Intel Xeon Phi による SCAN クラスタリングの分散並列化

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　SCAN はノード間の接続の強さを示す指標である構造的類似度に基いて，グラフからクラスタを抽出するグラフクラスタリング手法である．SCAN はクラスタとの関係性が薄いノードをハブや外れ値に分類することで従来の手法と比較して高い精度でクラスタを検出できる．しかしながら，SCAN は全てのエッジに対して構造的類似度の計算を行うため，大規模なグラフへの適用には多くの実行時間が必要とされる．そこで，本稿では Intel Xeon Phi による大規模並列計算により，SCAN を高速に計算する手法 DSCAN を提案する．DSCAN は，(1) 既存の Xeon Phi 上の並列手法 SCAN-XP に対して逐次の高速化手法 pSCAN の計算回数削減手法を組み込むことで更なる高速化を図り，(2)SCAN の分散並列化において問題となる隣接ノード情報の通信コストを事前計算によるエッジの枝刈りを行うことで削減し，効率的な分散並列化を行う．本稿では実データを用いた評価実験により，本手法の有用性を示す．

キーワード　Intel Xeon Phi，グラフクラスタリング，並列計算

1. は じ め に

グラフクラスタリングはグラフを解析するための基本的な解析手法である．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，グラフのコミュニティを解析することができる．その重要性から，min-max cut による手法 [1, 2] や Modularity [3] に基づく手法 [4–7] など，様々な手法が提案されてきた．

2007 年に Xu らにより提案された SCAN [8] 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手法である DBSCAN [9] 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，ノード間の接続密度を構造的類似度を計算することで評価し，図 1 のように従来手法でも検出できる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検出できることである．ハブは複数のクラスタ間の橋渡しをする役割を持ち，バイラルマーケティング等の分野で重要となる．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，ノイズとして扱われる．

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し，適用が難しい．

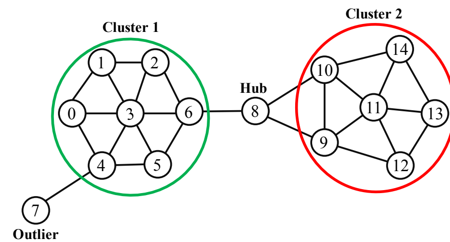

図 1: SCAN によるグラフクラスタリングの例

1.1 先行研究と課題

SCAN を大規模グラフに適用するために多くの高速化手法が提案されている．逐次の手法として，Shiokawa らによる SCAN++ [10] や Chang らによる pSCAN [11] や Son らによる anySCAN [12] がある．これらの手法は実世界のグラフに内包される構造特性をうまく捉えることで構造的類似度計算を巧妙に枝刈りし，大幅な高速化に成功した．しかしながら，いずれの手法も計算量は依然として $O(|E|)$ を必要とする．

また，SCAN を共有メモリ環境で並列化した手法として Takahashi らによる SCAN-XP [13] がある．SCAN-XP は Intel Xeon Phi 上で並列計算を行うことで SCAN の大幅な高速化に成功した手法である．しかし，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り，先に述べた逐次の高速化手法と比較して冗長な計算が発生してしまう．そのため，グラフの規模や構造によっては多くの実行時間がかかる．

以上のように，SCAN の高速化手法は多数提案されているが億規模のグラフに対してはいずれの手法でも多くの実行時間を必要とし，処理することは困難である．また，1 つの計算ノードに乗り切らない規模のグラフが実世界には存在し，これらの手法ではそういった莫大な規模のグラフに対処することができない．したがって，莫大な規模のグラフを処理するために SCAN の高速化は依然として重要な課題である．

表 1: 2. 節で定義する記号の一覧

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.
CC	G から抽出された core 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.
$CC(v)$	ノード v と同じ core 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.
$\sigma(u,v)$	u と v 間の構造的類似度.
$sd(v)$	v の *similar degree*.
$ed(v)$	v の *effective degree*.
$find(v)$	v の属するクラスタを特定する命令
$union(v,w)$	v の属するクラスタと w の属するクラスタをマージする命令

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 Intel Xeon Phi を用いた分散並列化により，大規模グラフを高速に並列処理する構造的類似度に基づくグラフクラスタリング手法 DSCAN (Distributed SCAN) を提案する．SCAN において，計算時間の多くを占める構造的類似度計算はグラフ中の各エッジごとに独立であるため，並列化による高速化が期待できる．また，近年では高性能計算分野を中心に Intel Xeon Phi などのメニーコアプロセッサを用いた並列化によるデータ処理の高速化が注目されている．Intel Xeon Phi に着目した手法としては，SCAN-XP が存在する．しかしながら，先に述べたように大規模グラフの処理には未だ多くの実行時間を必要とする．そこで，DSCAN においては SCAN-XP における問題点を解決する．具体的には (1) 逐次の高速化手法 pSCAN の計算回数削減回数を SCAN-XP に対して適用し，(2) SCAN を分散並列化をする上で大きなボトルネックとなる隣接ノード情報の通信を事前計算によるエッジの枝刈りにより削減することで複数の Intel Xeon Phi を用いた効率的な分散並列化を行う．本研究の貢献を以下に示す．

- **高速性：**DSCAN は従来手法 SCAN-XP と比較して，50 倍以上高速である (4.1 節).
- **スケーラビリティ：**DSCAN は分散並列化におけるボトルネックである通信コストをエッジの枝刈りにより軽減することで効率的な分散並列化を行う．(4.2 節).
- **正確性：**DSCAN は高い精度でクラスタを検出することができ，従来手法 SCAN 及び SCAN-XP と同じ結果を得る．

