Parallel Toolkit for Measuring the Quality of Network Community Structure

Mingming Chen  
Department of Computer Science  
Rensselaer Polytechnic Institute  
110 8th Street, Troy, NY 12180  
Email: chennm8@rpi.edu

Sisi Liu  
Information Technology and Web Science  
Rensselaer Polytechnic Institute  
110 8th Street, Troy, NY 12180  
Email: lius10@rpi.edu

Boleslaw K. Szymanski  
Department of Computer Science  
Rensselaer Polytechnic Institute  
110 8th Street, Troy, NY 12180  
Email: szymab@rpi.edu

Abstract—Many networks display community structure which identifies groups of nodes within which connections are denser than between them. Detecting and characterizing such community structure, which is known as community detection, is one of the fundamental issues in the study of network systems. It has received considerable attention in the last years. Numerous techniques have been developed for both efficient and effective community detection. Among them, the most efficient algorithm is the label propagation algorithm whose computational complexity is $O(|E|)$. Although it is linear in the number of edges, the running time is still too long for very large networks, creating the need for parallel community detection. Also, computing community quality metrics for community structure is computationally expensive both with and without ground truth. However, to date we are not aware of any effort that provides parallel computation for the community quality metrics with and without ground truth community structure [19], [20], [21], though it is computational expensive to do so. Hence, in this paper, we provide a parallel toolkit to calculate the values of these metrics. Although we are using parallel computing to speed up the processing, in most of the cases, algorithms are highly parallelizable, so the contributions of this paper focus on making the highly efficient social network analysis tools available to research community.

We implement the parallel algorithms with MPI (Message Passing Interface) and Pthreads (POSIX Threads). We perform runs on both distributed memory machine, such as Blue Gene/Q, and shared memory machine, like GANXIS. The network we adopt is LFR benchmark network [22]. The LFR benchmark network for testing the parallel programs to calculate the metrics with ground truth community structure has 100,000 nodes. We choose two sizes, ten million of nodes (10,000,000) and one hundred million of nodes (100,000,000), of the LFR benchmark network to test the parallel programs for computing the metrics without ground truth communities. The experimental results show that both parallel MPI algorithms and parallel Pthreads algorithms yield a significant performance gain over sequential execution. Moreover, we recommend using parallel MPI algorithms and parallel Pthreads algorithm respectively to calculate the metrics with and without ground truth communities on GANXIS (or shared memory machines) in terms of their speedup and efficiency.

I. INTRODUCTION

Many networks, including Internet, citation networks, transportation networks, e-mail networks, and social and biochemical networks, display community structure which identifies groups of nodes within which connections are denser than between them [1]. Detecting and characterizing such community structure, which is known as community detection, is one of the fundamental issues in the study of network systems. Community detection has been shown to reveal latent yet meaningful structures in networks [2].

Thus, numerous techniques were developed for both efficient and effective community detection, including Modularity Optimization [3], [4], Clique Percolation [5], Local Expansion [6], [7], Fuzzy Clustering [8], [9], Link Partitioning [10], and Label Propagation [11], [12], [13]. Among them, the most efficient algorithm is the label propagation algorithm whose computational complexity is $O(|E|)$, where $|E|$ is the number of edges in the network. Although it is a linear algorithm, the running time is still too long for very large networks. The primary examples are online social networks that are increasingly popular, the largest being Facebook with more than 800 million daily active users.$^2$ The WWW forms a network of hyperlinked webpages in excess of 30 billion nodes. Therefore, parallelism was introduced into community detection to alleviate computational costs [14], [15], [16], [17], [18]. However, to date we are not aware of any effort that provides parallel computation for the community quality metrics with and without ground truth community structure [19], [20], [21], though it is computational expensive to do so. Hence, in this paper, we provide a parallel toolkit to calculate the values of these metrics. Although we are using parallel computing to speed up the processing, in most of the cases, algorithms are highly parallelizable, so the contributions of this paper focus on making the highly efficient social network analysis tools available to research community.

II. COMMUNITY QUALITY METRICS

A. Metrics with Ground Truth Communities

The quality evaluation metrics with ground truth community structure we consider here can be divided into three categories: Variation of Information (VI) and Normalized Mutual Information (NMI) based on information theory; $F$-measure and Normalized Van Dongen metric (NVD) based on cluster matching; Rand Index (RI), Adjusted Rand Index (ARI), and Jaccard Index (JI) based on pair counting [19].

