Simulation of Information Propagation over Complex Networks: Performance Studies on Multi-GPU

Jiangming Jin, Stephen John Turner, Bu-Sung Lee, Jianlong Zhong, Bingsheng He
School of Computer Engineering, Nanyang Technological University, Singapore
{s080056, assjturner, ebslee, jzhong2, bshe}@ntu.edu.sg

***Abstract*—General Purpose Graphics Processing Units (GPGPU) have been used in high performance computing platforms to accelerate the performance of scientific applications such as simulations. With the increased computing resources required for large-scale network simulation, one GPU device may not have enough memory and computation capacities. It is therefore necessary to enhance the system scalability by introducing multiple GPU devices. It is also attractive to investigate the performance scalability of Multi-GPU simulations. This paper describes the simulation of information propagation on multiple GPU devices, including the optimized network simulation algorithms, the network partitioning and replication strategy, and the data synchronization scheme. The experimental results for scalable random networks show that the number of simulation steps, computation time, synchronization time, and data transfer time all affect the overall simulation performance. In order to compare with random networks, we also conduct simulations of scale-free networks. We can observe that the node replication ratio in scale-free networks is smaller than that in random networks and therefore the cost of data transfer and synchronization is significantly reduced. This indicates that the network structure is also an important factor that influences the simulation performance in a Multi-GPU system.**

***Index Terms*—Information Propagation, Multi-GPU, Distributed Simulation, Performance Analysis**

I. Introduction

The simulation of network-based information propagation has become a very important method to discover the propagation behaviors over complex networks. For example, simulations can be applied on a complex social network to explain how the network structure influences the propagation behaviors among the social entities over time. However, large-scale network simulations often require large computation and memory resources. It is therefore necessary to develop performance-oriented simulation techniques and map those optimized simulation methods onto high performance computing (HPC) platforms. For complex networks, the network structures are usually irregular due to the complicated relationships among the nodes. Such irregularity of the data structure may exhibit very poor locality and excessive memory access. Thus, the underlying network layout in the memory needs to be carefully optimized. It is an open research question how to optimize the data structure in the memory so that benefits can be obtained in both temporal and spatial domains during simulation.

Recently, the GPU has been widely applied in many scientific applications for general purpose computing [18]. With the emergence of new generations of GPUs, both the computation capacity and the memory bandwidth have exceeded the CPUs. Therefore General Purpose GPU (GPGPU) with many-core processors are used as accelerators for various applications, including network processing. The Multi-GPU system can provide more computing resources to process large-scale network simulations with massive parallelism. However, multiple GPUs also introduce extra communication cost between different GPU devices to handle the data consistency since the original network is partitioned into different GPU devices. The excessive data communication as well as the data synchronization may degrade the simulation performance.

In this paper, we introduce two optimized simulation algorithms based on the corresponding network structures. They are: (1) the Adaptive Vertex-Oriented Processing Loop (A-Loop) on the Vertex-Oriented Structure and (2) the Edge-Oriented Processing Loop (E-Loop) on the Edge-Oriented Structure. In addition, the A-Loop is derived from algorithmic adaptation between two basic information propagation models, the Independent Cascade Model [5] and the Linear Threshold Model [6]. According to the studies on model modification and the equivalence between the two basic models [10], we have developed optimization strategies that enable algorithmic adaptation at runtime in order to choose the most efficient algorithm at each simulation step [8].

We also note that the memory space of a single GPU device is not enough to contain very large networks. The space limit of a single GPU memory may become a limitation that constrains the scalability of GPU simulations. It is therefore necessary to design a simulation paradigm on multiple GPU devices for large-scale network simulation. Although the Multi-GPU system introduces a synchronization overhead that may reduce the parallel performance, this is acceptable if it is more important to improve memory scalability than to obtain a performance gain. With this motivation, we show a Multi-GPU simulation design for information propagation over complex networks in this paper. To obtain efficient simulation performance on multiple GPU devices, an efficient network partitioning and replication method is important. It should not only balance the workload in each GPU device but also reduce the data communication and synchronization cost. Metis [12], a widely used graph partitioning tool, is adopted for partitioning the input network and generating a partition file as the output. According to the original network file and the partition file, we propose a replication scheme to store the information of boundary nodes and the corresponding cross-partition edges. The specified network structure (vertex-oriented or edge-oriented) can then be built in different GPU devices. The boundary and replica nodes in different devices need to be synchronized in order to maintain the network state consistency at each simulation step.

According to the simulation performance of large scalable random networks (generated by GTgraph [1]) on multiple GPU devices, we observe that there are four factors that influence the performance: (1) the number of simulation steps, (2) computation time, (3) synchronization time, and (4) data transfer time. Furthermore, in order to investigate how simulation performance is influenced by different network structures, we generate a set of scale-free networks using the RMAT method [2] in GTgraph. With the performance comparison between random networks and scale-free networks, we can observe that the node replication ratio in scale-free networks is significantly reduced compared to random networks. Therefore, the cost of data transfer and synchronization at the end of each simulation step is reduced.

The rest of this paper is structured as follows. Some background information is introduced in section II. This is followed by a description of the optimized simulation algorithms and network structures in section III. In section IV, the detailed system design and synchronization scheme for Multi-GPU simulation are presented. In section V, we evaluate the experimental results of scalable random network simulation performance on multiple GPU devices. In section VI, we investigate how the network structure influences the Multi-GPU simulation performance by comparing random networks and scale-free networks. Finally, conclusions and future work are discussed in section VII.

II. Background

A. Simulation of Information Propagation

In conventional network studies using mathematical models, the individual is essentially ignored [16] [17]. However, many different types of relationships or attribute combinations usually form a complex network structure. It is therefore necessary to apply advanced agent-based modeling and simulation methods to analyze these behaviors. The simulation of information propagation aims to investigate the interactive behaviors between $Active$ nodes and $Inactive$ nodes within a given network. Currently, the independent cascade model and the linear threshold model are widely used in studying the behaviors of information propagation over networks. In the independent cascade model, we define an initial set of active nodes A_0 at step 0. The information propagation unfolds in discrete time steps: at step t, the newly active node v_i has a single chance to activate its inactive neighbor u with an independent probability $p_{(v_i,u)} \in [0,1]$. If v_i succeeds in activating u, u will transit its state from inactive to active at step $t+1$ [5]. Such a process continues until no more possible activations are available. In the linear threshold model, each node on the network is randomly assigned a threshold $T_u \in [0,1]$ with a specific probability distribution (usually a uniform distribution). At step t, each inactive node is influenced by its active neighbors (a set A_t, where A_t is ø if no active neighbor exists). The influence weight between the active node v_i and the inactive node u can be expressed as an independent probability $b_{(v_i,u)}$. Thus, node u's influence weight from its active neighbors can be calculated and represented by $\sum_{i=1}^{l} b(v_i,u)$, where l denotes the number of active neighbors [6]. If $\sum_{i=1}^{l} b(v_i,u) > T_u$, u's state will transit from inactive to active at step $t+1$.