我々の知る限り，DSCAN は複数の Intel Xeon Phi による分散並列化により，高速に構造的類似度に基づくグラフクラスタリングを行う唯一の手法である．我々の行った実データに対する評価実験により，DSCAN は大規模グラフ union を 19 秒で処理できることを確認した．

2. 前提知識

本稿で提案する手法を説明する上で必要な前提知識について概説する．本節で用いる記号の定義について表 1 に示す．

2.1 構造的類似度に基づくグラフクラスタリング

構造的類似度に基づくグラフクラスタリング SCAN [8] では，グラフ $G = \{V, E\}$ からクラスタ，ハブ，外れ値を抽出する．まず，定義 2.1 に示した構造的類似度に基づき，クラスタの核となる core を検出する．

定義 2.1 (構造的類似度) $v, w \in V$ に対する構造的類似度は $\sigma(v,w) = |\Gamma(v) \cap \Gamma(w)| / \sqrt{|\Gamma(v)||\Gamma(w)|}$．ただし，$\Gamma(v) = \{v, w \in V | \{v, w\} \in E\} \cup \{v\}$ とする．

ノード v とその隣接ノード w の構造的類似度が閾値 ϵ 以上を示した場合，v と w は構造的に類似であるとみなす．ノード v の構造的に類似なノードの集合を $N_\epsilon(v)$ としたとき，定義 2.2 から $|N_\epsilon(v)|$ が閾値 μ 以上のとき v は core である．

定義 2.2 (Core) $v \in V$ と $\epsilon \in \mathbb{R}, \mu \in \mathbb{N}$ が与えられたとき，$|N_\epsilon(v)| \geqq \mu$ ならば v は *core* である．

ノード v が core だった場合，v と $N_\epsilon(v)$ に含まれるノードを同じクラスタとする．ここで，同じクラスタとしたノード $u \in N_\epsilon(v)$ が core だった場合は更に $N_\epsilon(u)$ に含まれるノードを v と同一のクラスタとする．構造的類似度に基づくグラフクラスタリングでは G 中の全ての core がクラスタに含まれるまで，一連の処理を繰り返し行う．この処理により，定義 2.3 に基づくクラスタが抽出される．

定義 2.3 (クラスタ) *core* v が与えられたとき，v のクラスタを $C(v)$ とすると，$C(v) = \{u \in N_\epsilon(w) | w \in C(v) \wedge w = core\}$.

最後に，いずれのクラスタにも所属していないノードを定義 2.4 に基づきハブまたは外れ値に分類する．

定義 2.4 (ハブと外れ値) グラフ中のいずれのクラスタにも属していないノード u 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．そうでない場合 u は外れ値である．

2.2 SCAN-XP [13]

SCAN-XP は，Intel Xeon Phi 上でのスレッド並列化と SIMD 命令によるデータ並列化により，SCAN を高速化した手法である．SCAN-XP のアルゴリズムを Algorithm 1 に示す．2 SCAN-XP の各処理 (1) 並列 core 検出処理，(2) 並列クラスタ検出処理，(3) 並列ハブ・外れ値処理について概説する．

並列 core 検出処理：SCAN-XP はまず，並列に全てのエッジに対して構造的類似度計算を行うことで，全てのノードが core か否かを判定する．構造的類似度計算における隣接ノードの積集合計算は，Algorithm 2 に示す SIMD を用いた sort merge join のデータ並列化手法で高速化される．

並列クラスタ検出処理：全ての core が検出された後 SCAN-XP はクラスタ検出処理を行う．SCAN-XP は各 core ノードとその隣接ノードのみがクラスタを構築するかどうか判定してい

Algorithm 1 SCAN-XP

```
Input: G = {V, E}, ε ∈ ℝ and μ ∈ ℕ
Output: C(Set of clusters), H(Set of hubs), and O(Set of outliers)
 1: ∀v ∈ V are labeled as unclassified;
 2:
 3: // Step 1: 並列 core 検出処理
 4: for each edge (v, w) ∈ E do in parallel
 5:    run Algorithm 2;
 6: end for
 7:
 8: // Step 2: 並列クラスタ検出処理
 9: //C(v):Set of nodes that are belong to the same cluster as node v
10: for each core node v ∈ V do in parallel
11:    for each w ∈ N_ε(v) do
12:       if find(v) ≠ find(w) then
13:          get C(v) ∪ C(w) by using union(v, w) and CAS instruction;
14:          node v and node w are labeled as cluster-member;
15:       end if
16:    end for
17: end for
18:
19: // Step 3: 並列ハブ・外れ値検出処理
20: for each node v that is not included in any clusters of C do in parallel
21:    if ∃u, w ∈ Γ(v) s.t. find(u) ≠ find(w) then
22:       label node v as hub, and H = H ∪ {v};
23:    else
24:       label node v as outlier, and O = O ∪ {v};
25:    end if
26: end for
```

Algorithm 2 SIMD による構造的類似度計算

```
Input: v, w ∈ V,
Output: σ(v, w)
 1: // Initialization
 2: select α and β according to difference between two adjacent array size
 3: get head pointers vp and wp from Γ(v) and Γ(w), respectively;
 4: get tail pointers v_end and w_end from Γ(v) and Γ(w), respectively;
 5:
 6: while vp < v_end && wp < w_end do
 7:    load α and β nodes into SIMD register reg_v and reg_w from Γ(v) and
       Γ(w), respectively;
 8:    get the number of common nodes c between reg_v and reg_w by using
       SIMD instructions;
 9:    vw_common = vw_common + c;
10:    if vp+α == wp+β then
11:       vp = vp + α, wp = wp + β;
12:    else if vp + α > wp + β then
13:       wp = wp + β;
14:    else
15:       vp = vp + α;
16:    end if
17: end while
18: σ(v, w) = (vw_common + 2)/√(|Γ(v)||Γ(w)|);
```

