IMGPU: GPU-Accelerated Influence Maximization in Large-Scale Social Networks

Xiaodong Liu, Mo Li, *Member*, *IEEE*, Shanshan Li, *Member*, *IEEE*, Shaoliang Peng, *Member*, *IEEE*, Xiangke Liao, *Member*, *IEEE*, and Xiaopei Lu, *Student Member*, *IEEE*

Abstract—Influence Maximization aims to find the top-K influential individuals to maximize the influence spread within a social network, which remains an important yet challenging problem. Proven to be NP-hard, the influence maximization problem attracts tremendous studies. Though there exist basic greedy algorithms which may provide good approximation to optimal result, they mainly suffer from low computational efficiency and excessively long execution time, limiting the application to large-scale social networks. In this paper, we present IMGPU, a novel framework to accelerate the influence maximization by leveraging the parallel processing capability of graphics processing unit (GPU). We first improve the existing greedy algorithms and design a bottom-up traversal algorithm with GPU implementation, which contains inherent parallelism. To best fit the proposed influence maximization algorithm with the GPU architecture, we further develop an adaptive K-level combination method to maximize the parallelism and reorganize the influence graph to minimize the potential divergence. We carry out comprehensive experiments with both real-world and sythetic social network traces and demonstrate that with IMGPU framework, we are able to outperform the state-of-the-art influence maximization algorithm up to a factor of 60, and show potential to scale up to extraordinarily large-scale networks.

Index Terms—Influence maximization, GPU, large-scale social networks, IMGPU, bottom-up traversal algorithm

1 Introduction

Social networks such as Facebook and Twitter play an important role as efficient media for fast spreading information, ideas, and influence among huge population [12], and such effect has been greatly magnified with the rapid increase of online users. The immense popularity of social networks presents great opportunities for large-scale viral marketing, a marketing strategy that promotes products through "word-of-mouth" effects. While the power of social networks has been explored more and more to maximize the benefit of viral marketing, it becomes vital to understand how we can maximize the influence over the social network. This problem, referred to as influence maximization, is to select within a given social network a small set of influential individuals as initial users such that the expected number of influenced users, called influence spread, is maximized.

The influence maximization problem is interesting yet challenging. Kempe et al. [12] proved this problem to be NP-hard and proposed a basic greedy algorithm that provides good approximation to the optimal result. However, their approach is seriously limited in efficiency because it needs to run Monte-Carlo simulation for considerably long time period to guarantee an accurate estimate. Although a number of successive efforts have been made to improve the efficiency, state-of-the-art approaches still suffer from excessively long execution time due to the high-computational complexity for large-scale social networks.

On the other hand, graphics processing unit (GPU) has recently been widely used as a popular general-purpose computing device and shown promising potential in accelerating computation of graph problems such as breadth first search [9] and minimum spanning tree [20], due to its parallel processing capacity and ample memory bandwidth. Therefore, in this paper, we explore the use of GPU to accelerate the computation of the influence maximization problem.

However, the parallel processing capability of GPU can be fully exploited in handling tasks with regular data access pattern. Unfortunately, the graph structures of most real-world social networks are highly irregular, making GPU acceleration a nontrivial task. For example, Barack Obama, the U.S. president, has more than 11 million followers in Twitter, while more than 90 percent Twitter users' follower number is under 100 [13]. Such irregularities may lead to severe performance degradation. The main challenges of full GPU acceleration lie in the following aspects. First, the parallelism of influence-spread computation for each possible seed set is limited by the number of nodes at each level. Therefore, the computational power of GPU cannot be fully exploited if we directly map the problem to GPU for acceleration. Second, as the degree of nodes in most social networks mainly follow a power-law distribution, severe divergence between GPU threads will occur during influence-spread computation, seriously degrading the overall performance. Third, due to the irregular nature of real-world

- *X. Liu, S. Li, S. Peng, X. Liao, and X. Lu are with the School of Computer, National University of Defense Technology, No. 147 Yanwachi Main Street, Changsha, Hunan 410073, China. E-mail: {liuxiaodong, shanshanli, shaoliangpeng, xkliao, xiaopeilu}@nudt.edu.cn.*
- *M. Li is with School of Computer Engineering, Nanyang Technological University, N4-02c-108, 50 Nanyang Avenue, 639798 Singapore. E-mail: limo@ntu.edu.sg.*

Manuscript received 7 June 2012 ; revised 26 Nov. 2012; accepted 8 Jan. 2013; published online 14 Feb. 2013.
Recommended for acceptance by S. Papavassiliou.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-06-0539.
Digital Object Identifier No. 10.1109/TPDS.2013.41.

social networks, the memory accesses show poor spatial locality, making it hard to fit the GPU computational model.

To address the above challenges, we propose a GPU-accelerated influence maximization framework, IMGPU, which aims at fully leveraging the parallel processing capability of GPU. We first convert the social graph into a directed acyclic graph (DAG) to avoid redundant calculation. Then a Bottom-up traversal algorithm (BUTA) is designed and mapped to GPU with CUDA programming model. Our approach provides substantial improvement to the existing sequential approaches by taking advantage of the inherent parallelism in processing nodes within a social network. Based on the feature of the influence maximization problem, we propose a set of adaptive mechanisms to explore the maximum capacity of GPU and optimize the performance of IMGPU. In particular, we develop an adaptive K-level combination method to maximize the parallelism among GPU threads. Meanwhile, we reorganize the graph by level and degree distribution to minimize the potential divergence and coalesce the memory access to the utmost extent. We conduct extensive experiments with both real-world and synthetic social network traces. Compared with the state-of-the-art algorithm MixGreedy, IMGPU achieves up to $60\times$ speedup in the execution time and is able to scale up to extraordinarily large-scale networks which were never expected with the existing sequential approaches.

As a summary, the contributions of this paper are mainly twofold. First, we present BUTA, an efficient bottom-up traversal algorithm which contains inherent parallelism for the influence maximization problem. We further map BUTA to GPU architecture to exploit the parallel processing capability of GPU. Second, to best fit the GPU computational model, we propose several effective optimization techniques to maximize the parallelism, avoid potential divergence, and coalesce memory access.

The remainder of this paper is organized as follows: Section 2 provides preliminaries on influence maximization and reviews related work. The IMGPU framework and corresponding GPU optimizations are presented in Section 3 and Section 4, respectively. We evaluate the IMGPU design by extensive experiments and report the experimental results in Section 5. We conclude this work in Section 6.