B. GPU Architecture

As illustrated in [18], many-core streaming GPUs can also be used to perform computation in applications traditionally handled by CPUs. With a GPU parallel computing model such as CUDA [3], developers can ease their programming complexity to use stream processing on non-graphics data. Essentially, a GPGPU consists of multiple streaming multiprocessors (SMs) and each SM consists of multiple streaming processors (SPs). For example, a Nvidia Fermi C2050 GPU [14] card has 14 SMs and each SM consists of 32 SPs. A set of 32 concurrent threads are grouped into a scheduling unit, called a warp. The 448 CUDA cores can share 3 GBs device memory with bandwidth of 144 GBs/sec. However, the high bandwidth utilization is achieved by coalesced memory access. A memory request issued by a warp can be coalesced into one memory transaction if all the memory addresses fall into the same cache line of GPU memory with size of 128 bytes. Otherwise, the number of memory transactions is equal to the number of cache lines accessed. With the massive parallelism, GPUs have been used to accelerate graph processing which is relevant to network simulation. Some research works investigate the design and implementation of several graph algorithms such as Breadth-First Search (BFS) and Single Source Shortest Path (SSSP) [7][11][19]. In our previous work [8], we have shown that an average 15x parallel speedup can be gained on one Fermi C2050 GPU device compared to the serial CPU performance.

III. Simulations of Information Propagation over Complex Networks

A. Algorithm Description

In this paper, we assume that the node's threshold is equal to the system threshold $T = 1 - P$, where P is the transmissibility probability, such as the transmissibility in SIR infectious disease studies on SARS [13]. According to the model definition, we first introduce two types of agent-based simulation algorithms named as C-Loop, and T-Loop. The C-Loop and T-Loop can be considered as general simulation approaches to represent the cascade model and the threshold model respectively.

C-Loop: Starting from the active nodes in the network, each active node will go through its contact neighbors at each simulation step and test whether it can propagate the information to the inactive neighbors with a specific probability. If the inactive nodes receive the information, they will change state to be active at the next step.

T-Loop: In contrast to the C-Loop, the T-Loop starts from the inactive nodes and traces the contact neighbors' state at each step. The inactive node can be activated at the next step by any active neighbor if the transmissibility with a specific probability P is satisfied.

We can define the following notations: V: vertex set, E: edge set, V^{dst}: corresponding destination vertex set of the source vertex, src_i: state of the source vertex i, dst_j: state of the destination vertex j (for the E-Loop, src_i and dst_i are the states of the source and destination vertex associated with edge i), p: transmission probability.

1) Adaptive Vertex-Oriented Processing (A-Loop): According to the model equivalence between the C-Loop and the T-Loop, we can carry out the algorithmic adaptation at the proper time to obtain the best simulation performance gain. Since only one or a few active nodes are introduced into the network in the beginning, we can choose the C-Loop first and swap to the T-Loop at the simulation step where the workload of the T-Loop is less than that of the C-Loop [8].

Algorithm 1 A-Loop

```
Initialization:
    Set C-Loop as the Initial Configuration
    Set $AdaptFlag = false$
Iteration:
 1: if $AdaptFlag == false$ then
 2:     Process Network with C-Loop
 3:     for $i = 1 \to |V|$ do
 4:         if $src_i == Active$ then
 5:             for $j = 1 \to |V^{dst}|$ do
 6:                 if $dst_j == Inactive$ then
 7:                     $src_i \to dst_j$ with $p$
 8:                 end if
 9:             end for
10:         end if
11:     end for
12:     if the Adaptation Condition is Satisfied then
13:         Set $AdaptFlag = true$
14:     end if
15: else
16:     Process Network with T-Loop
17:     for $i = 1 \to |V|$ do
18:         if $src_i == Inactive$ then
19:             for $j = 1 \to |V^{dst}|$ do
20:                 if $dst_j == Active$ then
21:                     $dst_j \to src_i$ with $p$
22:                 end if
23:             end for
24:         end if
25:     end for
26: end if
```

2) Edge-Oriented Processing (E-Loop): We can also process the network simulation using an edge-oriented approach. The E-Loop starts from each edge element and checks the states of the connected pair of nodes. If the two connected nodes have a different state such as *Active-Inactive*, the information can be propagated from the active to the inactive node with the given probability. The E-Loop on the edge-oriented structure is an algorithm with constant computational complexity in network browsing at each simulation step.

Algorithm 2 E-Loop

```
Iteration:
 1: for $i = 1 \to |E|$ do
 2:     if $src_i == Active$ then
 3:         if $dst_i == Inactive$ then
 4:             $src_i \to dst_i$ with $p$
 5:         end if
 6:     else
 7:         if $dst_i == Active$ then
 8:             $dst_i \to src_i$ with $p$
 9:         end if
10:     end if
11: end for
```

B. Structures of Network Storage

In this sub-section, we introduce the two different types of data structures for the network storage in hardware.

1) Vertex-Oriented Structure: Figure 1 illustrates the vertex-oriented structure. With this structure, each node has a contact list for its connected neighbors, and it also appears in its neighbors' contact list. That is, we need to keep a bidirectional graph for network storage for efficient processing. Different from directed graphs, we use the state of the vertex attribute (*Active or Inactive*) instead of the directed edge to determine the direction of information flow between two connected nodes.

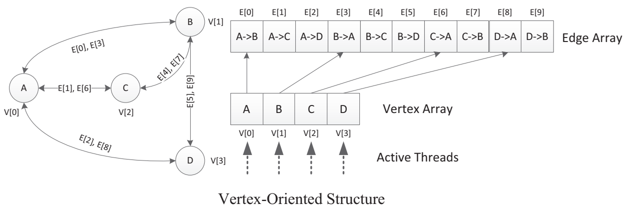

Fig. 1: Vertex-Oriented Structure

2) Edge-Oriented Structure: With this structure, simulation starts from the edge array and checks the connected pair of nodes' states. Since the network can be considered as an undirected graph, we only need one copy of each edge instead of two to represent the network structure. It is therefore possible to reduce spatial complexity in the network storage. In practice, the memory space of the edge-oriented structure is approximately one half of the vertex-oriented structure. The edge-oriented structure is shown in Figure 2.