1) Information Theory Based Metrics: Given partitions $C$ and $C'$, $VI$ quantifies the “distance” between those two partitions, while $NMI$ measures the similarity between them.

---

$^1$Please contact Mingming Chen via mileschen2008@gmail.com for the parallel toolkit if you are interested in it.

$^2$Facebook company info: http://newsroom.fb.com/company-info/
$VI$ is defined as

$$VI(C, C') = -\frac{1}{|V|} \sum_{e \in C, e' \in C'} |c \cap e'| \log \left( \frac{|c \cap e'|^2}{|c||e'|} \right), \tag{1}$$

and $NMI$ is given by

$$NMI(C, C') = -\frac{2}{|V|} \sum_{e \in C, e' \in C'} \frac{|c \cap e'| |c' \cap e'|}{|c||c'| |e||e'|} \log \frac{|c \cap e'| |c' \cap e'|}{|c||c'| |e||e'|}, \tag{2}$$

where $|V|$ is the number of nodes in the network, $|c|$ is the number of nodes in community $c$ of $C$, and $|c \cap e'|$ is the number of nodes both in community $c$ of $C$ and in community $e'$ of $C'$. The computational complexity to calculate $VI$ and $NMI$ is $O(|V||C||C'|)$, where $|C'|$ is the number of communities found by a community detection algorithm.

2) Cluster Matching Based Metrics: Measures based on cluster matching aim at finding the largest overlaps between pairs of communities of two partitions $C$ and $C'$. $F$-measure measures the similarity between two partitions, while $NVD$ quantifies the “distance” between partitions $C$ and $C'$. $F$-measure is defined as

$$F\text{-measure}(C, C') = \frac{1}{|V|} \sum_{e \in C} \max_{e' \in C'} \frac{2|c \cap e'|}{|c||e'|}. \tag{3}$$

$NVD$ is given by

$$NVD(C, C') = 1 - \frac{1}{2|V|} \left( \sum_{c \in C} \max_{e' \in C'} |c' \cap e'| + \sum_{c' \in C'} \max_{c \in C} |c \cap c'| \right). \tag{4}$$

The complexity to calculate $F$-measure and $NVD$ is $O(|V||C|+|C'|)$, where $|C|$ is the number of communities in the ground truth community structure.

3) Pair Counting Based Metrics: Metrics based on pair counting count the number of pairs of nodes that are classified (in the same community or in different communities) in two partitions $C$ and $C'$. Let $a_{11}$ indicate the number of pairs of nodes that are in the same community in both partitions, $a_{10}$ denote the number of pairs of nodes that are in the same community in $C$ but in different communities in $C'$, $a_{01}$ be the number of pairs of nodes which are in different communities in $C$ but in the same community in $C'$, and $a_{00}$ be the number of pairs of nodes which are in different communities in both partitions. By definition, $A = a_{11} + a_{10} + a_{01} + a_{00} = |V||V|-|E|$ is the total number of pairs of nodes in the network. Then, $RI$ which is the ratio of the number of node pairs placed in the same way in both partitions to the total number of pairs is given by

$$RI(C, C') = \frac{a_{11} + a_{00}}{A}. \tag{5}$$

Denote $M = \frac{1}{2}(a_{11} + a_{10})(a_{11} + a_{01})$. Then, $RI$’s corresponding adjusted version, $ARI$, is expressed as

$$ARI(C, C') = \frac{a_{11} - M}{\frac{1}{2}[(a_{11} + a_{10}) + (a_{11} + a_{01})] - M}. \tag{6}$$

$JI$ which is the ratio of the number of node pairs placed in the same community in both partitions to the number of node pairs that are placed in the same group in at least one partition is defined as

$$JI(C, C') = \frac{a_{11}}{a_{11} + a_{10} + a_{01}}. \tag{7}$$

Each of these three metrics quantifies the similarity between two partitions $C$ and $C'$. The complexity to calculate $RI$, $ARI$, and $JI$ is $O(|V|^2)$.

