Parallel Modularity-based Community Detection on Large-scale Graphs

Jianping Zeng, Hongfeng Yu
Department of Computer Science and Engineering
University of Nebraska Lincoln
Lincoln, Nebraska
Email: {jizeng, yu}@cse.unl.edu

***Abstract*—We present a parallel hierarchical graph clustering algorithm that uses modularity as clustering criteria to effectively extract community structures in large graphs of different types. In order to process a large complex graph (whose vertex number and edge number are around 1 billion), we design our algorithm based on the Louvain method by investigating graph partitioning and distribution schemes on distributed memory architectures and conducting clustering in a divide-and-conquer manner. We study the relationship between graph structure property and clustering quality, carefully deal with ghost vertices between graph partitions, and propose a heuristic partition method suitable for the Louvain method. Compared to the existing solutions, our method can achieve nearly well-balanced workload among processors and higher accuracy of graph clustering on real-world large graph datasets.**

***Index Terms*—large graph; community detection; graph clustering; parallel and distributed processing.**

I. Introduction

Retrieval of information or patterns hidden in graphs is of great interest in numerous fields. Researchers have made substantial efforts to analyze and understand the organization of graphs. One of the powerful techniques is *graph clustering*, or named *community detection*. There is no strict definition on community or cluster of a graph; however, one commonly accepted concept is that a graph is partitioned into sub-groups (communities or clusters) of vertices who have dense intra-connections, but sparse inter-connections [1].

As we enter the era of big data, graph clustering becomes a severely challenging task due to an exponential growth of graph size. A large graph can contain millions of vertices and billions of edges, which is hard to be processed efficiently using a single machine. Parallel clustering, which uses parallel machines with distributed memory, provides a natural solution to cope with large data. However, there is still a lack of scalable parallel algorithms for large graphs, mainly due to the following reasons.

First, most large and complex graphs are characterized with a scale-free structure such that only a few vertices have high degrees but most vertices have low degrees [2]. The sparse connections between the vertices incur a poor locality during data access for most primitive graph operations. Thus, when such a graph is partitioned and distributed among processors, parallel operations often suffer intensive data communication and imbalanced workload.

Second, after data partitioning and distribution, the global structure information of a graph is typically not preserved on each processor. Missing of such information can lower the cluster quality of local community detection and impair the accuracy of final aggregated results.

In this work, we aim to boost the scalability and the accuracy of parallel community detection. In particular, we implement a parallel modularity-based algorithm using distributed memory machines, and increase the clustering quality by considering the information of ghost vertices for local clustering. We study the interplay between graph structure property and clustering quality, and make use of graph degree distribution to study clustering accuracy. With considering modularity-based clustering, we develop a novel graph partitioning and distribution scheme to achieve workload balancing among processors and ensure the quality of clustering. We demonstrate the effectiveness and scalability of our algorithm using several real-world large datasets with up to 16,384 cores. The experimental study shows that our approach can achieve an improved accuracy that is close to the ground truth generated by the sequential algorithm. To the best of our knowledge, we present the first scalable distributed modularity-based algorithm to cluster the full extent of a large graph with over 1 billion edges.

II. Related Work

We refer interested readers to the literature survey [1] for an in-depth overview of graph clustering algorithms. We first review the sequential algorithms relevant to our work, and then discuss the corresponding work on parallelization.

Label propagation algorithm (LPA) [3] is one of the most representative graph clustering algorithms. In LPA, each vertex iteratively picks a label in its neighborhood that has the maximum frequency and the group of vertices with the same label forms a cluster. Label Propagation Algorithm (SLPA) [4] is an improvement on LPA. In 2004, Newman and Girvan [5] introduced the modularity measurement to quantify the quality of graph clustering, which laid the foundation of the modularity based clustering algorithms. Afterwards, Clauset, Newman, and Moore [6] proposed an agglomerative graph clustering algorithm (also known as Clauset-Newman-Moore algorithm, CNM for short), which merged the vertices achieving the global maximum modularity value. Blondel et al. [7] proposed

a heuristic method, known as the Louvain method, which can achieve a better result with a lower time complexity.

There have been some efforts in parallelizing graph clustering for large scale graphs based on shared memory architecture. Riedy et al. presented a parallel agglomerative algorithm [8] that extended the CNM algorithm. Their approach treated the problem as the maximum edge weighted matching problem and performed multiple pairwise community merges in parallel. In some case, the algorithm can produce a maximum matching within a factor of 50%. Kuzmin et al. [9] proposed a multi-thread SLPA algorithm for shared memory, but their method can only process a graph in the order of ten million edges. Recently, Bhowmick et al. [10] proposed an OpenMP implementation which adopted lock mechanism with a limited scalability. Staudt et al. [11] proposed a shared memory parallel clustering algorithm that combined the Louvain method and the LPA method. They conducted the algorithm using a computer with 512GB main memory. Lu et al. [12] proposed a parallel Louvain algorithm based on coloring preprocessing.

There is comparably limited work of parallel graph clustering on distributed memory architecture. Zhang et al. [13] proposed a parallel hierarchical graph clustering method that dynamically constructed the network topology. Soman et al. [14] built a parallel LPA graph clustering on GPUs cluster that was limited to a marginal size graph. Cheong et al. [15] presented a GPU-based Louvain algorithm that partitioned the original graph into several sub-graphs, ignored the edges connecting vertices residing in different sub-graphs, and conducted clustering using divide-and-conquer strategy. However, their accuracy of clustering result was relatively lower than the ground truth on multiple GPUs. Que et al. [16] implemented a distributed Louvain algorithm, but their experiment only showed the processing of real-world graphs with less than 1 billion edges.

III. Our Approach

The Louvain algorithm is characterized with a nearly linear time complexity and achieves the highest quality of community detection compared to any other sequential algorithms. However, the corresponding parallelization attempts have not demonstrated a desired scalability for large graphs. In addition, they have used simple graph partitioning and distribution schemes that cannot ensure the accuracy of clustering.

In our approach, we first revisit several key definitions and concepts in the Louvain algorithm (Section III-A), and study the relationship between graph structure property and clustering accuracy (Section III-B). We find that ghost vertices are critical to the accuracy of parallel clustering, which however have been largely neglected in existing work. Based on our observations, we design a new scheme for graph partitioning and distribution (Section III-C), leading to a scalable design of parallel Louvain algorithm that ensures balanced workload and accurate clustering for large graphs (Section III-D). In the following, we describe the details of our approach.

A. Background

Without loss of generality, we only consider undirected graphs in this work. However, our approach can be easily extended to directed graphs [15].