IV. System Design of Multi-GPU Simulation

In this section, we introduce the method to partition one logical network to multiple physical partitions on GPU devices. Furthermore, we describe the synchronization algorithm between different GPU devices that is executed at the end of each simulation step.

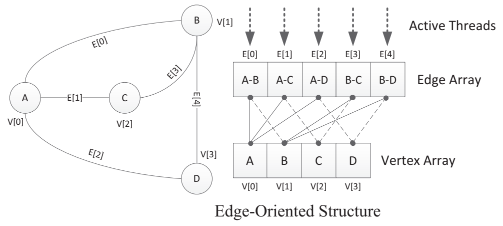

Fig. 2: Edge-Oriented Structure

A. Network Partitioning

In order to perform the network partitioning using Metis [12], both the vertex-oriented and the edge-oriented structures are considered as directed graphs. Metis balances the number of vertices in each partition and also minimizes the number of edges cut in order to minimize the data replication and cost of synchronization. As the result of graph partitioning, each cross-partition edge is cut and the connected *head-vertex* and *tail-vertex* are placed into two different partitions. It is therefore necessary to store the cross-partition edge and associated nodes' information in the proper partition.

For both vertex-oriented processing and edge-oriented processing, we maintain replicas for the head vertex of each cross-partition edge in the partition where the tail vertex resides. Also, each cross-partition edge is stored in its tail partition. Thus, we can process our network simulation on different GPU devices independently.

1) Partitioning with a Vertex-Oriented Structure: With a vertex-oriented structure, the complex network is represented by a bidirectional graph. For each edge that is cut, the nodes connected by that edge are both head vertices and tail vertices. As shown in Figure 3 (where the shaded circles denote replicas), the partitioning and replication scheme can ensure the correct interaction between the active and the inactive nodes. For example, suppose A is an active node in Partition 1 and node I is an inactive node in Partition 2. As the simulation executes, the C-Loop processes $A \to I$ in Partition 2 and the T-Loop processes $I \to A$ in Partition 1.

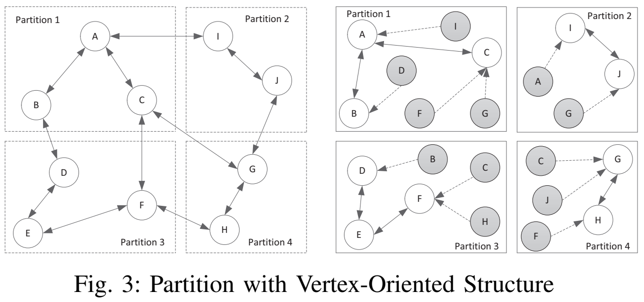

Fig. 3: Partition with Vertex-Oriented Structure

2) Partitioning with an Edge-Oriented Structure: As shown in Figure 4, we have only one copy of the edge between a pair of vertices in the edge-oriented structure. However, for the purpose of partitioning, we treat the graph as unidirectional, and for each edge that is cut, we store a replica of the head vertex (e.g. node I) in the tail vertex (e.g. node A) partition. During simulation execution, the direction of the edge is not important. With the edge-oriented structure, the number of replicas is less than that in the vertex-oriented structure. However, in some very large-scale complex networks, there are many cross-partition edges between vertices. Thus, nearly every vertex is replicated in each of the other partitions. In this case, the number of replicas may be almost the same as that in the vertex-oriented structure.

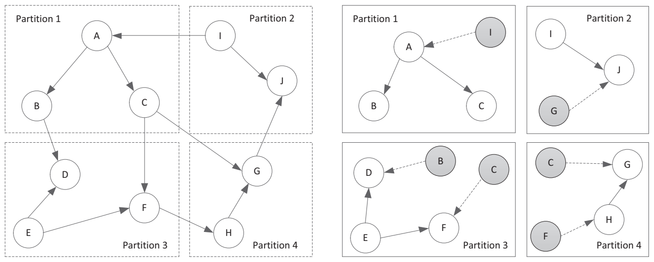

Fig. 4: Partition with Edge-Oriented Structure

In Multi-GPU simulation, each GPU device maintains a *Boundary Table* and a *Replica Table*. The *Boundary Table* stores the boundary vertices (nodes) in its partition which have replicas in other partitions that point to tail vertices. In contrast, the *Replica Table* is used to replicate the head vertices which reside in the other partitions but point to vertices in its own partition. As shown in Figure 5 (a), GPU 1 maintains a boundary table which consists of 3 segments for the boundary nodes with replicas that reside in GPUs 2, 3, and 4 respectively. The replica table in GPU 1 stores the replica nodes from GPUs 2, 3, and 4 respectively. Figure 5 (b) shows the boundary and replica tables of Partition 1 in the example given in Figure 3.

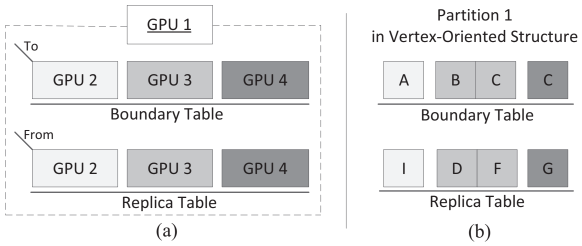

Fig. 5: Boundary and Replica Tables

B. Distributed GPU Simulation and Synchronization

Since the boundary node and its replicas in other partitions represent the same logical node in the original network, it is necessary to synchronize the state of the boundary and replica nodes at the end of each simulation step. Either the boundary or replica node may contain the updated state result. To perform the synchronization, we first need to copy the boundary nodes' state to the boundary table and copy the replica nodes's state to the replica table. We can then transfer the boundary and replica tables from the GPU devices to the *Boundary Buffer* and the *Replica Buffer* on the host CPU to process the synchronization. The boundary/replica buffer consists of boundary/replica tables from different GPU devices. The synchronization between the boundary table and replica table for 3 GPU devices is illustrated in Figure 6. The boundary buffer updates and records the correct nodes' state after the *Boundary-Replica* synchronization.

