Vol.2013-HPC-141 No.3
2013/9/30

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

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

概要：近年さまざまな分野で大規模なグラフに対する高速な処理が求められているが，その処理の特性上，妥当な性能を得るためには全てのデータを DRAM 上にロードして実行する必要があり，その結果，DRAM の容量を増設することによる消費電力，価格面でのコストの増加が問題となっている．そこで，Hybrid-BFS アルゴリズムに対して不揮発性メモリを補助的に利用した場合の I/O の最適化，性能低下要因の解析を行うことで性能低下を抑えながら大規模グラフ処理が実行可能かの評価を行った．その結果，一部データを不揮発性メモリに退避することで DRAM 用量が半分の環境において性能低下を 47.1%まで抑えることができた．また，参照され難いエッジデータをさらに退避することで性能の低下を抑えながらより DRAM 使用量が削減可能なことの確認，さらに，性能低下要因の特定とその改善案を示し，性能低下を抑えながら大規模グラフ処理の実現可能性が示唆された．

1. はじめに

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することで，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．

これまで著者らは Graph500 ベンチマークにおいて，Hybrid-BFS アルゴリズムに対して不揮発性メモリを補助的に利用した場合にアルゴリズムの特性を考慮しアクセス数の少ないデータを不揮発性メモリに退避することで性能低下を最小限に抑えながら DRAM 容量を超えるグラフを扱う手法を提案し，どの程度性能の低下が起きるか予備評価を行ってきた [1]．そこで，本論文では Graph500 ベンチマークにおいて，Hybrid-BFS アルゴリズムに対して不揮発性メモリを補助的に利用した場合の I/O の最適化，性能低下要因の解析を行うことで性能低下を抑えながら大規模グラフ処理が実行可能かの評価を行った．その結果，一部データを不揮発性メモリに退避することで DRAM 用量が半分の環境において性能低下を 47.1%まで抑えることができ，さらに，次数の小さい頂点が大量にあることによる非効率な I/O やデバイスの IOPS 性能が低く，性能低下の要因となっていることを確認し，対策案を提示する．また，参照され難いエッジデータをさらに退避することで性能の低下を抑えながら，より DRAM 使用量が削減可能なことの確認した．よって，不揮発性メモリを用いることで性能低下を最小限に抑えながら大規模グラフ実行の可能性が示唆された．

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

2. Graph 500 ベンチマーク

Graph 500 ベンチマーク [2] とはグラフ探索の性能を利用したベンチマークである．ここではまず，Graph 500 ベンチマークで使用される幅優先探索 (BFS) について説明し，次に Graph 500 ベンチマークの概要を示し，そして最後に，我々が対象とする BFS アルゴリズムについて述べる．

2.1 幅優先探索（BFS : Breadth First Search）

幅優先探索 (BFS) とは，重み無しグラフの探索において，ある頂点から他の頂点への最短経路を求める手法の一つである．探索は始点となる頂点からレベル（深さ）単位でステップを進めながら行い，あるレベルの頂点が全て探索されてから次のレベルの探索を行う．まず始点となる頂点（level 0）から隣接する全ての頂点（level 1）を探索する．次に，level 1 の頂点を始点とし，同様に，隣接する頂点（level 2）を全て探索する．レベル $i+1$ の頂点は，レベル i の頂点に隣接し，且つ，これまでに訪問されていない全ての頂点である．探索の始点となる頂点をレベル 0 とすれば始点に隣接する頂点の集合はレベル 1 となり，レベル i で訪問した頂点は始点から i ホップで隣接していることになる．

2.2 Graph 500 ベンチマークの概要

Graph 500 ベンチマークは，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 [3] 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 BFS を行い，始点から他の全頂点までの最短経路を求め，最後に結果の検証を行う．

Graph 500 で使用される，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．例えば Graph 500 ベンチマーク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*$Edgefactor$ 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．

Graph 500 ベンチマークでは，ベンチマークは探索開始点としてランダムに 64 個の点が選ばれ BFS を行う．よって，BFS の実行と探索結果の検証の組み合わせが 64 回行われ，64 回の BFS における TEPS 値の中央値がランキングに使用される．