2 Preliminaries and Related Work

In this section, we present preliminary introduction to influence maximization, and review related work.

In influence maximization, an online social network is modeled as a directed graph $G = (V, E, W)$, where $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of nodes in the graph, each of which corresponds to an individual user. Each node can be either active or inactive, and will switch from being inactive to being active if it is influenced by others nodes. $E \subset V \times V$ is a set of directed edges representing the relationship between different users. Take Twitter as an example. A directed edge (v_i, v_j) will be established from node v_i to v_j, if v_i is followed by v_j, which indicates that v_j is open to receive tweets from v_i, and thus may be influenced by v_i. $W = \{w_1, w_2, \ldots, w_n\}$ is the weight of each node which indicates its contribution to the influence spread. The weight is initialized as 1 for each node, meaning that if this node is influenced by other nodes, its contribution to the influence spread is 1. The size of node set is n, and the number of edges is m. Node v_i is called a sink if its outdegree is 0, and called a source if its indegree is 0.

The independent cascade (IC) model [12] is one of the most well-studied diffusion models. Given an initial set S, the diffusion process of IC model unfolds as follows: At step 0, only nodes in S are active, while other nodes stay in the inactive state. At step t, for each node v_i which has just switched from being inactive to being active, it has a single chance to activate each currently inactive neighbor v_w, and succeeds with a probability $p_{i,w}$. If v_i succeeds, v_w will become active at step $t+1$. If v_w has multiple newly activated neighbors, their attempts in activating v_w are sequenced in an arbitrary order. Such a process runs until no more activations are possible [12]. We use $\sigma(S)$ to denote the influence spread of the initial set S, which is defined as the expected number of active nodes at the end of influence propagation.

Given a graph $G = (V, E, W)$ and a parameter K, the influence maximization problem in the IC model is to select a subset of influential nodes $S \subset V$ of size K such that the influence spread $\sigma(S)$ is maximized at the end of influence diffusion process.

The influence maximization problem is first introduced by Domingos and Richardson in [7] and [19]. In 2003, Kempe et al. [12] proved that the influence maximization problem is NP-hard and proposed a basic greedy algorithm as shown in Algorithm 1. Their approach works in K iterations, starting with an empty set S (line 1). In each iteration, a node v_i which brings the maximum marginal influence spread $\sigma_S(v_i) = \sigma(S \cup v_i) - \sigma(S)$ is selected to be included in S (lines 3 and 4). The process ends when the size of S reaches K. However, computing the influence spread $\sigma(S)$ of an arbitrary set S is proved to be #P-Hard in [5]. To guarantee computation accuracy, Kempe et al. suggest running Monte-Carlo simulation for a long time to obtain a close estimate, resulting in low computation efficiency of the greedy algorithm.

Algorithm 1. Basic Greedy.
1: Initialize $S = \emptyset$
2: **for** $i = 1$ to K **do**
3: Select $v = \arg\max_{u \in (V \setminus S)}(\sigma(S \cup u) - \sigma(S))$
4: $S = S \cup \{v\}$
5: **end for**

Wei Chen et al. proposed MixGreedy [4] that reduces the computational complexity by computing the marginal influence spread for each node $v_i \in (V \setminus S)$ in one single simulation. MixGreedy first determines whether an edge would be selected for propagation or not with a given probability. Then all the edges not selected are removed to form a new graph $G' = (V, E', W)$, where $E' \subseteq E$. With this treatment, the marginal gain $\sigma_S(v_i)$ from adding node v_i to S is the number of nodes that are reachable from v_i, but unreachable from all the nodes in S. To compute the influence spread for each node, a basic implementation is doing BFS for all vertices which takes $O(mn)$ time. Therefore, MixGreedy incorporates Cohen's randomized algorithm [6] for estimating the marginal influence spread

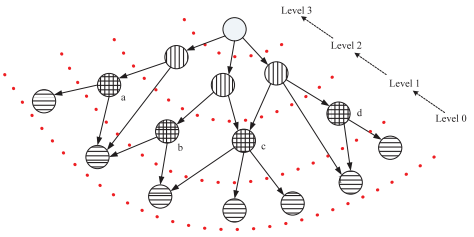


Fig. 1. Bottom-up traversal.

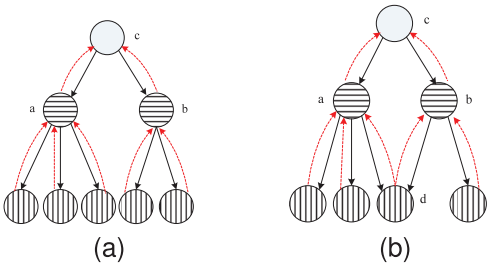


Fig. 2. Relation of nodes.

for every node, and then selects the node that offers the maximal influence spread. Adopting the above optimization methods, MixGreedy can run much faster. However, the improvement is not effective enough to reduce execution time to an acceptable range especially for large-scale social networks. Moreover, Cohen's randomized algorithm provides no accuracy guarantee.

In [17], Liu et al. propose ESMCE, a power-law exponent supervised Monte-Carlo method that efficiently estimates the influence spread by randomly sampling only a portion of the child nodes. In particular, ESMCE roughly predicts the number of child nodes needed to be sampled according to the power-law exponent of the given social network. Afterward, multiple iterative steps are employed to randomly sample more nodes until the precision requirement is finally achieved. Although ESMCE can considerably accelerate the influence maximization problem, it is obvious that the cost of its acceleration is the accuracy.

There have been also many other algorithms and heuristics proposed for improving the efficiency issue, such as [5], [8], [10], [11], [16], [21]. More details about these works can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.41. Different from all existing studies, in this work we demonstrate that by exploiting the parallel processing capability of GPU, we can greatly accelerate the influence maximization problem while performing consistently among the best algorithms in terms of accuracy.

3 IMGPU Framework

In this section, we describe the IMGPU framework that enables GPU-accelerated processing of influence maximization. First, we develop BUTA that can exploit inherent parallelism and effectively reduce the complexity with ensured accuracy. Then we map the algorithm implementation to GPU environment under CUDA programming model.