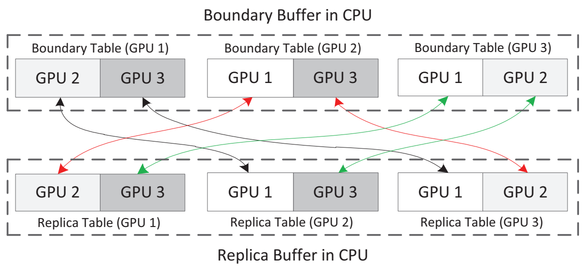

Fig. 6: Boundary-Replica Synchronization for 3 GPUs

We note that there may be multiple copies of boundary nodes that reside in the boundary table. For example, in Figure 5 (b), the node C has two copies in the boundary table of Partition 1. It is therefore necessary to synchronize the multiple copies of the boundary node after the *Boundary-Replica* synchronization. As shown in Figure 7 (a), we construct a *Redundancy Table* for each GPU device which resides in the CPU. With the same size as the boundary table, the redundancy table maintains the redundancy lists for each boundary node, where a list stores the boundary buffer index of multiple copies of that node.

As shown in Figure 5 (b), there are two copies of boundary node C which map to Partition 3 and Partition 4 respectively. Thus, each copy of C keeps the other copy's array index in the redundancy table, as shown in Figure 7 (b). According to the redundancy table of each partition, we are able to further update the boundary buffer in order to synchronize those boundary nodes with multiple copies.

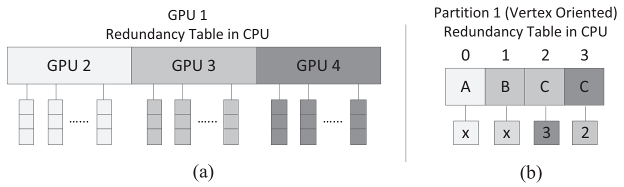

Fig. 7: Redundancy Table for Partition 1

After the synchronization process, we can obtain the updated node state in the boundary buffer on the CPU memory. We can transfer the synchronized data from the boundary buffer to each GPU's boundary table and replica table. Each GPU device can further update the boundary and replica nodes to complete the entire synchronization procedure. The details of the synchronization are illustrated in Algorithm 3.

Algorithm 3 Synchronization Algorithm

```
1:  // Data Copy on Devices (GPUs)
2:  for $i = 1 \rightarrow NumGPUs$ do
3:      Copy boundary nodes' status and replica nodes' status to
        Boundary and Replica Tables respectively
4:  end for
5:  // Data transfer from Devices to Host (GPUs $\rightarrow$ CPU)
6:  for $i = 1 \rightarrow NumGPUs$ do
7:      Transfer GPU's Boundary and Replica Tables to CPU
8:  end for
9:  // Synchronization on Host (CPU) with OpenMP
10: for $i = 1 \rightarrow NumGPUs$ do
11:     for $j = 1 \rightarrow NumSegments$ do
12:         Synchronization between Boundary and Replica nodes in
            different partitions
13:         Update the Boundary Buffer
14:     end for
15: end for
16: for $i = 1 \rightarrow NumGPUs$ do
17:     for $k = 1 \rightarrow RedundancyListSize$ do
18:         Check Redundancy Table and Synchronize multiple bound-
            ary nodes
19:     end for
20: end for
21: // Data transfer from Host to Devices (CPU $\rightarrow$ GPUs)
22: for $i = 1 \rightarrow NumGPUs$ do
23:     Transfer corresponding Boundary Buffer segments to other
        GPUs's Replica Tables
24:     Transfer corresponding Boundary Buffer segments to its GPU
        Boundary Table
25: end for
26: // Data Copy on Devices (GPUs)
27: for $i = 1 \rightarrow NumGPUs$ do
28:     Copy Boundary and Replica Tables to boundary nodes' status
        and replica nodes' status respectively
29: end for
```

V. Performance Evaluation with Random Networks

In this section, we evaluate the Multi-GPU simulation performance. The transmission probability for node activating is set to be 0.01. The experiments are conducted on a server with two Intel Xeon E5645 CPUs and four Fermi C2050 GPU devices. First, we show a case study on a random network with 1 million nodes and 256 million edges (1m256m). This is followed by a performance investigation on scalable networks.

A. Case Study with a 1m256m Random Network

The propagation behaviors on a 1m256m random network with both vertex-oriented processing and edge-oriented processing are shown in Figure 8. As each GPU now performs its own random number generation [4], the state results may not be exactly the same for different numbers of GPU devices. However, the state results are very similar at each simulation step. We note that all simulations converge within 20 steps. Compared to the edge-oriented structure, the vertex-oriented structure of the 1m256m network is too large to be executed

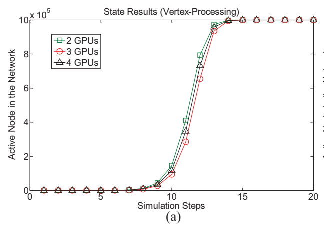

Fig. 8: 1m256m State Results during Simulation

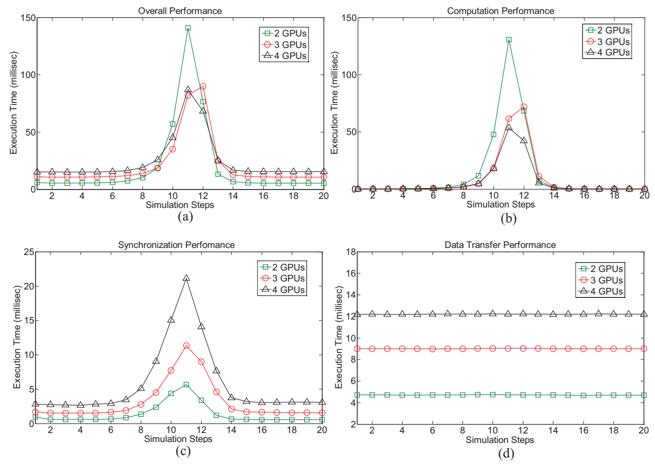

Fig. 9: Vertex Processing on 1m256m Random Networks

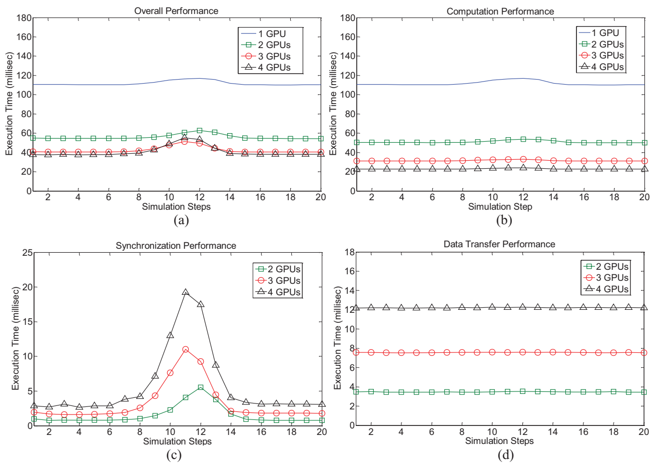

Fig. 10: Edge Processing on 1m256m Random Networks

on one GPU device. The simulation performance with vertex-oriented processing and edge-oriented processing is shown in Figure 9 and Figure 10 respectively. The execution times shown are for network browsing only and do not include the random number generation. The detailed analysis of random number generation in a Multi-GPU system can be found in [9]. In these figures, the overall execution time is broken down into separate times for computation, synchronization on CPU, and data transfer between CPU and GPU.

