Parallel Computing 58 (2016) 131–139

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A study of graph partitioning schemes for parallel graph community detection

Jianping Zeng, Hongfeng Yu*

University of Nebraska-Lincoln, NE 68588, United States

ARTICLE INFO

Article history:
Received 2 December 2015
Revised 6 May 2016
Accepted 15 May 2016
Available online 16 May 2016

Keywords:
Large graph
Community detection
Graph clustering
Parallel and distributed processing

ABSTRACT

This paper presents a study of graph partitioning schemes for parallel graph community detection on distributed memory machines. We investigate the relationship between graph structure and parallel clustering effectiveness, and develop a heuristic partitioning algorithm suitable for modularity-based algorithms. We demonstrate the accuracy and scalability of our approach using several real-world large graph datasets compared with state-of-the-art parallel algorithms on the Cray XK7 supercomputer at Oak Ridge National Laboratory. Given the ubiquitous graph model, we expect this high-performance solution will help lead to new insights in numerous fields.

1. Introduction

Community detection, also named as graph clustering, is a powerful technique for researchers to explore hidden patterns existing in graphs. However, it is still an open problem to design a scalable and accurate parallel community detection algorithm to tackle large graphs using a parallel machine with distributed memory. This is mainly due to the challenges in graph partitioning. First, traditional graph partitioning schemes cannot well preserve the global structure information of a graph on a processor, which can lower the cluster quality of local community detection and impair the accuracy of final aggregated results. Second, for real-world graphs (in particular scale-free graphs), it is difficult to create a balanced edge partitioning. Traditional 1D partitioning schemes often assign all the incident edges of one vertex to one processor, which can incur severe workload imbalance among processors and impair the scalability of parallel community detection. On the other hand, 2D partitioning can greatly improve the partition balance, which however can be unscalable for sparse graphs. In this case, each partition can have fewer edges than vertices, and thus is hyper-sparse [1].

In this paper, we present an experimental study of a simple graph partitioning scheme [2] designed to boost the accuracy and the scalability of parallel community detection in a typical high-performance computing environment. The scheme considers graph structure information to improve the quality of clustering, and uses a new heuristic algorithm to achieve balanced workload among processors. We preformed our tests using several real-world large graphs on the Cray XK7 supercomputer *Titan* at Oak Ridge National Laboratory (ORNL). The experimental study shows that the new graph partitioning scheme can achieve an improved accuracy of clustering that is close to the result generated by the sequential Louvain algorithm, and scale up to 16,384 cores of parallel community detection.

* Corresponding author.
E-mail addresses: jizeng@cse.unl.edu (J. Zeng), yu@cse.unl.edu (H. Yu).

http://dx.doi.org/10.1016/j.parco.2016.05.008
0167-8191/

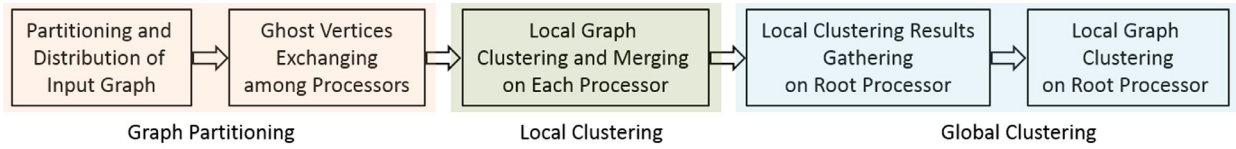

Fig. 1. The major steps of our parallel community detection approach.

2. Background

2.1. Sequential modularity-based community detection

Compared to many other sequential algorithms, modularity-based algorithms are characterized with a nearly linear time complexity and achieve a comparably higher quality of community detection. Newman and Girvan [3] first introduced the modularity measurement to quantify the quality of graph clustering, which laid the foundation of the modularity based clustering algorithms. Afterwards, Clauset et al. [4] 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. [5] proposed a heuristic method, known as the Louvain method, which can achieve a better result with a lower time complexity.

Although there is no strict definition on community or cluster of a graph, 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 [6]. Correspondingly, *modularity*, Q, is a measurement used to quantify the quality of communities detected in a graph [4], 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 Eq. (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, is the gain in modularity obtained by moving an isolated node v_i into a community C [5], which can be computed as:

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

where Σ_{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, Σ_{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.