B. Metrics without Ground Truth Communities

1) Newman’s Modularity: Modularity $Q$ measures the difference between the actual fraction of edges within the community and such fraction expected in a randomized graph with the same number of nodes and the same degree sequence. For the given community partition of a weighted and undirected graph $G=(V, E)$ with $|E|$ edges, modularity ($Q$) is given by

$$Q = \sum_{c \subseteq C} \left[ \frac{|E_{c}^{in}|}{|E|} - \frac{|2|E_{c}^{in}| + |E_{c}^{out}|}{2|E|} \right]^2, \tag{8}$$

where $C$ is the set of all the communities, $c$ is a specific community in $C$, $|E_{c}^{in}|$ is the number of edges between nodes within community $c$, and $|E_{c}^{out}|$ is the number of edges from the nodes in community $c$ to the nodes outside $c$.

2) Modularity Density: Modularity Density ($Q_{ds}$) [20], [21] is proposed to solve the two opposite yet coexisting problems of modularity: in some cases, it tends to favor small communities over large ones while in others, large communities over small ones. The latter tendency is also known as the resolution limit problem [23]. For unweighted and undirected networks, $Q_{ds}$ is defined as

$$Q_{ds} = \sum_{c, c' \subseteq C} \left[ \frac{|E_{c}^{in}|}{|E|} - \frac{|2|E_{c}^{in}| + |E_{c}^{out}|}{2|E|} \right]^2,$$

$$d_{c_{i}, c_{j}} = \left\lceil \frac{2|E_{c_{i}}^{in}|}{|c_{i}|(|c_{j}| - 1)} \right\rceil,$$

$$d_{c_{i}, c_{j}} = \frac{|E_{c_{i}}^{in}|}{|c_{i}| |c_{j}|}.$$
III. PARALLEL ALGORITHM DESIGN

In this section, we present the parallel algorithms, MPI and Pthreads versions, to calculate the quality metrics introduced in Section II. It can be seen from Section II that VI and NMI can be calculated together, F-measure and NVD can be computed together, RI, ARI, and JI can be calculated together, and the metrics without ground truth communities can be computed together. Thus, we will have four parallel algorithms based on MPI and four parallel algorithms based on Pthreads for these metrics. We denote the ground truth community structure as $C$ and the community structure detected with a community detection algorithm as $C'$.

A. Parallel Algorithms Based on MPI

In the parallel algorithms based on MPI, the problem to calculate metrics is partitioned with the unit of community. For the algorithms to calculate the metrics with ground truth community structure, each processor extracts the ground truth communities and the detected communities when $(\text{comId} \mod \text{numProcs} == \text{procId})$ to achieve rough load balance. comId is the id of a community, procId is the id of a processor, and numProcs is the total number of processors used. Hence, each processor will have $\lfloor |C|/\text{numProcs} \rfloor$ or $\lfloor |C|/\text{numProcs} + 1 \rfloor$ ground truth communities and $\lfloor |C'|/\text{numProcs} \rfloor$ or $\lfloor |C'|/\text{numProcs} + 1 \rfloor$ discovered communities. For the algorithms to compute the metrics without ground truth communities, each processor extracts the discovered communities using the same approach. Also, each processor gets its local network which contains the nodes in its own communities and the neighboring nodes of these nodes.

Algorithm 1 MPI_information_theory_metric($C_p$, $C_p'$)

