SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors*

Tomokatsu Takahashi
Gradiate School of Systems and Information Engineering
University of Tsukuba
shihakata@kde.cs.tsukuba.ac.jp

Hiroaki Shiokawa
Center for Computational Sciences
University of Tsukuba
shiokawa@cs.tsukuba.ac.jp

Hiroyuki Kitagawa
Center for Computational Sciences
University of Tsukuba
kitagawa@cs.tsukuba.ac.jp

ABSTRACT

The structural graph clustering method *SCAN*, proposed by Xu *et al.*, is successfully used in many applications because it not only detects densely connected nodes as clusters but also extracts sparsely connected nodes as hubs or outliers. However, it is difficult to applying SCAN to large-scale graphs since SCAN needs to evaluate the density for all adjacent nodes included in the given graphs. In this paper, so as to address the above problem, we present a novel algorithm *SCAN-XP* that performs over Intel Xeon Phi. We designed SCAN-XP in order to make best use of the hardware potential of Intel Xeon Phi by employing the following approaches: First, SCAN-XP avoids the bottlenecks that arise from parallel graph computations by providing good load balances among cores on the Intel Xeon Phi. Second, SCAN-XP effectively exploits 512 bit SIMD instructions implemented in the Intel Xeon Phi to speed up the density evaluations. As a result, SCAN-XP detects clusters, hubs, and outliers from large-scale graphs with much shorter computation time than SCAN. Specifically, SCAN-XP runs approximately 100 times faster than SCAN; for the graphs with 100 million edges, SCAN-XP is able to perform in a few seconds. In this paper, extensive evaluations on real-world graphs demonstrate the performance superiority of SCAN-XP over existing approaches.

CCS CONCEPTS

• **Information systems → Clustering**; • **Theory of computation → Graph algorithms analysis**; *Massively parallel algorithms*;

KEYWORDS

Graph, Clustering, Intel Xeon Phi Coprocessors

ACM Reference format:
Tomokatsu Takahashi, Hiroaki Shiokawa, and Hiroyuki Kitagawa. 2017. SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors. In *Proceedings of NDA'17, Chicago, IL, USA, May 19, 2017,* 7 pages.
DOI: http://dx.doi.org/10.1145/3068943.3068949

*Tomokatsu Takahashi and Hiroaki Shiokawa are contributed equally to this work.

NDA'17, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3068943.3068949

1 INTRODUCTION

Graph cluster analysis is one of the most fundamental techniques in various research areas such as data mining [1, 8] and social science [13]. A cluster can be regarded as a group of nodes that are densely connected within a group and sparsely connected to those of other groups. Besides extracting clusters, finding special role nodes, *hubs* and *outliers*, is also a worthwhile task for understanding the structures of large-scale graphs [11]. Hubs are generally thought of as bridging different clusters. In contrast, outliers are treated as noise. The hubs and outliers provide useful insights in mining graphs. For instance, hubs in web graphs act like authoritative web pages that link similar topics [12]. However, most traditional algorithms such as modularity-based methods [16, 17] only study the problem of cluster detection and so ignore hubs and outliers.

One of the most successful graph clustering method is *structural clustering algorithm (SCAN)* [21]. As well as density-based clustering, the main concept of SCAN is that densely connected nodes should be in the same cluster. However, unlike the traditional algorithms, SCAN successfully finds not only clusters but also hubs and outliers. Although SCAN is effective in finding highly accurate results, SCAN has a serious weakness; it requires high computational costs for large-scale graphs. This is because SCAN entails exhaustive density evaluations for all adjacent node pairs included in the large-scale graphs. This procedure involves the computational cost $O(m) = O(n^2)$ in the worst case where m and n represents number of edges and nodes, respectively. Furthermore, in order to evaluate the density, SCAN employs a criteria, called *structural similarity*, that is based on a set intersection between two node sets. Thus, as shown in the literature [18] and [4], SCAN requires $O(\frac{m^2}{n})$ and $O(m^{1.5})$ in average and worst case, respectively.