The Louvain algorithm is designed based on the modularity measure (Eq. (1)). It is a hierarchical agglomerative clustering method in that initially each vertex is regarded as a unique community, and then communities are merged iteratively. In each iteration, multiple communities are merged into a new community to maximize the modularity gain (Eq. (2)). This process goes on until there is no modularity gain among new communities.

2.2. Parallel community detection

Although there have been some efforts in parallelizing community detection for large scale graphs based on shared memory architecture [7,8], these techniques cannot be directly applied on distributed memory machines. There is comparably limited work of parallel graph clustering on distributed memory machines. Zhang et al. [9] proposed a parallel hierarchical graph clustering method that dynamically constructed the network topology. Soman et al. [10] built a parallel label propagation algorithm (LPA) graph clustering on GPUs cluster that was limited to a marginal size graph. Cheong et al. [11] presented a GPU-based Louvain algorithm using 1D partitioning and divide-and-conquer strategy. However, their accuracy of clustering result on multiple GPUs was relatively lower than the sequential Louvain algorithm. Que et al. [12] implemented a distributed Louvain algorithm that also used 1D partitioning.

3. Our parallel community detection approach

Fig. 1 shows the basic steps of our parallel community detection approach [2], which follows the conventional divide-and-conquer strategy. Cheong et al. stated that this strategy works for the parallel Louvain algorithm because the sub-graphs generated in the dividing stage can be efficiently merged in the reduction stage, and the size of the reduced graph is several

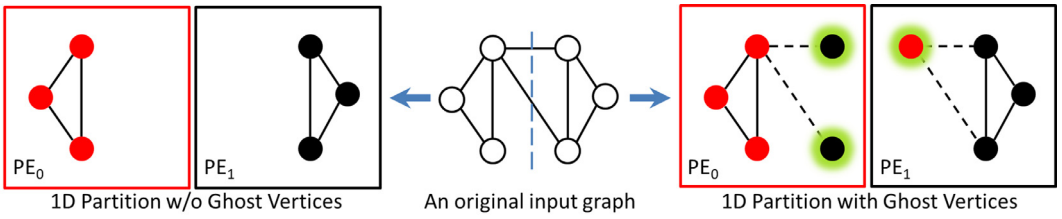

Fig. 2. An input graph is partitioned into two sub-graphs for two processors (PE_0 and PE_1) using the 1D partitioning without ghost vertices (left) and with ghost vertices (right). The sub-graphs are denoted in red or black, and the vertices circled in green are the ghost vertices. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Datasets used in our evaluation.

Name	Description	#Vertices	#Edges
uk-2005 [14]	the .uk domain	39.46M	936.4M
webbase-2001 [15]	a crawl graph by WebBase	118.14M	1.01B
Orkut [15]	a Google's social networking	3.07M	225.53M
LiveJournal [15]	a virtual-community social site	5.20M	76.94M
YouTube [16]	Youtube friendship network	11.34M	29.87M
DBLP [16]	a co-authorship network from DBLP	0.31M	1.04M
Amazon [16]	frequently co-purchased products	0.33M	0.92M
LFR Graph [17]	LFR graph generator with build-in communities	0.1M	1.6M

order less than the original one. In the reduction stage, the sequential Louvain algorithm will be applied on the reduced graph on the root processor, and the convergence of the algorithm is assured [11].

In our approach, we first partition and distribute a single large graph among the processors using a simple 1D partitioning similar to Cheong et al.'s method [11], and each processor is assigned with a sub-graph. Then we exchange ghost vertices among the processors, and add ghost vertices into the local sub-graph of a processor. In this way, we can preserve the local graph structure information more integrated compared to the existing methods.

After each processor builds a sub-graph consisting of both the local and ghost vertices, it conducts local clustering that is similar to the sequential Louvain algorithm on the sub-graph. The difference is that, in our local clustering, we assume the ghost vertices as read-only vertices. This means that we can 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 a sub-graph, and increase the local clustering accuracy. The communities of ghost vertices are only changed on their host processors. In our local clustering, only the modularity gain of local vertices is calculated; but for the neighbors of local vertices, the ghost vertices are also included.

After each processor generating its local communities, the local communities are merged to form a new global graph, and a local clustering is conducted on the root processor to obtain the final clustering result.

