Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 292 – 301

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

HPC Simulations of Information Propagation over Social Networks

Jiangming Jin[a,1], Stephen John Turner[a], Bu-Sung Lee[a,b], Jianlong Zhong[a], Bingsheng He[a]

[a]School of Computer Engineering, Nanyang Technological University, Singapore
[b]Service Platform Lab., HP Labs Singapore, Singapore

Abstract

Simulations provide a flexible and valuable method to study the behaviors of information propagation over complex social networks. High Performance Computing (HPC) is a technology that allows the implementation of efficient algorithms on powerful new hardware resources. With the increased computing resource usage in large-scale network based simulations, it is therefore attractive to apply the emerging HPC techniques to improve the simulation performance. This paper describes optimized simulation strategies based on algorithmic adaptation at runtime, which can facilitate the performance improvement of execution. In addition, the proposed optimization method is demonstrated on HPC architectures such as Multicore CPU and General Purpose GPU (GPGPU). Such a high performance simulation approach is evaluated by a range of experiments on different network structures. The experimental results show that we can obtain significant performance gain by an optimized algorithmic adaptation strategy in a serial CPU execution environment. Furthermore, the proposed optimization method is implemented on AMD Quad-core CPU and Nvidia Fermi C2050 GPGPU for performance acceleration.

Keywords: Information propagation, Network structures, Algorithmic adaptation, HPC architectures

1. Introduction

A social network refers to a specific social structure which consists of entities (such as individuals or organizations) and the patterns/implications of the relationships between these entities [18]. Social network approaches are useful for modeling and explaining many social phenomena. A social network can be employed to model the structure of a social group, or can be used to explain how this social structure influences the behaviors among the entities over time. In conventional social 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 modeling and simulation methods to analyze these behaviors. Examples include the spread of infectious disease or innovation over complex social networks.

Large-scale network-based simulations involving information propagation often require a large amount of computing resources. At the same time, with the emergence of new hardware architectures (such as Multi-core CPU, General Purpose GPU, etc.), the computing landscape has exciting technological advancements to be explored. It is therefore necessary to develop an efficient simulation mechanism and optimization strategies that can be used in

[1]Corresponding author. Email address: s080056@ntu.edu.sg

doi:10.1016/j.procs.2012.04.031

information propagation simulation. Furthermore, such optimization techniques need to be ported onto these different advanced computer systems in order to fully reap the performance benefits.

In this paper, we show how to design two different simulation algorithms from two basic information propagation models, including the Independent Cascade Model [5] and the Linear Threshold Model [6]. Our proposed network browsing algorithms can be used to propagate information over different network structures to represent the cascade model and the threshold model. They are named as the C-Loop and the T-Loop respectively (see section 2.1). Inspired by the C-Loop and T-Loop, we introduce another simulation algorithm, named as the E-Loop. In particular, according to the algorithm behaviors, we use a vertex-oriented structure for the C-Loop and T-Loop and an edge-oriented structure for the E-Loop. According to studies on the equivalence between the two basic models [8], we are able to show the algorithmic equivalence between the corresponding C-Loop and T-Loop. Such algorithm equivalence between the C-Loop and T-Loop provides us with an opportunity to set up optimization strategies which enable algorithmic adaptation at runtime in order to choose the most efficient algorithm at each simulation step. We present the detailed algorithmic adaptation optimizations for different network structures such as Random networks and Scale-free networks [15]. The proposed optimization strategies are shown to have very good performance in practice. With the optimized algorithmic adaptation strategy (named as the A-Loop), we can achieve significant performance benefits both on AMD Quad-core CPU and Nvidia Fermi GPGPU. In addition, according to the experiments on scalable networks, the optimized A-Loop on the vertex-oriented structure shows the best performance in reducing computational complexity and the E-Loop on the edge-oriented structure shows the smallest storage occupancy in spatial complexity.

Models of Information Propagation. The simulation of information propagation is 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 status 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 a 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 status will transit from inactive to active at step $t + 1$.