To address the performance limitation of SCAN, many efforts have been made to improve the clustering speed in the recent few years. *SCAN++* [18] and *pSCAN* [4] are the most representative methods for speeding up SCAN. These were recently proposed by Shiokawa *et al.* and Chang *et al.*, respectively. SCAN++ is designed to handle the property of real-world graphs; a node and its two-hop-away nodes tend to have lots of common neighbor nodes since real-world graphs have high clustering coefficients [18]. Based on the above property, SCAN++ effectively reduces the number of structural similarity computations. As well as SCAN++, pSCAN employs a new paradigm for structural graph clustering based on the three observations in real-world graphs [4]. By following the observations, pSCAN employs several nodes/edges pruning techniques and their optimizations for reducing the number of structural similarity computations. Although these algorithms surely succeeded

Table 1: Definition of main symbols

Symbol	Definition
G	Given graph.
V	Set of nodes in G.
E	Set of edges in G.
C	Set of clusters extracted from G.
H	Set of nodes classified as hubs from G.
O	Set of nodes classified as outliers from G.
$\Gamma(v)$	Set of nodes in the structural neighborhood of node v
$N_\epsilon(v)$	Set of nodes in the ϵ-neighborhood of node v
$D(v)$	Set of nodes in directly structure neighborhood of node v
$C(v)$	Set of nodes that are belong to the same cluster as node v
n	Number of nodes in G. ($n = \|V\|$)
m	Number of edges in G. ($m = \|E\|$)
ϵ	Threshold of the structural similarity, $0 \leq \epsilon \leq 1$.
μ	Minimum number of nodes in a cluster.
α, β	Block size parameters used in SCAN-XP.
$\sigma(u, v)$	Structural similarity between node u and v

in reducing the time complexity of SCAN for the real-world graphs, the computation time for large-scale graphs (*i.e.* graphs with more than 100 million edges) is still large. Thus, it is a challenging task to improving the computational efficiency for the structural graph clustering. Especially, most of existing approaches perform as a single-threaded algorithms; they do not fully exploit parallel computation architectures but this is time-consuming.

1.1 Contributions

Motivated by the above facts, we present a novel algorithm named *SCAN-XP*; SCAN-XP is a scalable and massively-parallel method that exploits a modern many-core processor *Xeon Phi*. Our goal is to efficiently detect clusters, hubs, and outliers from significantly large real-world graphs. To do so, SCAN-XP employs the following techniques in order to make best use of many-core processor on Xeon Phi: (1) We designed SCAN-XP to avoid *poor parallelism* and *unbalanced load distribution* coming from the irregular structure of real graphs, *i.e.* the skewed degree distribution (such as power-law), and (2) SCAN-XP effectively exploits 512bit SIMD instructions implemented in Xeon Phi to speed up each structural similarity computation. Instead of the original algorithm SCAN, our proposal SCAN-XP can find clusters in an efficient and a scalable manner for real-world large-scale graphs.

Our contributions of this paper are summarized as follows:

(1) **Efficient:** We developed significantly fast algorithm SCAN-XP for structural graph clustering. SCAN-XP is able to compute graphs with more than 850 million edges within 36 seconds. (Section 4.1)
(2) **Scalable:** As increasing the number of available processors, SCAN-XP effectively increases its runtime performances. (Section 4.2)
(3) **Exact:** Basically, SCAN-XP produces exactly same clustering results as SCAN since SCAN-XP is able to detect all of densely-connected node groups. (Section 3.4)

To the best of our knowledge, SCAN-XP is the first solution exploiting Xeon Phi for SCAN. We experiments confirmed that SCAN-XP computes results approximately 100 times faster than the original algorithm SCAN. Even though SCAN is effective in enhancing application quality, they have been difficult to apply large-scale graphs due to its performance limitations. However, by providing our massively parallel algorithm on Xeon Phi, SCAN-XP will help to enhance the effectiveness of a wider range of applications.

2 PRELIMINARY

In this section, we formally define the notations and introduce the backgrounds of this paper. Let $G = \{V, E\}$ be an unweighted and undirected graph, where V and E are a set of nodes and edges, respectively. We assume graphs are undirected and unweighted only to simplify the representations. Other types of graphs such as directed and weighted, can be handled with only slight modifications. Table 1 lists the main symbols and their definitions. In Section 2.1, we first review the baseline algorithm SCAN. After that, in Section 2.2, we introduce the Intel Xeon Phi coprocessor that is exploited by our proposal SCAN-XP.

2.1 Baseline algorithm: SCAN

SCAN [21], proposed by Xu *et al.*, is one of the most popular graph clustering method; it successfully detects not only clusters C but also hubs H and outliers O unlike traditional algorithms. SCAN extracts clusters as sets of nodes that have dense internal connections; it identifies the other non-clustered nodes, *i.e.* nodes that are not belong to any clusters, as hubs or outliers. Thus, prior to identifying hubs and outliers, it finds all clusters in a given graph.