4. Graph partitioning scheme and experimental results

Data partitioning is the key to the scalability of a parallel algorithm [13]. We also believe that a good data partitioning can influence the accuracy of a parallel algorithm. However, these two issues have not been extensively and holistically studied for parallel community detection. The salient feature of our approach is a new parallel graph partitioning scheme to enhance both the accuracy and scalability of parallel community detection. The existing algorithms often use a simple 1D partitioning that ignores ghost vertices residing in different sub-graphs. Alternatively, our method stores the ghost vertices for each sub-graph, and uses the degrees of the ghost vertices in the input graph. Fig. 2 illustrates the difference between these two schemes.

We have experimentally studied our graph partitioning scheme. In particular, we have compared our method to Cheong's method [11] because their method achieves one of the best clustering results among the existing distributed parallel algorithms. In our study, we used 7 real world input graphs and 1 synthetic graph, which are summarized in Table 1. Compared with the datasets used in Cheong's work, we used the full extent of these graphs. We have conducted our experiments on the *Titan* supercomputer at ORNL. 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. In order to compare with Cheong's work, we have implemented an MPI version of their algorithm that can process the full extent of each large graph in Table 1. We refer this implementation as Cheong's method in the following discussion.

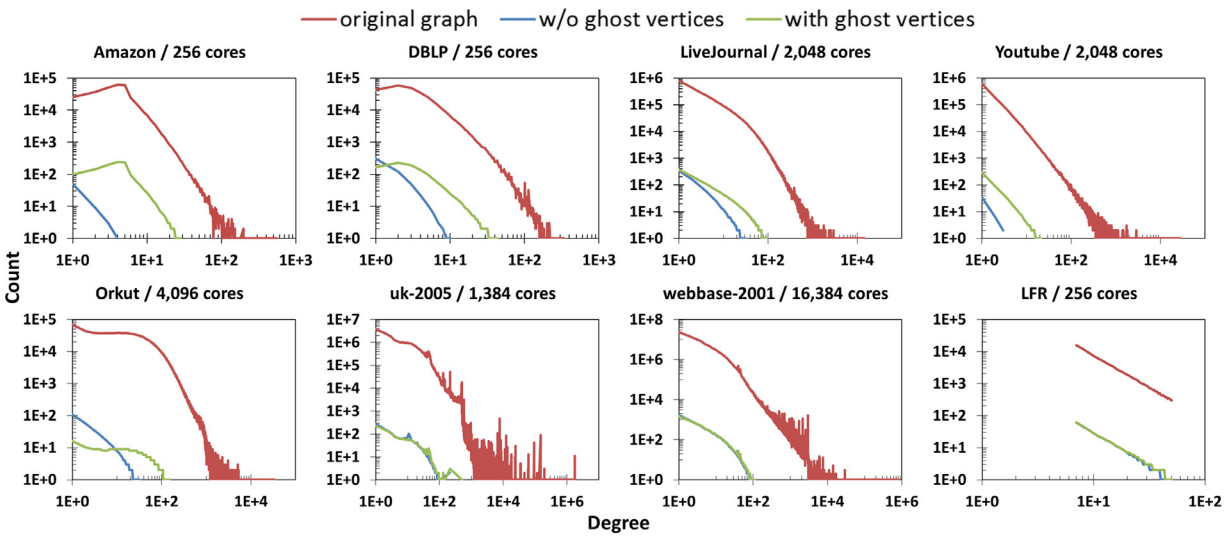

Fig. 3. Comparison of degree distribution using different datasets and different numbers of cores. In each plot, the blue curve corresponds to the degree distribution of the original graph; and the red and green curves correspond to the average degree distributions of sub-graphs with ghost vertices and 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. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4.1. Relationship between graph partitioning and clustering accuracy

In our previous work [2], we derived a lemma that shows clustering accuracy decreases without ghost vertices:

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

This lemma shows that an involvement of ghost vertices is a necessity for preserving graph structure on each processor and for improving clustering accuracy. We have verified this by evaluating the degree distribution and the community detection accuracy in our experimental results.

4.1.1. Experimental evaluation of degree distribution

According to previous research [18], we know that if the degree distribution of each sub-graph is preserved, then the other graph structure properties, such as clustering coefficient, can be preserved as well. Besides, we note that both Eqs. (1) and (2) depend on the degree information. Therefore, we have an intuition that *degree distribution* is a critical graph structure property related to the quality of graph clustering.