3.1 Bottom-Up Traversal Algorithm

As mentioned in Section 2, we can get a new graph $G' = (V, E', W)$ from the original graph after randomly selecting edges from G. Afterward, we need to compute the marginal gain for each node to find the best candidate. Instead of doing BFS for each node which is rather inefficient, we find that the marginal influence computation of each node only relies on its child nodes; thereby, we could get the influence spreads for all the nodes by traversing the graph only once in a bottom-up way. Moreover, as there is no dependence relationship among nodes at the same level in a DAG, their computation can be done in parallel. Here, the level of a node v_i, denoted as $level(v_i)$, is defined by the following equation:

$$level(v_i) = \begin{cases} \max\limits_{v_j \in out[v_i]} level(v_j) + 1, & \text{if } |out[v_i]| > 0, \\ 0, & \text{if } |out[v_i]| = 0 \end{cases}$$

where $out[v_i]$ denotes the set of child nodes of v_i. As a motivating example shown in Fig. 1, the influence-spread computation for nodes a, b, c, and d at level 1 can be done concurrently. Therefore, their computation can be mapped to different GPU threads and processed in parallel to overlap the execution time. To this end, we proposed BUTA to explore the inherent parallelism. We first convert the graph to a DAG to avoid redundant computation and potential deadlock. Then, the DAG is traversed by level in a bottom-up way to compute the marginal influence for all nodes in parallel on GPU.

To start with, we first prove a theorem that the marginal gains for nodes belonging to one strongly connected component (SCC) are identical. The detailed proof can be found in Appendix B, which is available in the online supplemental material. From such a theorem, we know that influence spreads of nodes in an SCC are equal in quantity; thus, it is unnecessary to do repeated computation for all of those nodes. Therefore, we merge all nodes belonging to the same SCC into one node with a weight being the size of the SCC. By doing so, first, we can avoid a large number of unnecessary computations. Second, the graph G' can be converted into a DAG $G^* = (V^*, E^*, W^*)$, and the following BUTA algorithm can be applied to compute the influence spread in parallel.

Note that influence spread of a node has close relation to its child nodes. As illustrated in Fig. 2a, node c has two child nodes, a and b. Every node reachable from a and b can also be reached from c. As the subgraph of node a has no overlap with that of b, influence spread of c can be easily calculated by the sum of influence spread of a, b as well as the weight of node c. Fig. 2b shows a special example where the influence spread of a and b overlaps at node d. When computing the influence spread of c, we need to subtract the overlap part (node d) from the sum. To summarize, the influence spread of source node v_i can thus be obtained by

$$\sigma_S(v_i) = \sum_{v_j \in out[v_i]} \sigma_S(v_j) + w_{v_i} - overlap(out[v_i]), \qquad (1)$$

where $overlap(out[v_i])$ denotes the number of overlaps among influence spreads of all the nodes in $out[v_i]$. Computation of the $overlap(\cdot)$ function is based on the label of nodes. Each node is assigned with a label recording

where this node may overlap with others. When computing the overlap for a given node set, we check the label of each node in the node set and compute the amount of redundant among their labels. In Appendix C, which is available in the online supplemental material, we give the details of label assignment criteria and overlap computation algorithm.

Inspired by 1, we find it not necessary to do BFS for each node. Instead, we can traverse the graph only once in a bottom-up way and obtain influence spreads of all the nodes. Moreover, as the influence spread of nodes at the same level depends only on their child nodes, the computation can be done in parallel by using GPU for acceleration.

Algorithm 2 presents the details of BUTA, where R denotes the number of Monte-Carlo simulations. In each round of simulation, the graph is first reconstructed by selecting edges at a given probability (line 5) and converting into a DAG (line 6). Then we start the bottom-up traversal level by level (lines 7-13). We use the "in parallel" construct (line 8) to indicate the codes that can be executed in parallel by GPU. Influence spreads of all nodes at the same level can be calculated in parallel (line 9) and the label of each node is then determined for future overlap computation (line 10). Such a process will iterate until all nodes are computed. As G^* is a DAG, there will be no deadlock at run time. After R rounds of simulation, the node providing the maximal marginal gain will be selected and added to the set S (lines 15-16).

Algorithm 2. BUTA.
Input: Social network $G = (V, E, W)$, parameter K
Output: The set S of K influential node
1: Initialize $S = \emptyset$
2: **for** $i = 1$ to K **do**
3: Set *Infl* to zero for all nodes in G
4: **for** $j = 1$ to R **do**
5: $G' \leftarrow$ randomly select edges under IC model
6: $G^* \leftarrow$ convert G' to DAG
7: **for** each level l from bottom up in G^* **do**
8: **for** each node v at level l **in parallel do**
9: Compute the influence spread $\sigma_S(v)$
10: Compute the label $L(v)$
11: $Infl[v] \leftarrow Infl[v] + \sigma_S(v)$
12: **end for**
13: **end for**
14: **end for**
15: $v_{max} = \arg\max_{v \in (V \setminus S)} Infl[v]/R$
16: $S = S \cup \{v_{max}\}$
17: **end for**

The total running time of Algorithm 2 is $O(KR(m + m^*))$, where m and m^* denote the number of edges in graphs G and G^*, respectively. The detailed complexity analysis can be found in Appendix D, which is available in the online supplemental material. Consequently, compared with the basic greedy algorithm taking $O(KRnm)$ time and MixGreedy taking $O(KRTm)$ time, BUTA can greatly reduce the time complexity.

To summarize, the advantages of BUTA are as follows: First, we can greatly reduce the time complexity to $O(KR(m + m^*))$ through DAG conversion and bottom-up traversal. Second, BUTA can guarantee better accuracy than MixGreedy as we accurately compute influence spread for each node while MixGreedy approximates them from Cohen's algorithm. Last but not least, BUTA is designed with inherent parallelism and can be mapped to GPU to exploit the parallel processing capability of GPU, thus further reducing the execution time, as we will show in Section 3.2.

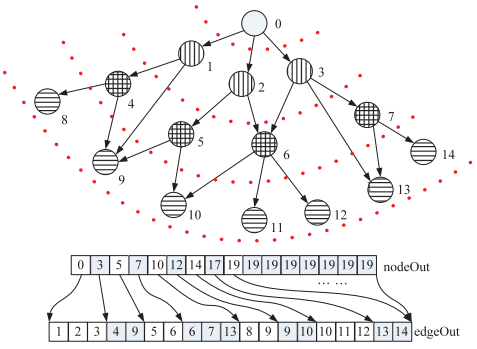


Fig. 3. Graph data representation.

3.2 Baseline GPU Implementation

In this section, we first describe the graph data structure used in this work, and then present the baseline implementation of IMGPU in detail. In Appendix E, which is available in the online supplemental material, we give a short introduction to the internal architecture of GPU and briefly describe the corresponding CUDA programming model.

