，さらに DRAM へ載せるグラフデータ容量を削減するための，backward graph の一部を退避する手法について予備評価を行う．

評価環境は必要なグラフデータが全て DRAM 上に確保できるケースと DRAM 容量が足りないケースで比較を行うために，DRAM 容量が 64GB と 128GB の 2 種類の環境を用意した．DRAM 容量が少ないノードでは SSD（600GB, Intel SSD 320 Series MLC SATA）を搭載している．両者の違いはメモリー容量のみであり，CPU は AMD Opteron（tm） 6172 （6 コア）が 8 ソケット，コンパイラは GCC 4.4.5 の-O2 オプションを使用している．

4.1 DRAM と不揮発性メモリの比率を変化させた場合の性能

不揮発性メモリを用いたグラフ探索の性能を調べるため，全てのグラフデータを DRAM に載せた場合 (DRAM only) と，forward graph を不揮発性メモリに退避した場合 (DRAM+SSD) の性能比較を行う．

今回実験に使用したグラフサイズを**表 1** に示す．グラフ

表 1 データサイズ (SCALE 27, Edge factor 16)

Forward graph	40.1GB
Backward graph	33.1GB
BFS data	15.1GB

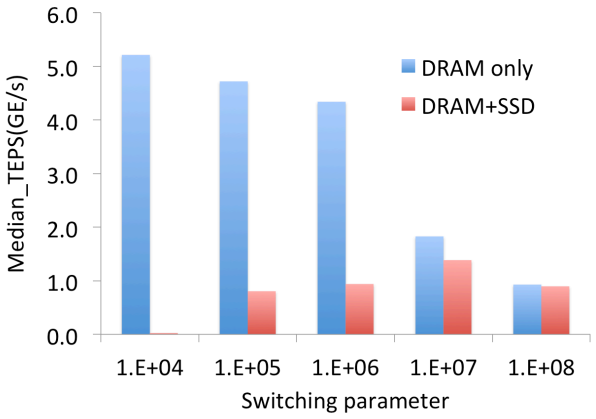

図 5 切替の閾値を変化させた場合の平均 TEPS(GE/s) 値

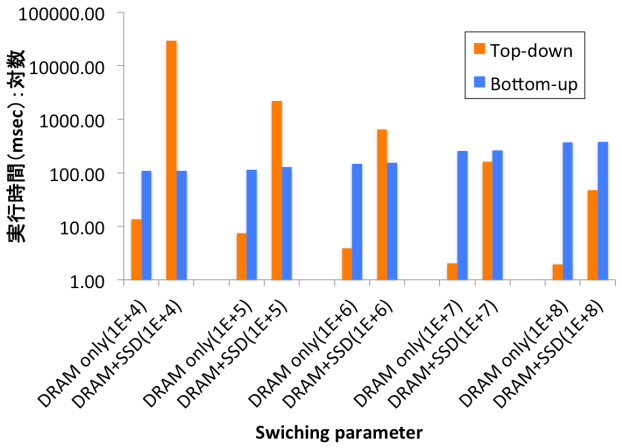

図 6 Top-down と Bottom-up 探索の実行時間

サイズは全て Scale 27, Edge factor 16 としている．backward graph と BFS data の領域 (合計 48.2GB) を DRAM 上に確保しているため，DRAM が 64GB の環境では Forward graph(40.1GB) を同時に DRAM 上に乗せることはできない．探索方向の切替を行う Switching parameter は，DRAM only において TEPS 値が最大となることが観測された 1.E+04 から，Bottom-up の割合が増える方向に 10 倍づつ 1.E+08 まで変化させた (**図 5**, **図 6**).

4.1.1 評価結果

図 5 より，DRAM only では Switching parameter を上昇させると性能が低下するのに対し，DRAM+SSD では 1.E+07 までは中央値で 1.38GTEPS となる性能向上を示し，DRAM only と比較しても 26.6%まで性能の低下を抑える性能を示した一方で，1.E+08 では性能低下が見られた．図 6 より，DRAM only については，Switching parameter を上昇させる Top-down アプローチの方が効率の良い探索に対しても Bottom-up アプローチによる探索が行われるため性能が減少してしまうことが観測された一方，DRAM+SSD では Bottom-up アプローチによる探索の性能低下を上回る Top-down の性能向上が観測された．

