DBSJ Journal, Vol.12, No.1
June 2013

SIMD 命令によるモジュラリティに基づくグラフクラスタリングの並列化

Parallel Approach for Modularity-based Graph Clustering with SIMD Instruction

塩川 浩昭 ◆　山室 健 ◆
藤原 靖宏 ◆　鬼塚 真 ◆

Hiroaki SHIOKAWA　Takeshi YAMAMURO
Yasuhiro FUJIWARA　Makoto ONIZUKA

大規模グラフデータに対して CPU 特性を考慮した高速なクラスタリング手法を提案する．グラフデータに対するクラスタリング手法として Newman 法や CNM 法，BGLL 法などが提案されてきたが，これらの手法を大規模なグラフデータに適用した場合，データのランダムアクセスと実行命令数の増大に伴い参照効率と実行効率の悪化が課題になることが判明した．そこで提案手法では，モジュラリティに基づくグラフクラスタリングに対して，CPU のキャッシュヒット率向上を考慮したデータ構造の適用，さらに近年注目されている CPU 内の SIMD 命令を活用することで従来手法の課題を解決する．

Modularity-based graph clustering algorithms have been applied to various applications. The goal of this paper is to efficiently compute clusters with high modularity from unprecedented size of graphs that have more than a few billion edges. Our approach is a data parallel method that is based on SIMD instructions. The heart of our solution is to compute the modularity gain between vertices, which takes most part of computing time, in parallel by transforming the modularity's formula into simple representations. Moreover, we implement the clustering algorithms with the data structure which improves a cache hit for efficiently computing.

1. はじめに

グラフはデータをノードとエッジで表現した基本的なデータ構造であり，情報推薦や情報検索，科学データ分析などの様々な分野で利用されている．特に近年では，数億ノードから構成される大規模なグラフが登場し，このようなデータに対する高速な解析処理技術への需要が高まっている．グラフ分析のひとつとして，クラスタリング手法が挙げられる．グラフには，クラスタと呼ばれる相互に密な接続を有する部分ノード集合が存在する．例えば，Webグラフではトピックや関心の近いページ集合が互いにリンクすることで，トピックや関心の類似したページ群がコミュニティを形成する傾向にある．このようにグラフ中のクラスタは互いに共通した性質を持つことから，グラフの理解や様々なアプリケーションに利用され，非常に重要な要素技術となっている．このような背景からこれまで様々なクラスタリング手法の研究が行われてきた．

Modularity[1]を用いた手法は大規模なグラフを高速にクラスタリングする手法として注目を集めている手法の一つである．Modularityは，よりクラスタ内のエッジが密であり，かつ，クラスタ間のエッジが疎であるクラスタリング結果であるほど良い値を示し，より適切にクラスタを抽出できていることを表す特徴を持つ．Modularityを用いた近年の研究[1,2,3,4,5]では，高いModularityの値を高速に求める課題となる．しかし，Modularity値の最大化はNP困難であることから，Modularityを用いたクラスタリング手法では高速にかつ可能な限り高いModularityの値をいかにして求めるかという点が重要な研究課題となっている．

BlondelらによるBGLL[5]は，数億ノード規模のグラフに対して，高速かつ高いModularityの値を示す手法として知られている．従来手法[2,3,4]ではクラスタ同士の統合を行う際に，クラスタに含まれる全てのノードとエッジに対し，Modularityの向上量を計算する必要があった．これに対し，BGLLでは収束した処理結果を1ノードへ集約することで，処理に必要となるノードとエッジの参照数を削減させることに成功している．またBGLLは，任意の順に選択されたノードが，それぞれが隣接するノードに対してのみ統合の可能性を検証する．ゆえに，抽出されるクラスタサイズの偏りを防ぐことができ，その結果高いModularityの値を示すことがわかっている．BGLLは1億ノード規模のグラフデータに対して約2.5時間程度でクラスタリング処理可能であることがBlondelらによる研究で報告されている[5].

本稿では，より大規模な数億～数十億ノード規模のグラフに対する高速なクラスタリングの実現に向け，処理の並列化によるBGLLの高速化手法を提案する．本手法では，BGLLに含まれるModularity計算部分に対し，複数の演算をCPUの1サイクルで同時実行することが可能なSIMD命令を活用することで処理の高速化を行う．さらに，クラスタリング処理時に生じるメモリ上のデータアクセスの効率化を実現する，キャッシュヒット率向上に向けたデータ構造を導入し，クラスタリング処理の更なる高速化を図る．SIMD命令によりModularity計算を並列実行する際に，Modularity計算の定義式をより単純な命令の組み合わせに置き換え，SIMD命令の実行数の増加を抑制する．本稿では，提案手法のプロトタイプを実装し実データを用いた評価実験を行う．評価実験により，BGLLに対してModularityの値を同程度に示しつつ，高速に処理可能であることを示す．本稿の構成は以下の通りである．2節で関連研究について述べ，3節で本研究の前提となる概念と手法について説明する．4節では本稿で扱うデータ構造について述べ，5節で提案手法であるSIMDによる並列化の詳細を説明する．6節で提案手法の評価を行い，最後に7節で本稿のまとめと今後の課題について述べる．