3.2.1 Data Representation

To implement IMGPU over the GPU architecture, the traditional adjacency-matrix representation is not a good choice especially for large-scale social networks. The reasons are twofold. First, it costs $n \times n$ memory space which significantly restricts the size of social network that can be handled by GPU. Second, the latency of data transfer from host to device as well as global memory access is high, degrading the overall performance. Therefore, we use the compressed sparse row (CSR) format which is widely used for sparse matrix representation [3]. As depicted by Fig. 3, the graph is organized into arrays. The *edgeOut* array consists of adjacency lists of all the nodes. Each element in the *nodeOut* array stores the start index pointing to the edges outgoing from that node in the *edgeOut* array. The CSR representation requires $n + 1 + m$ memory space in sum.

3.2.2 Baseline Implementation

In our CUDA implementation, the graph data is first transferred to the global memory of GPU. Then, we assign one thread for each node to run the influence-spread computation kernel. All threads are divided into a set of blocks and scheduled into GPU stream processors to execute in parallel. The influence-spread computation kernel works iteratively by level. In each iteration, the influence spreads of nodes at the same level are calculated in parallel by different stream processors. The iteration ends when all the influence-spread computations are finished. This process will run for R times to select the node that provides the maximal average influence spread. In this way, the parallel processing capability of GPU is exploited for influence maximization acceleration. In Appendix E,

which is available in the online supplemental material, we illustrate a figure that depicts the above CUDA implementation of BUTA on GPU architecture.

We adopt the fast random graph generation algorithm PPreZER [18] to select edges in the random and CUDA-based forward-backward algorithm [1] to find SCCs of the selected graph in parallel in GPU. Afterward, the kernel that computes the influence spread will run iteratively until all nodes are visited. Algorithm 3 shows the influence-spread kernel written in CUDA. *visited* is a Boolean array identifying whether a node has been visited or not, *unfinished* is an array recording the nodeID of the first child node whose influence-spread computation is yet unfinished, and $Infl[tid]$ denotes the influence spread of node *tid*. Lines 2-11 decide whether the thread is ready for influence computation. This is done by checking the *visited* variable of all its child nodes. We use the *unfinished* array to record the first unfinished child nodes, and thus we can avoid the repeated check on already finished child nodes. If all the child nodes have been visited, then lines from 12 to 16 compute the influence spread for each qualified thread in parallel according to (1). Finally, $visited[tid]$ is set to true denoting that the node has been visited (line 17).

Algorithm 3. Influence-Spread Computation Kernel.

```
 1: int tid = Thread_ID;
 2: if(visited[tid] == false){
 3:   int index = unfinished[tid];
 4:   int num_out = nodeOut[tid + 1] − nodeOut[tid];
 5:   int*outpoint = &edgeOut[nodeOut[tid]];
 6:   for(; index < num_out; index + +){
 7:     if(visited[outpoint[index]] == false){
 8:       unfinished[tid] = index;
 9:       break;
10:   }}
11:   if(index == num_out) {
12:     Infl[tid] = weight[tid];
13:     for(int i = 0; i < num_out; i + +) {
14:       Infl[tid]+ = Infl[outpoint[i]];}
15:     int overlapNo = overlap(Infl, label);
16:     Infl[tid]− = overlapNo;
17:     visited[tid] = true;
18: }}
```

In short, the influence-spread kernel works iteratively by level, and nodes at the same level take part in execution in parallel during each iteration. The parallelism depends on the number of nodes at each level. The worst case happens when the graph is extremely linear which will result in one node being processed at each iteration. According to tests over real-world social networks, however, such a case never happens. As modern social networks are typically of large scale and the interconnections are rich, much parallelism can be exploited to reduce execution time.

Note that, the baseline implementation is far from optimal. The actual parallelism is limited by the number of nodes at each level. Moreover, execution path divergence will occur when threads in a warp process nodes belonging to different levels and nodes with different degrees. All these elements can directly affect the performance. In Section 4, we describe how we tackle these challenges to pursue better performance.

4 GPU-Oriented Optimization

In this section, we analyze factors that affect the performance of baseline GPU implementation and provide three effective optimizations to achieve better performance. First, we reorganize the graph data by levels and degrees to minimize potential divergence. Second, we combine the influence-spread computation for nodes of multiple levels together for better parallelism. Third, we unroll the memory access loop to coalesce accesses to consecutive memory addresses.

4.1 Data Reorganization

As previously mentioned, execution path divergence may lead to serious performance degradation. In this paper, BUTA executes level by level in a bottom-up way. Threads in a warp are responsible for processing different nodes. However, due to the SIMT feature of GPU, threads in a warp execute the same instruction at each clock cycle. Consequently, if threads in a warp are assigned to process nodes at different levels (lines 2 and 11 in Algorithm 3), divergence will occur and induce different execution paths, which will significantly degrade the performance.

In addition, during BUTA execution, threads have to obtain the visit information and the influence spreads of their child nodes (lines 6 and 13 in Algorithm 3). As the degrees of nodes in real-world social networks mainly follow a power-law distribution, there may exist great disparity between the degree of different nodes. Therefore, the workloads of influence-spread computation for different nodes may vary widely, and thus a thread that processes a node with large degree will usually block the other threads in the same warp from successive running.

Such divergence will severely reduce the utilization of GPU cores and degrade the performance. To address these issues, we reorganize the graph by presorting the graph by level and degree, with the purpose of making threads in a warp process nodes that are at the same level and with similar degree as much as possible. In such a way, we hope that threads in the same warp would mostly follow the same execution path, minimizing the possible divergence.

However, since edges are selected in random during each Monte-Carlo simulation, a node may have different degrees and belong to different levels during various simulations. Therefore, a naive solution is that we sort the selected graph after each simulation to assure the least divergence. Obviously, such a method is extremely time-consuming and unrealistic to implement. Fortunately, we observe from experiments on real-world data sets that the probability that nodes with the same degree (resp. level) in the original graph still have the same degree (resp. level) in the randomly selected graph is as much as 95 percent (resp. 88 percent). Therefore, we can achieve the objective of making threads in the same warp processing nodes with the same level and similar degree by presorting the original graph, even though the level and degree of node in the randomly selected graph may be inconsistent with that in the original graph. To this end, we reorganize the original graph data through presorting the nodes by their levels and degrees, where level is the primary key, and outdegree is the secondary key. Experimental results shown in Section 5 validate the effectiveness of such an optimization approach.