3. Hybrid-BFS アルゴリズム

我々の提案手法は，Hybrid-BFS[4] 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において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．

3.1 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 [5]. **図 1** 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 neighbors とする．このとき neighbors 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

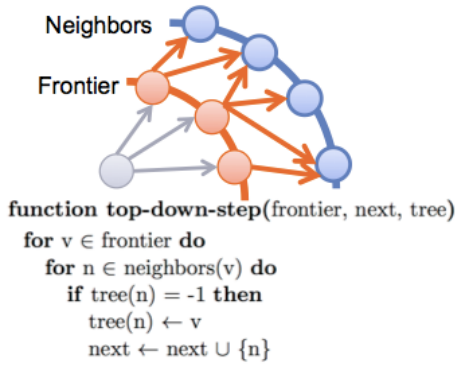

図 1 Top-down アプローチの概要と擬似コード

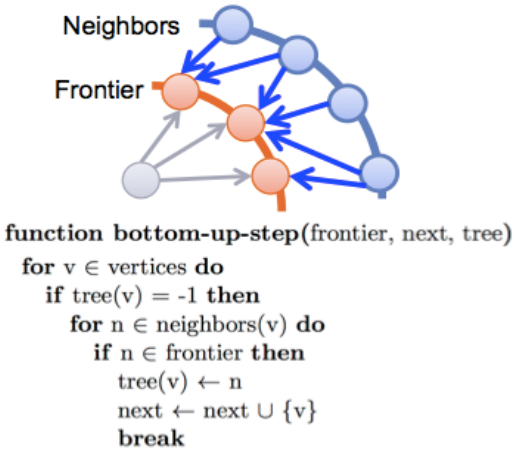

図 2 Bottom-up アプローチの概要と擬似コード

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．

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隣接する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，ある頂点に対して訪問済みとする操作を行う際にはアトミックな操作が必要になり，性能が低下してしまうという問題もある．

3.2 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未訪問頂点を訪問済とし，親の頂点をエッジの存在した 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 に属する頂点へのエッジ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，性能の向上が期待できる．

3.3 アプローチの切替

Top-down アプローチと Bottom-up アプローチを探索の状況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，が行われてしまう可能性が上昇する．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，frontier 数が多くなると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 数が減少した場合，Top-down アプローチでは，frontier 上の頂点が neighbors 上の同じ頂点の親になろうとする冗長な回数は減ることになり，探索効率は悪化しない．他方，Bottom-up アプローチでは，frontier に属す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 から出ているエッジの総数，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，レベル i おける frontier に含まれる頂点の数 ($n_{frontier(i)}$) が全ての頂点の数 (n_{all}) に対して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．具体的には探索方向切を切替えるためのパラメータ α, β を用いて，Top-down アプローチから探索を開始し，BFS のレベルが i の場合に，$n_{frontier(i-1)} < n_{frontier(i)}$ 且つ，$n_{frontier(i)} > \frac{n_{all}}{\alpha}$ の場合に Top-down アプローチから Bottom-up アプローチに切替え，$n_{frontier(i-1)} > n_{frontier(i)}$ 且つ，$n_{frontier(i)} < \frac{n_{all}}{\beta}$ の場合に Bottom-up アプローチから Top-down アプローチに切替える．本研究において，Graph 500 で使用されるグラフを対象として Hybri-BFS アルゴリズムを実行した場合には，1 回の BFS は 7 レベル程度で終了し，Top-down アプローチから始まり，Bottom-up アプローチに切替わり，最後 Top-down アプローチに戻り探索が終了する．探索のアプローチを切り替えるタイミングの詳細については [6] を参照されたい．

4. Hybrid-BFS アルゴリズムの特性を考慮したグラフデータの退避方針

Graph500 ベンチマークは，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 用グラフデータの構築

(3) BFS 実行
(4) BFS 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述べる．

4.1 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 ステータスデータ
探索結果（BFS 木）や探索に用いる Queue，ビットマップなどを持つ．

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 アルゴリズムを用いた探索時は探索用グラフデータと BFS ステータスデータ，そして，探索した結果の検証時にはエッジリストと BFS ステータスデータを使用する．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．