き，クラスタを形成する．このときクラスタの範囲が重複した場合は重複したクラスタは 1 つにまとめられる．SCAN-XP においては，Union-Find 木を用いてクラスタを表現しており，1 つのクラスタは 1 つの木を形成する．

並列ハブ・外れ値検出処理：最後に SCAN-XP はクラスタに属していないノードを並列に探索し，定義 2.4 に基づきハブか外れ値に分類する．

2.3 pSCAN [11]

pSCAN は Chang らにより提案された SCAN の逐次の高速化手法である．SCAN がグラフ中の全てのエッジに対して構造的類似度の計算を行っているのに対して，pSCAN は必要最小限のエッジに対して計算を行うことでクラスタリングを行う．pSCAN の各処理について概説する．pSCAN の処理の内，ハブ・外れ値検出処理は定義 2.4 と同様なため割愛する．

core クラスタ検出処理：最初に pSCAN は core の検出を *sd* (*similar degree*) と *ed* (*effective degree*) という 2 つの値を管理することで高速に行い，定義 2.5 で示される core のみのクラスタである core クラスタを検出する．

表 2: 3. 節で定義する記号の一覧

Symbol	Definition
P	分散並列処理を行う計算機の集合
p	自身の計算機番号
Gp	計算機 p が担当する G の部分グラフ
Vp	Gp 中のノードの集合
Ep	Gp 中のエッジの集合.
CCp	Gp 中の core のみで構成された core クラスタの集合
Cp	Gp から抽出されたクラスタの集合.
Hp	Gp からハブとして分類されたノードの集合.
Op	Gp から外れ値として分類されたノードの集合.
$CCp(v)$	core$v \in Vp$ と同じ core クラスタに属するノードの集合
$NCCp(v)$	core ではないノード $v \in Vp$ が属するクラスタの ID の集合
$S(r)$	計算機 $r \in P$ に隣接ノード情報を送るべきノード $v \in Vp$ の集合.

定義 2.5 (core クラスタ)　*core* $v \in V$ が与えられたとき，v の *core* クラスタを $CC(v)$ とすると，$CC(v) = \{u \in D(w) | u = core \wedge w \in CC(v)\}$．ここで $CC(v)$ の初期状態は $CC(v) = \{v\}$ である．

$sd(v)$ はノード v の持つ現時点で判明している構造的に類似な隣接ノードの数，$ed(v)$ は構造的に類似な隣接ノードと構造的類似度計算をしていないノードの合計数である．2 つの値を構造的類似度の結果ごとに更新し，$sd(v) \geqq \mu$ のとき v が core であり，$ed(v) < \mu$ のとき v は core ではないことは自明であるため，ノード v に対する構造的類似度計算を早期に終了できる．pSCAN は core を検出した後，隣接している core と構造的に類似であるならば同一の core クラスタとして検出する．

クラスタ検出処理：クラスタ検出処理では，全ての core と隣接している core ではないノードに対して，core と構造的に類似かどうか計算する．core と構造的に類似だった場合，その core と同じクラスタとして検出される．この処理により，定義 2.3 において示したクラスタを抽出することが可能である．

2.4 Intel Xeon Phi

Intel Xeon Phi [14] 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 つが提供されている．本講においては KNL を用いて実装を行う．

本稿において用いる KNL は現在最も新しい世代の Intel Xeon Phi 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．さらに，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において，3TFLOPS 以上の性能を発揮する．

3. 提案手法 DSCAN

複数の Intel Xeon Phi による高速化手法である DSCAN について概説する．Algorithm 3 に DSCAN のアルゴリズムを示す．また，本節で利用する記号の定義について表 2 に示す．

3.1 基本アイデア

本稿では複数の Intel Xeon Phi を用いて，SCAN を高速化する手法 DSCAN を提案する．提案手法 DSCAN では，(1) 単

Algorithm 3 DSCAN

Input: p,$Gp = \{Vp, Ep\}$, $\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
Output: C(Set of clusters), H(Set of hubs), and O(Set of outliers)
1: $\forall L(v,w) \in Lp = Unknown$;
2: $\forall sd(v) = 0$; $\forall ed(v) = |\Gamma(v)|$; $(v \in Vp)$
3:
4: run Algorithm 4; // Step 1:事前計算によるエッジの枝刈り（3.3 節）
5: run Algorithm 5; // Step 2:隣接ノード情報通信処理（3.4 節）
6:
7: // Step 3:分散並列 core 検出処理（3.5 節）
8: **for** each edge $(v,w) \in Ep$ **do in parallel**
9: run Algorithm 6;
10: **end for**
11: send $\forall sd(v)$ to other process; $(v \in Vp)$ //全ての計算機間で core の情報を共有
12: $//CCp(v)$:core v と同じ core クラスタに属するノードの集合
13:
14: // Step 4:分散並列 core クラスタ検出処理（3.6 節）
15: **for** each edge $(v,w) \in Ep$ **do in parallel**
16: run Algorithm 7;
17: **end for**
18: send CCp to processes 0 (master process);
19: **if** $p == 0$ **then**
20: merge $\forall CCp$ to CC;
21: send CC to other process; //全ての計算機で CC を共有
22: **end if**
23:
24: // Step 5:分散並列クラスタ検出処理（3.7 節）
25: $//NCC(v)$:core ではないノードが属するクラスタ ID の集合
26: **for** each node $v \in Vp$ $s.t.$ $sd(v) < \mu$ **do in parallel**
27: run Algorithm 8;
28: **end for**
29: send $\forall NCC(v)$ to other process; $(v \in Vp \wedge sd(v) < \mu)$
30:
31: //Step 6:分散並列ハブ・外れ値検出処理（3.8 節）
32: run Algorithm 9;
33: send Hp to process 0 (master process);
34:
35: C is created based on CC and NCC;$H = \Sigma_{r=0}^{P} Hr$;$O = \Sigma_{r=0}^{P} Or$