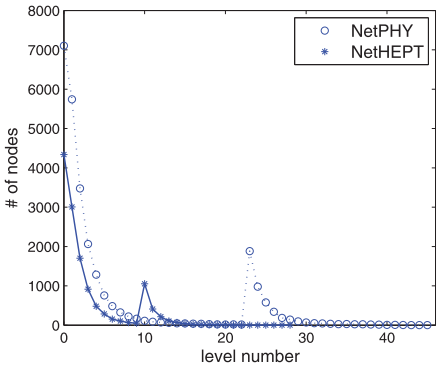


Fig. 4. Node number at different levels.

4.2 Adaptive K-Level Combination

Baseline IMGPU implementation computes influence spreads of nodes from bottom up by level, and thereby its parallelism is limited by the number of nodes at each level. We can benefit more if there are sufficient nodes belonging to the same level to be processed, otherwise the parallel processing capability of GPU would be underexploited. For most cases, there is adequate parallelism to exploit since the real-world social network is typically of large scale. However, there do exist some particular levels which only contain a small number of nodes due to the inherent graph irregularity of social networks. Fig. 4 depicts the statistics of number of nodes at different levels of two real-world data sets, NetHEPT and NetPHY. We can clearly see that the number of nodes varies seriously between different levels. More than 60 percent of levels just have less than 100 nodes. Apparently, the parallel computation capability of GPU cannot be fully utilized for these levels.

To exploit more parallelism and make full use of GPU cores, we propose an adaptive K-level combination optimization. We logically combine K-adjacent levels with small number of nodes into a virtual level, and compute their influence spread concurrently. Fig. 5 exemplifies such an approach. There are seven nodes at level 0, while the number of nodes at levels 1, 2, and 3 is relatively small. Consequently, we logically combine level 1, 2, and 3 into a virtual level $1'$. After the combination, we can compute the influence spreads for nodes $0, 1, \ldots, 7$ concurrently to pursue higher parallelism.

The K-level combination, though providing a good way to explore better parallelism, introduces another problem. That is, we cannot directly compute the influence spread for node at upper level (e.g., node 0 in Fig. 5) according to (1) because the influence spreads of its child nodes (node 1, 2, and 3 in this example) may be yet unknown. To address this problem, we divide the influence spread for nodes at upper levels into two parts. The first part is the number of nodes under the critical level. Here the critical level represents the level which is going to be visited next (e.g., level 1 in Fig. 5). Since the influence spread of nodes under the critical level have already been computed, this part can be easily computed according to (1). While the second part is the number of nodes above the critical level. For this part, we propose to do BFS from the target node till the critical level and count the number of reachable nodes. The influence spread of node at upper levels is thus the sum of these two

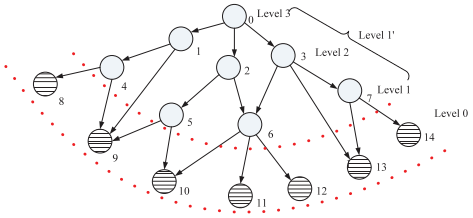


Fig. 5. K-level combination.

parts. In Fig. 5, since levels 1, 2, and 3 are logically combined into level $1'$, when computing the influence spread of node 0, we first perform BFS from node 0, and we get eight nodes above level 1. Then based on (1), we compute the number of nodes under level 1 and obtain seven nodes. Thus, the influence spread of node 0 is 15 in total.

It is important to notice that through such an optimization, we can obtain better parallelism because nodes at adjacent K levels can be processed concurrently. However, this is gained at the cost of additional time spent on BFS counting. The parameter K stands as a tradeoff between parallelism and extra counting time. If K is too small, there may still be room for mining parallelism. On the other hand, if K is too large, although more parallelism can still be exploited, the extra time cost for BFS will finally overwhelm the gained efficiency. To achieve the best performance, the parameter K is adaptively set for different graphs according to (2):

$$K = \left\{ \arg\max_{K} \sum_{i=l_0}^{l_0+K} nodeNO(i) \leq 256 \right\}, \tag{2}$$

where l_0 denotes the critical level and $nodeNO(i)$ denotes the number of nodes at level i. In other words, K is adaptively set to be the maximal number of adjacent levels so that the total number of nodes at these K levels is less than 256 which is half of the stream processor number of GPU used in this work. As tested in the experiments, the K-level combination optimization can significantly improve the parallelism, especially for large-scale social network.

4.3 Memory Access Coalescence

As described in lines 13-14 of Algorithm 3, when we compute the influence spread of a node, the thread needs to access the influence spreads of all the child nodes. Therefore, for nodes with large degree, this will result in a large number of memory accesses which will take very long execution time. Such nodes, though accounting for a small percentage of the entire graph, substantially exist in many real-world social networks. As depicted in Fig. 6, the degree distribution of Twitter users is highly skewed. Although the degree of 90.24 percent Twitter user is under 10, there do exist 1,089 nodes with a degree larger than $10K$, producing an enormous quantity of memory accesses.

Fortunately, these memory accesses, though large in amount, show good spatial locality after data reorganization. Fig. 7 depicts the child node distribution of five extra-large nodes in Twitter. We find a large percentage of target memory addresses are relatively consecutive so that we can unroll the memory access loop to coalesce these memory accesses. The loop is unrolled by a predefined step length.

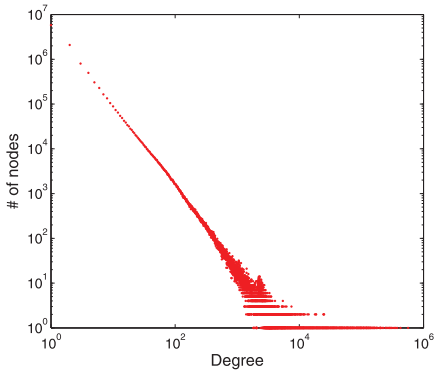


Fig. 6. Degree distribution of Twitter data set.

We tested the performance of loop unrolling under different step lengths (4, 8, 16, 32, 64, and 128), and 16 was finally selected for its best result. In such a way, first, 16 memory accesses are requested in each loop. Due to the spatial locality of child nodes distribution, these memory accesses could be coalesced to reduce the execution time. Second, we can reduce the execution of conditional statement (line 13 in Algorithm 3) by loop unrolling. This optimization, though simple in principle, can significantly improve the performance, especially on large-scale data sets. Our experimental results show that such a method significantly reduces the execution time.