In order to detect clusters, SCAN first finds a special node, called *core*. Core is a node that has a lot of neighbor nodes with highly dense connections; the core is regarded as the seed of a cluster. SCAN uses the structural neighborhood [21] to evaluate density. The structural neighborhood of a node is a node set composed of the node itself and all its adjacent nodes.

Definition 2.1 (Structural neighborhood)**.** *The definition of structural neighborhood of node v, denoted by $\Gamma(v)$, is given by $\Gamma(v) = \{u \in V | (v, u) \in E\} \cup \{v\}$.*

The density of adjacent nodes is computed by the common nodes in the structural neighborhoods. SCAN measures the number of common nodes in two structural neighborhoods normalized by the geometric mean of their structural neighborhood sizes. This measurement is called structural similarity and is defined as follows:

Definition 2.2 (Structural similarity)**.** *The structural similarity between node v and w, denoted by $\sigma(v, w)$, is defined as $\sigma(v, w) = |\Gamma(v) \cap \Gamma(w)| / \sqrt{|\Gamma(v)||\Gamma(w)|}$.*

The structural similarity is a score varying from 0 to 1 that indicates the scale of matching degree of structural neighborhoods. When adjacent nodes share many members of their structural neighborhoods, their structural similarity becomes high.

From Definition 2.2, SCAN detects the core by evaluating structural similarities for all neighborhoods. In order to specify core metrics, SCAN requires two user-specified parameters. First is the minimum score of the structural similarity to neighbor nodes, denoted by ϵ. Second is the minimum number of neighborhoods, denoted by μ, all of whose structural similarities exceed ϵ. SCAN regards a node as core when it has at least μ neighbors with structural similarity greater than ϵ:

Definition 2.3 (Core)**.** *Node u is core iff $|N_\epsilon(v)| \geq \mu$, where N_ϵ, called ϵ-neighborhood, is $N_\epsilon(v) = \{w \in \Gamma(v) | \sigma(v, w) \geq \epsilon\}$.*

Once SCAN finds core, it expands a cluster from the core. Specifically, nodes included in ϵ-neighborhood of the core are assigned to

Algorithm 1 Baseline method: SCAN [21]

Input: $G = \{V, E\}$, $\epsilon \in \mathbb{R}$, and $\mu \in \mathbb{N}$
Output: C, H, and O
1: $\forall v \in V$ are labeled as *unclassified*;
2:
3: // Step 1: Core detection based on Definition 2.3
4: **for** each edge$(v, w) \in E$ **do**
5: compute $\sigma(v, w)$ by Definition 2.2;
6: **end for**
7:
8: // Step 2: Cluster construction based on Definition 2.4
9: **for** each *unclassified* node $v \in V$ **do**
10: **if** $N_\epsilon(v) \geq \mu$ **then**
11: generate new clusterID, and add v into $C(v)$;
12: all $w \in N_\epsilon(v)$ into queue Q;
13: **while** $Q \neq \emptyset$ **do**
14: $u = Q$.pop;
15: **for** each $w \in D(u)$ **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** $\exists u, w \in \Gamma(v)$ *s.t.* $C(u) \neq C(w)$ **then**
33: label node v as hub, and $H = H \cup \{v\}$;
34: **else**
35: label node v as outlier, and $O = O \cup \{v\}$;
36: **end if**
37: **end for**

the same cluster as the core. The ϵ-neighborhood nodes of core, say node u, are called *directly structure neighborhood nodes*, denoted by $D(u)$. When node u is core and $D(u) \neq \emptyset$, SCAN assigned all nodes in $D(u)$ to the same cluster as node u. SCAN recursively expands the cluster by checking whether each node, which is included in the cluster, satisfies core condition defined by Definition 2.3 or not. Formally, the cluster that has node u is defined as follows:

Definition 2.4 (Cluster). *The cluster by node u, denoted by $C(u)$, is defined as $C(u) = \{w \in D(v) | v \in C(u)\}$, where $C(u)$ is initially set to $C(u) = \{u\}$.*

After termination of cluster expansion, SCAN randomly picks a new node from the nodes that have yet to be checked. SCAN continues this procedure until there are no undiscovered cores.

Finally, SCAN classify nodes, which are not belong to any clusters, into hubs or outliers.

Definition 2.5 (Hubs and Outliers). *Suppose node u does not belong to any cluster. $u \in H$ iff node v and w exist in Gamma(u) such that $C(v) \neq C(w)$. Otherwise $u \in O$.*

Algorithm 1 overviews pseudo code of SCAN. In the initial state, all nodes are labeled as *unclassified* (line 1). As shown in Algorithm 1, the subsequent procedures are composed of three steps.

