DEIM Forum 2017 E2-2

メニーコアプロセッサを用いた構造的類似度に基づくグラフクラスタリングの高速化

高橋　知克†　　塩川　浩昭††　　北川　博之††

† 筑波大学大学院システム情報工学研究科　〒305–8573 つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター　〒305–8573 つくば市天王台 1–1–1
E-mail: †shihakata@kde.tsukuba.ac.jp, ††{shiokawa,kitagawa}@cs.tsukuba.ac.jp

あらまし　グラフクラスタリングはグラフの中に存在するコミュニティ構造を理解する上で重要な要素技術である．その中でもノード間の構造的類似度に基づいた手法である SCAN は，高い精度でクラスタを検出するとともにハブや外れ値を同時に抽出できることから幅広く利用されている．しかしながら，SCAN は全てのエッジに対して構造的類似度計算を行う必要があることから計算量が大きく大規模なグラフへの適用が難しい．そこで本稿では，メニーコアプロセッサ Intel Xeon Phi を活用し，そのハードウェア特性を考慮した SCAN の高速化手法 SCAN-XP を提案する．提案手法では，(1) 実グラフにおける次数分布の偏りやクラスタ検出処理における計算の衝突等の並列化ボトルネックを解消するとともに，(2)Intel Xeon Phi の持つ 512 ビット SIMD 演算へのアルゴリズムの最適化を行う．本稿では実データを用いた評価実験を行い提案手法の有効性を示す．

キーワード　グラフクラスタリング，Intel Xeon Phi，SIMD 演算

1. はじめに

グラフとはデータエンティティ間の関係をノードとエッジにより表現したデータ構造であり，様々な分野で活用されている．近年では，グラフを解析することで有用な情報を抽出する技術に注目が集まっており，データマイニング [1,2] やソーシャルサイエンス [3] などの幅広い分野で研究されている．しかし，インターネットの普及に伴いグラフデータの大規模化が進んでおり，例えば Twitter のユーザをノード，ユーザ間のフォロー/フォロワー関係をグラフとして表現すると，数億ノードから構成される大規模グラフとなる．したがって，このような大規模グラフを高速に解析する手法の需要が高まっている．

グラフクラスタリングはグラフ構造解析手法のひとつである．グラフクラスタリングは密に接続されたノードの集合をクラスタとして検出することで，隠れたコミュニティ構造を抽出することができる．そのため，近年では多くの手法が提案されており，min-max cut による手法 [4,5] や Modularity [6] に基づく手法 [7–10] など様々な研究が行われてきた．

特に 2007 年に Xu らにより提案された SCAN [11] は多次元ベクトルに対する密度ベースのクラスタリング DBSCAN [12] を基に考案された構造的類似度に基づくグラフクラスタリング手法であり，高精度にクラスタリングが可能であることが知られている．SCAN の大きな特徴は，ノード間の構造的類似度を計算することで，従来手法で検出されるようなクラスタだけでなく，グラフ中の特別な役割を持つノードであるハブと外れ値を見つけることである．ハブは複数のクラスタ間の橋渡しをするように繋がっているノードであり，バイラルマーケティング等の分野で重要なデータである．一方で外れ値はクラスタとの関係性が低いノードであり，一般的にノイズとして扱われる．

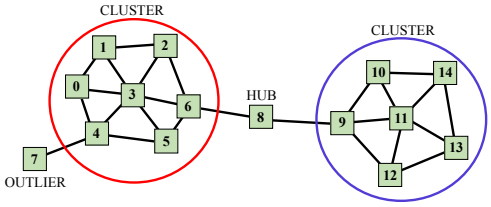

図 1: SCAN によるグラフクラスタリングの例

従来手法と SCAN の違いを示すために，実際に SCAN をグラフに対して適用した例を図 1 に示す．SCAN によりクラスタリングを行った場合，クラスタとの橋渡しをしているノードとクラスタとの関連性が薄いノードはそれぞれハブと外れ値として判定されるため，クラスタとして $\{0,\ldots,6\}$ と $\{9,\ldots,14\}$ が得られ，ハブとしてノード 8，外れ値としてノード 7 が検出される．一方で従来手法の中で最も代表的な Modularity を用いた手法を図 1 と同様のグラフに対して適応した場合，2 つのクラスタとして $\{0,\ldots,7\}$ と $\{8,\ldots,14\}$ の 2 つの集合が得られるが，ハブと外れ値を検出することはできない．このように，SCAN はグラフの構造をより高い精度で検出できる．

しかしながら，SCAN は計算コストが大きいという問題点がある．SCAN はクラスタを抽出するためにグラフ $G=\{V,E\}$ に含まれる全てのエッジに対しての構造的類似度と呼ばれるノード間の接続の強さを示す評価指標を計算する必要がある．この処理は $|E|$ をエッジ数，$|V|$ をノード数とした場合，最悪計算量が $O(|E|)=O(|V|^2)$ となる．さらに，各エッジの構造的類似度を計算するためには対象とするエッジに接続される 2 つのノードの共通の隣接ノードを計算する必要がある．結果として，SCAN の平均計算量は $O(|E|^2/|V|)$ となり，最悪計算量は $O(|E|^{1.5})$ となる．したがって，大規模グラフに対して膨大な計算時間を必要とするため，適用が難しいという問題がある．

1.1 先行研究と課題

SCAN を高速化するために，多くのクラスタリングアルゴリズムが提案されてきた． Lim らによって提案された LinkSCAN* [13] はクラスタリングの高速化を行うためにエッジサンプリング手法を取り入れた近似的手法である．LinkSCAN* はエッジのサンプリングにより，構造的類似度計算が必要なエッジの数を削減している．しかしながら，得られるクラスタリング結果は近似解となり，SCAN と同等のクラスタリング結果が得られる保証はされていない．

SCAN と比較してクラスタリングの精度を落とすこと無く，計算速度を向上させる手法として，Shiokawa らによる SCAN++ [14] や Chang らによる pSCAN [15] がある．これらの手法は実世界のグラフに内包された構造特性の利用や計算結果が明示であるエッジへの計算回数の削減により，SCAN の高速化に成功している．しかし，どちらの手法も計算量は依然として $O(|E|)$ を必要とする．ゆえに 1 億以上のエッジを持つ非常に大規模なグラフへの適用は未だ難しい．したがって，構造的類似度に基づくグラフクラスタリングの更なる処理性能の向上は依然として重要な課題となっている