2. 関連研究

2.1 Modularity に基づくグラフクラスタリング

Modularityによるクラスタリング手法について様々な研究が行われている．代表的な手法としてGirvan-Newman法

◆ 正会員　日本電信電話株式会社　NTT ソフトウェアイノベーションセンタ {shiokawa.hiroaki, yamamuro.takeshi, fujiwara.yasuhiro, onizuka.makoto}@lab.ntt.co.jp

[1]，Newman法[2]，CNM法[3]，WT法[4]，BGLL[5]などが挙げられるが，いずれの手法においても本研究で対象とする大規模なグラフを高速に処理することは難しい．

Girvanらはトップダウンにエッジを間引きグラフを分割するGirvan-Newman法[1]を提案した．グラフ全体を1つのクラスタとみなし，クラスタ間を横断する可能性の高いエッジを順に切り離すことで小さなクラスタへと分割する．エッジを切り離す度に各エッジに付与されたスコアを再計算することから，エッジが疎なグラフに対してノード数をnとすると$O(n^3)$の計算量を必要とする．計算量が膨大であるため1万ノード規模の処理には適用できないと報告されている．

Newmanは，貪欲法によりボトムアップ式にクラスタを抽出するNewman法[2]を提案した．この手法では，各ノードがそれぞれ別のクラスタである状態から処理を開始し，Modularityが最も向上するノード対を貪欲法により同一クラスタへと統合していく．統合させるべきノード対を全探索することから，エッジが疎なグラフに対してノード数をnとすると$O(n^2)$の計算量を必要とする．これに対し，ClausetらはNewman法に対してヒープ構造を導入することで高速化するCNM法[3]を提案し，計算量$O(n\log^2 n)$を達成した．また，脇田らはクラスタ統合時のクラスタサイズの不均衡による処理速度の低下を指摘し，新たにCNM法のさらなる高速化を行うWT法[4]を提案している．WT法ではクラスタ統合時にModularityの向上量をクラスタサイズで正規化するヒューリスティクスを用いることでクラスタサイズの不均衡を解消する．これらの手法では最大で数百万ノード規模までの処理可能と報告されている．

Blondelらは，Newman法の高速化手法としてBGLL[5]を提案した．BGLLはModularityの局所最適化とノードの一括集約により，処理におけるノードとエッジの参照数の削減に成功している．BGLLの最悪計算量は知られていないが，文献[5]において従来手法に対する高速性が実験的に示されており，我々の知る限り最も高速にクラスタリング処理が可能な手法である．文献[5]では，最大で数億ノード規模までの処理が可能であると報告されている．さらに，BGLLは任意の順でノードを選択することで抽出されるクラスタサイズに偏りが生じることを防ぎ，これまで述べた手法の中で最も良いModularityを示すことも文献[5]にて示されている．

本稿はこれらに対し，より大規模なグラフの高速なクラスタリング手法の実現に向け，SIMD命令とキャッシュヒット率向上を考慮したBGLLのさらなる高速化に取り組む．

2.2 SIMD 命令を用いたグラフクラスタリング

2002年頃以降，CPUやGPUのSIMD命令を用いた各種手法の高速化に関する研究は盛んに行われている．グラフクラスタリングの高速化に関する従来研究[6,7]は存在するが，これらの研究ではラベル伝搬やランダムウォークに着眼するものである．計算量の大きなこれらの手法に対して，局所的なデータ並列化を行うことで処理性能の向上とスケーラビリティの確保を図っている．しかしながら，これらの手法の計算量は依然として大きく，本研究で対象とするような数億〜数十億ノード規模のグラフを対象としたクラスタリングは難しい．これに対し，Modularityによる手法に着目し数億〜数十億ノードからなる大規模なデータに対するクラスタリングを行う処理にSIMD命令を適用したものは本研究がはじめてである．

3. 事前準備

3.1 Modularity

本稿で提案するクラスタリング手法では，抽出結果の良さを示す指標 Modularity を対象とする．そこで，クラスタリング指標 Modularity について概説する．

Modularity はクラスタ内に含まれたノード間のエッジが密であり，クラスタ間に存在するエッジが疎となる程良い値を示す指標である．クラスタリング手法により取得したクラスタ集合をC, クラスタiからクラスタjへ接続されているエッジ数をe_{ij}，グラフ全体に含まれる総エッジ数をmとするとき，Modularity Qは定義 1 のように定義される．Modularity が負の値を取る場合は$Q = 0$とし，常に Modularity Qは$0 \le Q \le 1$の値を示す．