4.2 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 [7]，2013 年 6 月の Graph 500 リストにおいて 1 台の計算ノード上 (4-way Intel Xeon E5-4640) で，11.1GTEPS を記録している [2]．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，ある頂点を訪問済みとする書き込みを確実にローカルの NUMA ノードに行うことができる．

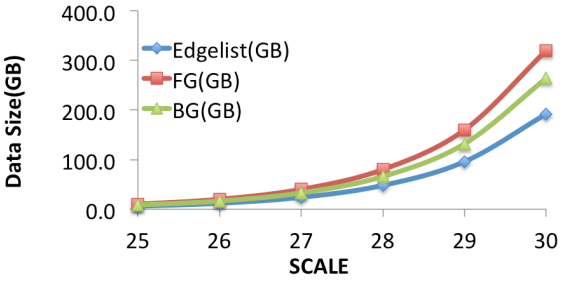

図 3 スケールを変化させた場合のデータサイズ

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ある頂点を訪問済みとする書き込みを確実にローカルの NUMA ノードに行うことができる．

4.3 不揮発性メモリを用いたグラフデータの配置

著者らはこれまで不揮発性メモリを用いた場合のデータ配置方針を提案している [1]．ここでは，その方針について述べる．既存手法として，グラフデータを全て不揮発性メモリに退避する手法があるが，これでは性能が大きく低下してしまう [8]．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 [9] らの指摘や，BFS ステータスデータのみを DRAM 上に乗せグラフデータは不揮発性メモリに載せる手法がある [10]．BFS ステータスデータを DRAM 上に固定する理由としては，BFS ステータス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．

そこで我々の提案手法は，BFS ステータスデータを DRAM 上に固定すると共に，全てのグラフデータを退避するのではなく，Hybrid-BFS アルゴリズムの特性を考慮し，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．

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ため，SCALE が上がればグラフデータ

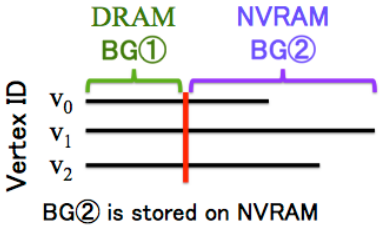

図 4 BG の一部を退避する手法

も指数関数的に上昇する．例えば，SCALE が 31 の場合では edge list, forward graph(FG), backward graph(BG) の合計は 1.5TB になる．なお，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．

提案手法 1

Hybrid-BFS アルゴリズムでは Bottom-up アプローチによって使用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 [4]．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の 1 つ目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し，Top-down アプローチ中は不揮発性メモリからエッジを読み込むようにする．

提案手法 2

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．backward graph では，各頂点毎に隣接している頂点 ID が連続した領域に保持されており，この領域に先頭からアクセスし frontier の頂点が見つかるまで探索を行う．よって，より後半の領域の方が参照される確率が低くなるため，2 つ目の提案手法として参照される可能性が高い前半のエッジを DRAM に保持し，参照される確率の低い後半の領域を不揮発性メモリに退避する（**図 4**）．Graph 500 ベンチマークで使用するようなスモールワールド性のあるグラフでは，特に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 の使用量を削減できると考える．