1: // Processor id is denoted as $p$. The number of processors used is denoted as $\text{numProcs}$. The local values of $VI$ and $NMI$ is denoted as $\text{rankVI}$ and $\text{rankNMI}$.
2: Calculate $\text{rankVI}$ and $\text{rankNMI}$ based on Equation (1) and Equation (2) using its own $C_p$ and its own $C_p'$.
3: // Circulate $C_p$ in a ring.
4: $\text{recvSrc} = (p + \text{numProcs} - 1) \mod \text{numProcs}$;
5: $\text{sendDst} = (p + 1) \mod \text{numProcs}$;
6: $\text{receivedMsgNum} = 0$;
7: while $\text{receivedMsgNum} < (\text{numProcs} - 1)$ do
8: $\text{sendBuf}[\cdot] \leftarrow C_p'$;
9: Send $\text{sendBuf}[\cdot]$ to $\text{sendDst}$;
10: Receive $C_p'$ from $\text{recvSrc}$ with $\text{recvBuf}[\cdot]$;
11: Calculate $\text{rankVI}$ and $\text{rankNMI}$ based on Equations (1) and (2) using its own $C_p$ and received $C_p'$;
12: $\text{++receivedMsgNum}$;
13: // Circulate $C_p$ in a ring.
14: $\text{receivedMsgNum} = 0$;
15: while $\text{receivedMsgNum} < (\text{numProcs} - 1)$ do
16: $\text{sendBuf}[\cdot] \leftarrow C_p$;
17: Send $\text{sendBuf}[\cdot]$ to $\text{sendDst}$;
18: Receive $C_p$ from $\text{recvSrc}$ with $\text{recvBuf}[\cdot]$;
19: // Circulate $C_p'$ in a ring.
20: $\text{receivedMsgNum} = 0$;
21: Calculation $\text{rankNMI}$ and $\text{rankNVD}$ for each community in $C_p'$
22: Algorithm 2 MPI_cluster_matching_metric($C_p$, $C_p'$)
23: 1: // Use $\text{maxNormedComs}[]$ and $\text{maxTComs}[]$ to record the max item for each ground truth community shown in Equation (3) and (4), respectively.
24: 2: Get $\text{maxNormedComs}[]$ and $\text{maxTComs}[]$ for each community in $C_p'$ with its own set of ground truth communities $C_p$;
25: 3: $\text{recvSrc} = (p + \text{numProcs} - 1) \mod \text{numProcs}$;
26: 4: $\text{sendDst} = (p + 1) \mod \text{numProcs}$;
27: 5: // Circulate $C_p$ in a ring.
28: $\text{receivedMsgNum} = 0$;
29: while $\text{receivedMsgNum} < (\text{numProcs} - 1)$ do
30: $\text{sendBuf}[\cdot] \leftarrow C_p'$;
31: Send $\text{sendBuf}[\cdot]$ to $\text{sendDst}$;
32: Receive $C_p'$ from $\text{recvSrc}$ with $\text{recvBuf}[\cdot]$;
33: Update $\text{maxNormedComs}[]$ and $\text{maxTComs}[]$ for each community in $C_p'$ with received $C_p'$;
34: $\text{++receivedMsgNum}$;
35: end while
36: // Use $\text{maxDComs}[]$ to record the max item for each discovered community shown in Equation (4).
37: Get $\text{maxDComs}[]$ for each community in $C_p'$ with its own set of ground truth communities $C_p$;
38: // Circulate $C_p$ in a ring.
39: $\text{receivedMsgNum} = 0$;
40: while $\text{receivedMsgNum} < (\text{numProcs} - 1)$ do
41: $\text{sendBuf}[\cdot] \leftarrow C_p$;
42: Send $\text{sendBuf}[\cdot]$ to $\text{sendDst}$;
43: Receive $C_p$ from $\text{recvSrc}$ with $\text{recvBuf}[\cdot]$;
44: Update $\text{maxDComs}[]$ for each community in $C_p'$ with received $C_p'$;
45: $\text{++receivedMsgNum}$;
46: end while
47: Calculate $\text{rankFMeasure}$ and also $\text{rankNVD}$ with $\text{maxNormedComs}[]$, $\text{maxTComs}[]$, and $\text{maxDComs}[]$ based on Equations (3) and (4);
48: Get $F$-measure and NVD by summation $\text{rankFMeasure}$ and $\text{rankNVD}$ of all $\text{numProcs}$ processors;
49: Return $F$-measure and NVD;

We first show the parallel MPI algorithm to calculate the information theory based metrics, $VI$ and $NMI$. Supposed there are $N$ processors (or MPI ranks), each processor reads its own set of ground truth communities $C_p$ and its own set of discovered communities $C_p'$. It can be learnt from the definitions of $VI$ and $NMI$ given by Equation (1) and Equation (2) respectively that each ground truth community should traverse all the discovered communities in order to get the values of them. Thus, in the algorithm, we circulate $C_p'$ to each processor in a “ring”. That is, processor 0 sends its $C_p'$ to processor 1, and processor 1 to processor 2, and processor $N - 1$ would send $C_p'$ to processor 0. This “shifting” of $C_p'$ will occur $N - 1$ times for $N$ processors. For its own $C_p'$ and each received $C_p''$, the processor will calculate its local values of $VI$ and $NMI$ with its own $C_p''$. Finally, the values of $VI$ and $NMI$ are the sum of the local values of all $N$ processors. Algorithm 1 shows our parallel algorithm for computing $VI$ and $NMI$ based on MPI. It takes $C_p$ and $C_p'$ as parameters.
both \( C_p \) and \( C_p' \) are circulated to each processor in a “ring”. This “shifting” of \( C_p' \) and \( C_p \) will both occur \( N-1 \) times for \( N \) processors. For its own \( C_p' \) and each received \( C_p \), the processor will calculate its local values of F-measure and NVD with its own \( C_p \). Moreover, for its own \( C_p \) and each received \( C_p' \), the processor will compute its local values of NVD with its own \( C_p' \). Finally, the values of F-measure and NVD are the sum of the local values of all \( N \) processors. Algorithm 2 shows our parallel MPI algorithm for computing F-measure and NVD. It takes \( C_p \) and \( C_p' \) as parameters.