1.2 本研究の貢献

上記の課題を解決するために，本稿ではメニーコアプロセッサである Intel Xeon Phi を用いて大規模グラフを高速に処理する構造的類似度に基づくクラスタリング手法 SCAN-XP (Structural Clustering Algorithm for Network over Xeon Phi) を提案する．SCAN において構造的類似度の計算はエッジごとに独立していることから，並列化による高速化が期待できる．特に近年では多くの演算コアを搭載したメニーコアプロセッサの発展がめざましく，高性能計算分野を中心に Intel Xeon Phi や GPU (Graphics Processing Unit) などを活用した並列化によるデータ処理の高速化が注目されている．

そこで SCAN-XP では Intel Xeon Phi を用いた高速かつスケーラブルなグラフクラスタリング手法を開発する．具体的には，SCAN-XP は並列化におけるボトルネックとグラフの次数の偏りによる負荷分散の問題を解消するとともに，Intel Xeon Phi の持つ 512 ビット SIMD 演算を各構造的類似度計算に対して適用し並列処理効率の向上を図る．

本研究の貢献は下記のとおりである．

- **高速性：**SCAN-XP は従来手法 SCAN と比べ非常に高速にクラスタリングを行うことが出来る．SCAN-XP は 8.5 億のエッジを持つグラフを約 36 秒でクラスタリングする (4.1 節).
- **スケーラビリティ：**SCAN-XP は従来手法 SCAN の持つ並列化ボトルネックを解消することにより，効率的な並列化を行う事ができる (4.2 節).
- **正確性：**SCAN-XP は従来手法 SCAN と同じクラスタリング結果を得ることができる (3.4 節).

我々の知る限り，SCAN-XP は構造的類似度に基づくグラフクラスタリングに対して Intel Xeon Phi の高い演算性能を利用した最初の手法である．我々の行った評価実験により，SCAN-XP はグラフ中のクラスタ，ハブ，外れ値を従来手法 SCAN と比較して 100 倍以上高速に検出できることを確認した．

表 1: 記号の定義

Symbol	Definition
G	与えられたグラフ.
V	G 中のノードの集合.
E	G 中のエッジの集合.
C	G から抽出されたクラスタの集合.
H	G からハブとして分類されたノードの集合.
O	G から外れ値として分類されたノードの集合.
$\Gamma(v)$	ノード v の構造的隣接ノード集合.
$N_\epsilon(v)$	ノード v の $\epsilon - neighborhood$ であるノードの集合.
$D(v)$	ノード v から Direct structure reachability であるノードの集合.
$C(v)$	ノード v と同じクラスタに属するノードの集合.
ϵ	構造的類似度の閾値, $0 \leqq \epsilon \leqq 1$.
μ	core となるために必要な構造的類似な隣接ノードの個数の閾値.
α, β	SCAN-XP で用いられる Block size のパラメータ.
$\sigma(u, v)$	u と v 間の構造的類似度.

SCAN やその拡張手法は幅広いアプリケーションで利用されているが，膨大な計算時間がボトルネックとなり大規模グラフに適応することは難しい．しかしながら，Intel Xeon Phi 上で大規模並列化を行うことにより，SCAN-XP はより広い範囲のアプリケーションの性能向上に貢献できる．

本稿の構成は以下の通りである. まず，2. 節において前提知識について説明する. 続いて 3. 節で本稿で提案する高速化手法について述べ，その評価を 4. 節で行う. そして 5. 節では関連研究について示し，6. 節ではまとめと今後の課題を述べる．

2. 前 提 知 識

本節では，構造的類似度に基づくグラフクラスタリングの基本的なアルゴリズムである SCAN および本研究で用いるメニーコアプロセッサ Intel Xeon Phi について述べる．

2.1 従来手法 SCAN

本節では，従来手法 SCAN を基に構造的類似度に基づくクラスタリングの定義を述べる．SCAN では，与えられた無向重みなしグラフ $G = \{V, E\}$ に対して，クラスタ集合 C，ハブ集合 H，外れ値集合 O を抽出する．

SCAN はエッジで繋がれた 2 つのノード間に対して，自身とその隣接ノードを含めたノードの集まりである定義 2.1 で示される構造的隣接ノード集合 $\Gamma(v)$ の比較に基づき，定義 2.2 で示される構造的類似度 $\sigma(v, w)$ を計算する．

定義 2.1 (構造的隣接ノード集合) $v \in V$ が与えられたとき，ノード v の構造的隣接ノード集合は $\Gamma(v)$ とすると，$\Gamma(v) = \{v, w \in V | \{v, w\} \in E\} \cup \{v\}$.

定義 2.2 (構造的類似度) $v, w \in V$ が与えられたとき，構造的類似度は $\sigma(v, w)$ とすると，$\sigma(v, w) = |\Gamma(v) \cap \Gamma(w)| / \sqrt{|\Gamma(v)||\Gamma(w)|}$.

定義 2.2 に基づき計算されたノード v とその隣接ノード w の構造的類似度が与えられた閾値 ϵ 以上を示すとき，ノード v とノード w は構造的に類似であるとみなされる．ここであるノード v と構造的に類似な隣接ノードの集合を $\epsilon - Neighborhood$ と呼び，$N_\epsilon(v)$ と表す．そして，$|N_\epsilon(v)|$ が閾値 μ 以上のとき v は core となる．core は以下の定義 2.3 で示される．

定義 2.3 (Core) $v \in V$ と $\epsilon \in \mathbb{R}$, $\mu \in \mathbb{N}$ が与えられたとき，v が *core* である必要十分条件は，$|N_\epsilon(v)| \geqq \mu$ である．

SCAN では，定義 2.3 で示された core を検出し，core をクラスタの中心として Direct structure reachability なノードを同一クラスタとして検出していくことでクラスタリングを行う．Direct structure reachability なノードとはある core ノード v から見て $N_\epsilon(v)$ に含まれるノードを指し，その集合を $D(v)$ とする．SCAN は $D(v)$ に含まれる全てのノード $w \in D(v)$ を v と同じクラスタに割り当てる．更に，クラスタに割り当てられたノード $w \in D(v)$ が core だった場合は $D(w)$ に含まれるノードを同一クラスタとして割り当てることでクラスタを拡張する．一方で，$w \in D(v)$ が core ではない場合はクラスタはそれ以上拡張されない．SCAN は同一クラスタとして検出された全ての core から Direct structure reachability なノードを同じクラスタに割り当てるまで繰り返される．このようにある core v から Direct structure reachability を満たすノードを辿った先にあるノードを core v からの Structure reachability を満たすという．構造的類似度に基づいたグラフクラスタリングでは，ある core から Structure reachability を満たすノードを全て同一クラスタとして検出する．構造的類似度に基づいたグラフクラスタリングのクラスタは次のように定義される．