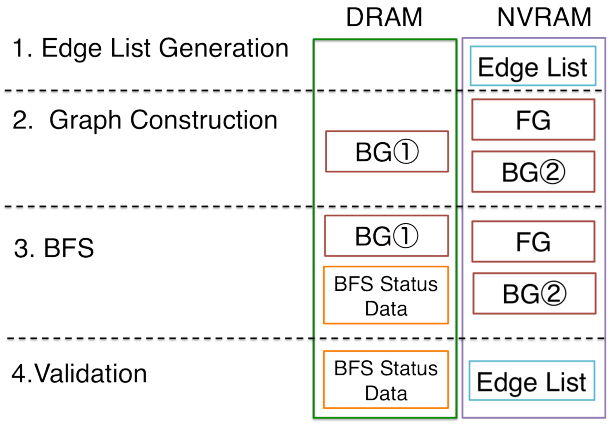

図 5 DRAM と不揮発性メモリを用いたデータ配置

よって，**図 5** のようなデータ配置を取ることで Graph 500 ベンチマークを行う．エッジリスト，forward graph（FG）や backward graph（BG）については不揮発性メモリ（NVRAM）を用いながら，グラフの生成・構築を行い，BFS は backward graph の一部と，BFS ステータスデータを DRAM 上に配置し探索を行う．最後，探索結果の検証時にはエッジリストを不揮発性メモリから読み込みながら，DRAM 上にある BFS ステータスデータと合わせて結果の検証を行う．

5. 評価

提案手法を用いた場合に不揮発性メモリと DRAM の使用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評価を行った．

5.1 評価 1：DRAM と不揮発性メモリの比率を変化させた場合の性能

まず提案手法 1 について forward graph を不揮発性メモリに退避した場合にどの程度性能が低下するのか確認するために提案手法 1 について実装を行い，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) の性能比較を行った．

実装

本論文では forward graph をファイルとして不揮発性メモリ上に保存し，POSIX 準拠の read(2) 関数を用いて，頂点毎に連続領域分，最大で 4KB 単位で読み込む．アクセスレイテンシーの大きいデバイスからデータを読み込む際に連続領域はできる限りまとめて読み込むことでレイテンシーを抑えることができるが forward graph を使用する Top-down アプローチでは探索を行う頂点の順番はランダムになってしまう．しかし，各頂点毎に保持するエッジを読み込むときは連続アクセスになるため，まとめて読み込むことで不揮発性メモ

表 1 データサイズ (SCALE 27, Edge factor 16)

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

リから読み込む際のレイテンシーを抑えることができる．連続領域の長さが実行時間にどの程度影響を与えるのかについての詳細は 5.1.2 で述べる．

評価環境

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不揮発性メモリとして Intel 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 4 ソケット，コンパイラは GCC 4.4 の-O2 オプション，OS は Debian 6.0（Linux kernel 2.6）を使用している．

グラフサイズ

評価に使用したグラフサイズを**表 1** に示す．グラフサイズは Scale 27，Edge factor 16 としている．BFS のステップにおいて必要なデータである，forward graph，backward graph，BFS status data の合計は 88.3GB となり，DRAM 容量が 128GB のノードでは全てのデータを DRAM 上に確保する．DRAM 容量が 64GB の環境では backward graph と BFS status data が合計 48.2GB となり DRAM 上に確保し，forward graph(40.1GB) を不揮発性メモリ上に確保する．

探索方向の切替を行うパラメータ α 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，Top-down アプローチの割合が減少し，Bottom-up の割合が増えるように 10 倍づつ 1.E+07 まで変化させ，パラメータ β は同じく DRAM only において TEPS 値が最大となることが観測された 10α と Bottom-up アプローチから Top-down アプローチへ戻り難くするために 0.1α の 2 種類で比較を行った．

5.1.1 評価結果

図 6 は Switching parameter を 1.0E+04 から 1.0E+07 まで変化させた場合の結果である．Switching parameter α の値が上昇するにつれて Top-down アプローチから Bottom-up アプローチへ切替るタイミングが遅れ，Top-down アプローチの割合が減少し，Bottom-up アプローチの割合が上昇する．逆に，Switching parameter β の値は減少するにつれて Bottom-up アプローチから Top-down アプローチへ切替るタイミングが遅れるため，Topdown アプローチの割合が減少し，Bottom-up アプローチの割合が上昇する．