HPC Architectures. The emerging multi-core CPU and many-core GPU processors are now widely used across many applications. In multi-core CPU, two or more independent processing units (cores) are integrated onto a circuit die. A multi-core CPU can run multiple instructions at the same time, which can significantly enhance the computing capacity and speedup for given applications, for example using OpenMP [12]. Many-core streaming GPU can also be used to perform computation in applications traditionally handled by CPU, so called General Purpose GPU. With the GPU environment 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 GPGPU card has 14 SMs and each SM consists of 32 SPs. The 448 CUDA cores share 3 GBs device memory with bandwidth of 144 GBs/sec.

Organization. The rest of this paper is structured as follows. The optimized strategies in modeling and simulation for information propagation are presented in section 2. This is followed by the detailed simulation design for different network structures on HPC architectures in section 3. In section 4, we evaluate the experimental results on scalable networks. Finally, conclusions are given in section 5.

2. Optimized Strategies in Modeling and Simulation for Information Propagation

In this section, we present some modifications to the original information propagation models and introduce some alternative strategies before presenting our optimization method. As described in section 1, the original independent cascade model and the linear threshold model have the constraint that only a newly active node can trigger the activation on its inactive neighbors with a specific probability. However, in many real applications such as infectious disease studies with SIR/SEIR models [7][10], any susceptible (inactive) individual has the possibility to catch the disease

from any infected (active) neighbor [11]. More importantly, the original constraint no longer allows the study of issues such as heterogeneity in dynamics of information propagation on networks [14] [13]. Thus, we assume instead that each active node is able to activate its inactive neighbors with a specific probability at any time step during simulation.

2.1. Model Modification and Algorithm Description

The detailed investigation on model equivalence between the original independent cascade and linear threshold models can be found in [8]. The same method can be followed for the model modifications and their equivalence in our simulations. If we assume an independent probability, we need to modify the linear threshold model. If there is any active neighbor whose influence weight is greater than the inactive node u's threshold T_u at step t, it succeeds in activating u. It is therefore very straightforward to conclude that this modified threshold model is equivalent to the cascade model with independent probability. Modifications can also be made to show model equivalence if an incremental probability is assumed.

Since this paper focuses on high performance network browsing algorithms, we can consider the information propagation with independent probability as a case study to illustrate our optimized simulation strategies. In addition, we can further assume that the node's threshold is equal to the system threshold, such as the transmissibility in SIR infectious disease studies on SARS [9]. According to the model definition, we first introduce two types of simulation algorithms named as C-Loop, and T-Loop. The C-Loop and T-Loop can be considered as the 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 status 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' status at each step. The inactive node can be activated at the next step by any active neighbor if the transmission probability is satisfied.

Besides the vertex-oriented processing approach such as the C-Loop or the T-Loop, we can also process the network-based simulation using an edge-oriented approach. It is therefore necessary to introduce another network browsing algorithm based on edge-oriented processing, named as the E-Loop.

- E-Loop: Different from the vertex-oriented approach, the E-Loop starts from each edge element and checks the status of the connected pair of nodes. If the two connected nodes have heterogeneous status such as *Active - Inactive*, the information can be propagated from active to inactive with the given probability.

Furthermore, we define the following notations: V: vertex set, E: edge set, V^{dst}: corresponding destination vertex set of the source vertex, src_i: status of the source vertex i, dst_j: status of the destination vertex j (for the E-Loop, src_i and dst_i are the status of the source and destination vertex associated with edge i), p: transmission probability. The pseudocode of the C-Loop, T-Loop, and the E-Loop can be found in Algorithm 1 - 3 in Figure 1.

2.2. Data Structures for Simulations of Social Networks

In this sub-section, we introduce the data structures for the network storage in hardware. As illustrated in the last sub-section, we have three different simulation algorithms based on two types of network processing scenarios. It is therefore necessary to design different data structures for the corresponding simulation algorithms.