定義 2.4 (クラスタ) *core* v のクラスタを $C(v)$ とすると，$C(v) = \{u \in D(w) | w \in C(v)\}$．ここで $C(v)$ の初期状態は $C(v) = \{v\}$ である．

定義 2.4 で示されたクラスタを全て検出した後，定義 2.5 に基づきクラスタに属していないノードを 2 つ以上のクラスタと隣接しているならばハブ，そうでない場合は外れ値に分類する．

定義 2.5 (ハブと外れ値) グラフ中のノード $u \notin C$ が与えられたとき， その隣接ノードが *2* つ以上のクラスタに属していた場合 u はハブである．一方で，ハブの条件を満たさない場合は u は外れ値である．

SCAN のアルゴリズムを Algorithm 1 に示す．初期状態で全てのノードにラベルとして unclassified が与えられている．SCAN アルゴリズムは Step 1 にて，グラフ中の全てのエッジに対して構造的類似度を計算する．そして，ϵ 以上の構造的類似度を持つエッジを μ 個持つノードを core として検出する．そして，Step 2 においては，検出した core を用いてクラスタの検出を行う．unclassified なノードに対して，それが core でない場合はラベルを non-menber に変更する．一方で core である場合はそのノードをクラスタの核とする．そして，構造的類似度に基づくクラスタリングの定義に基づき，core から Structure reachability を満たすノードを全て同一クラスタとして検出する．同一クラスタとして検出されたノードのラベルはクラスタ ID に書き換えられる．これらの処理は unclassified なノードがなくなるまで繰り返す．最後に Step 3 において，non-member とラベル付けされたノード，つまりクラスタに属していないノードに対して，ハブか外れ値かの判定を行う．それぞれクラスタ ID の異なる隣接ノードが存在する場合はハブ，そうでない場合は外れ値として判定する．

Algorithm 1 Baseline method: SCAN [11]

Input: $G = \{V, E\}$, $\epsilon \in \mathbb{R}$, and $\mu \in \mathbb{N}$
Output: C, H, and O

```
 1: ∀v ∈ V are labeled as unclassified;
 2:
 3: // Step 1: Core detection based on Definition 2.3
 4: for each edge(v, w) ∈ E do
 5:    compute σ(v, w) by Definition 2.2;
 6: end for
 7:
 8: // Step 2: Cluster construction based on Definition 2.4
 9: for each unclassified node v ∈ V do
10:    if N_ε(v) ≧ μ then
11:       generate new clusterID, and add v into C(v);
12:       all w ∈ N_ε(v) into queue Q;
13:       while Q ≠ ∅ do
14:          u = Q.pop;
15:          for each w ∈ D(u) do
16:             if w is unclassified or non-member then
17:                assign current clusterID to w, and add w into C(v);
18:             end if
19:             if w is unclassified then
20:                insert w into Q;
21:             end if
22:          end for
23:       end while
24:       addC(v) into C;
25:    else
26:       v are labeled as non-member;
27:    end if
28: end for
29:
30: // Step 3: Hubs/outliers detection based on Definition 2.5
31: for each non-member node v do
32:    if ∃u, w ∈ Γ(v) s.t. C(u) ≠ C(w) then
33:       label node v as hub, and H = H ∪ {v};
34:    else
35:       label node v as outlier, and O = O ∪ {v};
36:    end if
37: end for
```

2.2 Intel Xeon Phi

Intel Xeon Phi [16] は Intel から提供されているメニーコアプロセッサである．現在までに，KNC (KNights Corner) と KNL (KNights Landing) の 2 つが発売されている．

KNC は第 1 世代の Xeon Phi である．KNC は GPU と同じようなカードの形でのみ提供され，PCIe によりホストと通信する．内部では独立して Linux OS が動作しており，GPU とは異なり単体でのプログラムの実行も可能である．57 から 61 の物理コアを搭載し，各コアは 512 ビットの SIMD 演算機を持ち，1.10 GHz で動作する．また，6GB から 16GB のオンチップメモリを持ち，各コアは高速にオンチップメモリに高速にアクセスすることが可能である．KNC は倍精度浮動小数点演算で 1TFLOPS 以上の性能を発揮する．

KNL は第 2 世代の Xeon Phi であり，現在最も新しい機種である．KNL は KNC と同様のカードの形態だけでなく，単体で動作する CPU という 2 つの形で提供されている．KNL は最大 72 個の物理コアを持つ．各コアは KNC と同様に 512 ビットの SIMD 演算機が搭載されており，1.3GHz から 1.5GHz で動作する．更に，KNL は 16GB のオンチップメモリ MCDRAM と最大 384GB の DDR4 メモリを利用できる．KNL は倍精度浮動小数点演算で 3TFLOPS 以上の性能を発揮する．

上記で述べた通り，Xeon Phi は非常に高い演算性能を持っているが，その性能を発揮するためにはプログラムを Xeon Phi に最適化することが必要である．なぜならば，各コア一つ一つの性能は CPU に大きく劣るからである．したがって，演算性

Algorithm 2 SCAN-XP

Input: $G = \{V, E\}$, $\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
Output: C, H, and O

```
1: ∀v ∈ V are labeled as unclassified;
2:
3: // Step 1: Parallel core detection in Section 3.2
4: for each edge (v, w) ∈ E do in parallel
5:     run Algorithm 3;
6: end for
7:
8: // Step 2: Parallel cluster construction in Section 3.3
9: for each core node v ∈ V do in parallel
10:    for each w ∈ N_ε(v) do
11:        if find(v) ≠ find(w) then
12:            get C(v) ∪ C(w) by using union(v, w) and CAS instruction;
13:            node v and node w are labeled as cluster-member;
14:        end if
15:    end for
16: end for
17:
18: // Step 3: Parallel hubs/outliers detection
19: for each node v that is not included in any clusters of C do in parallel
20:    if ∃u, w ∈ Γ(v) s.t. C(u) ≠ C(w) then
21:        label node v as hub, and H = H ∪ {v};
22:    else
23:        label node v as outlier, and O = O ∪ {v};
24:    end if
25: end for
```