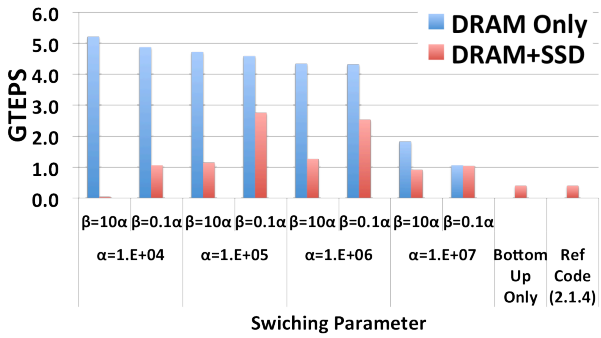

図 6 切替の閾値を変化させた場合の Median_TEPS(GE/s) 値

評価を行った結果，DRAM only では $\alpha = 1.E + 04, \beta = 10.0\alpha$ のとき最大で 5.2GTEPS であった．一方，DRAM+SSD ではパラメータを調整することで $\alpha = 1.E + 05, \beta = 0.1\alpha$ のとき最大で 2.8GTEPS となり，DRAM only の最大値に対して 47.1%まで性能の低下を抑えることができた．さらに，全ての探索を DRAM 上に確保したデータのみで探索を行う Bottom-up アプローチのみで探索を行ったところ，0.4GTEPS となった．また，DRAM 容量が 64GB の環境において参照実装 v2.1.4[2](SCALE=27, Edge factor=16) を実行した結果，0.04GTEPS となった．

なお, 同じく DRAM 容量 64GB の環境において, SCALE 27，Edge factor 16 とし，最初から使用するグラフデータとアプローチをどちらか一方のみに固定をし計測を行った．その結果，forward graph と BFS data のみを DRAM 上に確保し全ての探索を Top-down アプローチのみで行った場合の TEPS 値は 0.5GTEPS，図 6 で示したように，backward graph と BFS data のみを DRAM 上に確保し全ての探索を Bottom-up アプローチのみで行った場合は 0.4GTEPS となり，提案手法と同じく DRAM 容量を半分に減らすことはできるが，性能は大きく低下してしまい，Hybrid なアプローチを用いることの必要性が確認できた．

また，OS のスワップ機能を利用することで提案手法を用いず DRAM Only の実装のままで DRAM 容量を超えるサイズのグラフ処理を行う手法も考えられるが，同様に DRAM 容量 64GB の環境において，SCALE 27，Edge factor 16 とし性能を計測した結果，0.78MTEPS(但し，swiching parameter $\alpha = 1.E + 07$， $\beta = 10.0\alpha$) となり，提案手法のように明示的にグラフデータの配置や移動を制御する必要性が確認できた．

5.1.2 性能低下に対する考察

図 7 は，forward graph を退避した場合の Top-down 時間増加の原因について詳細を分析するために，Switching parameter α が $1.E + 04$〜$1.E + 07$，$\beta = 10.0\alpha$ の場合のベンチマーク結果について，各レベル単位での Top-down アプローチの DRAM Only に対する実行時間の倍率と頂点の平均次数の関係をプロットしたものである．なお，実

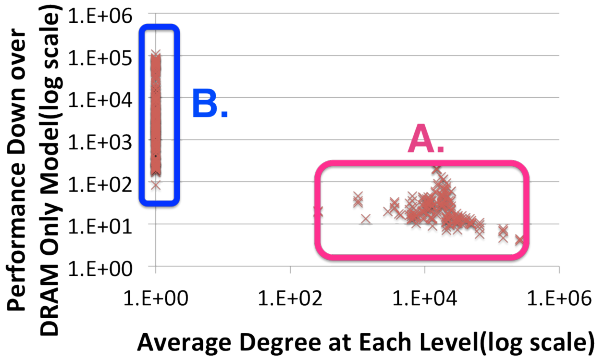

図 7 BFS の各レベルにおける DRAM Only に対する DRAM+SSD の実行時間の倍率と頂点の平均次数の関係（実行時間上位 50%のみ）

行時間が小さいものについては全体の性能に与える影響が小さいため，DRAM+SSD において実行時間が上位 50%のもののみについて考察する．

図 7 より，プロットした点は平均次数と実行時間の倍率の関係から明確に 2 つのグループに分類することができる．

グループ A.

平均次数が高いが実行時間の倍率が低い．実行時間の倍率は最大でも 207.6 倍，最小では 4.0 倍となる．

グループ B.