We show the qualitative and quantitative comparisons of degree distribution for all datasets between the conventional 1D graph partitioning without considering ghost vertices and our graph partitioning with considering ghost vertices. Fig. 3 shows a comparison of degree distribution using different datasets and different numbers of cores. We observe that the average degree distribution with ghost vertices is more similar to the original graph degree distribution. We can find that if we do not consider ghost vertices, there is a degree loss for high degree vertices in sub-graphs, which is the main difference between the conventional partitioning and our partitioning. We also observe that the degree distributions of these two partitioning schemes are nearly identical for the uk-2005 and webbase-2001 datasets. This is because these two datasets are very large graphs with respect to the core number. Compared to the other datasets, these two datasets have more vertices with the degrees larger than 10^4. For these vertices, the average degree values are similar between these two partitioning schemes. 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 the blue and green curves.

We further used the *Kolmogorov–Smirnov* test to quantitatively compare the degree distribution between 1D partitioning without considering ghost vertices and our partitioning with considering ghost vertices. The Kolmogorov–Smirnov test [19] is also called *D-statistic*, relying 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, we can calculate the greatest discrepancy between two degree distributions.

Table 2 shows the D-statistic comparison between the partitioning without considering ghost vertices (the "no ghost" columns) and our partitioning (the "ghost" columns) using different numbers of cores.[1] We can clearly see that our method can generate a lower value of D-statistic for each dataset over different core numbers, which means our partitioning 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.

4.1.2. Experimental evaluation of community detection accuracy

The degree distribution evaluation shows that the sub-graph on each processor well preserves the structure information of the original graph using our method, which theoretically can lead to superior clustering quality according to Lemma 1. In order to determine the accuracy of the communities detected by our algorithm, we have compared the modularity among the sequential Louvain algorithm, Cheong's method and our parallel algorithm. Besides, we have considered other measurements for similarity metrics between our parallel algorithm and the sequential Louvain algorithm, Finally, we have also compared the community size distribution between our algorithm and the sequential Louvain algorithm.

Modularity comparison. Modularity is designed to measure the quality of graph clustering. A higher modularity means graphs have dense intra-community connections but sparse inter-community connections. We compared the modularity among the sequential Louvain algorithm, Cheong's method, and our method.

Table 3 shows the result of modularity comparison.[2] The "difference" columns show the relative differences with respect to the Louvain algorithm. 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 Louvain algorithm and some of them are even higher than that. This shows that our method can achieve more accurate clustering results by considering both local vertices and ghost vertices. In Cheong's method, inaccuracy is introduced as the ghost vertices are ignored. Our method can effectively improve the accuracy and make the modularity values close to or even higher than those of the sequential Louvain algorithm.

We note that for the LiveJournal and Orkut datasets, the modularity values of our method are noticeably higher than Cheong's method. 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 1. 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.

We also note that our method can achieve slightly higher results than the sequential Louvain algorithm. This is mainly because of the involvement of ghost vertices in our method. On each processor, our method does not merge ghost vertices with local vertices into one community, but treats each ghost vertex as one independent community. In this way, our method can not only reduce the errors from local clustering, but also detect small clusters in a large graph, which is hard to resolve in the sequential Louvain algorithm.

Community structure comparison. We also considered several metrics to compare the similarity of the resulting community structure between the sequential Louvain algorithm and our parallel algorithm. The metrics include Normalized Mutual Information(NMI), F-measure, Normalized Van Dongen Metic (NVD), Rand Index (RI), Adjusted Rand Index (ARI) and Jaccard Index(JI). In general, a higher similarity corresponds to a lower value for NVD and higher values for the rest metrics [20]. Table 4 shows that our parallel algorithm has achieved the community detection results similar to the ones of the sequential algorithm. In particular, the most widely used measure NMI is close to 1.

Community size distribution. In order to provide a further insight of community detection between our algorithm and the sequential algorithm, we examined the detected community size distribution. Fig. 4 shows the community size distribution comparison on all datasets. We can find that the distribution of community size of our algorithm is similar to those of the sequential algorithm. This states that our parallel algorithm can guarantee the correctness of the Louvain algorithm and achieve similar community detection results.