As shown in Figure 9 (a) and (b), and Figure 10 (a) and (b), the computation time still dominates the overall performance both in vertex-oriented processing and edge-oriented processing. We also note that the multiple GPUs can share the original computation workload.

However, by increasing the number of GPU devices, the cost of data synchronization and the cost of data transfer between CPU and GPUs grows significantly. Figure 9 (c) and Figure 10 (c) indicate that the cost of synchronization increases with an increasing number of newly active nodes in the network and decreases as the propagation on the network eventually converges. This is comparable with the network propagation behavior shown in Figure 8. That is, most of the nodes are activated at step 11 and step 12, and the large number of newly active nodes may result in a large amount of state differences between boundary and replica nodes. Thus, the maximum synchronization cost appears between step 11 and step 12. Furthermore, according to the experimental results, we note that the more GPU devices we use, the higher the synchronization cost.

In Figure 9 (d) and Figure 10 (d), we show the data transfer performance between CPU and GPUs. Since the boundary and the replica tables are pre-built before simulation, the data transfer performance is quite stable at each simulation step. Interestingly, for this 1m256m random network, we note that the synchronization performance and the data transfer performance with vertex-oriented processing are comparable with edge-oriented processing. It indicates that in networks where the number of edges per node is very large, the number of boundary and replica nodes in the edge-oriented structure and the vertex-oriented structure are very similar since nearly every node is replicated in each of the other partitions.

As shown in Figure 11, according to the simulation performance at each step, we can calculate the total execution time for the entire simulation. With vertex-oriented processing, the simulation takes 397.9ms on 2 GPUs, 418.8ms on 3 GPUs, and 488.4ms on 4 GPUs. In contrast, with edge-oriented processing, the simulation spends 2234.9ms on 1 GPU, 1120.0ms on 2 GPUs, 846.9ms on 3 GPUs, and 816.4ms on 4 GPUs. The experimental results indicate that vertex-oriented processing can obtain the best performance gain on 2 GPUs. We also observe that edge-oriented processing shows excellent speedup on multiple GPUs compared to that of vertex-oriented processing.

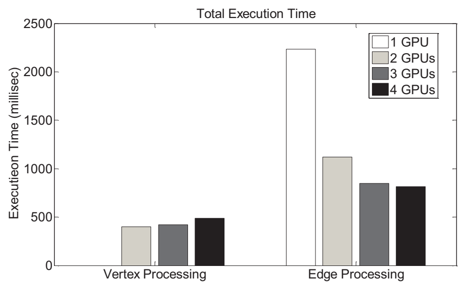

Fig. 11: Total Execution Time of 1m256m Random Network

B. Performance Studies with Scalable Random Networks

In this subsection, we investigate the Multi-GPU simulation performance on scalable networks. In order to show the performance scalability, we conduct two sets of experiments. They are: (1) *Node-Scaling* experiments: we fix the number of edges but scale up the number of nodes, and (2) *Edge-Scaling* experiments: we fix the number of nodes but scale up the number of edges ("m" denotes "million").

- In *Node-Scaling* experiments, we have 1m128m, 2m128m, 4m128m, 8m128m random networks
- In *Edge-Scaling* experiments, we have 4m32m, 4m64m, 4m128m, 4m256m random networks

We show the total time of edge-oriented processing in Figure 12 (a) and (b), and the total time of vertex-oriented processing in Figure 12 (c) and (d), excluding time for random number generation. Since the number of simulation steps may differ for different sizes of networks, this affects the total time (sum of the execution times for each simulation step). We therefore show the number of simulation steps on each network in the figure.

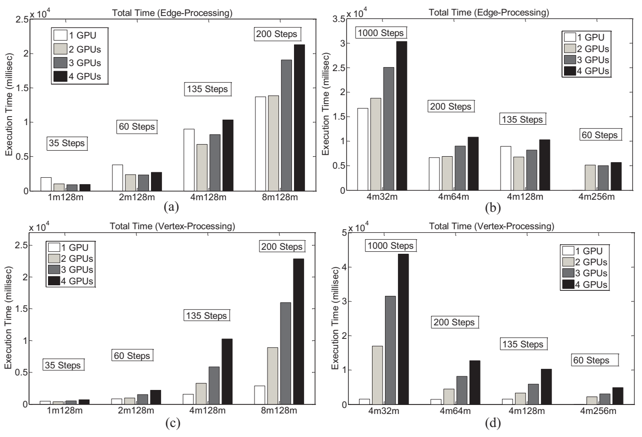

Fig. 12: Total Execution Time

In the *Node-Scaling* experiments, we observe that the number of simulation steps becomes larger as the number of nodes increases, from 35 steps (1m128m network) to 200 steps (8m128m network). Note that on networks with a smaller number of nodes, we can obtain a performance gain by multiple GPU devices for edge-oriented processing. In particular, we can achieve better performance with 2 GPUs for networks with 1m, 2m, and 4m nodes than that with a single GPU device. Compared to the constant network processing complexity in edge-oriented processing, the vertex-oriented processing can achieve optimized computational complexity by using the adaptive A-Loop algorithm. Thus, the cost of synchronization and data transfer may dominate the overall performance with multiple GPU devices. Furthermore, with an increased number of GPU devices, the cost of synchronization and data transfer increase significantly and may become performance bottlenecks and degrade the Multi-GPU simulation performance.