In a *graph* $G = (V, E)$, V is the set of vertices (or nodes) and E is the set of edges (or links). The weight of an edge between two vertices, u and v, is denoted as $w_{u,v}$, which is 1 in undirected unweighted graph. In order to partition the original graph into sub-graphs for each processor, *1D partition*, also called *vertex partition*, is a partition strategy that assigns vertices uniquely among the q processors $p_0, p_1, \ldots p_{q-1}$. After partitioning, the vertex number on each processor is roughly the same. The edges incident to the vertices assigned to p_i are all stored on p_i. After partitioning, the graph vertices that are directly assigned to one processor, p_i, are called the *local vertices* of the processor, represented as V_{Lp_i}. The vertices that locate on foreign processors, but are connected to the local vertices of p_i are called the *ghost vertices*, represented as V_{Gp_i}. The edges only connecting local vertices of p_i are represented as E_{Lp_i}, and the edges between ghost vertices and local vertices of p_i are represented as E_{Gp_i}.

Modularity, Q, is a measurement used to quantify the quality of communities detected in a graph [6], which can be formulated as:

$$Q = \frac{1}{2m}\sum_{vw}[A_{vw} - \frac{d_v d_w}{2m}]\delta(C_v, C_w), \tag{1}$$

where m is the number of edges in the graph, v and w are two vertices, and d_v and d_w are the degrees of v and w, respectively. A_{vw} represents the connectivity between v and w, which is 1 when v and w are connected by an edge and is 0 otherwise. C_v and C_w are the communities that contain v and w, respectively. The value of δ function is 1 when C_v and C_w is the same community; otherwise it is 0. The intuition of Equation 1 is that if the modularity value is high, there are many edges inside communities but only a few between different communities, indicating a high quality of community detection.

Modularity gain δQ [7] is the gain in modularity obtained by moving an isolated node v_i into a community C, which can be easily computed by:

$$\delta Q = [\frac{\sum_{in}+k_{v_i,in}}{2m} - (\frac{\sum_{tot}+k_{v_i}}{2m})^2] - [\frac{\sum_{in}}{2m} - (\frac{\sum_{tot}}{2m})^2 - (\frac{k_{v_i}}{2m})^2]. \tag{2}$$

where $\sum_{in}$ is the total edge weight inside C, $k_{v_i,in}$ is the sum of edge weight from a vertex v_i to C, $\sum_{tot}$ is the total weight of edges incident to vertices belong to C, k_{v_i} is the total weight of edges incident to v_i, and m is the sum of the weights of all edges in the graph. We can simplify Equation 2 and obtain:

$$\delta Q = \frac{1}{2m}(k_{v_i,in} - \frac{\sum_{tot} * k_{v_i}}{m}). \tag{3}$$

This equation can be further approximated as:

$$\delta Q \sim k_{v_i,in} - \frac{\sum_{tot} * k_{v_i}}{m}. \tag{4}$$

The Louvain algorithm is designed based on the modularity measure (Equation 1). It is a hierarchical agglomerative clustering method in that initially each vertex is regarded as a unique community, and then communities are merged

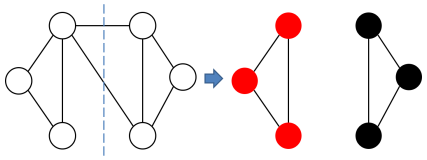


Fig. 1: The simplified 1D partitioning and distribution scheme. A graph is partitioned into 2 sub-graphs denoted with different colors. The edges between the sub-graphs are ignored.

iteratively. In each iteration, multiple communities are merged into a new community to maximize the modularity gain (Equation 2, or the approximation, Equation 4). This process goes on until there is no modularity gain among new communities.

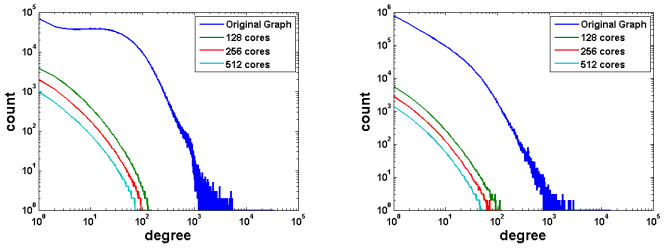
B. Clustering Accuracy Study

Divide-and-conquer is a conventional strategy widely used to parallelize algorithms on distributed memory architecture. For parallel graph community detection, a general idea is to partition a large graph into sub-graphs and distribute them among processors. Each processor then conducts local community detection and then collaboratively aggregates the final result. This strategy has been adopted in most existing distributed parallel algorithms [15], [14].

Data partitioning and distribution scheme is the key to the scalability of a parallel algorithm [17]. With respect to parallel community detection, we need to design an appropriate scheme by considering both the quality and the performance of solutions. However, these two issues have not been extensively and holistically studied. For example, Buluc and Madduri [18] discussed the relationship between the communication cost and the graph partitioning and distribution scheme for parallel graph traversal, but the relation between community quality and partition strategy is not their focus. Cheong et al. [15] adopted a simple 1D graph partitioning and distribution scheme. Although they achieved reasonable clustering results, they did not provide corresponding rationales. Hence, the fundamental relationship between community quality and graph partitioning and distribution scheme is not entirely clear in their method.

Because Cheong et al.'s approach achieves one of the best clustering results among the existing distributed parallel algorithms based on GPUs cluster, we are interested in studying the rationales behind their approach. Figure 1 revisits the simplified 1D partitioning and distribution scheme adopted in their method, where a graph is partitioned into a number of sub-graphs, each sub-graph only contains the local vertices and the edges between them, and the edges between the local vertices and the ghost vertices are ignored for each sub-graph. We represent this type of sub-graph on a processor p_i as $G_{Lp_i} = (V_{Lp_i}, E_{Lp_i})$.

1) Characterization of Sub-Graphs: We observe that *degree distribution* is a critical graph structure property related to the quality of graph clustering and can be used to explain Cheong et al.'s approach. This is mainly because of two reasons. First, according to Equations 1 and 4, we only need degree information and edge number to calculate modularity and modularity



(a) (b)

Fig. 2: Comparison of degree distribution between sub-graphs and original graph. (a) shows the results of an Orkut dataset. (b) shows the results of a Youtube dataset. In each plot, the blue curve corresponds to the degree distribution of the original graph; and the remaining three curves correspond to the average degree distribution of sub-graphs on 128, 256, and 512 cores, respectively.