2.2.1. Vertex-Oriented Structure

Figure 2 (a) illustrates the vertex-oriented structure. With the vertex-oriented 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 ensuring correctness of the C-Loop and T-Loop. Different from directed graphs, we use the status of the vertex attribute (*Active or Inactive*) instead of the directed edge to determine the direction of information flow between two connected nodes.

Algorithm 1 C-Loop

```
for i = 1 → |V| do
  if src_i == Active then
    for j = 1 → |V^dst| do
      if dst_j == Inactive then
        src_i → dst_j with p
      end if
    end for
  end if
end for
```

Algorithm 2 T-Loop

```
for i = 1 → |V| do
  if src_i == Inactive then
    for j = 1 → |V^dst| do
      if dst_j == Active then
        dst_j → src_i with p
      end if
    end for
  end if
end for
```

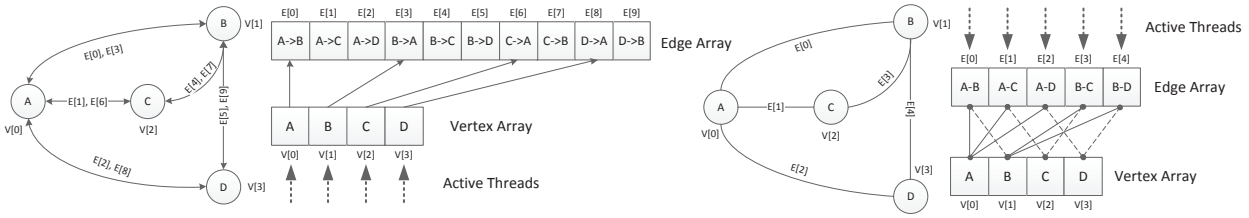
Algorithm 3 E-Loop

```
for i = 1 → |E| do
  if src_i == Active then
    if dst_i == Inactive then
      src_i → dst_i with p
    end if
  else
    if dst_i == Active then
      dst_i → src_i with p
    end if
  end if
end for
```

Figure 1: Simulation Algorithms

2.2.2. Edge-Oriented Structure

With the edge-oriented structure, simulation starts from the edge array and checks the connected pair of nodes' status. 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 (b).



(a) Vertex-Oriented Structure (b) Edge-Oriented Structure

Figure 2: Two Types of Data Structures for Network Simulation

2.3. Optimized Simulation Strategies

In the previous sub-section, we introduced two different simulation algorithms using vertex-oriented processing and one more algorithm using edge-oriented processing. The varying computational complexity of the three algorithms provides an opportunity to choose the best simulation algorithm with an analytical approach. In this subsection, we discuss the optimized simulation strategies for information propagation.

2.3.1. Vertex-Oriented Adaptive Processing

According to the vertex-oriented processing, we can calculate the computational complexity for the C-Loop and T-Loop at each simulation step. First, we introduce the following notations: V_A^{dst}: corresponding destination vertex set of the active vertices at step t, V_I^{dst}: corresponding destination vertex set of the inactive vertices at step t, E_A^{dst}: corresponding out-going edge set of the active vertices at step t, E_I^{dst}: corresponding out-going edge set of the inactive vertices at step t. Thus, we can present the algorithmic complexity of the C-Loop at each step t as

$$O(|V|) + O(|E_A^{dst}|) + O(|V_A^{dst}|) \tag{1}$$

The algorithmic complexity of the T-Loop can be expressed as

$$O(|V|) + O(|E_I^{dst}|) + O(|V_I^{dst}|) \tag{2}$$

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

$$|E_A^{dst}| + |V_A^{dst}| = |E_I^{dst}| + |V_I^{dst}| \tag{3}$$

2.3.2. Edge-Oriented Processing

For the edge-oriented processing, we introduce additional notations: $\bar{V}^{src}$: vertex set of all the source vertices associated with the edge array, $\bar{V}^{dst}$: vertex set of the destination vertices corresponding to set $\bar{V}^{src}$. Thus, the algorithmic complexity of the E-Loop at each simulation step can be expressed by

$$O(|E|) + O(|\bar{V}^{src}|) + O(|\bar{V}^{dst}|) \tag{4}$$

Note that, the size of E is equal to that of $\bar{V}^{src}$ and $\bar{V}^{dst}$ ($|E| = |\bar{V}^{src}| = |\bar{V}^{dst}|$). In contrast to the varying computational complexity at each simulation step with the C-Loop and T-Loop, the E-Loop is an algorithm with constant computational complexity in network browsing at each simulation step.