In the *Edge-Scaling* experiments, we fix the number of nodes but increase the number of edges. We note that with the very large 4m256m network, a single GPU device is not enough to contain the network structure. We also note that the number of simulation steps decreases as the number of nodes increases, from 1000 steps (4m32m network) to 60 steps (4m256m network). Due to the increasing number of edges between the same number of nodes, the propagation can converge in fewer simulation steps. As shown in Figure 12 (b) and (d), multiple GPUs with edge-oriented processing can share the computational workload. However, there is a trade-off between the computation performance improvement and the synchronization and data transfer cost. Compared to edge-oriented processing, the cost of synchronization and the cost of data transfer in vertex-oriented processing are the dominant factors that influence the simulation performance.

According to the performance comparison between Single-GPU and Multi-GPU simulation using edge-oriented processing, we can obtain better performance by Multi-GPU on 1m128m (3 GPUs), 2m128m (3 GPUs), 4m128m (2 GPUs), and 4m256m (3 GPUs) networks respectively. In vertex-oriented processing, we can obtain a performance gain by Multi-GPU only on the 1m128m (2 GPUs) network.

Essentially, there are four factors which affect the simulation performance, *Stimulation Steps*, *Computation Time*, *Synchronization Time*, and *Data Transfer Time*. It is therefore necessary to discuss each factor in detail.

The number of simulation steps measures how long the information propagation takes to converge. Such a convergence behavior is highly influenced by the average number of edges per node in random networks. However, in scale-free networks [15], the convergence behavior is also influenced by a few nodes which have a very large number of edges compared to others. In the experiments with random networks, we note that the number of simulation steps varies according to the average number of edges per node. In practice, all the networks can converge within the number of simulation steps shown in the figures except the 4m32m network. In particular, the simulation of the 4m32m network can propagate to 99.7% nodes within 400 steps and eventually reach to 99.9% nodes at 1000 steps. Therefore, we can conclude that if we double the average number of edges per node, the number of convergence steps is roughly halved.

The total computation time on GPU (sum of the computation times for each simulation step, excluding the data transfer and synchronization time at the end of each step) is shown in Figure 13. We can observe that the workload of network processing can be shared by multiple GPU devices, both in edge-oriented processing and in vertex-oriented processing. However, compared to the total execution time shown in Figure 12, the cost of synchronization and data transfer may become a bottleneck in Multi-GPU processing. In Figure 13 (a), the number of simulation steps is the main factor which affects the edge-oriented execution time in *Node-Scaling* experiments. However, as shown in Figure 13 (b), the computation time in *Edge-Scaling* experiments is influenced not only by the number of simulation steps but also by the number of edges. In vertex-oriented processing, the computation time is obviously influenced by the number of nodes. More importantly, we can observe that the Multi-GPU computation performance is highly optimized but the total performance is dominated by the cost of synchronization and data transfer.

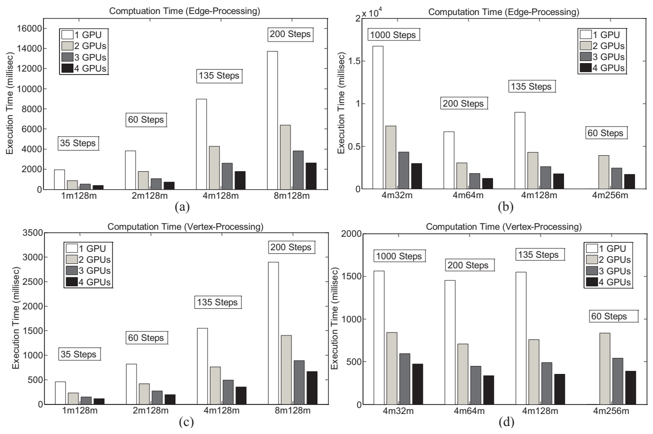
Fig. 13: Total Computation Time on GPU

We also identify the maximum synchronization time during simulation, which is shown in Figure 14. As we discussed in the case study of the 1m256m network, the maximum synchronization time appears in the middle of the simulation when many nodes are newly activated. In the *Node-Scaling* experiments, the maximum synchronization time grows significantly when increasing the number of nodes. In the *Edge-Scaling* experiments, with an increasing number of edges, the number of cross-partition edges also increases. Thus, the number of boundary nodes and replica nodes increases as well. This results in a high synchronization cost since more boundary-replica nodes need to be synchronized when we scale up the number of edges. The experimental results also indicate that the synchronization cost highly depends on the number of partitions.

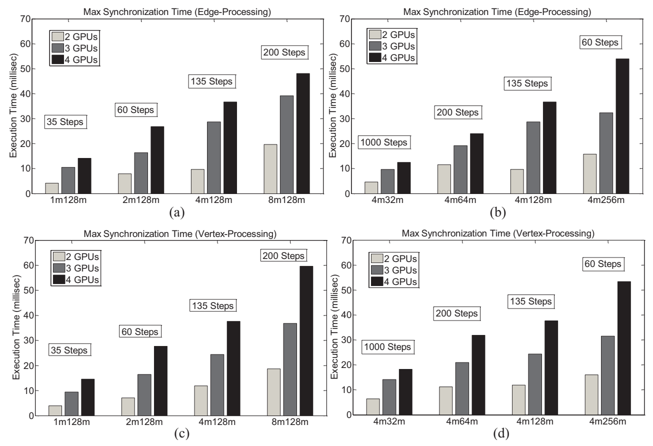
Fig. 14: Maximum Synchronization Time

The data transfer time measures the time spent to collect data from GPUs and the time spent to transfer the updated data from CPU back to GPUs. As the performance of data transfer only depends on the size of the boundary table and the size of replica table in the GPU devices, the *Data Transfer Time* is quite stable at each step. We can therefore measure the data transfer performance according to the average value at each simulation step, which is shown in Figure 15. According to the experimental results, we note that scaling the network size and changing the number of partitions change the size of boundary and replica tables. Furthermore, the boundary table size and the replica table size directly affect the data transfer performance.