能を十分に発揮するためには各コアを十分に利用した並列化と512 ビットの SIMD 演算を活用できるアルゴリズムを設計する必要がある．また，メモリアクセスのレイテンシが大きいためキャッシュヒット率の高いデータ構造や明示的なプリフェッチ命令の利用などメモリアクセスに伴うレイテンシの隠蔽を考慮してプログラムを作成する必要がある．

3. 提案手法 SCAN-XP

本節では提案手法 SCAN-XP について概説する．SCAN-XP のアルゴリズムを Algorithm 2 に示す．最初に SCAN-XP の基本的なアイデアを 3.1 節で述べる．その後，各手法の詳細について 3.2 節以降で説明する．

3.1 基本アイデア

前節で述べた通り，SCAN はグラフに含まれる全てのエッジに対して構造的類似度を計算する必要がある．したがって，高速化のためには精度を落とすことなく，いかにして構造的類似度を高速に計算するかが重要な課題となる．この課題を解決するために，Xeon Phi の高い並列計算性能用いた高速化手法 SCAN-XP を提案する．SCAN-XP は Algorithm 1 で示した (Step 1) core 検出処理，(Step 2) クラスタ検出処理，および (Step 3) ハブ・外れ値検出処理の 3 つのステップそれぞれを Xeon Phi を用いて並列化および最適化する．

Step 1 の core 検出処理では，構造的類似度計算がエッジごとに独立であることに着目し，エッジ単位でのスレッド並列化を行う．更に，構造的類似度計算の内部での共通隣接ノード検出処理は大きなボトルネックとなるため，SIMD 演算によるデータ並列化によりさらなる高速化を行う．Step 2 のクラスタ検出処理では，Union-Find 木と CAS (Compare And Swap) 命令を用いた並列化可能なアルゴリズムを導入する．既存手法 SCAN では，core となったノードから探索的にクラスタを構成するノードを見つける必要があったため，スレッド並列化が困難であった．そのため Xeon Phi の各物理コアを活用し，演算

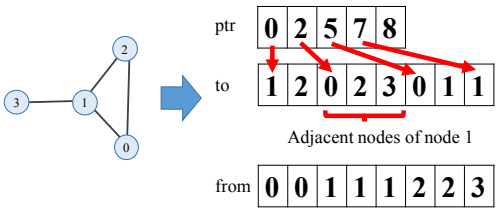

図 2: グラフを格納するデータ形式の例

性能を十分に発揮する実装が難しいという問題点がある．本手法で提案する Union-Find 木を用いた手法は並列化ボトルネックを改善し，効率的なクラスタ検出を実現する．最後に Step 3 のハブ・外れ値検出処理では，ノード単位でのスレッド並列化を行う．Step 3 については OpenMP を用いた単純な並列化であるため本稿では詳細な説明は割愛する．

3.2 core 検出の並列化

core 検出処理では全てのエッジに対する構造的類似度計算が多くの計算時間を占めている．そこで，Xeon Phi を用いたスレッド並列化と SIMD 演算を用いたデータ並列化による構造的類似度計算の高速化を行う．

3.2.1 構造的類似度計算のスレッド並列化

core 検出処理における構造的類似度の計算がグラフ中の各エッジに対して独立していることに着目し，Xeon Phi の持つ大量の物理コアを活用しスレッド並列化を行う．各構造的類似度の計算は独立しているが，単純にノード単位でスレッド並列化をした場合，グラフの次数分布の偏りからスレッド間の負荷分散に偏りが生じるという問題がある．

上記の問題について，グラフ処理で一般的に使用されているデータレイアウト CRS (Compressed Row Storage) 形式 [1,17] を用いてより詳細に説明する．CRS 形式は ptr 配列と to 配列からなるデータレイアウトである．to 配列はノード 0 の隣接ノードから順に各ノードの隣接ノードが連続して格納されている．ptr 配列は to 配列のどこにどのノードの隣接ノードが格納されているかを示すポインタである．CRS 形式は連結リスト方式などに比べて空間効率やキャッシュ効率の面で良いという利点があるため，Xeon Phi 上でグラフを処理する場合に非常に有効である．しかしながら，CRS 形式を用いてスレッド並列化をする場合，to 配列からは元のノードを参照できないため，ptr 配列の要素単位，すなわちノード単位での並列化を行う必要がある．ところが，グラフは一般的にべき条則に従う次数分布を持つことが知られているため [18]，各ノードの持つ隣接ノード集合の大きさに偏りが生じる．結果として，スレッド間のタスク割当量に不均衡が生じ，並列化効率の低下を招くことになる．

そこで本稿では，図 2 に示すように to 配列の要素と一対一で対応した逆ポインタである from 配列を新たに導入する．ptr 配列および to 配列に加えて from 配列を持つことで，CRS 形式の持つ空間効率やキャッシュ効率を維持しつつ，エッジ単位でのスレッド並列化を行うことが可能となる．結果として各スレッドはエッジ 1 つ分の構造的類似度計算を担当することになる．ゆえに，前述したグラフの次数分布の偏りに関して影響を受けることなく並列化が可能である．

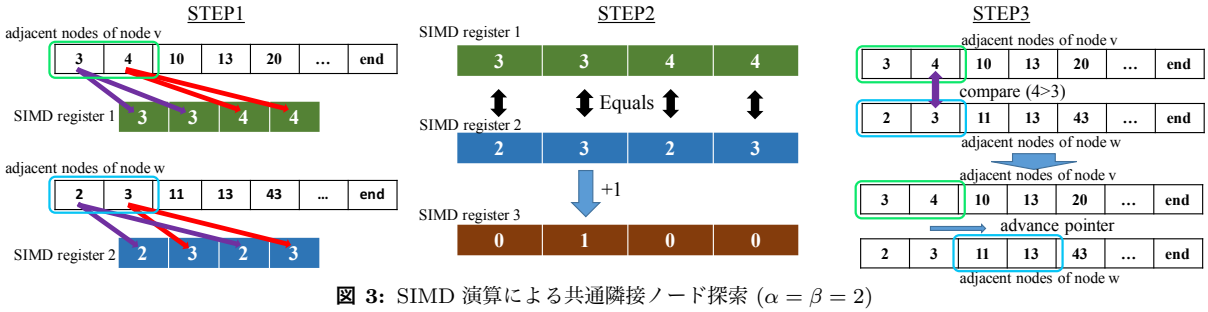

図 3: SIMD 演算による共通隣接ノード探索 ($\alpha = \beta = 2$)