一の Intel Xeon Phi による SCAN の並列化手法 SCAN-XP に対して，逐次手法 pSCAN における構造的類似度の計算回数削減手法を適用することで更なる高速化を図るとともに (2) 分散並列化をする上で大きなボトルネックとなる隣接ノード情報の送受信をエッジの枝刈りの事前計算により削減することで複数の Intel Xeon Phi による分散並列化を実現する．

Algorithm 3 に DSCAN のアルゴリズムを示す．DSCAN は各計算機に処理対象のグラフ $G = \{V, E\}$ を分割した部分グラフ $Gp = \{Vp, Ep\}$ を与える．最初に DSCAN は Algorithm 3 の 4 行目にて (Step 1) 事前計算によるエッジ $(v, w) \in Ep$ の枝刈りを行い，構造的類似度計算の削減および通信する隣接ノード情報の削減を行う．その次に Algorithm 3 の 5 行目に示す (Step 2) 隣接ノード情報通信処理にて，各計算機間で通信をすることで構造的類似度計算に必要な隣接ノード情報を取得する．そして Algorithm 3 の 7 行目から 11 行目の (Step 3) 分散並列 core 検出処理にてノード $v \in Vp$ が core か否かを判定し，Algorithm 3 の 13 行目から 22 行目の (Step 4) 分散並列 core クラスタ検出処理にて core のみのクラスタを構築する．その後，Algorithm 3 の 24 行目から 29 行目の (Step 5) 分散並列クラスタ検出処理にて，core ではないノード $v \in Vp$ の所属するクラスタを検出する．最後に Algorithm 3 の 31 行目から 33 行目の (Step 6) 分散並列ハブ・外れ値検出処理にてどのクラスタにも属していないノード $v \in Vp$ をハブか外れ値に分類する．DSCAN はグラフを各計算機上に分割してロードし，各処理を分散並列化する．さらに，各計算機内では，単一の Intel Xeon Phi を用いた並列手法 SCAN-XP を用いたスレッド並列化及び

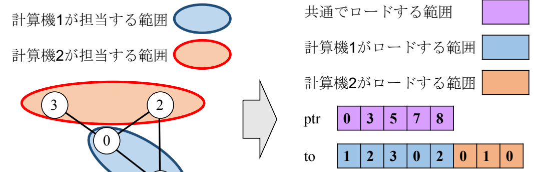

図 2: 計算機が 2 台の場合のグラフ分割の例

Algorithm 4 事前計算によるエッジの枝刈り

Input: p,$Gp = \{Vp, Ep\}$, $\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
1: **for** each edge $(v,w) \in Ep$ **do in parallel**
2: **if** $|\Gamma(v)| < \epsilon^2 \times |\Gamma(w)|$ or $|\Gamma(v)| < \epsilon^2 \times |\Gamma(w)|$ **then**
3: $L(v,w) = Dissimilar$;
4: **else if** $\epsilon \leqq \check{\sigma}(v,w)$ **then**
5: $L(v,w) = Similar$;
6: **end if**
7: **end for**

データ並列化を行う．加えて逐次手法 pSCAN による計算回数削減手法の活用により，処理の高速化を図る．

3.2 初期処理

3.2.1 グラフの分割

DSCAN は，図 2 内で示される ptr 配列と to 配列により，グラフを表現する．to 配列は各ノードの隣接ノードが連続的に並んでおり，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．このような ptr 配列と to 配列からなるグラフ表現形式を CRS (Compressed Row Strage) [15] と呼称する．

DSCAN はグラフ $G = \{V, E\}$ を計算機ごとにノード単位で分散して持つ．このとき，各計算機 $p \in P$ が分散して持つノードの集合を Vp とし，Vp 中のノードから接続するエッジの集合を Ep と定義する．そして，Vp と Ep により構成される G の部分グラフを Gp とし，$Gp = \{Vp, Ep\}$ と表現する．各計算機は部分グラフ Gp に対して，構造的類似度に基づくグラフクラスタリングを適用する

図 2 は 2 つの計算機上で DSCAN を実行した場合のグラフの分割と実際にロードするデータの例を示している．DSCAN は最初に ptr 配列全体を全ての計算機上にロードする．その後，各計算機には ptr 配列の先頭から順に処理範囲を割り当てる．処理範囲は，ptr 配列内に記録されている各ノードのエッジ数を参照し，計算機間でできるだけエッジ数が均等になるように決定する．図 2 では計算機 1 にはノード 0 とノード 1 を割り当て，計算機 2 にはノード 2 とノード 3 を割り当てる．その次に，各計算機は割り当てられたノードに対応する範囲の to 配列をロードする．

3.2.2 初期設定

初期値として DSCAN は各エッジ $(v, w) \in Ep$ に構造的に類似かどうか未分類であることを表すラベル $Unknown$ を与える．また，DSCAN は pSCAN と同様に sd(similar-degree) 及び ed(effective-degree) による core ノードの早期発見手法を採用している．初期値としてノード $v \in Vp$ の $sd(v)$ に対して 0 を $ed(v)$ に対して $|\Gamma(v)|$ を与える．