3. Simulations of Information Propagation on Multi-core CPU and GPGPU

In the previous section, we introduced three different simulation algorithms based on two types of processing approaches. For the vertex-oriented processing approach, we further proposed an optimized simulation strategy with algorithmic adaptation at runtime. In this section, we describe the detailed simulation designs on Fermi GPGPU and AMD Quad-core CPU.

3.1. Detailed Simulation Design for Information Propagation

We develop an HPC simulator in cooperation with Medusa [19] for the information propagation simulation. Medusa is a GPU-based graph processing package which can ease CUDA threads management. Before simulation, we assign consecutive arrays in the CPU main memory first to construct the network structures and copy the data structures to the GPU global memory via PCI-e bus. This is because the global memory takes up most of the GPU device memory. For example, Tesla C2050 GPGPU global memory occupies 2.62 / 3 GBs device memory. More importantly, the global memory is shared by all GPU streaming processors for massive parallel processing.

Figure 3 illustrates the overview of the simulation design using either CPU or GPU. Note that a random number generator is essential to cooperate with the network browsing algorithms for testing if the inactive nodes successfully receive the information from the active neighbors. In order to compare the simulation performance on CPU and GPU with the same baseline, we must ensure that the simulation on CPU and GPU generates the same network state results at each step. Curand [4] is a high performance random number generation library on GPU, and it is also available for CPU based random number generation. Therefore we adopt Curand to generate random number sequences on the CPU and GPU. At each simulation step, a new random number sequence is generated and attached to the edge array. At the same time, the Monitoring Module collects state results from both CPU and GPU. If the on-going simulation is using vertex-oriented processing, at each step the Steering Module selects the optimal simulation algorithm according to the network state results analysis from the Runtime Scheduler. The same operations are conducted whether the simulation is executed on CPU or GPU. Thus, by implementing the same network browsing algorithm on CPU and GPU, we can have identical state results at each step from both architectures. Since we are focusing on network browsing algorithms in this paper, we do not consider the cost of random number generation. For the proof of concept, we conduct a set of experiments in this section on two samples with 1 million vertices and 8 million edges (1M8M). In the beginning of the simulation, a single active node (the first node, $v[0]$) is introduced to the network. After that, according to the selected simulation algorithm, the active node can propagate the information to those inactive nodes in that network with the transmission probability. The detailed simulation parameters can be found in Table 1. Figure 4 illustrates the information propagation simulation on a 1M8M random network. In particular, Figure 4 (a) and Figure 4 (b) show the execution time per simulation step on Fermi GPU and AMD CPU respectively. Compared to the CPU serial simulation performance, the Fermi GPU based simulation shows 12.49x, 13.10x, 15.60x parallel speedup with the C-Loop, T-Loop, and the E-Loop respectively. Figure 4 (c) illustrates the state results of the network during simulation

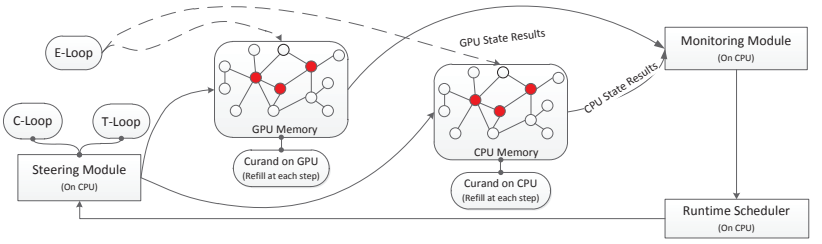


Figure 3: Overview of Simulation

Table 1: Simulation Parameters

Simulation Algorithm	Transmission Probability	Network Type	Hardware Architecture
C-Loop, T-Loop, E-Loop	0.01	Random, Scale-free	Tesla C2050, AMD Quad-core