Algorithm 3 SIMD-based structural similarity computation

Input: $v, w \in V$,
Output: $\sigma(v, w)$
1: // Initialization
2: **if** $|\Gamma(v)| \geqq 2|\Gamma(w)|$ (or $2|\Gamma(v)| \leqq |\Gamma(w)|$) **then**
3: $\alpha = 1, \beta = 16$ (or $\alpha = 16, \beta = 1$);
4: **else**
5: $\alpha = \beta = 4$;
6: **end if**
7: get head pointers vp and wp from $\Gamma(v)$ and $\Gamma(w)$, respectively;
8: get tail pointers v_end and w_end from $\Gamma(v)$ and $\Gamma(w)$, respectively;
9:
10: **while** $vp < v\_end$ && $wp < w\_end$ **do**
11: // STEP 1 and STEP 2
12: load α and β nodes into SIMD register reg_v and reg_w from $\Gamma(v)$ and $\Gamma(w)$, respectively;
13: get the number of common nodes c between reg_v and reg_w by using SIMD instructions;
14: $vw\_common += c$;
15: // STEP 3
16: **if** $vp+\alpha == wp+\beta$ **then**
17: $vp += \alpha$, $wp += \beta$;
18: **else if** $vp + \alpha > wp + \beta$ **then**
19: $wp += \beta$;
20: **else**
21: $vp += \alpha$;
22: **end if**
23: **end while**
24: $\sigma(v, w) = (vw\_common + 2)/\sqrt{|\Gamma(v)||\Gamma(w)|}$;

3.2.2 SIMD による共通ノード検出処理のデータ並列化

次に構造的類似度計算を SIMD によるデータ並列化することにより高速化する．定義 2.2 で示したように，構造的類似度計算には隣接する 2 つのノードが持つ隣接ノード集合の積集合を求める必要がある．この計算は，既存研究 [15] において，sort merge join が最も高速となることが示されている．そのため本研究では sort merge join による積集合計算を採用し，その処理を SIMD 演算を用いて実装する．

Algorithm 3 と図 3 に SIMD による積集合計算の概要を示す．提案手法は，Inoue らにより提案されたブロックベースの積集合計算手法 [19] に基づき，ポインタで示された複数個のノードを SIMD レジスタ上にロードし，並列に比較を行うことにより積集合計算を高速化する．

ここで我々は各レジスタ上にロードされるノード数を表すパラメータ α，β を導入する．積集合計算を行う配列のサイズ比に応じて，α，β の比率を変更することでグラフの次数の偏りによる SIMD 演算の効率の低下を抑制する．例えば，本研究で対象とする Xeon Phi では SIMD レジスタが 512 ビットであるため，32 ビットの integer で表現されたグラフを対象とする場合，(α,β)=(1,16)，(2,8)，(4,4) の 3 パターンが考えられる．基本的には比較する配列のサイズがほぼ同一の場合は α，β も同一とし，サイズが異なるほど α，β も異なる値を選択する．

図 3 は 128 ビットの SIMD レジスタ上で $\alpha = \beta = 2$ の場合の具体的なアルゴリズムの動作を示している．最初に Step 1 では構造的類似度を計算するノード v と w の隣接ノード配列からポインタで示された $\alpha(= 2)$， $\beta(= 2)$ 個の要素をそれぞれ選択し，SIMD レジスタ 1 及び 2 上にロードする．次に Step 2 にでは，SIMD レジスタ上にロードされたデータ同士の比較を compare 命令を用いて行う．比較を行った結果，値が一致した場合は，カウント用の SIMD レジスタ 3 の値が 1 加算される．最後に Step 3 において，ポインタで示された要素の内，一番大きな要素同士を比較し，小さい方のポインタが α もしくは β 分だけ進められる．この処理をどちらかのポインタが配列を全て参照するまで繰り返し行われる．

3.3 クラスタ検出処理のスレッド並列化

本節ではクラスタ検出処理の並列化について述べる．2 節で述べた SCAN のクラスタ検出処理は core からどこまでのノードがクラスタに入るのかは事前に明らかになっておらず，クラスタに含まれるノードを探索的に求める必要がある．その結果として，どのノードをどのスレッドで並列処理すれば良いのか決定できず，容易にスレッド並列化を行うことができない．

そこで本稿では，我々は Union-Find 木を用いた並列化アルゴリズムを提案する．Union-Find 木は素集合に対して，集合同士の統合処理 (union) と要素が属する集合の特定 (find) を高速に行うことが出来るデータ構造である．本稿では Union-Find 木と CAS 命令を組み合わせることでクラスタ検出処理の効率的な並列化を行う．

具体的な処理の詳細を Algorithm 2 を用いて説明する．Algorithm 2 の 12 行目の union(v, w) は，v の所属するクラスタと w の所属するクラスタを統合する命令である．また，Algorithm 2 の 11 行目の find(v) は v の所属するクラスタを検出する命令である．Algorithm 2 の 9 行目から 16 行目において，各スレッドは，core と判定されたノードをそれぞれ参照し，core とその N_ϵ で構成されるクラスタを検出する．このとき，各クラスタ間で格納されたノードに重複があった場合，それらのクラスタは union 命令により統合する．この処理は全ての core が参照されるまで繰り返し実行する．また，union 命令による統合処理は，スレッド並列間の書き換えの衝突を検知できる CAS 命令を使用して行われ，スレッド間のデータの整合性を保つようにする．

3.4 SCAN-XP の正確性

SCAN-XP 高速にクラスタリングを行う一方で，常に SCAN と同じクラスタリング結果を保証する．本節では SCAN-XP の正確性について以下の定理 3.1 が成り立つことを示す．

定理 3.1 (SCAN-XP の正確性) *SCAN-XP* は常に *SCAN* と同じクラスタリング結果を示す.