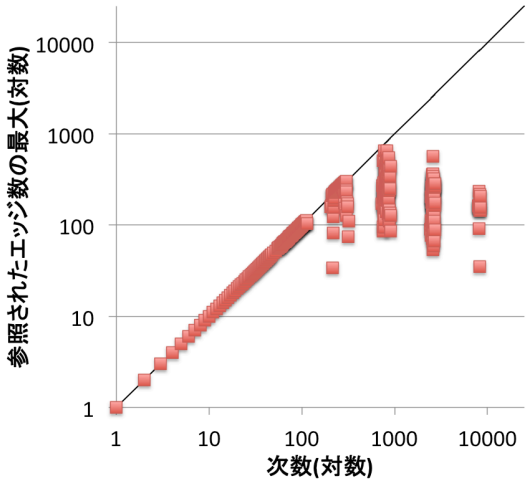

図 7 参照されたエッジ数（最大値）

表 2 backward graph に対して DRAM に乗せる最大エッジ数を変更した場合の DRAM サイズと SSD にあるエッジの割合

最大エッジ数	削減後のサイズ	参照されるエッジの内 SSD 上にあるエッジの割合
100	47%	1.740%
500	73%	0.003%
1000	87%	0.0%

4.2 backward graph の退避に向けた予備評価

DRAM に載せるグラフデータ容量をさらに削減する手法として，backward graph の一部を退避する手法について予備評価を行う．実際に探索に使用されたエッジ数について Scale 16, Edge factor 16 のグラフについて分析を行った．

4.2.1 評価結果

図 7 は 1 回のベンチマーク（同じグラフに対して始点を変えながら 64 回の BFS を実行）において各頂点に対して参照されたエッジ数を次数が同じ頂点毎にまとめ，参照された回数の最大値をプロットしたものである．次数が 100 以下の頂点では全てのエッジが参照されているが，次数が 2,000 を超えると参照されるエッジ数は一定の値以上は増加せずに，最大でも次数の 15%程度になる．参照したエッジ数が最も多いのは次数が 850 の頂点に対して 655 個のエッジを参照している．

表 2 は SCALE 16, Edge factor 16 のグラフに対して，DRAM 上に各頂点が DRAM に乗せるエッジ数の上限を調整した場合の backward graph のデータサイズである．例えば，上限エッジ数を 100 とすれば，次数が 100 以下の

頂点は全てのエッジを DRAM に乗せ，次数が 100 より多い頂点は 100 個までのエッジを DRAM 上に乗せ，超えたエッジは不揮発性メモリに退避することになる．上限エッジ数を 100 とすれば，データサイズは 47 %まで削減できるが，1.74%のアクセスは不揮発性メモリに対して行われる．

5. まとめ・今後の方針

大規模グラフ処理において消費電力やコストの問題から DRAM の容量を抑えながらシングルノード上でより容量の大きなグラフを扱えることが求められている．そこで，本研究では不揮発性メモリを利用することで性能低下を抑えながらより容量の大きなグラフデータを扱うことを目指している．本論文では DRAM 容量を超えるグラフを扱う手法を提案し，不揮発性メモリと DRAM の容量の比率が性能にどのような影響を及ぼすのかについて予備評価を行った．その結果，forward graph を不揮発性メモリに退避することで Top-down の性能が低下しても，探索方向を切り替えるパラメータを調整することで性能の低下を抑えることができた．また，backward graph についても一部のデータを性能の低下を抑えながら退避できることを確認し，本手法が有効である事が示唆された．今後の方針としては,Backward graph の一部を実際に 退避した場合の性能評価,Fusion-io NAND Flash などの デバイスの使用しての性能評価がある．

参考文献

[1] Graph500: `http://www.graph500.org/`.
[2] Beamer, S., Asanović, K. and Patterson, D.: Direction-optimizing breadth-first search, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, Salt Lake City, USA, IEEE Computer Society Press, pp. 12:1—-12:10 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2389013⟩ (2012).
[3] Van Essen, B., Pearce, R., Ames, S. and Gokhale, M.: On the Role of NVRAM in Data-intensive Architectures: An Evaluation, *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, Ieee, pp. 703–714 (online), DOI: 10.1109/IPDPS.2012.69 (2012).
[4] Pearce, R., Gokhale, M. and Amato, N. M.: Multithreaded Asynchronous Graph Traversal for In-Memory and Semi-External Memory, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC10)*, IEEE, pp. 1–11 (online), DOI: 10.1109/SC.2010.34 (2010).