2.2 Intel Xeon Phi

Intel Xeon Phi [10] (Xeon Phi for short) is a series of x86-compatible multicore processors for massively parallel computations targeted at high performance computing. Xeon Phi comprises of over 50 physical cores, and we can easily run parallel programs on Xeon Phi by using OpenMP or MPI. Recently, two versions are available for Xeon Phi: *Xeon Phi Knights Corner (KNC)* and *Knights Landing (KNL)*. KNL is the latest generation product of Xeon Phi. KNL is able to perform as a host CPU by using up to 72 physical cores; each core shows 1.3-1.5 GHz clock frequency with AVX-512 SIMD instruction set. Furthermore, KNL supports 16 GB on-chip MCDRAM and up to 384 GB DDR4 RAM. As a result, KNL is capable more than 6 TFLOPS of double precision floating point instructions.

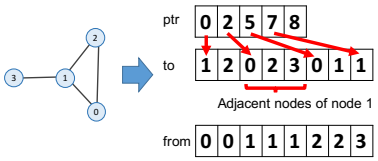

Figure 1: Example of graph data representation

As we described above, Xeon Phi is expected to show good performances for large-scale data processing, however it is not trivial task to design efficient algorithms on Xeon Phi. This is because each physical core on Xeon Phi has relatively lower clock frequency than Xeon CPUs (see Table 2 for details). Hence, in order to improve performances compared with single threaded algorithms, we have to fully exploit exhaustive resources on Xeon Phi, *i.e.* physical cores and 512 bit SIMD instructions. Furthermore, since the latencies for accessing main memory are significantly large, it is important to construct efficient programs and data structures so as to increase cache hit ratio and explicit prefetch instructions.

3 PROPOSED METHOD: SCAN-XP

Our goal is to find exactly same clustering results as SCAN from large-scale graphs in the scalable manner. In this section, we present details of our proposal, SCAN-XP, that effectively exploits Xeon Phi coprocessors for the structural graph clustering. We first overview SCAN-XP and then give a full description of algorithm.

3.1 Overview of SCAN-XP

As we described in Section 1, we need to reduce computation time of SCAN. To do so, SCAN-XP employs massively parallel computation approach for structural graph clustering by exploiting Intel Xeon Phi coprocessor shown in Section 2.2. In Algorithm 1, we reviewed that SCAN comprises three steps: (1) core detection step, (2) cluster construction step, and (3) hubs/outliers detection step. For each steps, SCAN-XP, thus, provides parallel computation algorithms that show good performance on Xeon Phi.

Specifically, as we described in Section 2.2, it is not trivial task for Xeon Phi to make good use of its hardware potential. In addition, real-world graphs generally lead *unbalanced load distribution* and *poor parallelization* to parallel computations since the graphs contain irregular and highly skewed structures (*e.g.* power-law degree distribution). Thus, in SCAN-XP, we design each step to balance loads among physical cores and to fully exploits 512 bit SIMD instructions implemented in each physical core.

3.2 Parallel core detection step

In this section, we describe our approach using Xeon Phi to reduce the computation time of core detection step. Since SCAN

needs to compute structural similarities for all edges in a graph prior to detecting cores, this step consumes the most part of the computational costs incurred by SCAN. Thus, we introduce two type of parallelization into the structural similarity computations: *thread-based parallelization* and *SIMD-based parallelization*.

Thread-based parallelization: First, we parallelize the structural similarity computations by using threads so as to made good used of many integrated physical cores on Xeon Phi. From step 1 in Algorithm 1, it is definitely reasonable to assign a thread into a edge that involves a structural similarity computation. This is because the structural similarity computation $\sigma(v, w)$ is independent among edges in E. If we assign a thread into a node, it unfortunately incurs unbalanced load distributions among threads since real-world graphs may have the power-law degree distribution.

To do so, we introduce an efficient graph data representation by extending the well-known graph data representation, called *CRS* (Compressed Row Storage) [6]. CRS is one of the cache and space friendly graph data representations that are generally used in the recent graph mining studies. As shown in Figure 1, CRS is composed of two arrays, `to` array and `ptr` array. `to` array stores neighbor nodes of each node in a graph, and `ptr` array saves pointers that indicate where neighbor nodes of a node are stored in `to` array. In order to obtain edges from CRS, we first access `ptr` array and then get edges from `to` array. However, this procedure incurs the load skewness to threads since we need to assign a node into a thread.