### Algorithm 3: MPI_pair_counting_metric\((C_p(map), C_p'(map))\)

1. Count \( rankA11, rankA10, rankA01, \) and \( rankA00 \) using its own \( C_p(map) \) and its own \( C_p'(map) \);
2. // Circulate \( C_p'(map) \) in a ring.
3. \( receivedSrc = (p + numProcs - 1) \mod numProcs; \)
4. \( sendDst = (p + 1) \mod numProcs; \)
5. \( receivedMsgNum = 0; \)
6. while \( receivedMsgNum < (numProcs - 1) \) do
   7. \( sendBuf[i] \leftarrow C_p'(map); \)
   8. Send \( sendBuf[i] \) to \( sendDst; \)
   9. Receive \( C_p''(map) \) from \( receivedSrc \) with \( receivedBuf; \)
10. Count \( rankA01 \) and \( rankA00 \) using its own \( C_p(map) \) and received \( C_p'(map) \);
11. \( ++ receivedMsgNum; \)
12. end while
13. Calculate \( rankRI, rankARI, \) and \( rankJI \) with \( rankA11, rankA10, rankA01, \) and \( rankA00; \)
14. Get \( RI, ARI, \) and \( JI \) by summing the \( rankRI, rankARI, \) and \( rankJI \) of all \( numProcs \) processors;
15. Return \( RI, ARI, \) and \( JI; \)

We now demonstrate how to calculate the pair counting based metrics, \( RI, ARI, \) and \( JI, \) in parallel with MPI. To calculate \( RI, ARI, \) and \( JI, \) each node in the network needs to traverse all the other nodes so as to get \( a_{11}, a_{10}, a_{01}, \) and \( a_{00} \). Therefore, each processor reads its own ground truth communities and saves as a map with key being the node id and value being the id of the ground truth community to which this node belongs. We denote the map of nodes with their communities from ground truth community structure as \( C_p(map) \). This processor also reads the community information for the nodes in \( C_p(map) \) from discovered community structure and saves as a map. We denote the map of nodes with their communities from discovered community structure as \( C_p'(map) \). \( C_p(map) \) and \( C_p'(map) \) have the same subset of nodes but with their community information from ground truth community structure and detected community structure, respectively. In our implementation, we use hash indexed map in order to search the community that a node belongs to quickly. Since each node needs to traverse all the other nodes, thus in the algorithm \( C_p'(map) \) is circulated to each processor in a “ring”. This “shifting” of \( C_p'(map) \) will also occur \( N-1 \) times for \( N \) processors. For each processor, each node in \( C_p(map) \) first traverses the other nodes in its own \( C_p'(map) \) to count its local values of \( a_{11}, a_{10}, a_{01}, \) and \( a_{00} \). Then, this node will traverse the nodes in received \( C_p'(map) \) to count only \( a_{01} \) and \( a_{00} \) because this node is in a different ground truth community with the nodes in received \( C_p'(map) \). With \( a_{11}, a_{10}, a_{01}, \) and \( a_{00} \), the processor could get the local values of \( RI, ARI, \) and \( JI \) based on Equations (5), (6), and (7).

Finally, the values of \( RI, ARI, \) and \( JI \) are the sum of the local values of all \( N \) processors. Algorithm 3 shows our parallel MPI algorithm for computing \( RI, ARI, \) and \( JI \). It takes \( C_p(map) \) and \( C_p'(map) \) as parameters.

Finally, we illustrate how to calculate the community quality metrics without ground truth community structure, such as modularity and Modularity Density, in parallel with MPI. From Section II-B, we could observe that in order to calculate the contribution of a community to these metrics, we only need to obtain the number of edges inside it, the number of edges on the boundary of it, the numbers of edges between it and its neighboring communities, and the sizes of it and its neighboring communities. There is no dependency between processors. Hence, there is no need to transfer communities or to transfer any message between processors. In the algorithm, each processor reads its own set of discovered communities \( C_p' \) and its local network, and then calculate its own part for these metrics. At last, the values of these metrics are the sum of the local values of all \( N \) processors. We will not show the outline of this algorithm here because of its simplicity.