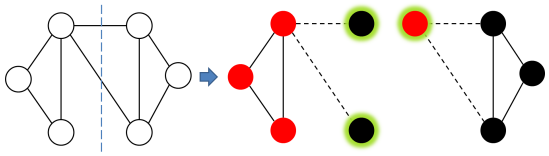
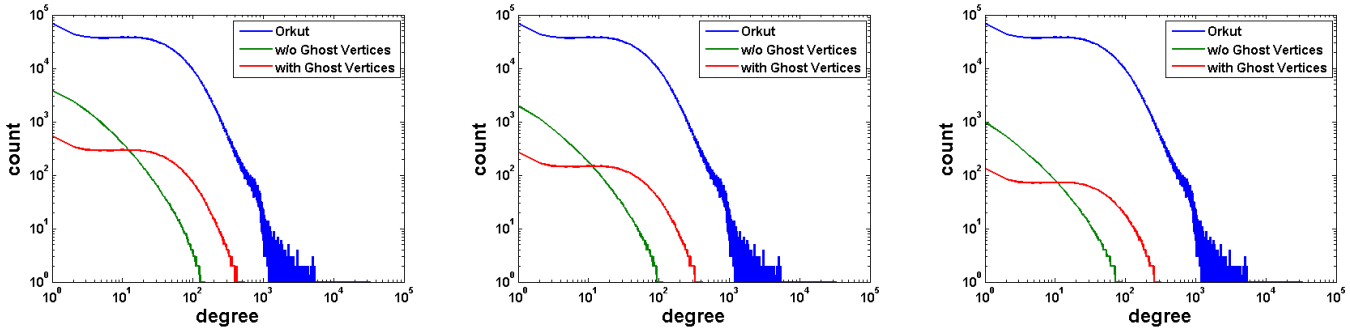


Fig. 3: The partitioning and distribution scheme with ghost vertices. A graph is partitioned into 2 sub-graphs with different colors. The vertices circled in green are the ghost vertices, and their colors correspond to their sub-graphs. The dotted lines represent the edges connecting the local vertices and the ghost vertices across the sub-graphs.

gain, where the number of edges can be derived from degree information. Thus, the measurement of modularity mainly depends on graph degree information. Second, if we preserve the degree distribution of each sub-graph, other graph structure properties (such as, connected component size and clustering coefficient) can be preserved as well [19].

We can further use degree distribution to explain the effectiveness of 1D graph partition in the result obtained by Cheong et al., that is why local clustering can be applied on each sub-graph and lead to a reasonable global result. We observe that when we use 1D graph partition, the degree distribution of each sub-graph is approximately coherent with the original global graph. Figure 2 shows a comparison of degree distribution between the sub-graphs and the original graph using two real-world datasets from Orkut [20] and Youtube [21] on 128, 256, and 512 cores, where the average degree distribution of sub-graphs is generally matched with that of the original graph. This verifies that 1D graph partition can preserve the structure property of a graph to a certain degree, and achieve a comparably appropriate quality of community detection.

However, we also observe that there is still some slight difference between the sub-graph degree distribution and the original graph degree distribution in Figure 2. This is because the 1D graph partition scheme used in Cheong et al.'s work ignored the edges between local vertices and ghost



(a) (b) (c)

Fig. 4: Comparison of degree distribution between sub-graphs with ghost vertices and without ghost vertices. (a), (b) and (c) show the comparison results using 128, 256, and 512 cores, respectively. In each plot, the blue curve corresponds to the degree distribution of the original graph; and the red and green curves correspond to the average degree distributions of sub-graphs with ghost vertices and without ghost vertices, respectively. We can see that the degree distribution of sub-graphs with ghost vertices is more similar to that of original graph.

vertices, and certain degree information of local vertices is lost. Thus, we naturally have an intuition that if we consider ghost vertices, we can make the structure information of sub-graph on each processor more close to the original graph. To testify this hypothesis, rather than ignoring the edges connecting the local vertices and the ghost vertices, we store all these edges and the ghost vertices for each sub-graph. We do not consider the edges between the ghost vertices, but use the degrees of the ghost vertices in the original graph. Figure 3 shows this partitioning and distribution scheme. We represent this new type of sub-graph on a processor p_i as $G_{Lp_i} = (V_{Lp_i} \cup V_{Gp_i}, E_{Lp_i} \cup E_{Gp_i})$.

We then compare the degree distribution between the sub-graphs with ghost vertices and the sub-graphs without ghost vertices using the Orkut dataset on different cores. As shown in Figure 4, we can clearly see that, with considering ghost vertices, we make the sub-graphs have a more similar degree distribution with the original graph. This means the sub-graph on each processor well preserves the structure information of the original graph. We present a detailed quantitative comparison between these two schemes in Section IV-C.

2) Clustering Accuracy: We derive a lemma that shows clustering accuracy decreases if ghost vertices are ignored.

Lemma 1: For a sub-graph, a local vertex can be clustered into a wrong community without considering ghost vertices.

Proof Given a local vertex u, if we ignore any ghost vertices u connects with, u will definitely be clustered with the local vertices. However, if u also connects with the ghost vertices, there can exist a ghost vertex v_{g_a} that gives us the maximum modularity gain according to Equation 4. As a result, the local vertex u should be clustered with v_{g_a} rather than any other local vertices, thus occurring a local clustering error.

This lemma shows that an involvement of ghost vertices is a necessity for preserving graph structure on each processor and for improving clustering accuracy.

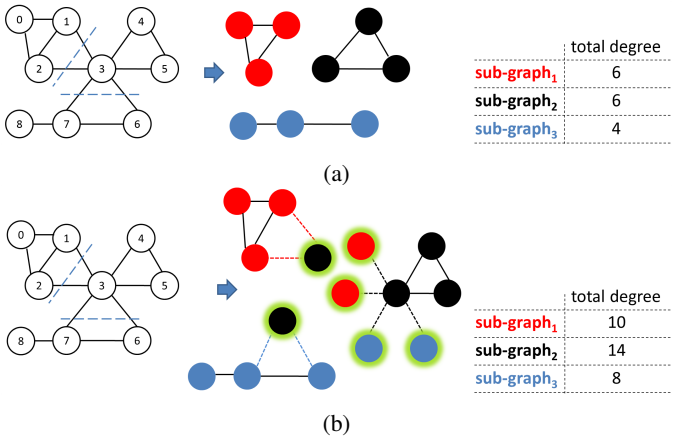


Fig. 5: (a) and (b) show the results of 1D partitioning without and with considering ghost vertices, respectively. The tables show the total degrees of sub-graphs in each case. The vertices circled in green are the ghost vertices. Three sub-graphs are distinguished by the colors of local vertices.

C. Graph Partitioning and Distribution Scheme

The involvement of ghost vertices, however, substantially increases the complexity in partitioning and distribution of a graph to achieve load balancing. We can illustrate the difficulty by first revisiting the workload estimation of the sequential Louvain algorithm: In an iteration of the sequential Louvain algorithm, for each vertex v, we need to find the maximum modularity gain by going through each neighbor of v to find the maximum modularity gain. Thus, we can easily see that the overall complexity of the sequential Louvain algorithm is proportional to the total degree of all vertices of a graph.