Thus, as shown in Figure 1, SCAN-XP add `from` array into CRS in order to efficiently assign a thread into each edge in E. `from` array stores adjacent nodeIDs corresponding to nodeIDs in the `to` array. As a result, SCAN-XP easily assigns a thread into each edges by obtaining a nodeID from `to` array and its corresponding nodeID of `from` array. From the above thread assignment strategy, SCAN-XP avoids the unbalanced load distributions.

SIMD-based parallelization: Then, we exploit 512 bit SIMD instructions to efficiently compute the structural similarity defined by Definition 2.2. As shown in Definition 2.2, we need to compute a set intersection between $\Gamma(v)$ and $\Gamma(w)$ to obtain a structural similarity $\sigma(v, w)$. Recent work [4] clarified that sort merge join (SMJ for short) based set intersection shows better performances compared with the other approaches. Thus, in this paper, SCAN-XP presents 512 bit SIMD-based set intersections by extending SMJ algorithm on each physical core of Xeon Phi.

Figure 2 and Algorithm 2 overview our SIMD-based structural similarity computation algorithm. Our algorithm is based on block-wise SIMD-based set intersection method [9] that is recently proposed by Inoue *et al.*, in 2015. In STEP 1, SCAN-XP first loads parts of nodes in $\Gamma(v)$ and $\Gamma(w)$ as blocks into SIMD registers. Then, in STEP 2, it counts the number of common nodes between the two blocks by comparing all node combinations on the 512 bit SIMD registers. Finally, as shown in STEP 3, SCAN-XP loads a new block from $\Gamma(w)$ (or $\Gamma(v)$), which has smaller nodeID than the other, into a 512 bit SIMD register. SCAN-XP continues STEP 1 to STEP 3 until the two blocks load all nodes in $\Gamma(v)$ and $\Gamma(w)$.

Based on the above algorithm, we further introduce two parameters α and β in order to optimize its performances. Since real-world graphs may have highly skewed degree distribution, the sizes of $\Gamma(v)$ and $\Gamma(w)$ could be imbalanced. In such cases, as reported in the literature [9], the algorithm degrades its computation speed compared with the performance if $|\Gamma(v)| \approx |\Gamma(w)|$. Hence, in order to speed up for real-world graphs, we use the parameters α and β; α and β specifies the number of nodes loaded in the two blocks. For example, since we set $\alpha = \beta = 2$ in Figure 2, SCAN-XP always loads two nodes in STEP 1 and STEP 3. If SCAN-XP uses 512 bit SIMD registers and 32 bit integers to represent a node in V, α and β could take 3 patterns based on the balance between $|\Gamma(v)|$ and $|\Gamma(w)|$: (i) $\alpha = 1$, $\beta = 16$, (ii) $\alpha = 2$, $\beta = 8$, and (iii) $\alpha = \beta = 4$. As shown in line 2-6 of Algorithm 2, we set $\alpha = 1$ and $\beta = 16$ if $|\Gamma(v)| \geq 2|\Gamma(w)|$ (or visa verse), otherwise $\alpha = \beta = 4$ since this setting shows the best performance in our evaluations. The automatic parameter optimization is a future work.

Algorithm 2 SIMD-based structural similarity computation

```
Input: v, w ∈ V,
Output: σ(v, w)
 1: // Initialization
 2: if |Γ(v)| ≥ 2|Γ(w)| (or 2|Γ(v)| ≤ |Γ(w)|) then
 3:    α = 1, β = 16 (or α = 16, β = 1);
 4: else
 5:    α = β = 4;
 6: end if
 7: get head pointers vp and wp from Γ(v) and Γ(w), respectively;
 8: get tail pointers v_end and w_end from Γ(v) and Γ(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 Γ(v) and Γ(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+α == wp+β then
17:       vp+ = α, wp+ = β;
18:    else if vp + α > wp + β then
19:       wp+ = β;
20:    else
21:       vp+ = α;
22:    end if
23: end while
24: σ(v, w) = (vw_common + 2)/√(|Γ(v)||Γ(w)|);
```

3.3 Parallel cluster construction step

As shown in Section 2.1, SCAN needs to expand cluster in the recursive manner, however it is not suitable for parallel computation manner. This is because that we need to run parallel threads iteratively in order to complete the cluster construction; this is time-consuming since the number of the iteration equals to the diameter of a graph in the worst case.

In order to improve the efficiency, SCAN-XP employs a parallelizable cluster construction method by using *union-find tree* [5]. Union-find tree is a data structure that keeps set of values partitioned into disjoint subsets. It supports two operations: *find*(u) and *union*(u, v). *find*(u) is an operation to check which subset does data u belongs to, and *union*(u, v) merges two subsets, which are data u and v belong to, into the same subset. It is known that each operation can be done in $\Omega(A(n))$ where A is Ackermann function.