(same on the CPU and the GPU). After 500 steps, the number of active nodes changes from 1 to 999459, 999457, and 999494 with the C-Loop, T-Loop, and the E-Loop, respectively. Furthermore, as shown in 4 (c), the state results using the three different algorithms are very similar at each simulation step. It indicates that our proposed three algorithms and assumptions of equivalence are valid in practice. In particular, according to Figure 4 (c), the inactive nodes are intensively changed to active status in the period of steps [150, 250]. It is the extra computing cost of state updating that causes the obvious curve protuberance (steps [150, 250]) in the CPU serial simulation performance, compared to that of massive parallel processing capacity in GPGPU.

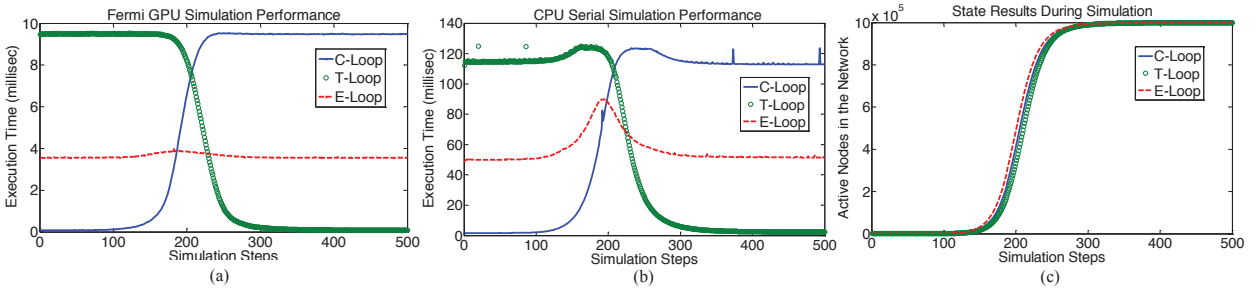


Figure 4: Random Network with 1M8M

As shown in Figure 5, different from the random networks, in the scale-free networks with power-law degree distribution, a few hubs occupy most of the edges and are very influential to the whole network. Furthermore, we note that the scale-free networks may not be fully connected, and it is therefore necessary to calculate the number of isolated nodes in the network before simulation. Figure 7 shows the simulation on a scale-free network with 1 million nodes and 8 million edges. However, there are 480551 isolated nodes without any connection to others. In addition, according to the feature of power-law degree distribution, a few nodes have very large contact lists compared to others. That is, the information will spread widely from the hubs if they receive the information. Figure 7 (c) shows that, the simulation converges very quickly among the connected nodes, and the number of active nodes finally reaches 519290, 519290, 519287 after 500 steps by the C-Loop, T-Loop, and the E-Loop respectively. Figure 7 (a) and Figure 7 (b) illustrate that the Fermi GPU based simulation performance shows 6.79x, 12.01x, and 14.43x parallel speedup compared to the CPU serial simulation performance with the C-Loop, T-Loop, and the E-Loop respectively. Due the the extra computing cost of state updating during steps [1, 50], the curve protuberance of the C, T, E Loops appears in the CPU serial simulation performance during these steps.