4.2. Relationship between graph partitioning and clustering scalability

In the Louvain algorithm, to calculate the maximum modularity gain for each vertex v, each neighbor of vertex v needs to be checked, which states the overall complexity of the algorithm is proportional to the total degree of all vertices of a graph. Using the simplified 1D partitioning without ghost vertices as Cheong et al. [11] can achieve a nearly balanced workload partition, because each sub-graph has an approximately equal summation of its local vertex degrees. However, the involvement of ghost vertices increases the complexity in partitioning of a graph, because a local vertex may have a large degree and connect to many ghost vertices.

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 have derived the following lemma [2] to cope with this issue according to Eq. (2):

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

[2] We further developed a heuristic partitioning method to improve the scalability of our method. The column of our method with heuristic partitioning will be discussed in Section 4.2.

Table 2
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												
LFR	0.042	0.009	0.082	0.014	0.123	0.023	0.256	0.054												

Table 3
Modularity comparison.

Graph	Louvain	Cheong's method		Our method		Our method$_{\text{heuristic partitioning}}$	
		Modularity	Difference	Modularity	Difference	Modularity	Difference
uk-2005	0.979	0.979	0.00%	0.980	0.10%	0.980	0.10%
webbase-2001	0.984	0.984	0.00%	0.985	0.11%	0.985	0.11%
Orkut	0.661	0.600	−9.22%	0.661	0.00%	0.660	−0.30%
LiveJournal	0.734	0.704	−4.08%	0.749	2.04%	0.749	2.04%
YouTube	0.715	0.709	−0.80%	0.719	0.56%	0.715	0.00%
DBLP	0.820	0.800	−2.44%	0.818	−0.24%	0.816	−0.49%
Amazon	0.926	0.920	−0.65%	0.926	0.00%	0.925	−0.11%
LFR	0.608	0.604	−0.66%	0.608	0.00%	0.608	0.00%

Table 4
Community structure comparison.

Dataset	NMI	F-measure	NVD	RI	ARI	JI
Amazon	0.9738	0.8133	0.1480	0.9989	0.6675	0.5123
LFR Graph	0.9933	0.9144	0.0399	0.9999	0.9425	0.8611

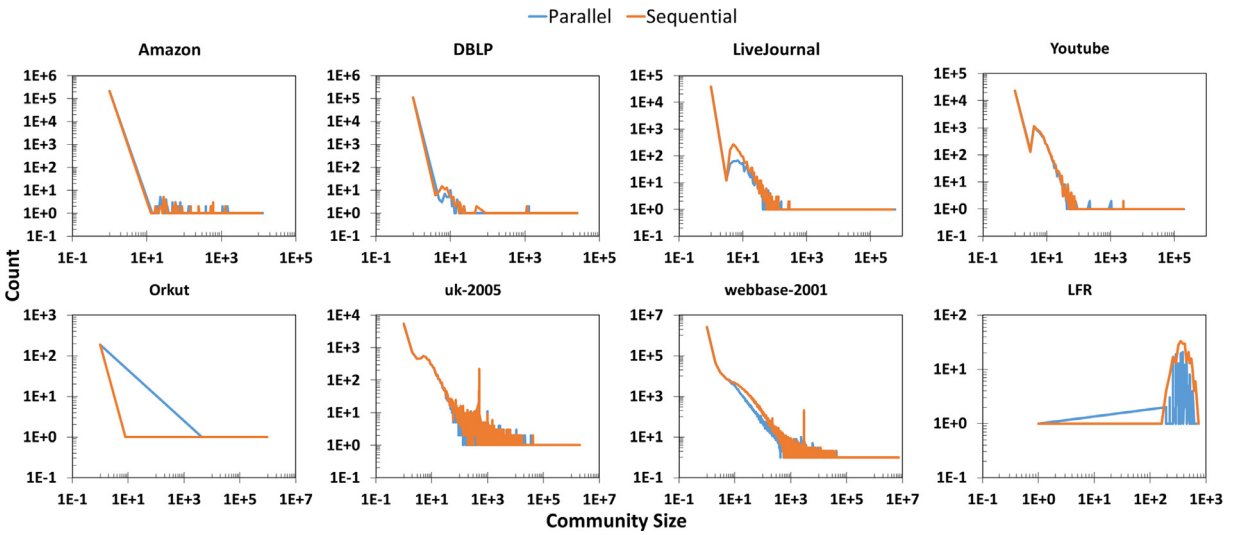

Fig. 4. Comparison of community size distribution using different datasets between our parallel algorithm and the sequential algorithm.