If we use a simple 1D partitioning and distribution scheme without considering ghost vertices as Cheong et al. [15], we can achieve a nearly balanced workload partition because each sub-graph has an approximately equal summation of its local vertex degrees, where the edges between local vertices and ghost vertices are neglected. Figure 5 (a) shows an example

where a graph is partitioned into 3 sub-graphs without considering ghost vertices. However, if we involve ghost vertices, the total degree of each sub-graph can become significantly different, although each sub-graph may still have a similar number of local vertices. As shown in Figure 5 (b), the sub-graph, $sub\text{-}graph_2$, has the same set of local vertices as in Figure 5 (a); however, its total degree is significantly higher than the others. This typically happens when a local vertex has a large degree and is connected to many ghost vertices. In this case, if we conduct the Louvain algorithm on each sub-graph, the workload can be significantly different, although we can achieve more accurate clustering on each sub-graph.

1) Workload Estimation: This issue seems inevitable given that a large graph with a scale-free structure may always have a few vertices with very high degrees. We derive the following lemma to cope with this issue according to Equation 4.

Lemma 2: Given an sub-graph on a processor p_i, $G_{Lp_i} = (V_{Lp_i} \cup V_{Gp_i}, E_{Lp_i} \cup E_{Gp_i})$, we have a set of ghost vertices $v_{g_1}, v_{g_2}, ..., v_{g_n} \in V_{Gp_i}$, which only connect the same local vertex v_l. If $\text{argmin}(degree(v_{g_1}), degree(v_{g_2}), ..., degree(v_{g_n})) = v_{g_a}$, then in the final clustering result of this sub-graph, $C(v_l) \neq C(v_{g_b})$, where $g_1 \leq g_b \leq g_n$ and $g_b \neq g_a$.

Proof We prove this lemma from two aspects:

First, we show that if in some iteration $C(v_l) = C(v_{g_a})$, then in the following iterations, $C(v_l) \neq C(v_{g_b})$, where $g_1 \leq g_b \leq g_n$ and $g_b \neq g_a$.

According to Equation 4, the modularity gain between v_l and a ghost vertex v_{g_i} is:

$$\delta Q_{v_l v_{g_i}} \sim d_{v_l, v_{g_i}} - \frac{\sum_{tot(v_{g_i})} * d_{v_l}}{m}. \tag{5}$$

In Equation 5, we note that $d_{v_l, v_{g_i}} = 1$, as there is only one edge between v_l and v_{g_i}. In addition, d_{v_l} and m are constant. Thus, given v_{g_a} has the lowest degree among $v_{g_1}, v_{g_2}, ..., v_{g_n}$, we can easily see that $\sum_{tot(v_{g_a})} < \sum_{tot(v_{g_b})}$ and $\delta Q_{v_l v_{g_a}} > \delta Q_{v_l v_{g_b}}$, where $g_1 \leq g_b \leq g_n$ and $g_b \neq g_a$. This means that once $C(v_l) = C(v_{g_a})$, $C(v_l)$ cannot become $C(v_{g_b})$ in the following iterations.

Second, even if in some iteration $C(v_l) = C(v_{g_b})$, we can also easily see that $C(v_l) \neq C(v_{g_b})$ in a later iteration based on our proof in the first aspect.

Lemma 2 allows us to prune a considerable amount of ghost vertices for one sub-graph. For example, considering the sub-graph, $sub\text{-}graph_2$, in Figure 5 (b), the ghost vertices, v_1, v_2, v_6, and v_7, all connect and only connect to the local vertex v_3. The degree of v_6 is the smallest among these four vertices. According to Lemma 2, only v_6 needs to be considered for local clustering, and the rest ghost vertices can be discarded. This reduces the total degree of the second sub-graph from 14 to 8, and thus significantly reduces the associated workload.

Based on Lemma 2, we develop a heuristic method to estimate the workload of each partition according to the edge numbers in adjacency matrix. We note that each non-empty entity in an adjacency matrix represents an edge. Figure 6 (a) shows the adjacency matrix of the graph in Figure 5. Each

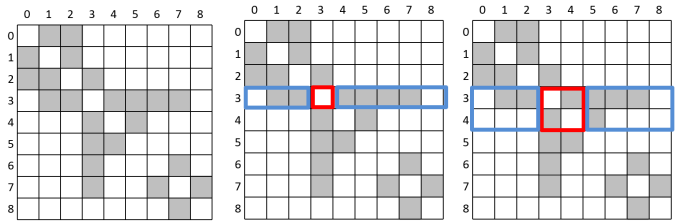


Fig. 6: (a) shows the adjacency matrix of the graph in Figure 5. It contains 9 vertices, and each non-empty entity represents an edge. We partition the matrix vertically. For each partition, the local vertices are circled in red, and the ghost vertices are circled in blue. (b) shows a partition only containing v_3, and (c) shows a partition containing both v_3 and v_4.

gray cell corresponds to one edge. If a partition only contains a local vertex v_i, and all ghost vertices of this partition connect to this local vertex, then in this case we only need to examine at most one ghost vertex to determine the modularity gain, according to Lemma 2. Figure 6 (b) shows an example, where we have one partition containing only v_3. The blue region corresponds to the ghost vertices and the red region indicates the local vertex. Obviously, there is no workload associated with the local vertex because it lacks edges in the red region. Although there are multiple edges in the blue region, only one vertex (v_6 in this case, because v_6 is the ghost vertex with minimum degree connecting with v_3 in the partition) needs to be considered.

The workload of one partition increases if more local vertices are added. For the local vertices, we simply count the total number of edges among local vertices. For the ghost vertices that have more than one edges connecting to the local vertices, we count these edges. For the ghost vertices that have only one edge connecting to the local vertices, we use a simplified heuristic that only counts one of these edges. Figure 6 (c) shows an example, where we have one partition containing v_3 and v_4. The red region contains two edges with respect to the local vertices. The blue region totally contains six edges, where v_5 has two edges connecting to the local vertices, but v_1, v_2, v_6, or v_7 only has one edge. In this case, we count the two edges of v_5, but only count one edge for the rest of ghost vertices. Therefore, the workload of this partition is proportional to five edges (i.e., two local edges plus three ghost edges). This method can be easily extended to estimate the workload of a partition containing more local vertices.

2) Graph Partition Algorithm: We next examine how to partition a graph into a set of sub-graphs. Given q processors and a graph $G = (V, E)$, the task is equal to vertically split the adjacent matrix of G into q partitions, where each partition contains multiple rows and is associated with an approximately equal amount of workload.