証明：SCAN-XP と SCAN は共に定義 2.5 に基づき，クラスタに属さないノードをハブと外れ値に分類する．したがって，SCAN-XP が SCAN と同等のクラスタをグラフから抽出するなら，定理 3.1 は証明される．定義 2.4 により，SCAN と同じクラスタを抽出するために必要な十分条件は (1) SCAN-XP が SCAN で検出する全ての core を検出することおよび (2) SCAN-XP がエッジで繋がった全ての core とその隣接ノードが同じクラスタに属するかどうかを判定することである．

Algorithm 2 により，SCAN-XP が上記の必要十分条件を満たすことが示される．Algorithm 2 の 3 行目から 6 行目において，SCAN-XP はグラフ G の全てのエッジ E に対して，構造的類似度計算を行う．これにより，グラフ G 中に含まれる core ノードは全て検出される．したがって，必要十分条件 (1) は満たされる．同様に，Algorithm 2 の 3 行目から 6 行目において，SCAN-XP は全てのエッジの構造的類似度を計算する．したがって，エッジで繋がれた全ての core とその隣接ノードが Direct structure reachability であるかどうかを計算している．よって，必要十分条件 (2) は満たされる． □

4. 評価実験

本節では，実データに対して提案手法 SCAN-XP および既存手法 SCAN の実行時間を比較し，提案手法の有効性を検証する．実験に使用した計算機は Xeon E5-1620 (以下，CPU)，Xeon Phi 3120A (以下，KNC)，Xeon Phi 7250 (以下，KNL) である．KNC の MPSS は 3.7.2 を用い，KNL のメモリはフラットモードで使用した．また，コンパイラは icpc 17.0.0 を用い，最適化オプションとして-O3 を使用した．実験環境の詳細は表 3 に示す．また，本実験において，測定に使用した 7 つの実データセットは以下の表 2 に示す．

表 2: データセットの詳細

Dataset	Nodes	Edges	Data source
com-youtube	1134890	2987624	SNAP [20]
web-BerkStan	685230	6649470	SNAP [20]
soc-Pokec	1632803	22301964	SNAP [20]
com-LiveJournal	3997962	34681189	SNAP [20]
soc-LiveJournal1	4846609	42851237	SNAP [20]
com-Orkut	3072441	117185083	SNAP [20]
webbase2001	115554441	854809761	LAW [21]

データセットの内，webbase2001 は非常に大きなデータセットであり，CPU 及び KNC ではメモリの容量が原因で実行が不可能であったため，KNL での実行結果のみを示す．また，SCAN は逐次手法であるため CPU 上での逐次実行による測定を行った．一方で，SCAN-XP は CPU，KNC，KNL 上でスレッド数を変えて測定を行った．

構造的類似度に基づくグラフクラスタリングにおけるユーザ定義の閾値 ϵ 及び μ について，SCAN と SCAN-XP はともに処理時間に対し影響を受けることはほぼ無かったため，本稿においては $\epsilon = 0.3$，$\mu = 2$ と定める．また，SCAN-XP における α，β は予備実験により，CPU ではでは比較する配列のサイズの違いが 2 倍未満であるとき $\alpha = 2$，$\beta = 4$，2 倍以上であるとき $\alpha = 1$，$\beta = 8$ とし，KNC 及び KNL では比較する配列のサイズの違いが 2 倍未満であるとき $\alpha = 4$，$\beta = 4$，2 倍以上であるとき $\alpha = 1$，$\beta = 16$ とするのが全体として良い傾向を示すことがわかっているため，本実験ではその値を設定する．

表 3: 実験環境

CPU	Xeon E5 1620	Xeon Phi 3120A	Xeon Phi 7250
Main memory	16GB	6GB	MCDRAM 16GB DDR4 96GB
Clock rate	3.5 GHz	1.10GHz	1.4GHz
# of Core/# of Thread	4/8	57/228	68/272
OS	Cent OS 7	Linux 3.10	Cent OS 7
SIMD	AVX2	IMCI	AVX-512

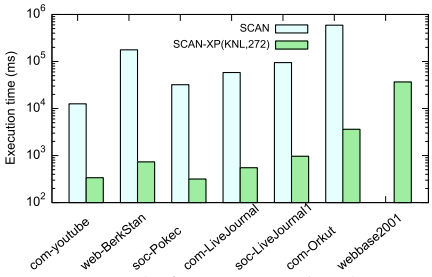

図 4: 実データに対する実行時間

4.1 実行時間の比較

CPU 上で 1 スレッドで実行した従来手法 SCAN と KNL 上で 272 スレッドで実行した SCAN-XP の実行時間の比較を図 4 に示す．SCAN-XP(KNL，272) は一貫して SCAN に対して 100 倍以上の高速であることが示された．特に 1 億以上のエッジを持つ com-Orkut に対して，SCAN-XP (KNL，272) は 3.6 秒でクラスタリングを行った．これは SCAN と比較して約 164 倍高速である．更に，8 億 5 千万エッジを持つ，webbase2001 に対して SCAN-XP はわずか約 36 秒でクラスタリングした．

4.2 スケーラビリティの比較

SCAN-XP のスケーラビリティを評価するために，CPU，KNC，KNL 上でスレッド数を変化させて実験を行った．その結果を図 5 に示す．図から，SCAN-XP (CPU)，SCAN-XP (KNC)，SCAN-XP (KNL) は一貫して SCAN より高速であることが示された．また，SCAN-XP (KNC) と SCAN-XP (KNL) はスレッド数の増加に対して実行速度も向上し，最終的に一貫してそれぞれ約 50，100 倍以上 SCAN より高速となった．しかし，com-youtube 及び web-Berkstan では SCAN-XP(CPU) と比較して，そこまでの高速化はされていない．これは，これら 2 つのグラフの大きさが小さいため Xeon Phi に搭載された多量の物理コアを十分に働かせることができなかったからだと考えられる．以上により，SCAN-XP は大規模なグラフに対して良いスケーラビリティを示すことがわかる．

4.3 高速化手法の効果の検証

前述したように，SCAN-XP は core 検出処理とクラスタ検出処理に対して高速化手法を適用した．各処理に適用した高速化手法の評価を行うために，各処理の実行時間を測定した．

図 6 に，各アルゴリズムの core 検出処理の実行時間を示す．図 5 と同様に SCAN-XP は com-youtube，web-Berkstan を覗いたグラフに対して，良いスケーラビリティを示している．更に SCAN-XP (KNL) は 272 スレッドでの実行時に 1 億エッジを持つ com-Orkut に対して 3.4 秒で core 検出処理を終える．これは従来手法 SCAN より 168 倍高速である．