SCAN-XP constructs clusters from the union-find tree in massively parallel manner on Xeon Phi. Initially, by using union-find tree, SCAN-XP manages clusters what nodes belong to. If node u is core, ϵ-neighborhood $N_\epsilon(u)$ could be in the same cluster of node u from Definition 2.4. Unlike SCAN shown in Algorithm 1, SCAN-XP does not immediately merge $N_\epsilon(u)$ into the same cluster of node u. For each node in $N_\epsilon(u)$, say node v, SCAN-XP first invokes *find*(u) and *find*(v) operators. If *find*$(u) \neq$ *find*(v), node

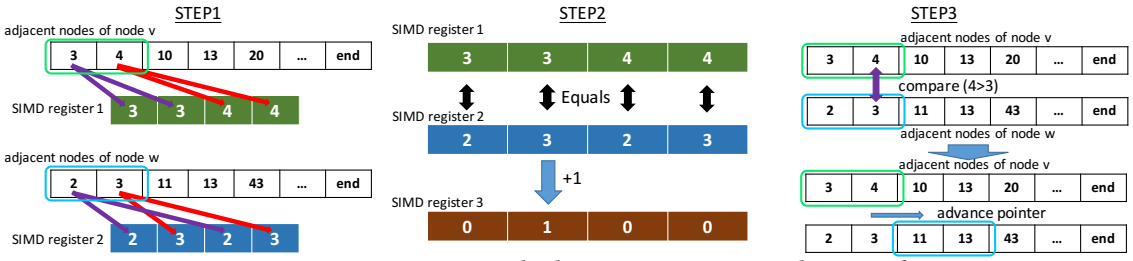

Figure 2: Counting common nodes by SIMD instruction where $\alpha = \beta = 2$.

Algorithm 3 SCAN-XP

Input: $G = \{V, E\}$, $\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
Output: C, H, and O
1: $\forall v \in V$ are labeled as *unclassified*;
2:
3: // Step 1: Parallel core detection in Section 3.2
4: **for** each edge $(v, w) \in E$ **do in parallel**
5: run Algorithm 2;
6: **end for**
7:
8: // Step 2: Parallel cluster construction in Section 3.3
9: **for** each core node $v \in V$ **do in parallel**
10: **for** each $w \in N_\epsilon(v)$ **do**
11: **if** $find(v) \neq find(w)$ **then**
12: get $C(v) \cup C(w)$ 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** $\exists u, w \in \Gamma(v)$ *s.t.* $C(u) \neq C(w)$ **then**
21: label node v as hub, and $H = H \cup \{v\}$;
22: **else**
23: label node v as outlier, and $O = O \cup \{v\}$;
24: **end if**
25: **end for**

u and node v are not in the same cluster until now; thus, SCAN-XP performs $merge(u, v)$ so as to construct a cluster from node u and v. For improving the computation speed, SCAN-XP performs the above procedure in parallel by using all of physical cores on Xeon Phi. SCAN-XP uses CAS (Compare-And-Swap) instruction to $union(u, v)$ so as to avoid write conflicts among parallel threads.

3.4 Algorithm of SCAN-XP

We show overall algorithm of SCAN-XP in Algorithm 3. As we described in Section 3, SCAN-XP consists of three parallel computation steps: (1) parallel core detection in line 3-6, (2) parallel cluster construction in line 8-16, and (3) parallel hubs/outliers detection in line 18-25. From Algorithm 3, SCAN-XP has the following theorem:

Theorem 3.1 (Exactness of SCAN-XP). *SCAN-XP always obtains exactly same clustering results as SCAN.*

We omit the proof of Theorem 3.1 due to space limitation.

4 EVALUATION

We evaluate the effectiveness of our proposed method SCAN-XP by using several real-world graphs. We compared the following structural graph clustering algorithms including our proposal SCAN-XP:

- **SCAN-XP:** Our proposal. We implemented three types of SCAN-XP that runs on CPU, Xeon Phi KNC, and Xeon Phi KNL denoted by *SCAN-XP (CPU)*, *SCAN-XP (KNC)*, and *SCAN-XP (KNL)*, respectively.