### Algorithm 4: Pthreads_pair_counting_metric\((nodes)\)

1. // Thread id is denoted as threadId. The number of threads used is denoted as numThreads.
2. // The number of nodes in the network.
3. \( numNodes = nodes.size(); \)
4. for \( i = 0 \) to \( numNodes \) do
5. \( if \ i \mod numThreads == threadId \) then
6. \( iNode = nodes[i]; \)
7. // Traverse all the other nodes for \( iNode \).
8. \( for \ j = 0 \) to \( numNodes \) do
9. \( jNode = nodes[j]; \)
10. if \( iNode \neq jNode \) then
11. Count \( a_{11}, a_{10}, a_{01}, \) and \( a_{00} \) based on the community information of \( iNode \) and \( jNode \) from ground truth community structure and discovered community structure;
12. end if
13. end for
14. end if
15. end for
16. Calculate \( RI, ARI, \) and \( JI \) with \( a_{11}, a_{10}, a_{01}, \) and \( a_{00} \) based on Equations (5), (6), and (7);
17. Return \( RI, ARI, \) and \( JI; \)

### B. Parallel Algorithms Based on Pthreads

The parallel Pthreads algorithms for all the metrics, except the ones based on pair counting, assign subsets of ground truth communities and discovered communities, and also local network to each thread using the same approach adopted in the parallel MPI algorithms introduced in Section III-A. The difference between the parallel Pthreads algorithms and the parallel MPI algorithms is that the ground truth communities, the detected communities, and the network are globally accessible in Pthreads, while they are locally stored in MPI. The cores of the algorithms to calculate these metrics do not change compared with the parallel MPI algorithms, so we will not present their outlines here.

In the parallel Pthreads algorithm to calculate the pair counting based metrics, the problem is partitioned with the
We calculate speedup using Equation (10)
\[ \text{Speedup} = \frac{T_1}{T_p} \] (10)
where \( T_1 \) is the running time of the sequential program and \( T_p \) is the running time of the parallel program when \( p \) processors is adopted. We also compute efficiency according to Equation (11)
\[ \text{Efficiency} = \frac{\text{Speedup}}{p} \] (11)

Notice that for our experimental results on Blue Gene/Q, \( p \) is denoted as the number of computing nodes adopted, \( T_1 \) is the running time of the parallel program when only 1 node adopted, and \( T_p \) is the running time of the parallel program when \( p \) nodes adopted.

C. LFR Benchmark Network

We run our parallel MPI and Pthreads programs to calculate the metrics for LFR benchmark networks [22] which have known ground truth community structure. The average node degree of the LFR benchmark networks is set to be 15 and the maximum node degree is set to be 50. The exponent \( \gamma \) for the degree sequence is 2. The exponent \( \beta \) for the community size distribution is 1. The mixing parameter \( \mu \) is equal to 0.3.

The LFR benchmark network for testing the parallel programs for calculating the metrics with ground truth community structure has 100,000 nodes. The ground truth community structure is given when generating the network. The discovered community structure is obtained by using a community detection algorithm called Speaker listener Label Propagation Algorithm (SLPA) [12] with threshold parameter \( r = 0.5 \). SLPA gets disjoint communities when \( r = 0.5 \).

We choose two sizes, ten million of nodes (10,000,000) and one hundred million of nodes (100,000,000), of LFR network to test the parallel programs to compute the metrics without ground truth communities. We calculate the values of these metrics for the ground truth community structure instead of for the discovered community structure since it takes too long to get the detected communities with SLPA.

D. Experimental Results

In this part, we will report the performance results of the parallel MPI and Pthreads algorithms for community quality metrics both with and without ground truth community structure. For the running time, we do not take into account the I/O time. That is, the time of the program to read the ground truth communities, the discovered communities, or the network. Moreover, since GANXIS has 64 processors,
every thread in our parallel Pthreads algorithms executes on its dedicated processor. Therefore, threads do not compete for central processing unit (CPU) processors. They execute in parallel, and we can completely ignore thread scheduling issues in our considerations. Because of this we use terms ‘thread’ and ‘processor’ interchangeably when describing the results of the parallel Pthreads algorithms on GANXIS.