At first glance, this problem is similar to 1D workload balancing problem, and Pinar and Aykanat provided a survey of related solutions [22]. However, they only considered one variable during partitioning. In our problem, we have two

variables, which are the workload associated with only local vertices and the workload associated with both local vertices and ghost vertices. More specifically, due to our pruning strategy, the workload associated with local vertices and ghost vertices depends on how we partition the graph, and thus is not deterministic. Hence, the existing solutions [22] cannot be directly applied to our workload estimation. Besides, there is some other heuristic method for 1D partition [23]. However, their workload estimation for each row in the matrix can be done independently, and this is not the case in our problem.

We develop a new heuristic partition algorithm suitable for our own parallel Louvain algorithm. The key is to find an appropriate average workload h to partition the adjacent matrix. However, given our estimation method in Section III-C1, the workload of a partition depends on the connectivity of local vertices and ghost vertices that can be dramatically changed even if we only add or remove a few local vertices. To cope with this issue, we begin with an initial guess of an average workload h_0 for each partition, where $h_0 = \mid E \mid /q$. Then we sequentially process the adjacent matrix and split it into q_0 partitions. We can easily let each of the first $q_0 - 1$ partitions has a workload around h_0. However, we cannot guarantee that $q_0 = q$ and the workload of the last partition h_{q_0-1} is around h_0. We adjust our guess of h according to two cases:

First, if $q_0 < q$ or if $q_0 = q$ but $h_{q_0-1} \ll h_0$, this means that our initial guess h_0 is relatively large. In this case, we decrease h_0. Our next guess h_1 is $(h_0 + h_{low})/2$, where h_{low} is the lower bound of h and its initial value is 1. We also update the upper bound h_{up} to h_0.

Second, if $q_0 > q$, this means that our initial guess h_0 is relatively small. In this case, we increase h_0. Our next guess h_1 is $(h_0 + h_{up})/2$, where the initial value of upper bound h_{up} is $\mid E \mid$. We also update the lower bound h_{low} to h_0.

We continue this iterative change of h_i, h_{low} and h_{up}, and let h_i be the average of h_{low} and h_{up}. The iteration stops when $q_i = q$ and the difference between h_{q_0-1} and h_i is less than a threshold ε. Algorithm 1 sketches the procedure. In the algorithm, q_i is the partition number in an iteration i, h_{q_i-1} is the workload of the last partition in the iteration i and $\mid N \mid$ is the vertex number of the graph.

In our algorithm, we essentially visit all the edges in each iteration, and thus the complexity of each iteration is $O(\mid E \mid)$. Moreover, because initially $h_{low} = 1$ and $h_{up} = \mid E \mid$ and we search h in a binary fashion, it takes $O(\log \mid E \mid)$ iterations for us to find the desired h. Therefore, the total complexity of our approximation algorithm is $O(\mid E \mid \log \mid E \mid)$.

Algorithm 1 Graph Partition Algorithm

Input:
undirected graph $G = (V, E)$;
stopping threshold ε;
required partition number q;
Output:
Q: resulting partitions

```
1: set h_0 = | E | /q
2: set h_low = 1 and h_up = | E |
3: set i = 0 // the iteration number
4: repeat
5:    clear Q
6:    set start = 0; end = 0
7:    repeat
8:       estimate the workload w of the partition between
         start and end
9:       if w < h_i then
10:         end = end + 1
11:      else
12:         create a partition p between start and end, and add
            p into Q
13:         start = end + 1
14:         end = start
15:      end if
16:   until end > | N |
17:   q_i = the size of Q
18:   h_{q_i-1} = the workload of the last partition
19:   if q_i < q or (q_i = q and h_{q_i-1} << h_i) then
20:      h_up = h_i
21:   end if
22:   if q_i > q then
23:      h_low = h_i
24:   end if
25:   h_i = (h_up + h_low)/2
26:   i = i + 1
27: until q_i == q and abs(h_i - h_{q_i-1}) < ε
```

D. Parallel Modularity Based Clustering

Our parallel clustering algorithm follows the conventional divide-and-conquer strategy. In Cheong's method [15], they state this strategy works for parallel Louvain algorithm because in dividing stage the graph can be efficiently merged and in the reduction stage the graph size is several order less than the original one. Therefore, the convergence of the algorithm is assured. We first partition and distribute a single large graph among the processors using our graph partition algorithm (Algorithm 1), where we divide a graph among q processors, and each processor is assigned with a set of local vertices, and duplicate ghost vertices among the processors to improve the degree distribution of sub-graph. In this way, we recreate a sub-graph consisting of local vertices and ghost vertices. According to Lemma 1, this 1D partition can increase the accuracy of local clustering.

After that, the processors exchange ghost vertices. Each processor then conducts local clustering where both local and ghost vertices are considered. After each processor generates its local clusters, a local clustering is conducted on the root processor to obtain the final clustering result. Algorithm 2 shows the framework of our parallel clustering.

Our local clustering algorithm is similar to the sequential Louvain algorithm with involving both local and ghost vertices. The difference is that, in our local clustering, we assume the ghost vertices as read-only vertices. This means that we can

Algorithm 2 Parallel Clustering Algorithm

Input:
undirected graph $G = (V, E)$;
modularity gain threshold θ;
1: partition and distribute the input graph
2: exchange ghost vertices among processors
3: **for all** processor p_i **do**
4: local graph clustering
5: **end for**
6: **for all** processor p_i **do**
7: local graph merging
8: **end for**
9: root processor gathers local clustering results
10: do local graph clustering on root processor

change the communities of local vertices into the communities of other local vertices or ghost vertices, but we always keep the communities unchanged for ghost vertices. In this way, we use the degree information of ghost vertices to enhance the structure information of sub-graph, and increase the local clustering accuracy. The communities of ghost vertices are only changed on their host processors. Algorithm 3 shows the detailed local clustering algorithm. In the algorithm, C_u^k is the community of vertex u in iteration k. We note that, in Line 10, only the modularity gain of local vertices is calculated. But for the neighbors of local vertices, both the local vertices and the ghost vertices are considered, as shown in Line 16.

After each processor generating its local communities, we need to merge the local communities to form a new global graph. This requires each vertex has a globally unique community index. However, in the local clustering, a local vertex can be assigned a community index of other local vertices or the ghost vertices. In this case, a local vertex can have the same community index with a certain ghost vertex, and we need to distinguish them using different community indices. To this end, we assign a different unique index for those local vertices. Because the Louvain method is a heuristic clustering, we can leave these communities to global clustering for deciding whether they should be merged together or not.

IV. Experimental Results

A. Setup

In this section we present the experimental evaluation of our parallel graph clustering algorithm with respect to modularity comparison, degree distribution comparison, and scalability analysis. We tested our algorithm on the supercomputer, *Titan*, at Oak Ridge National Laboratory. The system contains 18,688 nodes with Gemini interconnect. Each node has 16-core AMD Opteron CPUs with 32 GB of RAM. Our parallel program is entirely written in C++ with MPI for parallelism. Table I shows the real-world large-scale graphs used in our study. We use the full extent of these graphs, rather than use sampling to reduce the graphs as in Cheong's work [15].