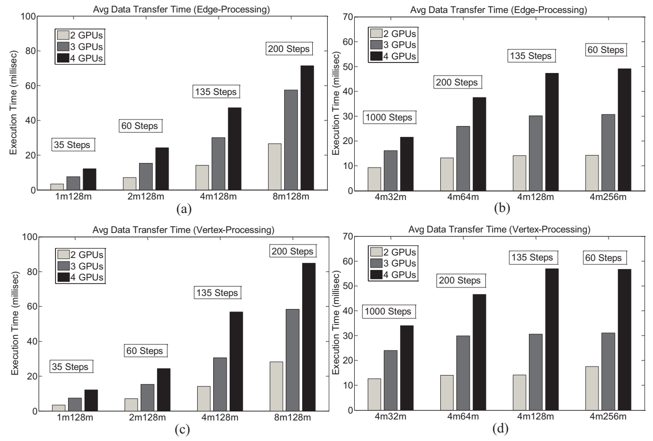
Fig. 15: Average Data Transfer Time

According to the synchronization performance we have discussed above, we note the maximum synchronization cost and the average data transfer cost are directly influenced by the same factors, including the size of the boundary and replica tables, and the number of partitions. Therefore, we can define a metric to investigate the data replication performance in the Multi-GPU system. The *Replication Ratio* R denotes the average node replication, where $R = \frac{ReplicaBufferSize}{NumberofNodes}$. Given a network which is partitioned into P partitions, we can have $R \in [0, P-1]$. Here, $R = 0$ if there are no edges between different partitions. However, $R = P-1$ if each node is replicated in each of the other $P-1$ partitions. To compare the replication ratio between the vertex-oriented structure and the edge-oriented structure in the Multi-GPU system, we show the replication ratio of scalable networks in Figure 16.

We can observe that the replication ratio increases with an increased number of partitions. In addition, the vertex-oriented structure shows a higher replication ratio than the edge-oriented structure in the cases where the average number of edges per node is relatively small, both in the node-scaling experiments and in the edge-scaling experiments. However, in the cases with a large number of edges per node, the vertex-oriented structure and the edge-oriented structure show a similar performance in data replication. That is, the replication ratio R approximates to $P - 1$.

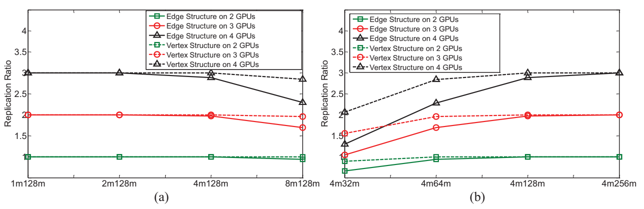

Fig. 16: Replication Ratio of Scalable Networks

C. Discussion of Simulation Performance of Random Networks on Multi-GPU

According to the detailed performance analysis of scalable random networks, we can observe that the replication ratio of random networks is high. The high replication ratio results in an expensive cost of data transfer and synchronization at the end of each simulation step. It offsets the massive processing capability of multiple GPU devices. Furthermore, in the simulations with vertex-oriented processing, we note that the cost of data transfer and synchronization becomes a bottleneck and dominates the total execution time. As the adaptive simulation algorithm is optimized according to its algorithmic complexity, the pure computation time on GPU is very small. It approximates to zero in the beginning and the end of the simulation (e.g. Figure 9 (b)). The Multi-GPU system can only show its computing advantages in the middle of the simulation where a large network browsing workload is required for processing. In the beginning and the end of the simulation, the cost of data transfer and synchronization can be considered as a constant overhead (e.g. Figure 9 (c) and (d)). Such an overhead is accumulated with the number of simulation steps. Therefore, those vertex-oriented processing experiments with a larger number of simulation steps are significantly influenced by the cumulative cost of data transfer and synchronization at the end of each simulation step. However, simulation performance with edge-oriented processing is not significantly influenced by the number of simulation steps since all performance factors are stable at each simulation step except for the synchronization (e.g. Figure 10).

VI. Performance Studies with Scale-free Networks

We have noted that the high replication ratio of random networks on Multi-GPU results in an expensive cost of data transfer and synchronization. In this section, we show how the simulation performance is influenced by the network structures. We use the RMAT method to generate a set of scale-free networks. The simulation performance between random networks and scale-free networks are compared on multiple GPUs.

As shown in Figure 17, the nodes in a scale-free network show an unbalanced number of connected edges. With the power-law degree distribution, a few nodes have very large contact lists compared to others and form node clusters within the network. The number of connections between these clusters may be very small compared to the random networks with uniform degree distribution. Thus, we may benefit from the reduced cost of data transfer and synchronization. As shown in the previous section, the replication ratio is an important factor that influences the cost of data transfer and synchronization. Such a cost of data transfer and synchronization at each simulation step further influences the performance at each step.

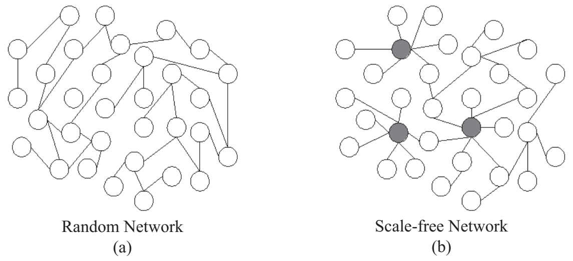

Fig. 17: Random and Scalable Networks

The replication ratio of certain scale-free networks is shown in Figure 18. It shows that the replication ratio of the scale-free networks is smaller than that of the random networks.

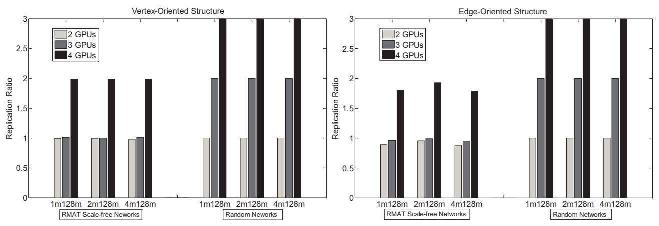

Fig. 18: Replication Ratio Comparison

In addition, it is very interesting that for the scale-free networks the replication ratio of 3 GPUs is close to that of 2 GPUs. With the detailed analysis of the replica buffer, we can observe there are only a few boundary nodes in one of the three partitions so that the total size of the replica buffer of 3 GPUs is comparable with that of 2 GPUs. Thus, the cost of data transfer and the cost of maximum synchronization on 3 GPUs are comparable with that on 2 GPUs. The performance analysis of data transfer and synchronization of a 1m128m RMAT scale-free network is shown in Figure 19.

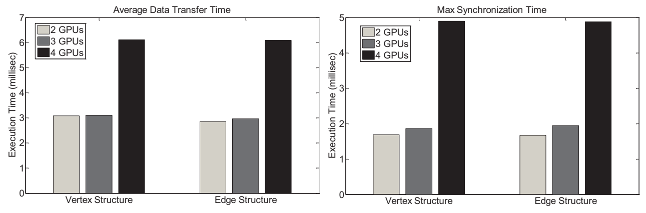

Fig. 19: Cost of Data Transfer and Synchronization (1m128m)

As the cost of data transfer and synchronization is reduced significantly in simulations of scale-free networks, we may achieve performance improvement using multiple GPU devices. The total execution time is shown in Figure 20, again excluding time for random number generation. The experiments terminate at 20 steps, 37 steps and 85 steps for 1m128m, 2m128m, 4m128m networks respectively. We can observe that vertex-oriented processing is still the best algorithm in simulations of scale-free networks. Furthermore, compared to the Multi-GPU vertex-oriented processing performance on random networks, we note that a performance gain can be obtained in the Multi-GPU vertex-oriented processing on scale-free networks.