平均次数が低く（平均 1.001）実行時間の倍率が高い．実行時間の倍率は最小でも 84.2 倍，最大で 108047.5 倍となる．

このように 2 つのグループに分類できたのは，SSD から読み込みを行う際のバッファリングの効果が関係していると考えられる．次数が大きい場合は連続領域にアクセスするため SSD から読み込みを行う際にバッファリングの恩恵を受けやすいが，一方，次数が小さい場合にはバッファリングの効果がなく，実行時間が大きく上昇したと考えられる．実行時間の倍率が最大である 108047.5 倍の場合についてバッファリングの効果について詳細な分析を行った結果，参照されたエッジ間の距離が SSD の読み込みブロックサイズである 512 バイト以内である確率（一度のアクセスで複数のエッジを読み込める確率）は各頂点間の距離が最も小さくなるように頂点の ID 順にエッジをソートした場合でも 0.25%となり，アクセスパターンは非常にスパースなものとなりバッファリングの効果が得られ難いことが確認された．

また，評価を行ったグラフについてはどれも 7 レベル程度で探索が終了しており，Switching parameter の値に左右されるが Top-down アプローチから始まり最初の数ステップと後半の数ステップを Top-down アプローチが担当する．その際に，前半と後半の Top-down アプローチにおいて参照される頂点の平均次数について分析を行った結果，

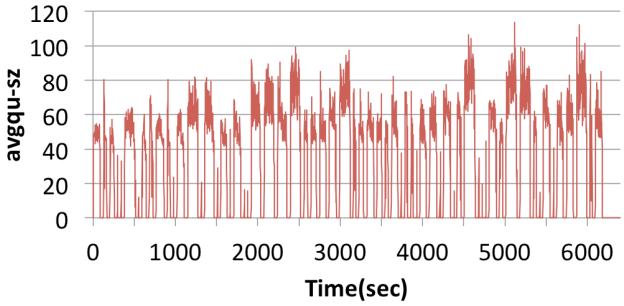

図 8 BFS 実行時の iostat の結果（avgqu-sz）

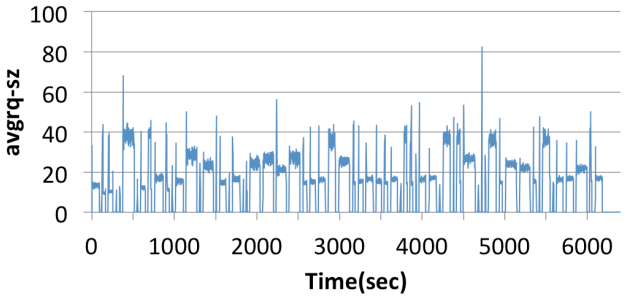

図 9 BFS 実行時の iostat の結果（avgrq-sz）

前半の Top-down アプローチでは平均次数は平均で 9634.6 と非常に大きな値である一方，後半の Top-down アプローチでは平均次数は最大でも 1.007 となり後半の Top-down 探索に次数が極めて小さい頂点が集中していることが観測された．

さらに，iostat コマンドを用いて，1 秒単位で BFS 実行時のデバイスの I/O 状況を調べた（swiching parameter $\alpha = 1.E + 04$, $\beta = 10.0\alpha$）．**図 8** と**図 9** は 1 回のベンチマークで 1 回目の BFS の実行時から，64 回目の BFS と結果の検証が完了するまでの iostat の結果である．実際には BFS を実行後，毎回，結果の検証ステップにおいてエッジリストの読み込みが発生するが，エッジリストを BFS に使用するグラフデータ（forward graph）とは別のデバイスに保存することで，BFS 時におけるデバイスの状態（forward graph の読み込み）のみを表示している．その結果，I/O キューの長さの平均を表す avgqu-sz の値が BFS 実行時の値のみを抽出すると平均で 56.1 となり，BFS 実行時は多くの I/O リクエストが処理待ち状態になっていることが観測された．これはより高性能なデバイスを用いることで性能の向上が期待できる一方，I/O リクエスト辺りの平均セクタ数を表す avgrq-sz の値は 22.8 となり，I/O の粒度は細かく，デバイスの性能を引き出せてない可能性が観測された．