Algorithm 3 Modified Local Clustering Algorithm

Input:
$G^0 = (V^0, E^0)$: undirected graph, where V^0 contains local vertices and ghost vertices;
C^0 : initial community of G^0;
θ : modularity gain threshold;
$max_{iteration}$: maximum iteration number;
Output:
C: resulting community
Q: resulting modularity
1: $k = 0$ // k indicates the iteration number
2: **repeat**
3: **for all** $u \in$ local vertices of V^k **do**
4: set $C_u^k = u$
5: set $\sum_{in}^{C_u^k} = w_{u,u}, (u,u) \in E^k$
6: set $\sum_{tot}^{C_u^k} = w_{u,v}, (u,v) \in E^k$
7: **end for**
8: randomize the order of vertices
9: **repeat**
10: **for all** $u \in$ local vertices of V^k **do**
11: //$weight(u)$ is the total edge weights incident to u
12: $\sum_{tot}^{C_u^k} = \sum_{tot}^{C_u^k} - weight(u)$
13: //$weight(u, C_u^k)$ is the sum of edge weight from u to C_u^k, and $in(u)$ is the weight of u's self loop edge
14: $\sum_{in}^{C_u^k} = \sum_{in}^{C_u^k} - 2 * weight(u, C_u^k) - in(u)$
15: // $neighbor(u)$ contains the adjacent vertices of u
16: **for all** $v \in neighbor(u)$ **do**
17: $C_v^k = argmax(\delta Q_{C_v^k \leftarrow C_u^k})$
18: **end for**
19: $\sum_{tot}^{C_v^k} = \sum_{tot}^{C_v^k} - weight(u)$
20: $\sum_{in}^{C_v^k} = \sum_{in}^{C_v^k} - 2 * weight(u, C_v^k) - in(u)$
21: **end for**
22: **until** $\delta Q < \theta$
23: //Build a new graph
24: $V^{k+1} \leftarrow C^k$
25: $E^{k+1} \leftarrow e(C_u^k, C_v^k)$
26: $k = k + 1$
27: **until** $k \leq max_{iteration}$ **and** $\delta Q < \theta$

TABLE I: Datasets used in our experimental study.

Name	Description	#Vertices	#Edges
uk-2005 [24]	the .uk domain	39.46M	936.4M
webbase-2001 [20]	a crawl graph by WebBase	118.14M	1.01B
Orkut [20]	a Google's social networking	3.07M	225.53M
LiveJournal [20]	a virtual-community social site	5.20M	76.94M
YouTube [21]	Youtube friendship network	11.34M	29.87M
DBLP [21]	a co-authorship network from DBLP	0.31M	1.04M
Amazon [21]	frequently co-purchased products	0.33M	0.92M

Besides, in order to compare with Cheong's work, we implemented an MPI version of their algorithm that can process the full extent of each large graph in Table I. We refer this implementation as Cheong's method in the following. In order to verify if our method can achieve a more balanced workload computation, we also implemented an MPI version graph clustering algorithm using the conventional 1D partition

considering both local vertices and ghost vertices.

B. Modularity Comparison

Modularity is designed to measure the quality of graph clustering. A higher value of modularity means graphs have dense intra-community connections but sparse inter-community connections. In this section, we compare the modularity values among the sequential Louvain algorithm, Cheong's method, and our method. We use the modularity obtained by the sequential Louvain algorithm as the ground truth.

Table II shows the result of modularity comparison. The difference columns show the relative differences with respect to the ground truth. A positive (negative) difference value means a higher (lower) modularity, implying a better (worse) clustering quality. We can observe that the modularity values of our clustering results are closer to the ground truth and some of them are even higher than the ground truth. This shows that our method can achieve more accurate clustering results by considering both local vertices and ghost vertices. In original Cheong's method, inaccuracy is introduced as the ghost vertices are ignored. In our method, we can effectively improve the accuracy and make our modularity value close to or even higher than the modularity value of the sequential Louvain algorithm.

TABLE II: Modularity Comparison

Graph	Louvain	Cheong's Method		Our Method	
		modularity	difference	modularity	difference
uk-2005	0.979	0.979	0.00%	0.980	0.10%
webbase-2001	0.984	0.984	0.00%	0.985	0.11%
Orkut	0.661	0.600	-9.22%	0.660	-0.30%
LiveJournal	0.734	0.704	-4.08%	0.749	2.04%
YouTube	0.715	0.709	-0.80%	0.715	0.00%
DBLP	0.820	0.800	-2.44%	0.816	-0.49%
Amazon	0.926	0.920	-0.65%	0.925	-0.11%

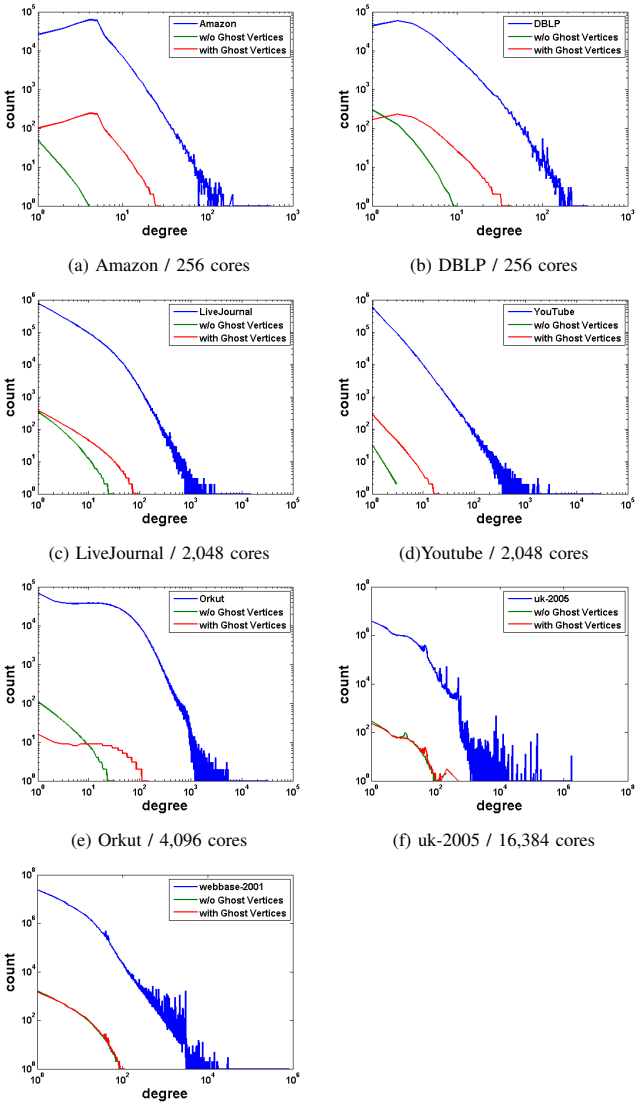
We note that for the LiveJournal and Orkut datasets, the modularity values of our method are noticeably higher than Cheong's. These two datasets correspond to dense social networks where the average degree of vertices is significantly higher than the other graphs, as indicated by the high edge/vertex ratios in Table I. This shows that the structure information, particularly degree distribution, is critical to dense graphs. Our method can well preserve graph structure information by appropriately involving ghost vertices.