3.3 事前計算によるエッジの枝刈り

事前計算によるエッジの枝刈りでは，構造的類似度の上限値，

Algorithm 5 隣接ノード情報通信処理

Input: $p,P,Gp=\{Vp,Ep\},\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
1: **for** each $r \in P \setminus \{p\}$ **do**
2: $\quad S(r)=\emptyset$
3: **end for**
4: **for** each $Unknown$ edge $(v,w) \in Ep$ **do in parallel**
5: $\quad$ **if** $(v,w) \in Ep \wedge v \in Vp \wedge w \in Vr (r \in P \setminus \{p\})$ **then**
6: $\quad\quad S(r)=S(r) \cup \{v\}$
7: $\quad$ **end if**
8: **end for**
9: create data structure based on $S(r)$ in Figure 3;
10: send data structure to process $r \in P \setminus \{p\}$;

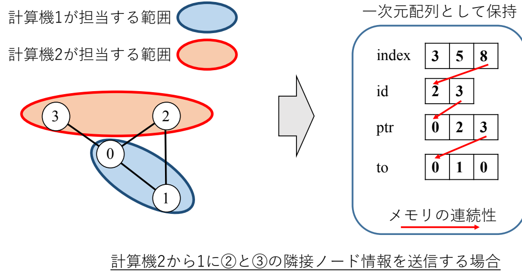

図 3: 隣接ノード情報を送信するためのデータ構造の例

下限値を計算することでエッジの枝刈りを行う．枝刈りされたエッジは，隣接ノード情報を用いて構造的類似度を計算する必要がなく，計算機間で送受信される隣接ノード情報は減少する．結果として，分散並列化においてボトルネックとなる通信コストを削減できる．アルゴリズムを Algorithm4 により示す．

構造的類似度の上限値は Chang らにより提案された Pruning rule [11] を用いて計算する．Pruning rule を定義 3.1 に示す．Pruning rule を満たすエッジは構造的に類似でないので枝刈りし，*Dissimilar* をラベル付けする．

定義 3.1 (Pruning rule) $v,w \in V$ が与えられたとき，$\Gamma(v) < \epsilon^2 \times \Gamma(w)$ もしくは $\Gamma(w) < \epsilon^2 \times \Gamma(v)$ を満たすならば，$\sigma(v,w) < \epsilon$ である．

また，定義 2.1 の構造的隣接ノード集合は自身と隣接ノードの和集合であり，定義 2.2 の式における $|\Gamma(v) \cap \Gamma(w)|$ は最低でも 2 以上であることは明白である．したがって，構造的類似度の下限値を定義 3.2 に示す．下限値が ϵ 以上であるエッジは構造的に類似であるため枝刈りし，*Similar* をラベル付けする．

定義 3.2 (構造的類似度の下限値) $v,w \in V$ が与えられたとき，$\sigma(v,w)$ の下限値 $\check{\sigma}(v,w)=2/|\Gamma(v)||\Gamma(w)|$．

3.4 隣接ノード情報通信処理

隣接ノード情報通信処理は，グラフを分割して持つ各計算機間で構造的類似度計算を行うために各計算機が必要な隣接ノード情報を共有する処理である．各計算機は自身に分配されたノード $v \in Vp$ とその隣接ノードの情報を持つ．しかし，ノード v に接続するエッジは他の計算機 $r \in P$ のノード $w \in Vr$ $(r \neq p)$ と接続されている可能性がある．このように計算機間をまたいでノードを接続するエッジ (v,w) 間の構造的類似度を計算するためには計算機 r からノード w の隣接ノード情報を通信して受け取る必要がある．そこで DSCAN の隣接ノード情報通信処理のアルゴリズムを Algorithm 5 に示す．

Algorithm 6 分散並列 core 検出処理

Input: $v,w \in Gp, sd(v), ed(v)$, $\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
1: **if** $sd(v) < \mu$ and $ed(v) \geqq \mu$ **then**
2: $\quad$ **if** $L(v,w) == Unknown$ **then**
3: $\quad\quad$ run Algorithm 2
4: $\quad\quad$ **if** $\sigma(v,w) \geqq \epsilon$ **then**
5: $\quad\quad\quad L(v,w)=Similar;$
6: $\quad\quad$ **else**
7: $\quad\quad\quad L(v,w)=Dissimilar;$
8: $\quad\quad$ **end if**
9: $\quad$ **end if**
10: $\quad$ **if** $L(v,w) == Similar$ **then**
11: $\quad\quad sd(v)=sd(v)+1$ by using atomic instraction;
12: $\quad$ **else if** $L(v,w) == Dissimilar$ **then**
13: $\quad\quad ed(v)=ed(v)-1$ by using atomic instraction;
14: $\quad$ **end if**
15: **end if**

まず，各計算機は自身と他の計算機との間にまたがって貼られているエッジを検出する．そのエッジが 3.3 節の処理により，枝刈りされていない場合は，構造的類似度計算を行うために，他の計算機に対して隣接ノードの情報を送信する．DSCAN では，頻繁な通信による通信オーバーヘッドを軽減するために送る必要のあるノードの情報を最初に全て検出する．その後，図 3 に示されるデータ構造を構築することで 1 度の通信で全ての隣接ノード情報を目的の計算機に送る．

図 3 で示されるデータ構造は，index，id，ptr，to の 4 つの配列からなるデータ構造である．先に述べた CRS 形式と同様に，to 配列には送信するべきノードの隣接ノード情報が連続的に並んでおり，ptr 配列は to 配列のどの範囲にどのノードの隣接ノードが格納されているのかを示している．しかし，送信する必要があるノードの情報のみを格納するため，配列の要素番号とノード番号は必ずしも一致しない．そこで，id 配列には ptr 配列の要素がどのノードの情報を示しているのか，そのノード番号を格納する．さらに，本稿では通信回数の削減のために id，ptr，to 配列を 1 つの 1 次元配列にまとめることで 1 回の通信で情報の送信を完了する．このときに，送信する配列の先頭には新たに index 配列を持つ．index 配列は id，ptr，to の各配列が送信した 1 次元配列のどの範囲に格納されているかを示す配列である．