よって，forward graph を不揮発性メモリに退避した場合に性能が低下する原因について，以下の 3 つが確認された．1）アクセスパターンは非常にスパースなものとなり，バッファリングの効果が得られ難い次数が小さい頂点が大

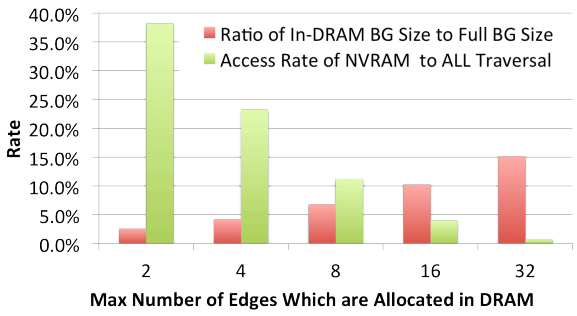

図 10 DRAM 上に保存するエッジ数の上限を変化させた場合の backward graph の一部を退避した場合の DRAM 使用量と不揮発性メモリへのアクセスの割合

量にある．2）I/O の粒度が細かく，デバイスの性能を引き出せていない可能性がある．3）デバイスの性能が低く，要求される I/O を処理できていない．

そこで，これらの問題点に対して，具体的な対策として，1）提案手法 2 で示すように DRAM の使用量を増加しても次数が小さい頂点のみは DRAM 上に確保する．2）高い IOPS 性能を持つデバイスや白幡らが提案しているような複数のフラッシュデバイスを RAID を組まずに並列的に使用するような仕組み [11] を使用する．が考えられる．

5.2 評価 2：backward graph（BG）の退避に向けた予備評価

提案手法 2 に示した backward graph の一部を退避することで DRAM の使用量をさらに削減する手法に対して，DRAM 使用量と性能低下の関係について，実際に Scale 27，Edge factor 16 に対して探索を行い，その結果から backward graph がどの程度削減可能かシュミレーションを行った．

5.2.1 評価結果

図 10 は SCALE 27， Edge factor 16 のグラフの実際に BFS を行ったケースに対して，各頂点が DRAM に確保するエッジ数の上限を調整した場合の backward graph のデータサイズと bottom-up アプローチ中に backward graph へ行われた全アクセスの中で不揮発性メモリへのアクセスの割合をシュミレーションしたものである．例えば，各頂点に対して上限エッジ数を 2 とすれば，それを超えたエッジは不揮発性メモリに退避することで DRAM 上の backward graph のサイズは 2.6%まで削減できるが，不揮発性メモリへのアクセスは全アクセス中 38.2%行われる．DRAM 上の上限エッジ数を 32 とすれば，DRAM 上の backward graph のサイズは 15.1%まで削減され，不揮発性メモリへのアクセスは 0.7%となる．

よって，5.1.2 で示したように不揮発性メモリを使用した場合に次数の低い頂点へのアクセスが性能の著しい低下を招いており，そのようなエッジを優先的に DRAM 上に確保することで，性能の低下を抑えながら DRAM の使用量を大きく削減することができると考えられる．

6. 関連研究

不揮発性メモリをグラフ処理に用いた研究として Kyrola らはグラフデータを一定のチャンクに分割しディスクに保存するこで，DRAM への読込みをシーケンシャルに行い，レイテンシーの隠蔽を行うモデルがある [12]．しかし，Hybrid-BFS の場合は，特に Top-down アプローチにおいてメモリアクセスがランダムになってしまうため，この手法の適応は難しい．

Pearce らは BFS 以外のグラフ処理にも使用できるアルゴリズムを採用し，コア数を超える大量のスレッドを使用することで不揮発性メモリとのレイテンシーを隠蔽している．しかし，アルゴリズムの特性上、全てのエッジを探索するため，BFS 時の性能は Hybrid-BFS に比べ低くなってしまう [9] [10]．

マルチノードに関する研究では，Beamer らが Hybrid-BFS のマルチノード化を行っており [13]，Pearce らも不揮発性メモリを搭載したマルチノード環境での研究を行っている [14]．