[定義 1] （Modularity Q）

$$Q = \sum_{i \in C} \left\{ \frac{e_{ij}}{2m} - \left(\frac{\sum_{j \in C} e_{ij}}{2m} \right)^2 \right\}$$

3.2 既存手法 BGLL

本稿では BGLL に対して，キャッシュヒットを考慮したデータ構造と SIMD 命令を用いた並列化を導入することで，処理のさらなる高速化を行う．そこで本節では，既存手法 BGLL について概説する．

BGLL は 2 つのフェーズから構成されるパスを繰り返すことでクラスタを抽出する手法である．第 1 フェーズにて，各ノードと隣接するノード間のみにおいてModularity の値を準最適化し，第 2 フェーズにて，第 1 フェーズで得られた結果を 1 ノードに一括集約する．BGLL は Modularity の値が増加する限り，2 つのフェーズからなるパスを反復する．各フェーズの詳細について以下に説明する．

3.2.1 第 1 フェーズ：Modularity の局所最適化

第 1 フェーズではまず，入力されたグラフに対して，任意の順（ランダム順）にノードを選択する．その後，選択されたノードの全隣接ノードに対して，クラスタを統合した際に上昇する Modularity の値ΔQを計算する．一般的に定義 1 で示した Modularity Qを求めるためには，全てのノードとエッジを参照する必要があるため効率的ではない．しかしBGLL では Modularity の変化量にのみ着目し，定義 1 からModularity 変化量ΔQを導出し，計算の効率化を図っている．2 つのクラスタiとjを統合した際の Modularity 変化量ΔQを定義 2 に示す．

[定義 2] （Modularity 変化量ΔQ）

$$\Delta Q = 2 \left\{ \frac{e_{ij}}{2m} - \left(\frac{\sum_{k \in C} e_{ik}}{2m} \right) \left(\frac{\sum_{k \in C} e_{jk}}{2m} \right) \right\}$$

第 1 フェーズでは，上記の定義に従いΔQを計算し，最もModularity 変化量ΔQが大きくなるノード対を選出する．選出したノード対を同一のクラスタとしてラベル付する．この処理を Modularity の値が向上するあいだ継続する．

3.2.2 第 2 フェーズ：グラフの一括集約

第 2 フェーズでは第 1 フェーズで得られた処理結果に対してノードとエッジを一括集約し，処理で扱うグラフのサイズを縮小する．同一のクラスタに含まれるノードは，1 つのノードにまとめられる．クラスタ内に存在するエッジについては，自己ループのエッジ 1 本へ集約し元のエッジ本数の 2 倍の重みをつける，クラスタ間に存在するエッジについては，

表 1 BGLL の処理時間と Modularity の増加量の変化
Table.1 Increment of time/modularity for each pass

パス	1	2	3	4	5	6	Total
Time (sec)	418.7	3.34	0.17	0.03	0.01	0.01	422.3
Modularity	0.669	0.06	0.01	0.01	0.01	0	0.759

同一のクラスタ対につきエッジ 1 本に集約し, エッジ本数分の重みをつける. 第 2 フェーズで生成された集約された重み付きグラフは, 次回以降のパスで入力データとして与えられ利用される.

3.2.3 BGLL の課題

BGLL は第 1 パスの第 1 フェーズにて入力された全てのノードをランダム順に参照し, Modularity 変化量ΔQが増加する可能性を検証することでクラスタの統合を行う. この処理は Modularity 変化量ΔQが増加するあいだ継続するため, 入力されるグラフのノード数増加に伴い, 第 1 パスにおける処理時間は増加することとなる. これにより, より大規模なグラフであるほど, 第 1 パスにおける処理時間がボトルネックとなり, クラスタリング処理時間の短縮が困難となる. 具体例を表 1 に示す. 表 1 では, ノード数 5,363,260, エッジ数 79,023,142 から構成されるグラフデータに対して BGLL を適用した際の処理時間と Modularity 値の変化の内訳を示している. 表 1 のパスは BGLL を実行した際の反復回数, Time は各パスに消費した処理時間, Modularity は各パスで増加した Modularity の値を表している. 表 1 からも分かる通り, BGLL では, 第 1 パスにその処理時間の 99%以上を消費している. また Modularity については, 第 2 パス以降では大きな向上は見られないことがわかる. この傾向は, 予備実験で使用したデータセットのみならず, 他のデータセットにおいても同様に見られるものである. このことから, 我々は BGLL の第一パスを効率化することで, クラスタリング処理全体を大幅に高速化できると考えた.