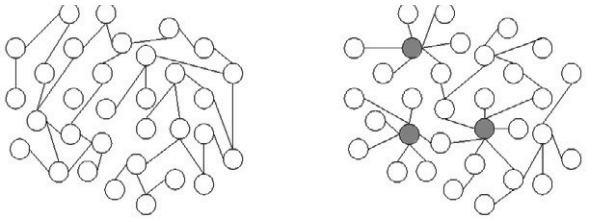


Figure 5: Network Structures (adopted from [15])

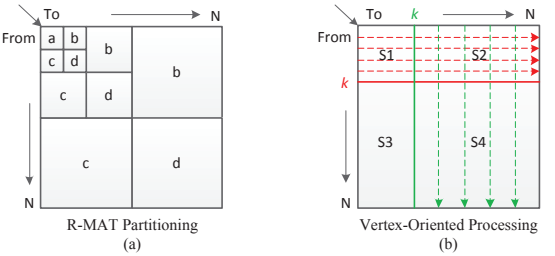


Figure 6: R-MAT Graph Generation Model

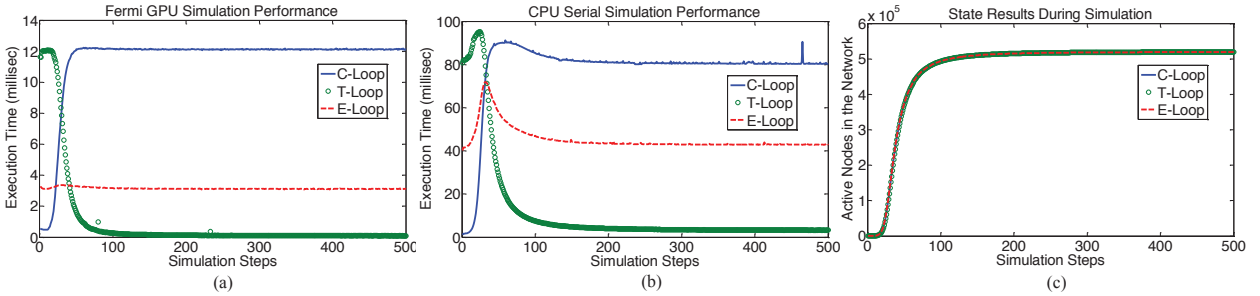


Figure 7: Power-Law Network with 1M8M

3.2. Optimized Algorithmic Adaptation Strategies for Different Networks

In the previous sub-section, we have shown the simulation performance with three different algorithms on the random network with uniform degree distribution and also on the scale-free network with power-law degree distribution. In this sub-section, we introduce the optimized algorithmic adaptation strategies for the vertex-oriented processing on different network structures. According to the experimental results shown in Figure 4 and Figure 7, the optimized algorithmic adaptation chooses the C-Loop first and then swaps to the T-Loop at a proper time step. In section 3, we have also formulated the computational complexity for each algorithm. That is, the optimal algorithm swapping should be conducted at the step where the C-Loop's workload equals that of the T-Loop's. Such an optimized simulation algorithm with runtime adaptation is named as the A-Loop.

3.2.1. Random Networks

As shown in Figure 5 (a), edges in the random networks are randomly assigned to nodes with uniform degree distribution. It means that the contact degree of each node can be considered as the same. It is therefore not difficult to derive an optimization strategy which satisfies Equation (3). That is, the algorithmic adaptation can be conducted when the number of active nodes is more than the number of inactive nodes in the network, expressed as $|V_{Act}| > |V| - |V_{Act}|$. Figure 8 (a) and Figure 8 (c) illustrate that the A-Loop executes the most optimal algorithm at each step of the simulation, both on the GPGPU and CPU. Furthermore, we observe that the vertex-oriented processing with the A-Loop obtains better simulation performance than that with the E-Loop. The experimental results also demonstrate that the A-Loop achieves 8.18x, 5.92x, and 4.98x speedup compared to that of the C-Loop, T-Loop, and the E-Loop on Fermi GPGPU. In addition, the A-Loop also achieves 5.74x, 4.36x, and 4.37x compared to that of the C, T, and E Loops in the CPU serial execution environment.