7. まとめ・今後の方針

近年様々な分野で大規模グラフ処理が求められており，消費電力やコストの問題から，DRAM の使用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，不揮発性メモリを利用することで性能低下を最小限に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おり，Graph500 ベンチマークにおいて，Hybrid-BFS アルゴリズムに対して不揮発性メモリを補助的に利用した場合の I/O の最適化，性能低下要因の解析を行うことで性能低下を抑えながら大規模グラフ処理が実行可能かの評価を行った．その結果，一部データを不揮発性メモリに退避することで DRAM 用量が半分の環境において性能低下を 47.1%まで抑えることができ，さらに，次数の小さい頂点が大量にあることによる非効率な I/O やデバイスの IOPS 性能が低く，性能低下の要因となっていることを確認し，対策案を提示した．また，参照され難いエッジデータをさらに退避することで性能の低下を抑えながら，より DRAM 使用量が削減可能なことの確認した．よって，不揮発性メモリを用いることで性能低下を最小限に抑えながら大規模グラフ実行の可能性が示唆された．

今後の方針としては，次数の小さい頂点への対応，backward graph の一部を実際に退避した場合での性能評価，アルゴリズムの特性，メモリへのアクセスパターン，DRAM と不揮発性メモリの容量やデバイスの性能を踏まえたモデル式の構築がある．

謝辞 本研究の一部は JST CREST「ポストペタスケールシステムにおける超大規模グラフ最適化基盤」の援助による.

参考文献

[1] 岩渕圭太, 佐藤仁, 安井雄一郎, 藤澤克樹, 松岡聡：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, 情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2013, No. 31, pp. 1–6 (2013).

[2] Graph500: `http://www.graph500.org/`.

[3] Leskovec, J., Chakrabarti, D., Kleinberg, J. and Faloutsos, C.: Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication, *Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'05, Berlin, Heidelberg, Springer-Verlag, pp. 133–145 (2005).

[4] Beamer, S., Asanovic, K. and Patterson, D.: Searching for a Parent Instead of Fighting Over Children: A Fast Breadth-First Search Implementation for Graph500, *. . . , University of California, Berkeley, Tech. Rep. . . .*, pp. 1–9 (2011).

[5] Suzumura, T., Ueno, K., Sato, H., Fujisawa, K. and Matsuoka, S.: Performance Characteristics of Graph500 on Large-Scale Distributed Environment, *Proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC 2011)*, pp. 149–158 (2011).

[6] 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).

[7] Yasui, Y., Fujisawa, K. and Goto, K.: NUMA-optimized Parallel Breadth-first Search on Multicore Single-node System, IEEE BigData '13, Santa Clara, USA, IEEE Computer Society (2013).

[8] Ajwani, D., Dementiev, R. and Meyer, U.: A computational study of external-memory BFS algorithms, *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, New York, NY, USA, ACM, pp. 601–610 (online), DOI: 10.1145/1109557.1109623 (2006).

[9] Van Essen, B., Pearce, R., Ames, S. and Gokhale, M.: On the Role of NVRAM in Data-intensive Architectures: An Evaluation, *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 703–714 (online), DOI: 10.1109/IPDPS.2012.69 (2012).

[10] 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).

[11] 白幡晃一, 佐藤仁, 松岡聡：GPU アクセラレータと不揮発性メモリを考慮した I/O 性能の予備評価, 情報処理学会研究報告. [ハイパフォーマンスコンピューティング] (2013).

[12] Kyrola, A., Blelloch, G. and Guestrin, C.: GraphChi: large-scale graph computation on just a PC, *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, Berkeley, CA, USA, USENIX Association, pp. 31–46 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2387880.2387884⟩ (2012).

[13] Beamer, S., Buluc, A., Asanovic, K. and Patterson, D.: Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search, *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '13, Washington, DC, USA, IEEE Computer Society, pp. 1618–1627 (online), DOI: 10.1109/IPDPSW.2013.159 (2013).

[14] Pearce, R., Gokhale, M. and Amato, N. M.: Scaling Techniques for Massive Scale-Free Graphs in Distributed (External) Memory, *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, Washington, DC, USA, IEEE Computer Society, pp. 825–836 (online), DOI: 10.1109/IPDPS.2013.72 (2013).