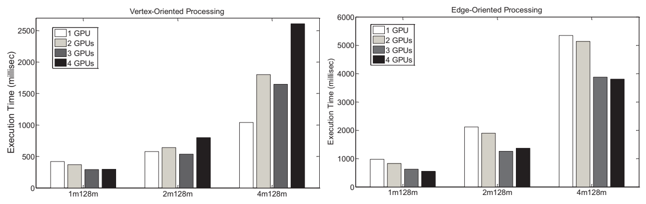

Fig. 20: Total Execution Time of RMAT Scale-free Networks

Using a baseline of single GPU vertex-oriented processing performance, the Multi-GPU system can achieve 11.3%, 30.7%, 29.1% performance improvement for simulation of the 1m128m scale-free network on 2, 3 and 4 GPUs respectively. For the simulation of the 2m128m scale-free network, we can achieve 7.4% performance improvement on 3 GPUs compared to the single GPU vertex-oriented processing. For the simulation of the 4m128m scale-free network, we cannot obtain a performance gain in vertex-oriented processing on multiple GPUs. This is because the number of simulation steps for the 4m128m network is larger than that for the 1m128m and 2m128m scale-free networks. Its cumulative cost of data transfer and synchronization during simulation increases significantly compared to the other two cases. Compared to the optimized computation time of vertex-oriented processing, the cost of data transfer and synchronization dominates the total execution time. Thus, the Multi-GPU system cannot show its computing advantage in vertex-oriented processing of the 4m128m network.

However, we can obtain performance improvement in edge-oriented processing in scale-free networks. This is because the algorithmic complexity of edge-oriented processing is approximately constant at each simulation step. Thus, we can observe that the Multi-GPU system can show its massive parallel processing advantages in simulation of information propagation. Using a baseline of single GPU edge-oriented processing performance, we can obtain an average 10.0%, 34.4%, and 36.1% performance improvement on 2, 3, and 4 GPUs.

VII. Conclusions

With the motivation of extending the space scalability of GPU simulation, we have presented the simulation performance of information propagation over complex networks on Multi-GPU. With an efficient data replication and synchronization scheme, we can map the optimized A-Loop and E-Loop simulation algorithms onto multiple GPU devices. The simulation performance is comprehensively evaluated in large scalable networks. The experimental results show that the number of simulation steps, computation time, synchronization time, and data transfer time affect the overall simulation performance. Although the Multi-GPU system introduces an overhead, this is acceptable if it is more important to enhance the system scalability for very large network simulations than to obtain a performance gain. Furthermore, we generate certain scale-free networks using the RMAT method in GTgraph. It shows that the replication ratio of scale-free networks is much lower compared to random networks. The cost of data transfer and synchronization at the end of each simulation step is also reduced significantly. Therefore, we can obtain performance improvement using multiple GPU devices.

References

[1] D. A. Bader, K. Madduri, “GTgraph: A Synthetic Graph Generator Suite”, 2006.
[2] D. Chakrabarti, Y. Zhan, C. Faloutsos, “R-MAT: A Recursive Model for Graph Mining”, In Proceedings of the fourth SIAM International Conference on Data Mining, 2004.
[3] CUDA, Nvidia “http://www.nvidia.com/object/cuda_home_new.html”, 2012.
[4] CUDA CURAND Library,“http://developer.nvidia.com/cuRAND”, 2011.
[5] J. Goldenberg, B. Libai, E. Muller, “Talk of the Network: A Complex Systems Look at the Underlying Process of Word-of-Mouth”, Marketing Letters 12(3), pp. 211-223, 2001.
[6] M. Granovetter, “Threshold Models of Collective Behavior”. American Journal of Sociology 83(6), pp. 1420-1443, 1978.
[7] P. Harish, P. J. Narayanan, “Accelerating Large Graph Algorithms on the GPU using CUDA”, In Proceedings of the 14th HiPC, pp. 197-208, 2007
[8] J. Jin, S. J. Turner, B.-S. Lee, J. Zhong, B. He, “HPC Simulations of Information Propagation over Social Networks”, Procedia CS (9), pp. 292-301, 2012.
[9] J. Jin, S. J. Turner, B.-S. Lee, J. Zhong, B. He, “Simulation Studies of Viral Advertisement Diffusion on Multi-GPU”, In Proceedings of the 2013 Winter Simulation Conference, 2013.
[10] D. Kempe, J. Kleinberg, E. Tardos, “Maximizing the Spread of Influence through a Social Network”, In Proceedings of the 9th ACM SIGKDD, pp. 137-146, 2003.
[11] L. Luo, M. Wong, W.-M. Hwu, “An Effective GPU Implementation of Breadth-first Search”, In Proceedings of the 47th DAC, pp. 52-55, 2010.
[12] Metis, “http://glaros.dtc.umn.edu/gkhome/views/metis/”, 2012.
[13] L. A. Meyers, B, Pourbohloul, M. E. J. Newman, D. M. Skowronski, R. C. Brunham, “Network Theory and SARS: Predicting Outbreak Diversity”, Journal of Theoretical Biology 232, pp. 71-81, 2004
[14] Nvidia Fermi, “NVIDIA’s Next Generation CUDA Compute Architecture: Fermi”, Nvidia Whitepaper, 2009.
[15] Scale-free Network, “http://en.wikipedia.org/wiki/Scale-free_network”, 2012
[16] J. P. Scott, “Social Network Analysis: A Handbook (2nd edition)”, Thousand Oaks, CA: Sage Publications, 2000.
[17] S. H. Strogatz, “Exploring Complex Networks”, Nature 410, pp. 268-276, 2001.
[18] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, T. J. Purcell, “A Survey of General-Purpose Computation on Graphics Hardware”, In Eruographics, State of the Art Reports, 2005.
[19] J. Zhong, B. He, “An Overview of Medusa: Simplified Graph Processing on GPUs”, In Proceedings of ACM SIGPLAN symposium on Principles and Practice of Parallel Programming 2012, pp. 283-284, 2012.