3.5 分散並列 core 検出処理

分散並列 core 検出処理では，各計算機 p は，自身の持つ全てのノード $v \in Vp$ を core かどうか判定する．そのアルゴリズムを Algorithm 3 の 10–14 行目と Algorithm 6 に示す．

提案手法では，core かどうか判明していないノード v $(sd(v) < \mu \wedge ed(v) \geq \mu)$ に接続されているエッジ $(v,w) \in Ep$ を選択する．そして，エッジ (v,w) が 3.3 節の処理により枝刈りされていない場合，Algorithm 2 により，構造的類似度を計算する．計算の結果，エッジ (v,w) が構造的に類似である場合は *Similar* がラベル付され、構造的に類似でない場合は *Dissimilar* がラベル付けされる．

そして，最後に構造的類似度計算を行ったかどうかに関わらず，エッジ (v,w) のラベルが *Similar* なら $sd(v)$ が更新され，*Dissimilar* なら $ed(v)$ が更新される．この $sd(v)$ 及び $ed(v)$ の更新はスレッド間の衝突を防ぐために atomic 命令により排他制御される．$sd(v) \geq \epsilon$ の場合，ノード v は core である．一方

Algorithm 7 分散並列 core クラスタ検出処理

```
Input: v, w ∈ Gp, sd(v), sd(w), CCp(v), CCp(w) ε ∈ ℝ and μ ∈ ℕ
 1: if sd(v) ≧ μ ∧ sd(w) ≧ μ ∧ find(v) ≠ find(v) then
 2:    if L(v, w) == Unknown then
 3:       run Algorithm 2
 4:       if σ(v, w) ≧ ε then
 5:          L(v, w) = Similar;
 6:       else
 7:          L(v, w) = Dissimilar;
 8:       end if
 9:    end if
10:    if L(v, w) == Similar then
11:       get C(v) ∪ C(w) by using union(v, w) with CAS instruction;
12:    end if
13: end if
```

で，$ed(v) < \mu$ の場合，ノード v は core ではない．sd 及び ed により，ノードの全てのエッジを構造的類似度を計算すること無く，core であるかどうか判定できる．

全ての計算機における分散並列 core 検出処理が終了した後，core の情報は全ての計算機間で共有される．

3.6 分散並列 core クラスタ検出処理

分散並列 core クラスタ検出処理では，前節の処理により，検出された core $v \in Vp$ から core のみで構成される core クラスタ CCp を構築する．検出される CCp は Gp 上で構築される core クラスタの集合であり，G 上の core クラスタの集合 CC の部分集合である．分散並列 core クラスタ検出処理のアルゴリズムを Algorithm 7 に示す．

各計算機 p は，core $v \in Vp$ とエッジ $(v, w) \in Ep$ で繋がった core $w \in V$ が，構造的に類似ならば同一の core クラスタとして検出する．core クラスタの検出は Union-Find 木を用いて行う．Union-Find 木は，v の属する素集合の特定を $find(v)$ 命令，v と w の属する素集合同士の結合を $union(v, w)$ 命令を用いることで高速に行うデータ構造である．初期状態の Union-Find 木では各 core は自分自身へのポインタを持つが，2 つの core が同一の core クラスタに統合された場合，一方のポインタをもう一方を指すように書き換え，木を形成する．このポインタの書き換えは CAS 命令を使用して行い，スレッド間の整合性を保つ．

各計算機上に分割された部分グラフ Gp 上での core クラスタ CCp が検出した後，全ての CCp を 1 つの計算機に集約される．集約された計算機上で，CCp はマージし，グラフ G 上の core クラスタ CC を形成する．その後，通信により CC の情報を全ての計算機間で共有する．

3.7 分散並列クラスタ検出処理

分散並列クラスタ並列処理では，core ではないノードがどの core クラスタに属するかを判定することで，定義 2.3 において示したクラスタを構築する．分散並列クラスタ検出処理のアルゴリズムを Algorithm 8 に示す．

各計算機は，core ではないノード $v \in Vp$ を並列に探索していき，隣接している core w と構造的に類似である場合は，core クラスタの ID を $find(w)$ により取得し，ノード v が所属しているクラスタを示すラベルの集合 $NCC(v)$ に追加する．全ての core ではないノードを探索した後，全ての $NCC(v)$ は通信により，全ての計算機間で共有される．

Algorithm 8 分散並列クラスタ検出処理

```
Input: v ∈ Vp, ε ∈ ℝ and μ ∈ ℕ
 1: for each node w s.t. (v, w) ∈ Ep ∧ sd(w) ≧ μ ∧ find(w) ∉ NCC(v) do
 2:    if L(v, w) == Unknown then
 3:       run Algorithm 2
 4:       if σ(v, w) ≧ ε then
 5:          L(v, w) = Similar;
 6:       else
 7:          L(v, w) = Dissimilar;
 8:       end if
 9:    end if
10:    if L(v, w) == Similar then
11:       NCC(v) = NCC(v) ∪ {find(v)}
12:    end if
13: end for
```