4. キャッシュヒットを考慮したデータ構造設計

本稿ではグラフG=(V,E)を隣接リスト表現で扱う. 隣接リスト表現では, 各ノードにノードuとそのノードに隣接するノード集合Γ(u)のみをリスト状にして表現したデータ構造であるため, 隣接行列表現に対してより空間効率よくデータを保持することができる. しかしながら, Modularity によるグラフクラスタリングにおいてはメモリ上のデータに対する参照が頻繁に発生するため, ノード毎に離散したメモリ番地へのアクセスが生じる可能性のある隣接リスト表現はキャッシュヒット率の低下を招き, 参照効率が悪化する可能性がある. 本稿では隣接リスト表現をメモリ上において連続した領域に配置することで, データ参照時のキャッシュヒット率を向上させる. 概要を図 1 に示す.

本データ構造は 2 つの連続した配列から構成される. 一つはノード集合を格納した配列V_a, もう一つは隣接ノード集合を格納した配列E_aである. まず, 配列V_aでは, 配列の番地をノードの ID とし, その要素として配列の先頭から現在の番地までに含まれるノードの次数の合計値を格納する. ここで格納した次数は, 配列E_aのインデックスとして利用される. 次に配列E_aにおいて, 配列V_aに格納したノードの順にそれぞれのノードの隣接ノードを格納する.

このようにノードとエッジを格納することにより, 全てのデータを連続領域に格納しつつ, それぞれのデータに対して

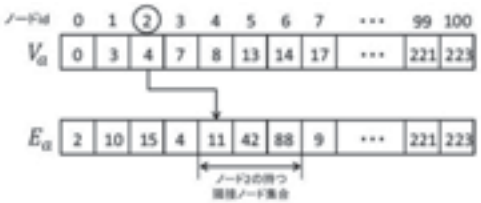

図 1　キャッシュヒットを考慮したデータ構造
Fig.1 Reconstructed data structure for CPU cache

効率良くアクセスすることが可能にする. 例えば, あるノードuの隣接ノード集合Γ(u)を参照する場合を考える. この場合, まず, 配列V_aの番地uにアクセスし, 次数の合計値$V_a(u)$および$V_a(u+1)$を取得する. この合計値$V_a(u)$および$V_a(u+1)$はそれぞれ配列E_aに存在するノードuおよびノード$u+1$の隣接ノード集合を格納した領域の先頭番地を指すため, 配列$E_a(V_a(u))$から$E_a(V_a(u+1)-1)$にアクセスすればノードuの隣接ノード集合Γ(u)が取得可能である.

例えば図 1 では, ノード 2 に隣接するノード集合を取得する場合を示している. ノード 2 の隣接ノード集合を取得する場合, 配列E_a中の$E_a(V_a(2)) = E_a(4)$から$E_a(V_a(2+1)-1) = E_a(7-1) = E_a(6)$までアクセスすれば, 隣接ノード集合$\{11, 42, 88\}$を取得できる. このデータ構造では, データをメモリ上の連続した領域の載せた状態で処理を行うことが可能となるため, キャッシュヒット率が向上しやすく, メモリ上のデータ参照が頻繁に発生するグラフクラスタリングにおいてデータアクセスを効率化できると考えられる.

5. SIMD 命令による Modularity 計算の並列化

本稿では, 複数のデータに対する演算を CPU の 1 サイクルで同時実行することが可能な SIM 命令を活用することで, Modularity によるクラスタリングをデータ並列化し高速化する. 3. 2 節で示したように, BGLL は各パスの第 1 フェーズにおいて, 全てのノードを走査する. そして, それぞれのノードの隣接ノード集合に対して Modularity 変化量ΔQを計算し, ΔQを最大化するノードとクラスタの併合を繰り返すことで最終的なクラスタを抽出する. しかし, BGLL ではこの処理を全てのエッジに対し複数回の計算を実行する必要があり, 処理時間に対して支配的となり, その性能を劣化させる原因となっている. そこで, 本稿では BGLL の第一フェーズにおける, 各ノードの Modularity 変化量ΔQの計算に対して SIMD 命令を用いたデータ並列化を導入し, その処理時間を短縮する. 提案手法における SIMD 命令を用いたデータ並列化の概要を図 2 に示す.

5.1 Modularity 変化量の並列計算

任意のノードが与えられた際にその全ての隣接ノードに対して SIMD 命令を用いて並列に Modularity 変化量を計算する. 提案手法では, BGLL と同様に任意の順にノードuを選択する. 次に, 選択されたノードuの隣接ノード集合Γ(u)を 4 節に示した手順で取得する. 取得した隣接ノード集合Γ(u)に対して, SIMD 命令を用いて並列に Modularity 変化量の計算を実行する. この際に, 従来手法では定義で示した, Modularity 変化量ΔQを計算する必要がある. 定義 2 の示した Modularity 変化量ΔQに従うと, 計算に必要なe_{ij}や$\sum_{k\in C} e_{ik}$, $\sum_{k\in C} e_{jk}$が事前に求められていた場合, SIMD 命令によるΔQの計算には, (I)「e_{ij}と2mの商」, (II)「$\sum_{k\in C} e_{ik}$と2m