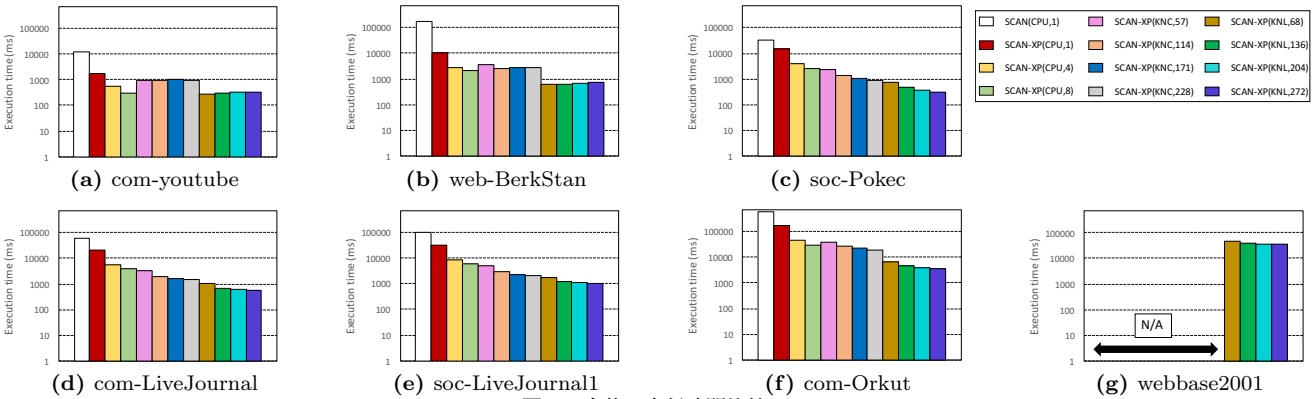

図 5: 全体の実行時間比較

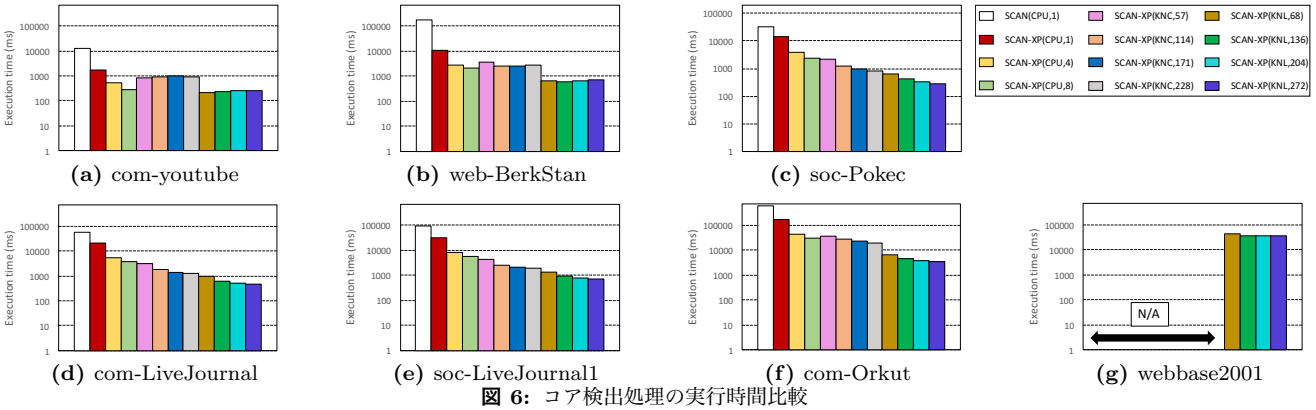

図 6: コア検出処理の実行時間比較

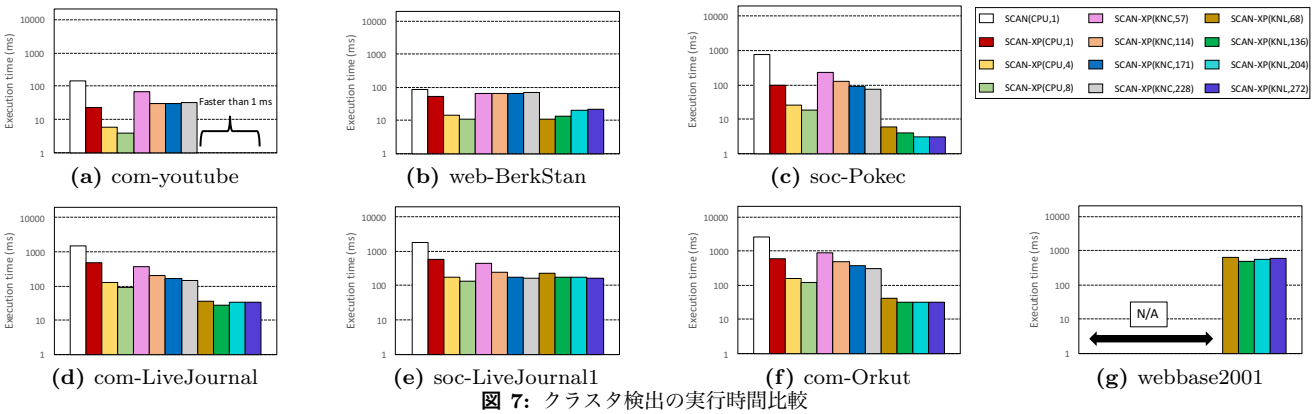

図 7: クラスタ検出の実行時間比較

次に図 7 にクラスタ検出処理の実行時間を示す．core 検出処理と同様に SCAN-XP (KNL) は他のアルゴリズムに対して良いスケーラビリティを示す．対照的に，SCAN-XP (KNC) は SCAN-XP (CPU) より遅い．これは，Union-Find 木を用いたクラスタ検出アルゴリズムが SCAN のクラスタ検出処理と比較して分岐予測ミスやキャッシュミスが起こりやすいこと，更に KNC がアウトオブオーダー実行に対応していないため，そういったレイテンシの影響を受けてしまうからだと考えられる．

5. 関 連 研 究

グラフから隠されたクラスタ構造を抽出する手法は非常に重要な技術であり，現在に至るまで様々なアルゴリズムが提案されてきている．min-max cut に基づく手法 [4,5] や Modularity に基づく手法 [9,10] はグラフクラスタリングにおける代表的な手法である．また，近年においては，より高精度にグラフに内包されたクラスタ構造を捉えることが可能な構造的類似度に基づく手法が考案され，幅広く利用されている．本節では，主な構造的類似度に基づく手法について述べる．