5 Experiments

5.1 Experimental Setup

In our experiments, we use traces of four real-world social networks of different scales and different types, i.e., NetHEPT [14], NetPHY [14], Amazon [15], and Twitter [22]. Table 1 summarizes the statistical information of the data sets. We compare IMGPU and its optimization version IMGPU_O with the two existing greedy algorithms and two heuristic algorithms, i.e., MixGreedy [4], ESMCE [17], PMIA [5], and Random. Moreover, we also implement a CPU-based version of BUTA, referred to as BUTA_CPU, to evaluate the performance of BUTA and the effect of parallelization. The detailed description of the data sets and algorithms can be found in Appendix F, which is available in the online supplemental material.

We examine two metrics, influence spread and running time in the experiments for evaluating the accuracy and efficiency of different algorithms. These experiments are carried out on a PC equipped with a NVIDIA GTX 580 GPU and an Intel Core i7 CPU 920.

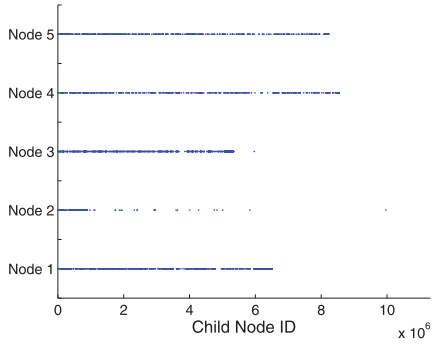


Fig. 7. Child node distribution of five large nodes of Twitter data set.

TABLE 1
Statistics of Real-world Data Sets

Datasets	Types	Nodes	Edges	Avg. Degree
NetHEPT	Real-world	15,233	32,235	4.23
NetPHY	Real-world	37,154	180,826	9.73
Amazon	Real-world	262,111	1,234,877	9.42
Twitter	Real-world	11,316,811	85,331,846	15.08

5.2 Performance Analysis

We first evaluate the performance of different algorithms in terms of influence spread and running time on real-world social networks. Then we examine the effectiveness of different optimization methods. To study the generality and scalability of different algorithms, we also test their performance on synthetic networks generated by GTgraph [2] following Random and Power-law distribution. The detail can be found in Appendix G, which is available in the online supplemental material.

5.2.1 Influence Spread on Real-World Data Sets

Influence spread directly indicates how large the portion of the network is finally influenced. A larger influence spread represents a better approximation to influence maximization, i.e., better accuracy. We test the influence spreads of different algorithms and heuristics under the IC model with propagation probability $p = 0.01$. To obtain an accurate estimate of $\sigma(S)$, we execute 20,000 runs for each possible seed set S. The number of seed sets K varies from 1 to 50. Such an experiment is carried out over four real-world data sets and the results are shown in Fig. 8.

Fig. 8a depicts the experimental result of influence spread on NetHEPT. Note that NetHEPT has only $15K$ nodes, which is the smallest among all data sets. In accordance with expectation, the BUTA-series algorithms, IMGPU, IMGPU_O, and BUTA_CPU, perform well and their accuracies match that of MixGreedy, producing the best results at all values of K. Such a result validates the proposed BUTA algorithm. The performance of ESMCE is slightly lower than IMGPU in terms of influence spread; the influence spread of ESMCE is 4.53 percent less than that of IMGPU when K is 50. This is mainly because ESMCE estimates the influence spread by supervised sampling, thus, affecting the accuracy. Meanwhile, Random performs the worst among all test algorithms (as much as 42.83 percent less than IMGPU), because it contains no consideration of influence spread when selecting seeds. Compared with Random, PMIA performs much better. However, the influence spread of PMIA is still 3.2 percent lower than that of IMGPU on average.

Fig. 8b depicts the influence spread on NetPHY. In this case, IMGPU, IMGPU_O, and BUTA_CPU slightly outperform MixGreedy by 1.5 percent on average. The reason is that MixGreedy estimates the influence spread which inevitably induces errors, while IMGPU accurately calculates the influence spread. Compared with IMGPU, ESMCE

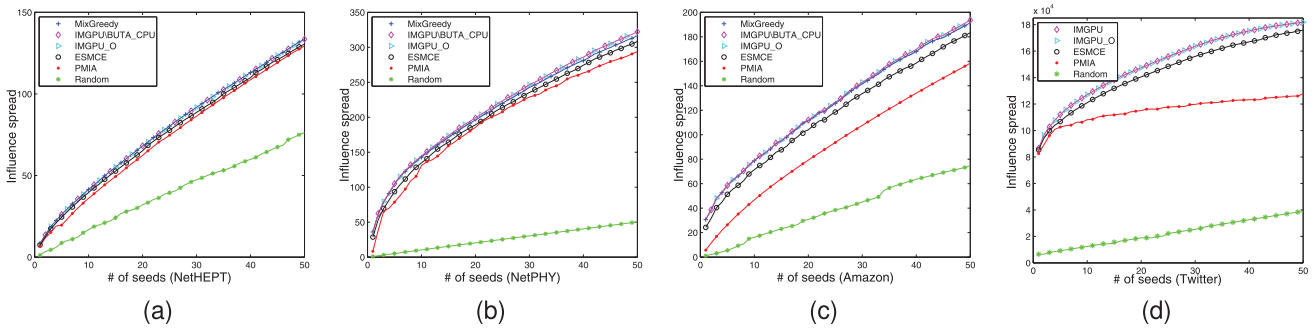


Fig. 8. Influence spread of different algorithms on four real-world data sets.

is 3.35 percent lower on average for K from 1 to 50. Similar to NetHEPT, Random still produces the worst influence spread which is only 15.90 percent of IMGPU. This clearly demonstrates that it is important to carefully select the influential nodes so as to obtain the largest influence spread. The gap between PMIA and IMGPU becomes larger (3.2 percent and 5.4 percent on NetHEPT and NetPHY, respectively) with the growth of social network size, showing that heuristics perform poorly particularly on large-scale networks.

For the case of Amazon ($262K$ nodes and $1.23M$ edges), as shown in Fig. 8c, the accuracies of BUTA-series algorithms are slightly better than that of MixGreedy. As expected, the BUTA-series algorithms perform much better than ESMCE, PMIA, and Random (4.18, 20.82, and 256.38 percent on average, respectively). We can see that ESMCE clearly outperforms the MIA heuristic in this case; this is due to the reason that ESMCE controls the estimation error with a specified error threshold and iteratively samples more needs until the specified accuracy is finally achieved.