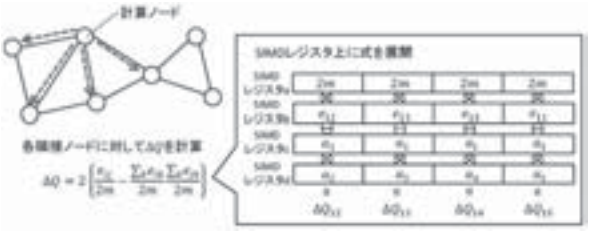

図 2　SIMD 命令による並列化の概要
Fig.2 Overview of SIMD Implementation strategy

の商」，(III)「$\sum_{k\in C} e_{jk}$と2mの商」，(IV)「(II)と(III)の積」，(V)「(I)と(IV)の差」，(VI)「2 と(V)の積」の 6 回の計算が必要となる．しかしながら，この計算で本来求める必要がある事項は，Γ(u)に含まれるノードのうち Modularity 変化量ΔQ を最大化し得るノードの発見である．すなわち，ΔQに示した計算を全て実行する必要はなく，Γ(u)に含まれる全てのノードがΔQに関する相対的な順位が求まれば良い．そこでクラスタi,j間の Modularity 変化量ΔQ_{ij}をより容易な計算の組合せに変換した相対 Modularity 変化量$\Delta Q'_{ij}$を以下に定義する．

[定義 3]（相対 Modularity 変化量$\Delta Q'_{ij}$）

$$\Delta Q'_{ij} = 2\mathrm{m}e_{ij} - \sum_{k\in C} e_{ik} \sum_{k\in C} e_{jk}$$

定義 3 で定義した相対 Modularity 変化量$\Delta Q'_{ij}$から下記の補題が導出される．

[補題 1]（$\Delta Q'_{ij}$とΔQ_{ij}の大小関係）相対 Modularity 変化量$\Delta Q'_{ij}$は，Modularity 変化量ΔQ_{ij}によって得られる値の大小関係と等価な大小関係を示す．

（証明）定義 2 より，

$\Delta Q_{ij} = 2\left\{\frac{e_{ij}}{2m} - \left(\frac{\sum_{k\in C} e_{ik}}{2m}\right)\left(\frac{\sum_{k\in C} e_{jk}}{2m}\right)\right\}$となり，これを変形すると，$\frac{4m^2}{2}\Delta Q_{ij} = 2\mathrm{m}e_{ij} - \sum_{k\in C} e_{ik} \sum_{k\in C} e_{jk}$が得られる．したがって，$\Delta Q'_{ij} = 2\mathrm{m}e_{ij} - \sum_{k\in C} e_{ik} \sum_{k\in C} e_{jk}$となり，$\Delta Q'_{ij}$と$\Delta Q_{ij}$は相対的に等価となる．　□

$\Delta Q'_{ij}$を用いることで Modularity 変化量ΔQ_{ij}では SIMD 命令上 6 回の計算が必要であったものを，(I)「e_{ij}と2mの積」，(II)「$\sum_{k\in C} e_{ik}$と$\sum_{k\in C} e_{jk}$の積」，(III)「(I)と(II)の差」の 3 回の演算で相対的に等価に求めることが可能になる．ゆえに，提案手法では，SIMD レジスタ上において$\Delta Q'_{ij}$を用いて並列に計算を行うことにより，従来の BGLL よりも少ない CPU サイクルでクラスタを求めることができる．

5.2 Modularity 変化量の最大ノードの取得

Modularity 変化量を最大化する隣接ノードを SIMD 命令により取得する．Modularity 変化量を最大化するノードの取得を行うために，4 つの SIMD レジスタ，*reg_id*, *reg_result*, *reg_max_id*, *reg_max*, *reg_mask* を使用する．*reg_id* は計算対象となっている隣接ノード集合の ID を格納した SIMD レジスタ，*reg_result* は *reg_id* に対応した Modularity 変化量の計算結果が格納された SIMD レジスタ，*reg_max_id* は Modularity 変化量を最大化する可能性のある隣接ノード ID が格納された SIMD レジスタ，*reg_max* はノードuと *reg_max_id* に存在する隣接ノードによって得られる Modularity 変化量を格納した SIMD レジスタ，そして *reg_mask* は最大化ノードを取得するために利用するマスク用の SIMD レジスタである．

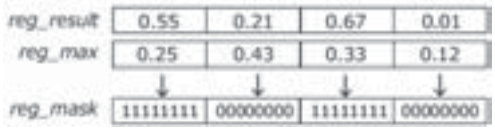

図 3　*reg_result* と *reg_max* から *reg_mask* の作成
Fig.3 Build a *reg_mask* from *reg_result*/*reg_max*