Lemma 2. *Given an sub-graph on a processor p_i, we have a set of ghost vertices $v_{g_1}, v_{g_2}, \ldots, v_{g_n}$, which only connect the same local vertex v_l. Assume v_{g_a} has the minimal degree among the ghost vertices, then in the final clustering result of this sub-graph, there exists a ghost vertex v_{g_b} where the community of v_l is not equal to the community of v_{g_b} and $v_{g_b} \neq v_{g_a}$.*

Lemma 2 allows us to prune a considerable amount of ghost vertices for one sub-graph. Based on this lemma, we have developed a heuristic partitioning algorithm [2] to partition a graph into a set of sub-graphs, and each sub-graph is associated with an approximately equal amount of workload. The total complexity of our heuristic algorithm is $O(|E|\log|E|)$, where $|E|$ is the total number of edges of the input graph. Table 3 shows the modularity values of our method with heuristic partitioning, where we can see that the modularity values are unchanged or slightly decreased. We have further verified that the application of Lemma 2 can improve parallel scalability and workload balancing of our method.

4.2.1. Experimental evaluation of parallel scalability

Fig. 5 shows the detailed performance results of graph clustering using our method with heuristic partitioning, our method without heuristic partitioning, and Cheong's method on different datasets. In Fig. 5, the running time of each test is the maximum local clustering time among all processors.

We first compared our method with and without heuristic partitioning, and observed that heuristic partitioning can help our method achieve a more scalable local clustering performance. For the Amazon and DBLP datasets that are comparably small, we processed them using up to 256 cores, and our method can achieve nearly 2 × speedup if enabling heuristic partitioning. For the LiveJournal, Youtube and Orkut datasets, heuristic partitioning can help our method achieve nearly 4 × speedup. Even when using 4096 cores for Orkut, our method can still have 2 × speedup using heuristic partitioning.

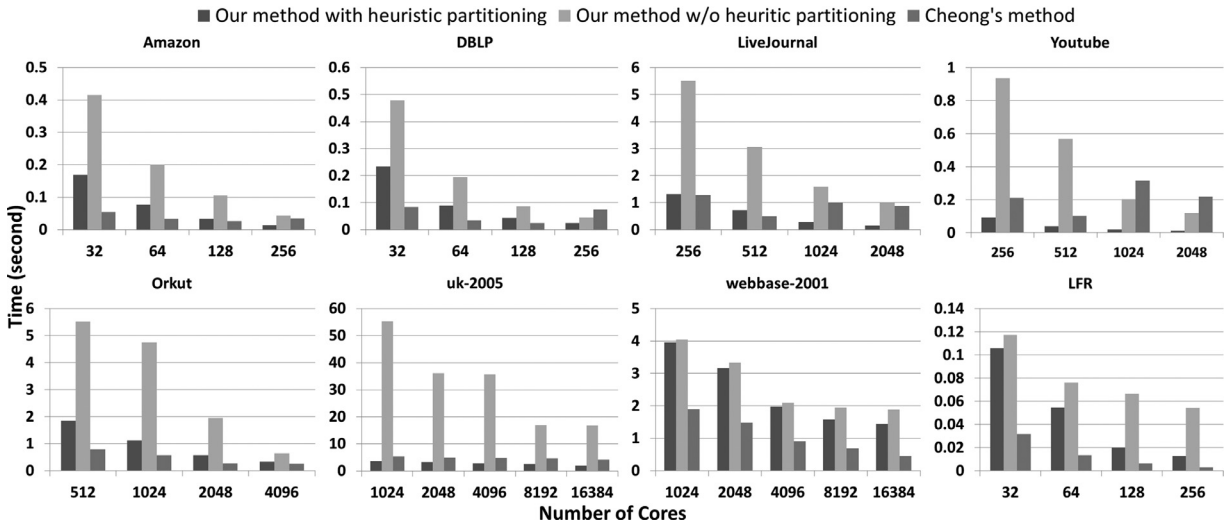

Fig. 5. Scalability study using different datasets.