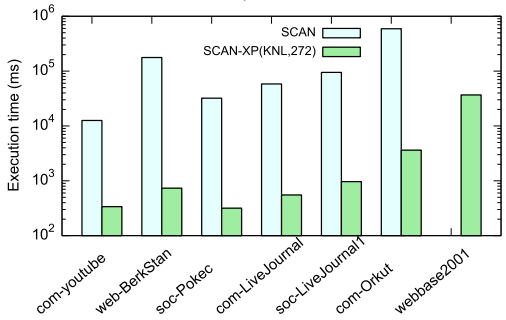

Figure 3: Runtimes for real-world datasets

- **SCAN [21]:** The most standard structural graph clustering that runs as a single-threaded algorithm on a CPU.

SCAN-XP and SCAN requires user-specified parameters ϵ and μ. In our experiments, we evaluated performances by varying the parameters, however the parameter settings showed almost same performances. Thus, we used $\epsilon = 0.3$ and $\mu = 2$.

In addition to the above parameters, SCAN-XP also requires to specify parameters α and β for SIMD-based set intersections shown in Algorithm 2. As we described in Section 3.2, we set $\alpha = 1$ and $\beta = 16$ if $|\Gamma(v)| \geq 2|\Gamma(w)|$ when SCAN-XP computes $\sigma(v, w)$, otherwise, we used $\alpha = \beta = 4$. This is because that, in our evaluations, the above settings showed the best performances.

Datasets: We used seven real-world datasets in the evaluations. Details of each dataset are shown in Table 3. In each datasets, we removed self-loop edges and nodes without any connections from the datasets in order to run SCAN and SCAN-XP.

Experimental environments: All experiments were conducted on Intel Xeon CPU E-1620 (hereinafter, referred as CPU), Intel Xeon Phi Knights Corner 3120A (KNC), and Intel Xeon Phi Knights Landing 7250 (KNL). KNC uses Intel MPSS version 3.7.2, and we set memory mode of KNL to `flat mode`. All algorithms were implemented by using `icpc 17.0.0` with `-O3` option. Table 2 summarizes the details of our experimental environment settings.

4.1 Efficiency

We evaluated the clustering performance of each method through wall clock time for the real-world datasets. In the evaluation, we compared SCAN and SCAN-XP (KNL) using 272 threads. Figure 3 shows the runtimes for each real-world dataset. In Figure 3, we omitted the result of SCAN for webbase-2001 since it does not return the result within 1 hour. Figure 3 shows that SCAN-XP is much faster than SCAN under all conditions examined. As described earlier, SCAN computes all edges in a single-threaded algorithm, while SCAN-XP efficiently exploits 272 threads for structural graph

Table 2: Experimental environment

CPU	Memory	Clock frequency	# of cores	Maximum # of threads	OS	SIMD type
Xeon E5-1620 v3	16GB	3.5 GHz	4	8	Cent OS 7	AVX2
Xeon Phi 3120A	6GB	1.10 GHz	57	228	Linux 3.10	IMCI
Xeon Phi 7250	16GB	1.4 GHz	68	272	Cent OS 7	AVX-512

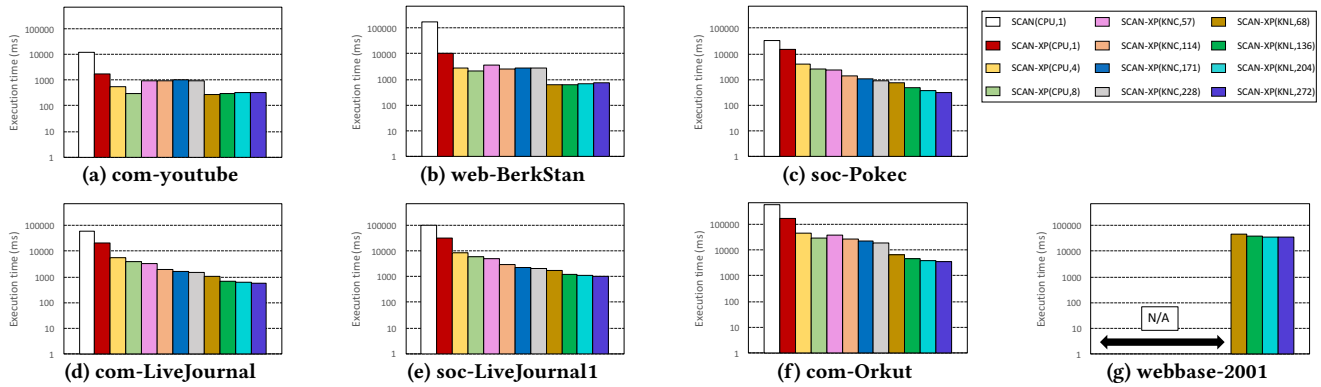

Figure 4: Scalability (overall)