まず，5.1 節に示した手順により，ノードuの隣接ノード集合Γ(u)のうち，1 つの SIMD レジスタにデータ入力可能な個数の隣接ノード部分集合 *reg_id* に対する Modularity 変化量計算が終了した場合を考える．この時，*reg_result* には各隣接ノードに対する Modularity 変化量の値が入力されている．次に図 3 のように，*reg_result* と *reg_max* の 2 つの SIMD レジスタ間の大小比較を行い，*reg_result* の方が大きな値を持っている SIMD レジスタ上のアライメントのビットを全て 1，そうでないアライメントのビットを全て 0 とし，その結果を *reg_mask* で保持する．*reg_mask* 取得後，このマスクの値と *reg_id* および *reg_max_id* を用いて，Modularity 変化量が大きくなったノード ID を下記のように抽出する．

reg_max_id = (!reg_mask&reg_max_id)|(reg_mask&reg_id)

ノードuにおける全てのΓ(u)に対する処理が終了するまで，上記の処理を繰り返し，*reg_max_id* および，*reg_max* に Modularity 変化量を最大化する可能性のある隣接ノードの ID 及び Modularity 変化量を更新し保持し続ける．Modularity 変化量の計算が終了後，*reg_max_id* から *reg_max* が最大値となるようなアライメントを操作により取得することで，ノードuに対して Modularity 変化量を最大化する隣接ノードを取得する．

6. 評価実験

提案手法の有効性を評価するために，Blondel らによる BGLL に対して，本提案手法を適用し比較実験を行う．本実験で下記の実データを利用し評価を行った．

- Live[8][1]：LiveJournal と呼ばれる SNS の 2008 年時点でのユーザの友人関係から構成されるグラフ．ノード数 5,363,260，エッジ数 79,023,142.
- uk-2005[8][2]：2005 年に収集した uk ドメインの Web グラフ．ノード数 39,459,925, エッジ数 936,364,282.
- webbase-2001[8][3]：WebBase[4]によって 2001 年に収集された us ドメインの Web グラフ．ノード数 118,142,155，エッジ数 1,019,903,190.

これらのデータセットの次数分布を図 4 に示す．
また，本実験では Dynamic Graph Generator (DIGG)[5]を用いて，人工のグラフを生成した．DIGG では，生成するグラフのノード数とグラフ中に含まれるエッジの偏りを表すパラメータβの値を指定することで，グラフデータを生成することが可能である．βの値は大きな値を持つほど，ノード間の次数に偏りが生じ，小さな値を持つほどノード間で均一な次数をもつグラフデータが生成される．本実験では，DIGG

[1] http://law.di.unimi.it/webdata/ljournal-2008/
[2] http://law.di.unimi.it/webdata/uk-2005/
[3] http://law.di.unimi.it/webdata/webbase-2001/
[4] http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/
[5] http://digg.cs.tufs.edu/

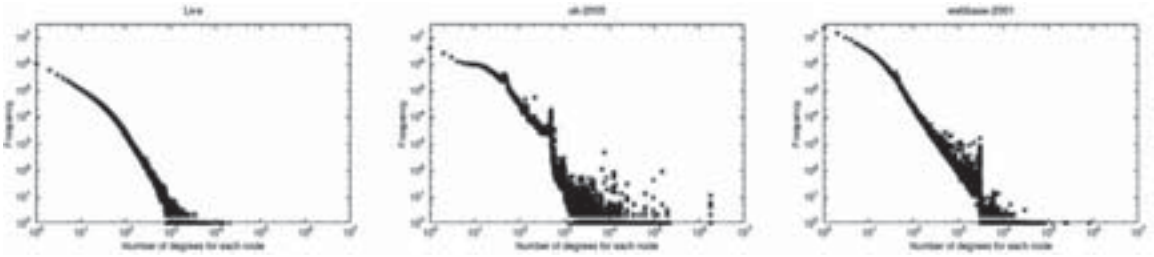

図 4　各データセットの次数分布
Fig.4 Degree distribution for each dataset

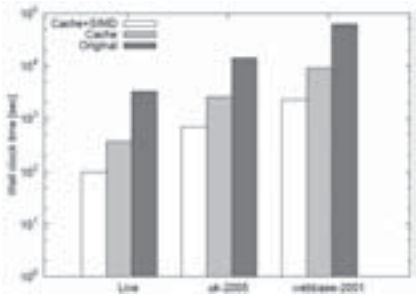

図 5　実データによる実行速度の比較
Fig.5 Clustering time for each dataset

により生成したノード数を 10,000 ノードから 10,000,000 ノードまで 10 倍ずつ変化させた 4 つのグラフデータを利用し評価を行った．また，DIGG で生成した全てのグラフデータは$\beta = 2.0$と設定されている．本実験では，これらのデータセットに対して，クラスタリング処理が終了するまで処理を行った際の処理時間を示し，比較を行う．本実験には CPU が Intel Xeon Quad-CoreL5640，メモリが 144GB の Linux サーバを利用した．また，本実験で使用した SIMD レジスタのサイズは 128bit であり，計算に利用するノード ID やエッジ数は 32bit のデータ型として扱った．