For the two large-scale datasets uk-2005 and webbase-2001, there is a noticeable difference between the running times without heuristic partitioning on 1024 cores. Through a comparison of degree distribution of these two datasets, we found that although uk-2005 has a smaller vertex number and edge number than webbase-2001, the highest vertex degree 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 without heuristic partitioning, it is easy to incur a highly imbalanced workload among processors. For the webbase-2001 dataset, the difference between our method with and without heuristic partitioning is less obvious. This is because this dataset has a less amount of high degree vertices, and our method without heuristic partitioning can also achieve a relatively balanced workload assignment.

We also compared our method with heuristic partitioning and Cheong's method. We observed several interesting phenomena. From our initial inference, Cheong's method ignores ghost vertices and can achieve a relatively balanced workload among processors, and correspondingly a higher scalability. This has been illustrated by the results of the Orkut, webbase-2001 and LFR datasets in Fig. 5. However, we can find that for the Amazon, DBLP, LiveJournal and Youtube datasets, the running time of Cheong's method increases with more cores used. This shows that only reducing the edges incident to the ghost vertices cannot make the parallel Louvain algorithm scalable. We think this is because the Louvain algorithm detects communities based on graph structure, and reducing the edges incident to the ghost vertices can make the algorithm use more iterations to converge, thereby increasing the running time. On the other hand, the running time of our method decreases with the increasing number of cores. Thus, for the Amazon, DBLP, LiveJournal, Youtube and uk-2005 datasets, our method is more scalable than Cheong's method. We think the reason is that our partitioning can well preserve global graph structure information on sub-graphs (Section 4.1), and our local clustering can converge using a much fewer number of iterations than Cheong's method that neglects ghost vertices. The comparison shows that our method can achieve better scalability and more accurate results on these datasets.

4.2.2. Experimental evaluation of workload balancing

To explore more details about workload on each core, we have plotted the running time of each core for each dataset partitioned on the different numbers of cores, as shown in Fig. 6. We can see that for the Amazon, LFR and uk-2005 datasets, our heuristic partitioning method can make each core have a balanced workload than the method without heuristic partitioning. Especially for the uk-2005 dataset, our heuristic partitioning can effectively decrease the maximum local clustering time, while, without heuristic partitioning, an extremely high workload is observed on one particular core. For the DBLP, LiveJournal and YouTube datasets, our partitioning method makes each core have similar workload while the conventional method makes each core have highly varied workload, and thus our local clustering with heuristic partitioning can achieve a much shorter running time. For the Orkut and webbase-2001 datasets, although most cores have higher workload with heuristic partitioning, we note that one or two cores in 1D partitioning without heuristic strategy can have much higher local computing time, while our heuristic partitioning has distributed this heavy load to all of the cores.

5. Conclusion

Our graph partitioning scheme directly accounts for graph structure through ghost vertices, and leads to scalable and higher-quality parallel graph clustering results with large-scale graphs on massively parallel machines. Our evaluation study, which has used up to 16,384 cores of Titan at ORNL and more than 1 billion edges of graphs, demonstrates convincing