C. Degree Distribution Comparison

We show the qualitative and quantitative comparisons of degree distribution for all datasets between the conventional 1D graph partition without considering ghost vertices and our graph partition with considering ghost vertices.

Figure 7 is the degree distribution comparison using different datasets and different numbers of cores[1]. Through comparison, we can find that the degree distribution with ghost vertices is more similar with the original graph degree distribution. From the figure, we can find that if we do not consider ghost vertices, there will be a degree loss for high degree vertices, which is the main difference between partition without ghost vertices and our partition. We also observe that for uk-2005 and webbase-2001, the degree distributions of two partition schemes are nearly identical. This is because the uk-2005 and webbase-2001 datasets are very large graphs with respect to the core number. Thus, even ignoring ghost vertices, the sub-graph on each core does not lose much structure information. For the other datasets, there are noticeable discrepancies between the degree distributions conveyed by

[1]Due to the page limit, we only show the maximum number of cores that we use to run our algorithm for each dataset.



(a) Amazon / 256 cores
(b) DBLP / 256 cores
(c) LiveJournal / 2,048 cores
(d)Youtube / 2,048 cores
(e) Orkut / 4,096 cores
(f) uk-2005 / 16,384 cores
(g) webbase-2001 / 16,384 cores

Fig. 7: Comparison of degree distribution using different datasets and different numbers of cores. In each plot, the blue curve corresponds to the degree distribution of the original graph; and the red and green curves correspond to the average degree distributions of sub-graphs with ghost vertices and without ghost vertices, respectively. We can see that the degree distribution of sub-graphs with ghost vertices is more similar to that of original graph.

TABLE III: D-statistics comparison.

Graph	Core=32		Core=64		Core=128		Core=256		Core=512		Core=1024		Core=2048		Core=4096		Core=8192		Core=16384	
	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost	No Ghost	Ghost
uk-2005									0.059	0.004	0.065	0.007	0.072	0.012	0.088	0.027	0.110	0.045	0.130	0.060
webbase-2001									0.026	0.002	0.027	0.040	0.029	0.006	0.032	0.008	0.037	0.012	0.045	0.016
Orkut	0.599	0.005	0.648	0.009	0.682	0.016	0.707	0.024	0.727	0.036	0.747	0.054	0.768	0.084	0.794	0.130				
LiveJournal	0.435	0.002	0.463	0.004	0.484	0.006	0.501	0.010	0.516	0.016	0.536	0.023	0.547	0.036						
YouTube	0.796	0.005	0.834	0.007	0.862	0.010	0.879	0.014	0.888	0.020	0.897	0.030	0.900	0.040						
DBLP	0.425	0.004	0.466	0.006	0.495	0.010	0.512	0.017												
Amazon	0.543	0.002	0.551	0.004	0.569	0.005	0.581	0.010												

the red and green curves.

We further use the *Kolmogorov-Smirnov* test to quantitatively compare the degree distribution between 1D graph partition without considering ghost vertices and our partition with considering ghost vertices. Kolmogorov-Smirnov test [25] is also called *D-statistic*, which relies on the fact that the value of the sample cumulative density function is asymptotically normally distributed. This is a goodness-of-fit test for any statistical distribution. In order to apply the Komologrov-Smirnov test, we need to first calculate the cumulative frequency of the observations as a function of class. We then calculate the cumulative frequency of the ground truth. The greatest discrepancy between the observed and expected cumulative frequencies is called D-statistic. A lower discrepancy means the distribution of sample is more accordance with that of the ground truth. In our case, we can easily calculate the degree distribution of original graph, and then treat the average degree distribution of sub- graph on each processor as a sample. In this way, the greatest discrepancy between two degree distributions can be calculated.

Table III shows the D-statistic comparison between partition without considering ghost vertices (the "No Ghost" columns) and our partition (the "Ghost" columns) using different numbers of cores[2]. We can clearly see that our partition method can generate a lower value of D-statistic for each dataset over different core numbers, which means our partition can generate an average degree distribution more consistent with that of the original graph. This quantitatively verifies that our method can well preserve graph structure information on a sub-graph, and thus explains that our method achieves superior clustering quality with higher modularity values than Cheong's method as shown in Table II.

D. Scalability Analysis

Although Cheong's method is less accurate, their method can relatively easily achieve nearly balanced workload among processors as ghost vertices are ignored. Therefore, in this section we focus on a comparison between parallel graph clustering using the conventional 1D partition with considering ghost vertices and our method. Figure 8 shows the detailed performance results using different datasets. In the figure, the running time is the maximum local clustering time among all processors. We can see that our method can achieve a more scalable local clustering performance compared to the conventional graph clustering using 1D partition with considering ghost vertices.

[2]For each dataset, we choose the range of core numbers according to the graph size.