1) Performance of Parallel Algorithms for Metrics with Ground Truth Community Structure: Figures 1(a), 1(b), and 1(c) respectively show the total running time, computation time, and message passing time of the three parallel MPI algorithms, Algorithm 1, Algorithm 2, and Algorithm 3, to compute the community quality metrics with ground truth communities on GANXIS. Figure 1(a) indicates that the total running time decreases as the number of processors increases. Figure 1(b) implies that the computation time decreases as the growth of the number of processors. However, it is shown in Figure 1(c) that the message passing time goes as a saw shape when the number of processors grows. The saw behavior is the result of GANXIS architecture in which quads of cores uses one shared global memory module. Increasing the number of processors from 1 to 2 reduces the message passing for the first time, adding two more processors helps as the message are placed in the same share memory module for all four cores. In case of 8 processors, some messages start to be moving between two memory modules, which decreases the advantage of having more processors and so on. In addition, Figures 2(a) and 2(b) present the corresponding speedup and efficiency. From Figure 2(a), we can observe that the speedup of all three algorithms grows as the number of processors increases. Figure 2(b) indicates that the efficiency of Algorithm 3 first increases from 1 to 16 processors and then decreases from 16 to 32 processors, while the efficiency of the other two algorithms always decreases. It is worth noting that the speedup and efficiency of Algorithm 1 and Algorithm 2 are almost the same with each other, but both are much smaller than those of Algorithm 3 that is to calculate the pair counting based metrics. The efficiency of Algorithm 3 is even larger than 1, achieving a super-linear speedup. The super-linear speedup is the result of increasing larger cache available on many processors. As the number of processors increases, the volume of data processed on each processor decreases but cache is the same size. Thus, the number of cache misses decreases on each processor, speeding up the execution beyond linear speed up.

Figures 3(a), 3(b), and 3(c) respectively present the total running time, computation time, and message passing time of the three parallel MPI algorithms, Algorithm 1, Algorithm 2, and Algorithm 3, for computing the metrics with ground truth communities on Blue Gene/Q. Figure 3(a) indicates that the total running time decreases as the number of nodes times 16. Figure 3(b) demonstrates that the total running time decreases as the number of nodes grows. Figure 3(b) implies that the computation time decreases as the growth of the number of nodes. However, there is no obvious trend of the message passing time. Moreover, Figures 4(a) and 4(b) show the corresponding speedup and efficiency. It can be observed from Figure 4(a) that the speedup of the three algorithms grows when the number of nodes increases, except for Algorithm 1 and Algorithm 2 when there are 256 nodes, the reason of which is that the message passing time instead of the computation time is the dominant part of the total running time at this case. Figure 4(b) implies that the efficiency of all three algorithms decreases as the growth of the number of nodes.

Figures 5(a), 5(b), and 5(c) respectively show the total running time, speedup, and efficiency of the three parallel Pthreads algorithms for computing the metrics with ground truth community structure on GANXIS. Figure 5(a) implies that the total running time decreases when the number of processors increases. Figure 5(b) indicates that the speedup increases as the growth of the number of processors. However, we could learn from Figure 5(c) that the efficiency first grows from 1 to 2 processors and then decreases from 2 to 32 processors. Comparing the total running time in Figure 5(a) and in Figure 1(a), we could see that the total running time of the three parallel Pthreads algorithms is generally larger than that of the three parallel MPI algorithms. Therefore, we recommend using the parallel MPI algorithms instead of the parallel Pthreads algorithms to calculate the metrics with ground truth community structure on GANXIS (or shared memory machines). Also, it is interesting that the speedup and efficiency of the parallel Pthreads algorithms to calculate the information theory and cluster matching based metrics shown.
in Figures 5(b) and 5(c) are larger than those of the corresponding parallel MPI algorithms shown in Figures 2(a) and 2(b). However, the speedup and efficiency of the parallel Pthreads algorithm to calculate the pair counting based metrics shown in Figures 5(b) and 5(c) are much smaller than those of the corresponding parallel MPI algorithm shown in Figures 2(a) and 2(b). This phenomenon leads to an interesting result that the speedup and efficiency of the parallel Pthreads algorithm to calculate the pair counting based metrics are smaller than those of the parallel MPI algorithm. This phenomenon leads to an interesting result that the speedup and efficiency of the parallel Pthreads algorithm are respectively much smaller, larger, and higher than those of the parallel MPI algorithm.