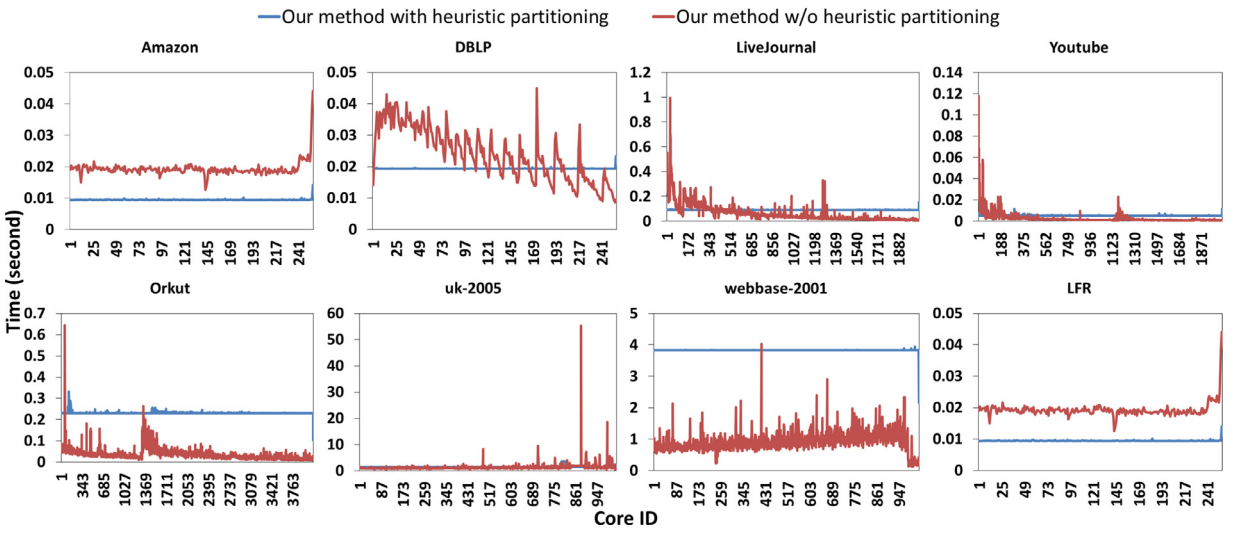

Fig. 6. Comparison of workload on each core using different datasets.

results, and reveals the relationship between graph partitioning schemes and parallel clustering quality and scalability. Although this paper presents our experimental study of graph partitioning schemes for parallel community detection, the accuracy and scalability challenges are common in the context of other types of large-scale graph applications.

Acknowledgments

This research has been sponsored in part by the National Science Foundation through grants IIS-1423487 and ICER-1541043, and the Department of Energy through the ExaCT Center for Exascale Simulation of Combustion in Turbulence.

References

[1] A. Buluç, J.R. Gilbert, On the representation and multiplication of hypersparse matrices, in: IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, IEEE, 2008, pp. 1–11.
[2] J. Zeng, H. Yu, Parallel modularity-based community detection on large-scale graphs, in: 2015 IEEE International Conference on Cluster Computing (CLUSTER), 2015, pp. 1–10.
[3] M.E. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[4] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004) 066111.
[5] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. 10 (2008) 8.
[6] S. Fortunato, Community detection in graphs, Phys. Rep. 486 (3-5) (2010) 75–174.
[7] E.J. Riedy, D.A. Bader, H. Meyerhenke, Scalable multi-threaded community detection in social networks, in: 2012 IEEE 26th International on Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE, 2012, pp. 1619–1628.
[8] H. Lu, M. Halappanavar, A. Kalyanaraman, Parallel heuristics for scalable community detection, Parallel Comput. 47 (2015) 19–37.
[9] Y. Zhang, J. Wang, Y. Wang, 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, KDD '09, 2009, pp. 997–1006.
[10] J. Soman, A. Narang, Fast community detection algorithm with GPUs and multicore architectures, in: 2011 IEEE International on Parallel Distributed Processing Symposium (IPDPS), 2011, pp. 568–579.
[11] C.Y. Cheong, H.P. Huynh, D. Lo, R.S.M. Goh, Hierarchical parallel algorithm for modularity-based community detection using GPUs, in: Proceedings of the 19th International Conference on Parallel Processing, Euro-Par'13, 2013, pp. 775–787.
[12] X. Que, F. Checconi, F. Petrini, J.A. Gunnels, Scalable community detection with the louvain algorithm, in: 2015 IEEE International on Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2015, pp. 28–37.
[13] P. Pacheco, An Introduction to Parallel Programming, Morgan Kaufmann, 2011.
[14] P. Boldi, B. Codenotti, M. Santini, S. Vigna, Ubicrawler: a scalable fully distributed web crawler, Software 34 (8) (2004) 711–726.
[15] P. Boldi, S. Vigna, The WebGraph framework I: compression techniques, in: Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), 2004, pp. 595–601.
[16] J. Yang, J. Leskovec, Defining and evaluating network communities based on ground-truth, in: Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics, MDS '12, 2012, pp. 3:1–3:8.
[17] A. Lancichinetti, S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities, Phys. Rev. E 80 (1) (2009) 016118.
[18] C. Hübler, H.-P. Kriegel, K. Borgwardt, Z. Ghahramani, Metropolis algorithms for representative subgraph sampling, in: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, 2008.
[19] G.W. Corder, D.I. Foreman, Nonparametric Statistics: a Step-by-Step Approach, Wiley, 2014.
[20] J. Xie, S. Kelley, B.K. Szymanski, Overlapping community detection in networks: the state-of-the-art and comparative study, ACM Comput. Surv. 45 (4) (2013) 43:1–43:35.