6.1 実データによる評価

本実験では，前節で述べた実データに対して提案手法と BGLL の処理速度，処理制度を比較する．図 5 に処理速度の比較結果を示す．図 5 の Original，Cache，Cache+SIMD はそれぞれ，BGLL，キャッシュを考慮したデータ構造のみを用いた BGLL，キャッシュを考慮したデータ構造と SIMD による並列化を行った BGLL を示している．図 5 からわかるように，Cache および，Cache+SIMD は全てのデータセットにおいて，Original よりも高速に処理できていることがわかる．Cache は Original に対して約 5〜8 倍程度高速化に成功している．また，Cache+SIMD は，いずれのデータセットに対しても Cache に対して約 4 倍の高速化に成功している．本実験で用いた実験環境では SIMD レジスタのサイズ 128bit に対して 32bit のデータ型を利用した実装を行ったため，Modularity 計算部分の並列化上限値は 4 である．図 5 の結果はその並列化の上限値に達した結果となっている．このような結果が得られた理由として，キャッシュを考慮したデータ構造の導入が挙げられる．従来の BGLL では，大量のノードとエッジに対する参照と Modularity 計算の両方に計算コストが生じていたが，本稿ではキャッシュを考慮したデータ構造を導入したことによりノードとエッジの参照コストを削減することに成功している．そのため，BGLL のボトルネックが参照と演算の 2 つから演算のみにシフトし，演算部分の並列化による高速化が処理全体に寄与する割合が大きくなっているからだと考察できる．

表 2　Modularity 値の比較
Table.2 Modularity score for each dataset

	Live	uk-2005	webbase-2001
Proposed	0.755	0.977	0.980
BGLL	0.754	0.979	0.981

図 5 では，webbase-2001 のように規模が大きく次数分布が広いデータセットに対してより高い高速化性能を示している．SIMD 命令を用いる際に，SIMD レジスタ上に展開できるノード数の上限を超える次数を持つノードが多く存在する場合，並列化された計算がより効果的に実行できるためだと考えられる．この場合，SIMD レジスタの領域を残すことなく実行され易くなるため，1 回の CPU サイクルで同時に計算が可能となる隣接ノードの数の期待値が増加する．ゆえに webbase-2001 のような規模が大きく，広く次数が分布したグラフほど，高速化し易くなると考えられる．

表 2 に提案手法 Cache+SIMD と BGLL によるクラスタリング結果の Modularity 値の比較を示す．いずれのデータセットに対しても同程度の値を示している．提案手法によるデータ構造の変更と SIMD による並列化は BGLL 法と本質的に同様の処理を行うため，最終的に得られるクラスタリング結果は BGLL と同程度のものが得られる．

6.2 人工データによる評価

次数の分布傾向を示す DIGG のパラメータβを 2.0 に固定し，ノード数が増加した際のスケーラビリティを評価する．図 6 に実験結果を示す．ノード数が 10,000 ノードや 100,000 ノード程度の時には提案手法 Cache+SIMD と BGLL はほぼ同程度の処理時間を要している．データ規模が小さいことでグラフ全体に占める高次数ノードが少なく，並列化による高速化の効率が低下していることが理由として考えられる．1,000,000 から 10,000,000 ノード規模のより大きなデータセットに対しては各データセットに対しても 3 倍から 4 倍程度の高速化性能を示している．規模の大きなデータセットになるほど，高次数ノードがグラフ全体に占める割合が大きくなり，計算の並列度が高くなりやすくなるためだと考えられる．提案手法は，よりノード数やエッジ数が比較的大きなグラフデータに対して，BGLL よりも約 3 倍から約 4 倍程度高速に処理可能である．

7. おわりに

本稿では，グラフデータに対する高速なクラスタリングの実現に向けた，SIMD 命令を用いたグラフクラスタリングの並列化手法を提案した．本稿ではまず，BGLL のボトルネ