Table 3: Real-world datasets

Dataset	n	m	Data source
com-youtube	1,134,890	2,987,624	SNAP [14]
web-BerkStan	685,230	6,649,470	SNAP [14]
soc-Pokec	1,632,803	22,301,964	SNAP [14]
com-LiveJournal	3,997,962	34,681,189	SNAP [14]
soc-LiveJournal1	4,846,609	42,851,237	SNAP [14]
com-Orkut	3,072,441	117,185,083	SNAP [14]
webbase-2001	115,554,441	854,809,761	LAW [2]

clustering. As a result, our proposal is up to 100 times faster than the original algorithm SCAN for average; SCAN-XP computes clusters from webbase-2001 that has more than 850 million edges within 36 seconds. Besides the results in Figure 3, Shiokawa *et al.* reported that the state-of-the-art method SCAN++ requires almost 1,000 seconds to compute webbase-2001 [18]. Thus, SCAN-XP runs considerably faster than the existing approaches.

4.2 Scalability

We evaluated scalability of SCAN-XP by varying the number of threads for all experimental conditions we examined. In order to evaluate SCAN-XP on CPU, we used 1-8 threads on CPU by using OpenMP; SCAN-XP (KNC) and SCAN-XP (KNL) varied the number of threads from 1 to 228 and from 1 to 272, respectively. Figure 4 represents the results of scalability evaluations. As well as Figure 3, SCAN and SCAN-XP (KNC) do not return results for webbase-2001 whithin 1 hour, we thus omitted the results from Figure 4 (g). Figure 4 shows that both SCAN-XP (KNC) and SCAN-XP (KNL) outperform SCAN. As increasing the number of threads, SCAN-XP (KNL), SCAN-XP (KNC), and SCAN-XP (CPU) gradually improve their clustering efficiency. SCAN-XP (KNL) with 272 threads performs 164 times faster than SCAN for com-Orkut; SCAN-XP (KNL) completes the clustering in 3.6 seconds. In contrast, SCAN-XP (KNC) and SCAN-XP (KNL) degrade their performances for small dataset such as com-youtube and web-Berkstan.

5 RELATED WORK

Structural graph clustering [3, 4, 18–20, 22, 23] is an algorithm that detects clusters, hubs and outliers. Since it is effective to avoid resolution limit problem of modularity-based methods, structural graph clustering is able to find clusters with high accuracy; structural graph clustering is effective to reproduce the ground truth [18, 21]. SCAN [21], proposed by Xu *et al.*, is the most standard algorithm for the structural graph clustering. It is an extension of the traditional density-based clustering DBSCAN [7]. Xu *et al.* reported that SCAN outperforms modularity-based methods in producing clustering results that resemble the ground-truth. However, SCAN suffers from its performance limitation.

In order to overcome the limitation, several methods [4, 15, 18] have been proposed in a recent few years. SCAN++ [18] and pSCAN [4] are efficient and exact structural graph clustering method for large-scale graphs. By focusing on several observations of real-world graphs, the methods are designed to reduce the computational cost for core detection and cluster construction. PSCAN [23], proposed by Zhao *et al.*, is a distributed structural graph clustering method using MapReduce framework. PSCAN first detects cores by using several map and reduce processes, and then it just drops non-core nodes as noises. As a result, PSCAN produces an approximated clustering results as well as LinkSCAN* in the distributed computation manner. Since PSCAN needs to iteratively write/read intermediate results into/from storages, it requires much larger computation time than SCAN++ and pSCAN.

6 CONCLUSION

We present a novel structural graph clustering algorithm *SCAN-XP* that exploits Intel Xeon Phi coprocessors. As a result, our evaluations and theoretical analysis show that SCAN-XP achieves efficient and scalable clustering without sacrificing clustering quality compared with SCAN. SCAN is a fundamental to many current and prospective applications in various disciplines. Our proposal will improve the effectiveness of future applications.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Numbers JP26280037, JP16H07410, and Interdisciplinary Computational Science Program in CCS, University of Tsukuba, and University of Tsukuba Basic Research Support Program Type A.

REFERENCES

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

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

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

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

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.

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

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. 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)*. 226–231.

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

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

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

[11] U Kang and Christos Faloutsos. 2011. Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining. In *Proceedings of the IEEE 11th International Conference on Data Mining (ICDM)*. 300–309.

[12] Jon M. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (Sep 1999), 604–632.

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

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

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

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

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

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

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

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

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

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

[23] Weizhong Zhao, Venkata Swamy Martha, and Xiaowei Xu. 2013. 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)*. 862–869.