Algorithm 9 分散並列ハブ・外れ値検出処理

```
Input: v, w ∈ Gp, sd(v), sd(w), Cp(v), Cp(w) ε ∈ ℝ and μ ∈ ℕ
 1: for each node v ∈ Vp ∧ v ∉ C do in parallel
 2:    if ∃u, w ∈ Γ(v) s.t. ∃C(u) ≠ ∃C(w) then
 3:       label node v as hub, and H = H ∪ {v};
 4:    else
 5:       label node v as outlier, and O = O ∪ {v};
 6:    end if
 7: end for
```

3.8 分散並列ハブ・外れ値検出処理

各計算機は，どのクラスタにも属していないノードに対して，2 つ以上のクラスタに隣接するならばハブ，そうでない場合は外れ値に分類する．この処理によって各計算機 p 上でのハブの集合 Hp 及び外れ値の集合 Op が取得される．Hp 及び Op は検出した後，1 つの計算機に集約する．アルゴリズムを Algoreithm 9 に示す．

3.9 結果の出力

最後に，各計算機上で得られた結果を 1 つに集約することで，グラフ G 中のクラスタ集合 C 及びハブ集合 H 及び外れ値集合 O を求める．C は，core ノードの所属クラスタは CC，core ではないノードの所属クラスタは NCC をそれぞれ参照することで求める．H および O は各計算機 $r \in P$ における Hr，Or の和集合として求めることが出来る．

4. 評価実験

本節では，DSCAN の性能を示すために，従来手法 SCAN-XP との比較実験を行う．実験は東京大学及び筑波大学所有のスーパーコンピュータシステムである Oakforest-PACS を用いる．Oakforest-PACS は第二世代の Intel Xeon Phi である Knights Landing のクラスタである．Oakforest-PACS は総数 8208 の計算ノードで構成されているが，本実験ではそのうち 32 計算ノードのみ使用した．表 3 に 1 計算ノードあたりの構成を示す．また，本実験で使用した実データセットの詳細を表 4 に示す．

表 3: 実験環境

CPU	Intel Xeon Phi 7250
# of Processor	1
Clock rate	3.5 GHz
# of Core/# of Thread	68/272
SIMD	AVX-512
Main memory	16GB (MCDRAM) + 96(DDR4)
OS	Cent OS 7
Interconnect	Intel Omni-Path ネットワーク (100Gbps)

また，本実験では SCAN におけるユーザ定義の閾値は $\epsilon = 0.4$，$\mu = 2$ とした．特に言及がない限り，従来手法及び提案手法は

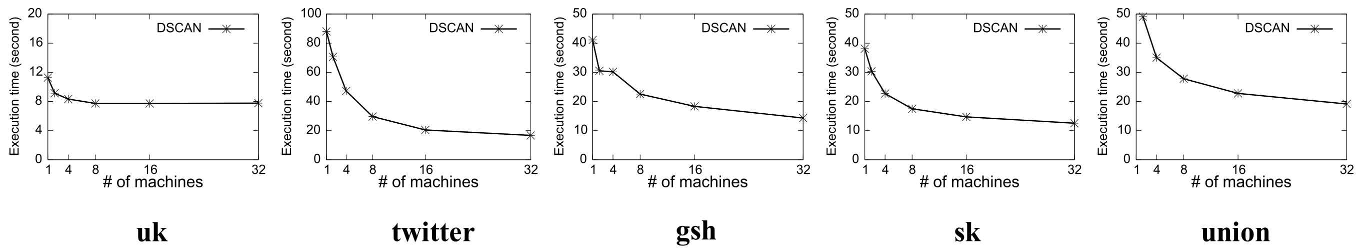

図 4: スケーラビリティ

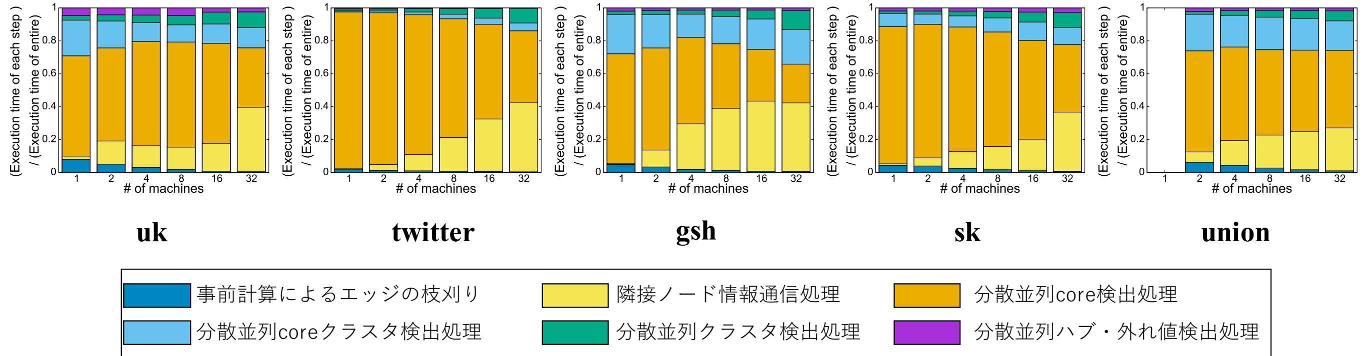

図 5: 全体の実行時間に対して各処理が占める割合

表 4: データセットの詳細

Dataset	Nodes	Edges	Data source
uk	39,454,463	783,027,125	uk-2005 [16]
twitter	41,652,230	1,202,513,046	twitter-2010 [16]
gsh	68,660,142	1,502,666,069	gsh-2015-host [16]
sk	50,636,059	1,810,063,330	sk-2005 [16]
union	131,572,430	4,663,392,591	uk-union-2006-06-2007-05 [16]