Finally we carry out experiments on Twitter. This data set, containing more than $11M$ nodes and $85M$ edges, is so far the largest data set tested in influence maximization experiments. The result is presented in Fig. 8d. MixGreedy and BUTA_CPU becomes infeasible to execute due to the unbearably long running time. The disparity between IMGPU and ESMCE is stably kept around 4 percent when the number of seed K is large than 10. On the other hand, IMGPU performs substantially better than the PMIA and Random as much as 42.25 and 368.21 percent, respectively when K is 50. This evidently demonstrates the effectiveness of IMGPU in terms of influence spread.

5.2.2 Running Time on Real-World Data Sets

Fig. 9 reports the execution time of different approaches for selecting 50 seeds on four data sets, by which we examine the efficiency. Note that the overhead of DAG conversion is relatively low, and takes only 1.74 and 3.7 percent of the execution time of IMGPU on NetHEPT and Amazon. The execution time of IMGPU and IMGPU_O includes such a preprocessing time. The y-axis of Fig. 9 is in logarithmic scale.

As we pointed out, MixGreedy and BUTA_CPU can only run on NetHEPT, NetPHY, and Amazon. Furthermore, MixGreedy costs the longest running time; it takes as much as 1.93×10^3 seconds for NetHEPT and 2.02×10^4 seconds for NetPHY. Compared with MixGreedy, BUTA_CPU performs much better; BUTA_CPU could accelerate the influence computation by $2.58\times$, $6.49\times$, and $8.07\times$, respectively. These results are in accordance with the complexity analysis and demonstrate the efficiency of BUTA.

ESMCE utilize supervised sampling to estimate the influence spread, thus greatly reducing its running time when compared with MixGreedy. As shown in Fig. 9, the execution time of ESMCE is only 10.21 and 3.73 percent of MixGreedy on NetHEPT and Amazon.

IMGPU and IMGPU_O perform significantly better in comparison with MixGreedy. IMGPU achieves $9.69\times$, $15.61\times$, and $46.87\times$ speedups on NetHEPT, NetPHY, and Amazon, respectively. The optimizations of IMGPU_O work effectively and produce as much as $13.39\times$, $19.00\times$, and $60.14\times$ speedups, respectively. Moreover, IMGPU and IMGPU_O show better scalability; IMGPU and IMGPU_O is able to handle larger data sets, such as Twitter, which is definitely beyond the capability of MixGreedy. When compared with ESMCE, the efficiency of IMGPU is slightly lower on NetHEPT and NetPHY. However, IMGPU completely outperforms ESMCE when they are applied to large-scale data sets such as Amazon and Twitter. The reason is that large-scale networks contain more inherent parallelism; thus, the parallel processing capability of GPU can be fully exploited for acceleration.

The two heuristics approaches, Random and PMIA run much faster. However, according to our previous accuracy evaluation, they show poor performance in terms of influence spread. This drawback severely limits their application to real-world social networks.

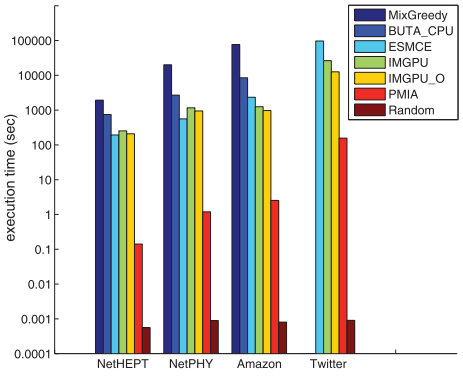


Fig. 9. Running time on four real-world data sets.

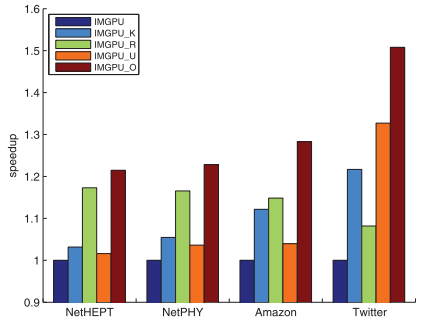


Fig. 10. Effects on different optimizations.

5.2.3 Evaluation of Optimization Methods

To evaluate the effect of proposed optimization methods, we separately test the proposed three optimization methods. We denote them as follows:

- IMGPU_K: IMGPU with K-level combination.
- IMGPU_R: IMGPU with data reorganization.
- IMGPU_U: IMGPU with memory access coalescence optimization.

The experiments are performed on four real-world social networks. All the speedups are based on the running time of baseline IMGPU. Fig. 10 presents the experimental results. IMGPU_K performs much better on large-scale networks, such as Amazon ($1.12\times$) and Twitter ($1.21\times$), for the reason that the number of levels in small networks is relatively small and there is little room to combine the computation. On the contrary, IMGPU_R achieves much performance gain for networks of small scale. The speedup on Twitter is $1.08\times$ while that on other three data sets is $1.15\times$ on average. This is because, although we can minimize possible divergence by reorganizing the graph, such a gain is obtained at the cost of presorting nodes, which will consume much time on large-scale networks. IMGPU_U shows substantial improvement on Twitter ($1.33\times$) due to the fact that there are many nodes with large amount of child nodes in Twitter. Threads processing these "hub" nodes need frequent memory accesses and thus degrade the overall performance. IMGPU_U thereby significantly reduces the cost by coalescing memory access to consecutive memory addresses. As those optimization methods are orthogonal in nature, by integrating them together, IMGPU_O achieves accumulative speedup and thus significantly outperforms the baseline IMGPU.

6 Conclusions

In this paper, we present IMGPU, a novel framework that accelerates influence maximization by taking advantage of GPU. In particular, we design a bottom-up traversal algorithm, BUTA, which greatly reduces the computational complexity and contains inherent parallelism. To adaptively fit BUTA with the GPU architecture, we also explore three effective optimizations. Extensive experiments demonstrate that IMGPU significantly reduces the execution time of the existing sequential influence maximization algorithm while maintaining satisfying influence spread. Moreover, IMGPU shows better scalability and is able to scale up to large-scale social networks.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. This research was supported partly by KJ-12-06, NSFC under grant no. 61272483, 61272056, 61272482, and 61170285, as well as NAP grant of Nanyang Technological University under M4080738.020.