2) Performance of Parallel Algorithms for Metrics without Ground Truth Community Structure: Figures 6(a), 6(b), and 6(c) respectively show the total running time, speedup, and efficiency of the parallel Pthreads algorithm for calculating the metrics without ground truth community structure on GANXIS with the size of the LFR benchmark network being 10,000,000. These three subfigures imply that the total running time decreases, the speedup increases, and the efficiency decreases as the growth of the number of processors. Similarly, the results shown in Figure 6, Figure 8 demonstrates that there is also a performance degradation when there are 32 processors.

Figures 7(a), 7(b), and 7(c) present the corresponding total running time, speedup, and efficiency of the parallel Pthreads algorithm. We could observe that the total running time decreases, the speedup grows, and the efficiency decreases as the number of processors increases. Also, the total running time, the speedup, and the efficiency of the parallel Pthreads algorithm achieve 21.06 seconds at 16 processors, while the smallest running time the parallel Pthreads algorithm achieves is 7.02 seconds at 32 processors. The largest speedup the parallel Pthread algorithm gets is 8.87 at 16 processors, while the largest speedup the parallel Pthreads algorithm gets is 20.9 at 32 processors. When the size of the LFR benchmark network is 100,000,000, the smallest running time the parallel MPI algorithm can achieve is 208.82 seconds at 32 processors, while the smallest running time the parallel Pthreads algorithm can achieve is 65.66 seconds at 32 processors. The largest speedup the parallel MPI algorithm can get is 10.98 at 32 processors, while the largest speedup the parallel Pthread algorithm can get is 23.92 at 32 processors.

When the size of the LFR benchmark network is 10,000,000, the smallest running time the parallel MPI algorithm achieves is 21.06 seconds at 16 processors, while the smallest running time the parallel Pthreads algorithm achieves is 7.02 seconds at 32 processors. The largest speedup the parallel MPI algorithm gets is 8.87 at 16 processors, while the largest speedup the parallel Pthread algorithm gets is 20.9 at 32 processors. When the size of the LFR benchmark network is 100,000,000, the smallest running time the parallel MPI algorithm can achieve is 208.82 seconds at 32 processors, while the smallest running time the parallel Pthreads algorithm can achieve is 65.66 seconds at 32 processors. The largest speedup the parallel MPI algorithm can get is 10.98 at 32 processors, while the largest speedup the parallel Pthread algorithm can get is 23.92 at 32 processors. Thus, we recommend using the parallel Pthreads algorithm instead of the parallel MPI algorithm to calculate the metrics without ground truth community structure on GANXIS.

Moreover, the parallel Pthreads algorithm can get a speedup of 8.87 at 16 processors, while the largest speedup the parallel Pthread algorithm gets is 20.9 at 32 processors. When the size of the LFR benchmark network is 100,000,000, the smallest running time the parallel MPI algorithm can achieve is 208.82 seconds at 32 processors, while the smallest running time the parallel Pthreads algorithm can achieve is 65.66 seconds at 32 processors. The largest speedup the parallel MPI algorithm can get is 10.98 at 32 processors, while the largest speedup the parallel Pthread algorithm can get is 23.92 at 32 processors.
true communities on GANXIS (or shared memory machines).
There is another point we could get is that the speedup and the efficiency of both parallel MPI algorithm and parallel Pthreads algorithm grow as the size of the network increases.

V. CONCLUSION
In this paper, we provide a parallel toolkit, implemented with MPI and Pthreads, to calculate the community quality metrics with and without ground truth community structure. We evaluate their performance on both distributed memory machine, such as Blue Gene/Q, and shared memory machine, for instance GANXIS. We conduct experiments on LFR benchmark networks with the number of nodes being 100,000; 10,000,000; and 100,000,000. The experimental results indicate that both the parallel MPI programs and the parallel Pthreads programs yield a significant performance gain over sequential execution. In addition, we discover that the parallel MPI algorithms perform better than the parallel Pthreads algorithms in terms of total running time, speedup, and efficiency on calculating the metrics with ground truth community structure, while the situation reverses on computing the metrics without ground truth community structure. Therefore, we recommend using the parallel MPI algorithms and the parallel Pthreads algorithm respectively to calculate the metrics with and without ground truth community structure.

ACKNOWLEDGMENT
This work was supported in part by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 and by the the Office of Naval Research Grant No. N00014-09-1-0607. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies either expressed or implied of the Army Research Laboratory or the U.S. Government.

REFERENCES