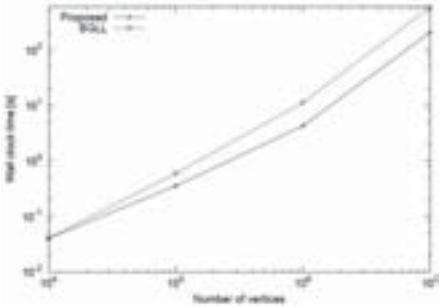

図 6　ノード数の変化による実行速度の比較
Fig.6 Clustering time for each node size

ックが処理の第 1 パスにあることを明らかにした．そして本手法では，第 1 パスの計算時間を短縮するために，複数の命令を 1 つの CPU サイクルで実行可能な SIMD 命令をクラスタリングの Modularity 計算部に適用することによる計算時間を短縮し，メモリ上のデータ参照に対するキャッシュヒット率向上を目的としたデータ構造および索引構造を導入することでデータアクセスを効率化した．本稿では，128bit の SIMD レジスタを利用して実データセット及び人工データセットに対して計算速度の評価を行い，数百万以上のノードを持つグラフデータに対しては高速化の理論上限値である 3 倍から 4 倍程度の高速化が可能であることを確認した．本手法は，SIMD レジスタのサイズの大きさに伴い，その処理の並列度を増加させることが可能であり，将来普及する可能性のあるより大きな SIMD レジスタを搭載した計算機においてさらなる高速化が期待できる．

本研究の今後の課題として次の点が挙げられる．

(a) 並列度の向上

本稿の評価実験で示したように，SIMD 命令による並列化はキャッシュを考慮したデータ構造と組み合わせることにより，並列化上限に近い高速化倍率を得られることがわかっている．そこで本研究ではこの特性を利用して，SIMD レジスタ上での並列化上限を向上させることによる本手法のさらなる高速化を検討する．具体的な方針として下記の 2 つのアプローチを検討している．

i.　ハードウェアによるアプローチ

本実験で用いた SIMD レジスタのサイズは 128bit である．しかし，近年の各種ハードウェアには SIMD レジスタが 256bit のものや 512bit のものが存在する．今後はこれらのハードウェアを利用したさらなる並列度の向上を検討する．

ii.　データ圧縮に寄るアプローチ

SIMD 命令による Modularity 変化量計算の並列度は SIMD レジスタのサイズと各ノード ID や各ノードの持つ次数のデータ型に依存して決定する．例えば，本稿の評価実験では 128bit の SIMD レジスタに対し 32bit のデータ型を利用したため 4 並列が理論上現値であった．これに対し，ノード ID や次数を圧縮することで SIMD レジスタ上に展開可能なノード数を増やすことが可能になり，さらなる高速化が期待できる．今後は SIMD レジスタ上での並列度向上を目的とし，グラフの圧縮技術について検討を進める．

(b) グラフデータの特性による性能への影響の調査

本稿では，SIMD 命令を用いたグラフクラスタリングの並列化の初期検討として，3 つの実データと 4 つの人工データを用いた性能の評価を行った．本稿の評価結果より，次数の分布の傾向や平均エッジ数などに依存して処理の並列度が影響を受ける可能性が考えられる．今後の課題として，ノードの計算順序や，グラフの傾向にあわせた手法の検討を行う．

[文献]

[1] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Phys. Rev. E, Vol. 69, p.026113, Feb 2004.
[2] M. E. J. Newman. Fast algorithm for detecting community structure in networks. Phys.Rev. E, Vol. 69, P066133, Jun 2004.
[3] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. Phys.Rev. E, Vol. 70, p. 066111, Dec 2004.
[4] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks. In Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 1275-1276, May 2007.
[5] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, Vol. 2008, P10008, October 2008.
[6] Di Wu, Tianji Wu, Yi Shan, Yu Wang, Yong He, Ningyi Xu, and Huazhong Yang. Making human connectome faster: GPU acceleration of brain network analysis. In 16th International Conference on Parallel and Distributed Systems, 2010.
[7] Jyothish Soman and Ankur Narang. Fast community detection algorithm with gpus and multicore architectures. In 2011 IEEE International Parallel & Distributed Processing Symposium, 2011.
[8] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 595-601, May 2004.

塩川 浩昭　Hiroaki SHIOKAWA
日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ勤務．2011 筑波大学大学院システム情報工学研究科博士前期課程修了，修士（工学）．グラフマイニングの研究開発に従事．日本データベース学会会員．

山室 健　Takeshi YAMAMURO
日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ研究員．2008 上智大学理工学研究科博士前期課程修了，修士（工学）．DBMS のコア技術、およびハードウェア特性を考慮した探索/圧縮アルゴリズムの研究・開発に従事．日本データベース学会会員．

藤原 靖宏　Yasuhiro FUJIWARA
日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ研究員．2011 東京大学大学院情報理工学系研究科博士課程修了，博士（情報理工学）．時系列データ処理及びグラフマイニングの研究開発に従事．情報処理学会，電子情報通信学会，日本データベース学会各会員．

鬼塚 真　Makoto ONIZUKA
日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ特別研究員．電気通信大学客員教授．1991 東京工業大学工学部情報工学科卒業，博士（工学）．2000～2001 ワシントン大学客員研究員. DBMS, XML データベース，分散処理，グラフマイニングの研究開発に従事．ACM，情報処理学会，電子情報通信学会，日本データベース学会各会員．