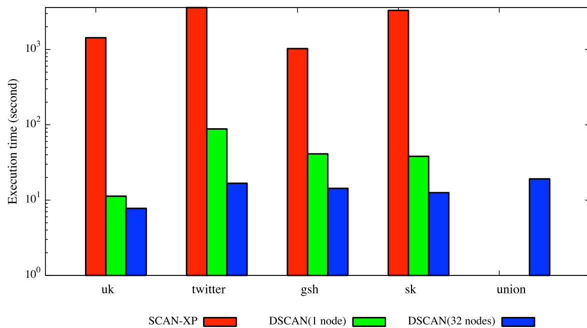

図 6: 実行時間

常に 1 つの Intel Xeon Phi 上では OpenMP を用いて 272 スレッド並列され、更に AVX-512 を用いてデータ並列化する．

4.1 実行時間の比較

図 6 に従来手法 SCAN-XP と 1 つの計算ノードで実行した DSCAN（以下 DSCAN(1)），そして一番良い性能を示した 32 個の計算ノードで実行した DSCAN（以下 DSCAN (32)）の実行時間の比較を示す．実験に用いたデータセットの内，union は非常に大きなデータセットであるため単一の計算ノード上ではメモリ不足で実行が不可能であった．したがって，union に関しては DSCAN (32) の実行時間のみを示す．

図より，全てのデータセットに対して，SCAN-XP に比べ DSCAN が大幅に高速であることを確認した．SCAN-XP に対して pSCAN の計算回数削減手法を取り入れた DSCAN は分散並列を行っていない DSCAN (1) においても 20 倍以上高速である．更に，複数の Intel Xeon Phi を用いて分散並列化を行うことにより，32 ノードでの並列実行時に DSCAN は SCAN-XP より 50 倍以上高速である．

また，約 46 億のエッジを持つグラフ union は，単一の計算ノードで処理を行うことが出来ないほど巨大なグラフである．しかし，DSCAN (32) は分散並列化を行うことにより，約 19 秒でクラスタリングを終えることが出来る．

4.2 DSCAN のスケーラビリティ

図 4 に DSCAN の計算ノード数に伴う性能のスケーラビリティを示す．図より，uk を覗いた全てのデータセットで計算ノード数の増加に伴い，実行時間が減少することを確認した．特にデータセット twitter では，ノード数とともに性能は良好に向上し，最終的に 1 ノードでの実行と比較して 32 ノードで 5 倍以上高速であることが示された．

一方で uk では分散並列化による高速化率は非常に小さい．これは，データセットの中で uk は一番小さな規模のグラフであるため，1 計算ノード上での pSCAN による枝刈り手法に加えて SCAN-XP に基づいたスレッド並列化とデータ並列化により十分に高速化が行われたためだと考えられる．

4.3 DSCAN におけるボトルネック

4.2 節により，DSCAN は計算ノード数に伴い性能が向上することを確認した．しかしながら，図 4 から確認できるように，いずれのデータセットにおいても計算ノード数が 16 を超えたあたりから，高速化率が低くなってしまう．

この原因を示すために，図 5 に各処理の実行時間が全体の実行時間のうちに占める割合を示す．図から，計算ノードの増加ともに隣接ノード情報通信処理が占める割合は増大することを確認した．このことから，DSCAN は先に述べた枝刈り手法を用いて隣接ノード情報通信処理を削減するが隣接ノード情報通信処理は未だ大きなボトルネックであることが分かる．したがって，隣接ノード情報通信処理の最適化は依然として大きな課題である．

5. 関連研究

構造的類似度に基づくグラフクラスタリング手法は [10,11,17–21]，グラフからクラスタ，ハブ，外れ値を検出する唯一の手法であり，様々な拡張手法が考案されている．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．

Xu らにより提案された SCAN [8] 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 は多次元ベクトルに対する密度ベースのクラスタリング手法として有名な DBSCAN [9] のグラフデータに対する拡張である．Xu らは SCAN が Modularity に基づく手法と比較してより高い精度でクラスタを検出することを示している．しかし，1. 節で述べたように SCAN は構造的類似度の計算量が原因となり，大規模グラフに対して膨大な計算時間を必要とする．

SCAN を高速化するため，幾つかの手法 [10–12,22] 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++ [10]，pSCAN [11]，anySCAN [12] は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．これら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 [21] は MapReduce を利用し，SCAN を分散並列化した手法である．PSCAN は，最初に map，reduce 処理により core を検出する．その後，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近似解となる．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．したがって，SCAN++や pSCAN より多くの計算時間を必要とする．

6. まとめと今後の課題

本稿では，複数の Intel Xeon Phi を用いた分散並列化による高速化手法 DSCAN を提案した．我々の行った評価実験により，DSCAN は SCAN-XP よりも大幅に高速であり，46 億以上のエッジを持つグラフ union をわずか 19 秒で処理することを確認した．今後の課題としては，依然として大きなボトルネックである隣接ノード情報通信コストの最適化や 100 億エッジ規模の非常に巨大なグラフへの適用について検討する．

謝　　辞

本研究は，JSPS 科研費 JP16H06650，JST ACT-I 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の助成を受けたものである．

文　　献

[1] 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.

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

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Son T Mai, Martin Storgaard Dieu, Ira Assent, Jon Jacobsen, Jesper Kristensen, and Mathias Birk. Scalable and interactive graph clustering algorithm on multicore cpus. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 349–360. IEEE, 2017.

[13] Tomokatsu Takahashi, Hiroaki Shiokawa, and Hiroyuki Kitagawa. Scan-xp: Parallel structural graph clustering algorithm on intel xeon phi coprocessors. In *Proceedings of the 2nd International Workshop on Network Data Analytics*, page 6. ACM, 2017.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.