構造的類似度に基づく手法 [14, 15, 22–26] はグラフから高い精度でクラスタを検出するだけでなく，ハブや外れ値といったグラフ上で特別な役割を持つノードを検出することが可能なグラフクラスタリング手法である．

Xu らにより提案された SCAN [11] は，最も代表的な構造的類似度に基づくグラフクラスタリング手法である．SCAN は密度ベースのクラスタリング手法として有名な DBSCAN [12] のグラフデータに対する拡張である．Xu らは SCAN は Modularity に基づく手法と比較してより正確なクラスタリング結果が得られることを示している．しかし，2. 節で述べたように SCAN は構造的類似度計算の計算量が原因となり，大規模グラフに対して計算時間が膨大となる問題がある．

SCAN の計算速度を向上させるために，最近になって幾つかの手法 [13–15] が提案されている．Lim らによって提案された LinkSCAN* はクラスタリングの速度を速めるためにエッジサンプリング手法を取り入れ高速に近似解を求める手法である．一方で，SCAN++ [14] と pSCAN [15] は，大規模グラフに対する高速かつ正確な構造的類似度に基づくグラフクラスタリング手法である．この 2 つの手法は実世界に存在するグラフの頻出構造に着目し，構造的類似度の計算回数を効率的に削減し，正確なクラスタリング結果を高速に求めることができる．

Zhao らによって提案された PSCAN [26] は MapReduce を利用した分散メモリ環境で SCAN を並列化した手法である．PSCAN は，最初に map，reduce 処理により core を検出し，そして core でないノードをノイズとして排除する．結果として，PSCAN で得られるクラスタリング結果は LinkSCAN* と同様の近似解である．MapReduce の特性上，中間結果をストレージに繰り返し読み書きする必要があるため，SCAN++や pSCAN より多くの計算時間を必要とする．

6. ま と め

Xeon Phi を用いて高速に構造的類似度に基づいたグラフクラスタリングを行う SCAN-XP を提案した．SCAN-XP は従来手法である SCAN と比較し，KNC 上で一貫して 50 倍以上，KNL 上で 100 倍以上の高速化を達成した．更に KNL 上で実行された SCAN-XP は 8 億 5 千万以上のエッジを持つグラフをたった 36 秒でクラスタリングした．

今後の課題として，512 ビット SIMD 演算を活かすことを含めた Xeon Phi への更なる最適化や，複数の Xeon Phi を用いた分散メモリ環境での並列化手法，更にメモリの限られた Xeon Phi 上で更に大規模なグラフデータを処理する方法の検討を行う予定である．また，SCAN++や pSCAN で提案されている計算回数削減アプローチの取り入れについて検討する．

謝　　辞

本研究は，JSPS 科研費 JP26280037, JP16H07410，平成 28 年度筑波大学研究基盤支援プログラムならびに筑波大学計算科学研究センター学際共同利用プロジェクト (COMA) の助成を受けたものである．

文　　献

[1] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. Rabbit Order: Just-in-Time Parallel Reordering for Fast Graph Analysis. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 22–31, 2016.

[2] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Efficient Ad-hoc Search for Personalized PageRank. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 445–456, 2013.

[3] Pei Lee, Laks V. S. Lakshmanan, and Evangelos E. Milios. Incremental Cluster Evolution Tracking from Highly Dynamic Network Data. In *Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE)*, pages 3–14, 2014.

[4] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A Min-max Cut Algorithm for Graph Partitioning and Data Clustering. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 107–114, 2001.

[5] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[6] M. E. J. Newman and M. Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review E*, 69(2):026113, Feb 2004.

[7] M. E. J. Newman. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review*, E 69(066133), 2004.

[8] Aaron Clauset, M. E. J. Newman, , and Cristopher Moore. Finding Community Structure in Very Large Networks. *Physical Review*, E 70(066111), 2004.

[9] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E.L.J.S. Mech. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[10] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Fast Algorithm for Modularity-based Graph Clustering. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1170–1176, 2013.

[11] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. SCAN: A Structural Clustering Algorithm for Networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 824–833, New York, NY, USA, 2007. ACM.

[12] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[13] S. Lim, S. Ryu, S. Kwon, K. Jung, and J. G. Lee. LinkSCAN*: Overlapping Community Detection Using the Link-space Transformation. In *Proceedings of the IEEE 30th International Conference on Data Engineering (ICDE)*, pages 292–303, March 2014.

[14] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs. *Proceedings of the Very Learge Data Bases (PVLDB)*, 8(11):1178–1189, August 2015.

[15] L. Chang, W. Li, X. Lin, L. Qin, and W. Zhang. pSCAN: Fast and Exact Structural Graph Clustering. In *Proceedings of the IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 253–264, May 2016.

[16] James Jeffers and James Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[17] James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[18] Michalis Faloutsos, Petors Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 1999)*, pages 251–262, New York, NY, USA, 1999. ACM.

[19] Hiroshi Inoue, Moriyoshi Ohara, and Kenjiro Taura. Faster Set Intersection with SIMD instructions by Reducing Branch Mispredictions. *Proceedings of the Very Learge Data Bases (PVLDB)*, 8(3):293–304, August 2015.

[20] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`, June 2014.

[21] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[22] T. R. Stovall, S. Kockara, and R. Avci. GPUSCAN: GPU-Based Parallel Structural Clustering Algorithm for Networks. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3381–3393, Dec 2015.

[23] Dustin Bortner and Jiawei Han. Progressive Clustering of Networks Using Structure-Connected Order of Traversal. In *Proceedings of the IEEE 26th International Conference on Data Engineering (ICDE)*, pages 653–656, March 2010.

[24] Heli Sun, Jianbin Huang, Jiawei Han, Hongbo Deng, Peixiang Zhao, and Boqin Feng. gSkeletonClu: Density-Based Network Clustering via Structure-Connected Tree Division or Agglomeration. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM)*, pages 481–490, 2010.

[25] Nurcan Yuruk, Mutlu Mete, Xiaowei Xu, and Thomas A. J. Schweiger. AHSCAN: Agglomerative Hierarchical Structural Clustering Algorithm for Networks. In *Proceedings of the International Conference on Advances in Social Network Analysis and Mining (ASONAM)*, pages 72–77, 2009.

[26] Weizhong Zhao, Venkata Swamy Martha, and Xiaowei Xu. PSCAN: A Parallel Structural Clustering Algorithm for Big Networks in MapReduce. In *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 862–869, 2013.