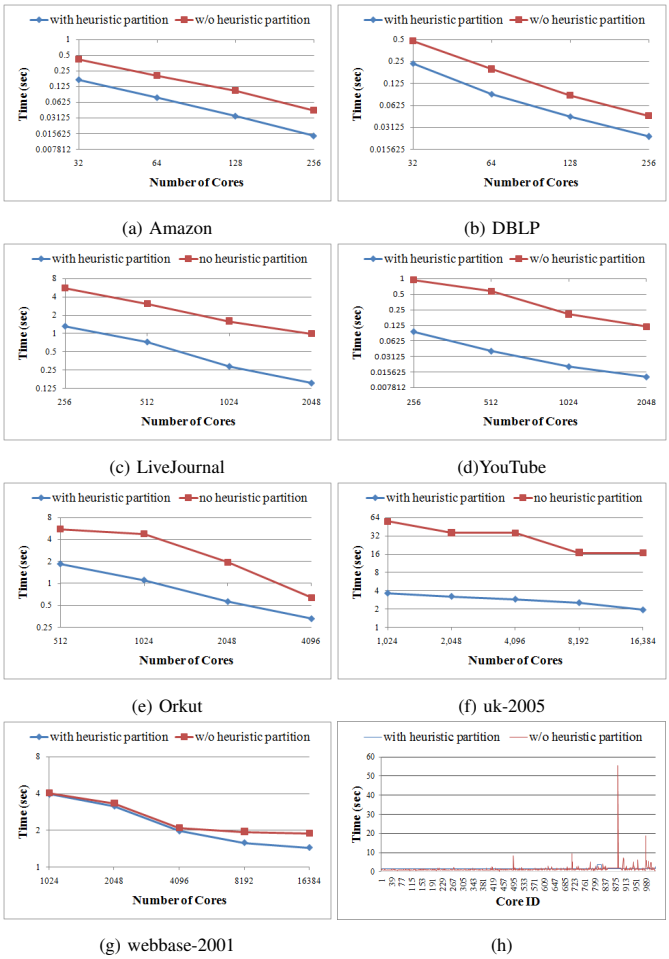


Fig. 8: (a)-(g): Scalability study using different datasets. The running time is plotted on a log-log scale. (h): The running time of each core using the uk-2005 dataset on 1,024 cores. In each plot, the red curve corresponds to the running time using the conventional 1D partition, and the blue curve corresponds to the running time using our method.

For the Amazon and DBLP datasets that are comparably small, we process them using up to 256 cores, and our method can achieve nearly 2× speedup compared to the conventional method. For the LiveJournal, Youtube and Orkut datasets, our method can achieve nearly 4× speedup. Even when using 4,096 cores for Orkut, our method can still have 2× speedup.

For two large-scale datasets uk-2005 and webbase-2001, there is a noticeable difference between the running times using the conventional method on 1,024 cores. Through a

comparison of degree distribution of these two datasets, we find that although uk-2005 has smaller vertex number and edge number than webbase-2001, the highest degree of vertex in uk-2005 (more than 10^6) is much larger than that in webbase-2001(less than 10^6), and the number of high degree vertices in uk-2005 is also larger than that in webbase-2001. This means that if only using the conventional 1D partition, it is easy to incur a highly imbalanced workload among processors. On the other hand, our heuristic partition can make cores have balanced workload, and thus can effectively decrease the maximum local clustering time. Figure 8 (h) shows the running time of each core when 1,024 cores are used with these two methods on dataset uk-2005. We can clearly see that our heuristic partition achieves a more balanced workload partition. For the webbase-2001 dataset, our speedup is less obvious compared to the conventional method. This is because this dataset has a less amount of high degree vertices, and the conventional method can also achieve a relatively balanced workload partition for 1,024 cores.

V. Conclusions

We study the relationship between the graph structure property and the clustering quality. This leads us to investigate the importance of ghost vertices in parallel clustering, which however has been overlooked in the existing approaches. Our parallel solution carefully takes ghost vertices into account, and achieves higher-quality clustering results over different types of graph datasets with a large number of processors. In particular, for dense graphs, the involvement of ghost vertices can well preserve the graph structure information in graph partitioning and distribution, which makes the results of our solution be significantly superior to the existing work. In addition, thanks to our graph partition algorithm, our solution achieve a well-balanced workload among processors, and has demonstrated a highly scalable performance using real-world graphs with over 1 billion edges on thousands of processors.

In both Cheong's approach and our approach, a root processor needs to gather all local clusters to generate the final clustering result. This step can become a performance bottleneck with increasing graph sizes. We would like to address this issue and conduct more detailed scalability study in the future. In addition, we also plan to study the acceleration techniques for our local clustering using GPUs. In this case, we expect the final aggregation stage would introduce server communication overhead. We would like to address this challenging issue in the future as well.

VI. Acknowledgment

This research has been sponsored in part by the National Science Foundation through grants IIS-1423487, and the Department of Energy through the ExaCT Center for Exascale Simulation of Combustion in Turbulence.

References

[1] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[2] A. Clauset, C. Shalizi, and M. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[3] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, no. 3, p. 036106, Sep. 2007.

[4] J. Xie and B. K. Szymanski, "Towards linear time overlapping community detection in social networks," *ArXiv e-prints*, Feb. 2012.

[5] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[6] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, no. 6, p. 066111, Dec. 2004.

[7] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 10, p. 8, Oct. 2008.

[8] E. J. Riedy, D. A. Bader, and H. Meyerhenke, "Scalable multi-threaded community detection in social networks," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 1619–1628.

[9] K. Kuzmin, S. Y. Shah, and B. K. Szymanski, "Parallel overlapping community detection with slpa," in *Social Computing (SocialCom), 2013 International Conference on*. IEEE, 2013, pp. 204–212.

[10] S. Bhowmick and S. Srinivasan, "A template for parallelizing the Louvain method for modularity maximization," in *Dynamics On and Of Complex Networks, Volume 2*. Springer, 2013, pp. 111–124.

[11] C. L. Staudt and H. Meyerhenke, "Engineering high-performance community detection heuristics for massive graphs," in *Parallel Processing, 2013 42nd International Conference on*. IEEE, 2013, pp. 180–189.

[12] H. Lu, M. Halappanavar, and A. Kalyanaraman, "Parallel heuristics for scalable community detection," *Parallel Computing*, vol. 47, pp. 19 – 37, 2015.

[13] Y. Zhang, J. Wang, Y. Wang, and L. Zhou, "Parallel community detection on large networks with propinquity dynamics," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09, 2009, pp. 997–1006.

[14] J. Soman and A. Narang, "Fast community detection algorithm with gpus and multicore architectures," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 568–579.

[15] C. Y. Cheong, H. P. Huynh, D. Lo, and R. S. M. Goh, "Hierarchical parallel algorithm for modularity-based community detection using GPUs," in *Proceedings of the 19th International Conference on Parallel Processing*, ser. Euro-Par'13, 2013, pp. 775–787.

[16] X. Que, F. Checconi, F. Petrini, T. Wang, and W. Yu, "Lightning-fast community detection in social media: A scalable implementation of the louvain algorithm," Department of Computer Science and Software Engineering, Auburn University, Tech. Rep. AU-CSSE-PASL/13-TR01, 2013.

[17] P. Pacheco, *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011.

[18] A. Buluc and K. Madduri, "Graph partitioning for scalable distributed graph computations," in *Proceedings of 10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering*, 2012.

[19] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, 2008.

[20] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, 2004, pp. 595–601.

[21] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, ser. MDS '12, 2012, pp. 3:1–3:8.

[22] A. Pınar and C. Aykanat, "Fast optimal load balancing algorithms for 1D partitioning," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 974–996, 2004.

[23] S. Miguet and J.-M. Pierson, "Heuristics for 1D rectilinear partitioning as a low cost and high quality answer to dynamic load balancing," in *High-Performance Computing and Networking*. Springer, 1997, pp. 550–564.

[24] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[25] G. W. Corder and D. I. Foreman, *Nonparametric Statistics: A Step-by-Step Approach*. Wiley, 2014.