References

[1] J. Barnat, P. Bauch, L. Brim, and M. Ceska, "Computing Strongly Connected Components in Parallel on CUDA," *Proc. IEEE 25th Int'l Parallel Distributed Processing Symp. (IPDPS)*, pp. 544-555, 2011.

[2] D. Bader and K. Madduri, "GTgraph: A Suite of Synthetic Graph Generators," http://www.cse.psu.edu/madduri/software/GTgraph/, Nov. 2012.

[3] N. Bell and M. Garland, "Efficient Sparse Matrix-Vector Multiplication on CUDA," Technical Report NVR-2008-04, NVIDIA, Dec. 2008.

[4] W. Chen, Y. Wang, and S. Yang, "Efficient Influence Maximization in Social Networks," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 199-208, 2009.

[5] W. Chen, C. Wang, and Y. Wang, "Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 1029-1038, 2010.

[6] E. Cohen, "Size-Estimation Framework with Applications to Transitive Closure and Reachability," *J. Computer and System Sciences*, vol. 55, no. 3, pp. 441-453, 1997.

[7] P. Domingos and M. Richardson, "Mining the Network Value of Customers," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 57-66, 2001.

[8] A. Goyal, W. Lu, and L.V.S. Lakshmanan, "CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks," *Proc. Int'l Conf. World Wide Web (WWW)*, pp. 47-48, 2011.

[9] P. Harish and P.J. Narayanan, "Accelerating Large Graph Algorithms on the GPU Using CUDA," *Proc. High Performance Computing (HiPC)*, pp. 197-208, 2007.

[10] Q. Jiang, G. Song, G. Cong, Y. Wang, W. Si, and K. Xie, "Simulated Annealing Based Influence Maximization in Social Networks," *Proc. 25th AAAI Int'l Conf. Artificial Intelligence (AAAI)*, pp. 127-132, 2011.

[11] K. Jung, W. Heo, and W. Chen, "IRIE: A Scalable Influence Maximization Algorithm for Independent Cascade Model and Its Extensions," *CoRR arXiv rapid post arXiv:1111.4795*, pp. 1-20, http://arxiv.org/abs/1111.4795/, 2011.

[12] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the Spread of Influence through a Social Network," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 137-146, 2003.

[13] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a Social Network or a News Media?" *Proc. Int'l Conf. World Wide Web (WWW)*, pp. 591-600, 2010.

[14] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 177-187, 2005.

[15] J. Leskovec, L. Adamic, and B. Huberman, "The Dynamics of Viral Marketing," *ACM Trans. Web*, vol. 1, no. 1, p. 5, May 2007.

[16] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective Outbreak Detection in Networks," *Proc. 13th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2007.

[17] X. Liu, S. Li, X. Liao, L. Wang, and Q. Wu, "In-Time Estimation for Influence Maximization in Large-Scale Social Networks," *Proc. ACM EuroSys Workshop Social Network Systems*, pp. 1-6, 2012.

[18] S.H. Nobari, X. Lu, P. Karras, and S. Bressan, "Fast Random Graph Generation," *Proc. 14th Int'l Conf. Extending Database Technology (EDBT)*, pp. 331-342, 2011.

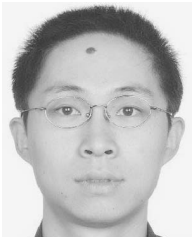
[19] M. Richardson and P. Domingos, "Mining Knowledge-Sharing Sites for Viral Marketing," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pp. 61-70, 2002.

[20] V. Vineet, P. Harish, S. Patidar, and P.J. Narayanan, "Fast Minimum Spanning Tree for Large Graphs on the GPU," *Proc. High Performance Graphics Conf. (HPG),* pp. 167-171, 2010.
[21] Y. Wang, G. Cong, G. Song, and K. Xie, "Community-Based Greedy Algorithm for Mining Top-K Influential Nodes in Mobile Social Networks," *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining (SIGKDD),* pp. 1039-1048, 2010.
[22] R. Zafarani and H. Liu, "Social Computing Data Repository at ASU," http://socialcomputing.asu.edu/, Nov. 2012.

Xiaodong Liu received the BS and MS degrees from the School of Computer Science, National University of Defense Technology, Changsha, China, in 2007 and 2009, respectively. He is currently working toward the PhD degree at National University of Defense Technology. His main research interests include parallel computing, social network analysis and large-scale data mining and so on. He is a student member of the ACM.

Mo Li received the BS degree in the Department of Computer Science and Technology from Tsinghua University, Beijing, China, in 2004 and the PhD degree in the Department of Computer Science and Engineering from Hong Kong University of Science and Technology. Currently, he is working as an assistant professor in the School of Computer Engineering of Nanyang Technological University of Singapore. He won ACM Hong Kong Chapter Professor Francis Chin Research Award in 2009 and Hong Kong ICT Award Best Innovation and Research Grand Award in 2007. His research interests include sensor networking, pervasive computing, mobile and wireless computing and so on. He is a member of the IEEE and the ACM.

Shanshan Li received the MS and PhD degrees from the School of Computer Science, National University of Defense Technology, Changsha, China, in 2003 and 2007, respectively. She was a visiting scholar at Hong Kong University of Science and Technology in 2007. She is currently an assistant professor in the School of Computer, National University of Defense Technology. Her main research interests include distributed computing, social network, and data center network. She is a member of the IEEE and the ACM.



Shaoliang Peng received the PhD degree from the School of Computer Science, National University of Defense Technology, Changsha, China, in 2008. Currently, he is an assistant professor at National University of Defense Technology. His research interests include distributed system, high performance computing, cloud computing, and wireless networks. He is a member of the IEEE and the ACM.

Xiangke Liao received the BS degree in the Department of Computer Science and Technology from Tsinghua University, Beijing, China, in 1985, and the MS degree from National University of Defense Technology, Changsha, China, in 1985. He is currently a full professor and the dean of School of Computer, National University of Defense Technology. His research interests include parallel and distributed computing, high-performance computer systems, operating systems, cloud computing, and networked embedded systems. He is a member of the IEEE and the ACM.

Xiaopei Lu received the BS degree from Tsinghua University, Beijing, China, in 2006, and the MS degree from National University of Defense Technology, Changsha, China, in 2008. He is currently working toward the PhD degree at the School of Computer Science, National University of Defense Technology. His research interests include distributed systems, wireless networks, and high-performance computer systems. He is a student member of the IEEE.