3.2.2. Scale-free Networks

With scale-free networks, the number of links connecting to a node follows the power-law degree distribution. In practice, such networks can be generated by the recursive matrix (R-MAT) model [2]. For the scale-free networks, we could also use the same condition for algorithmic adaptation as that on random networks. However, we discuss a more accurate algorithmic adaptation strategy in this section. As shown in Figure 6 (a), with the R-MAT model, the

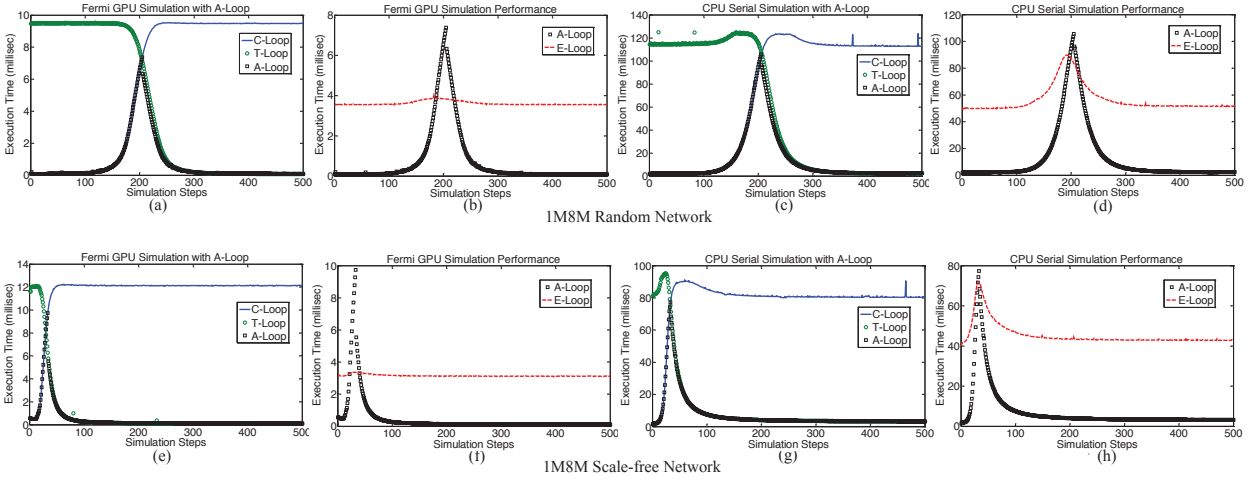


Figure 8: Algorithmic Adaptation

adjacency network matrix is recursively divided into four equal sized sub-matrices. Moreover, the edge assignment on the sub-matrices is under biased probabilities. That is, each new edge recursively chooses one sub-matrix with probabilities a, b, c, d (here $a + b + c + d = 1$) until it finally reaches a single cell in that adjacency matrix. In order to match the power-law degree distribution, we have $a \geq b, a \geq c, a \geq d$ [2]. The bidirectional network modified from the original directed network still follows the power-law distribution if $b = c$ [2]. In practice, we use the values of 0.45, 0.15, 0.15, 0.25 for the parameters a, b, c, d to generate networks. That is the default setting in GTgraph and it is approximate in many real world graphs [1].

In Figure 6 (b), the $Left \rightarrow Right$ arrows show the C-Loop, and the $Top \rightarrow Down$ arrows show the T-Loop. In practice, we choose the C-Loop first to conduct information propagation simulation starting from the first node $v[0]$. According to the given edge distribution in the network matrix, the information will be propagated to the $S1$ zone with very high probabilities. We can therefore assume that there exists a simulation step t at which there are k percent of nodes with active status in the network (see Figure 6 (b)). Thus, the execution cost of the C-Loop can be expressed as $O(S1 + S2)$ at the simulation step t. The execution cost of the T-Loop at step t can then be represented as $O(S2 + S4)$. The problem of finding the algorithmic adaptation point can be considered as one of finding out the simulation step t at which there are k percent of nodes active and associated with half of the total edges. In addition, as the scale-free network contains isolated nodes (denoted as V^{Iso}), we can ignore these. We can therefore carry out the algorithmic adaptation on the scale-free networks when $|V_{Act}| > k \cdot (|V| - |V^{Iso}|)$. In particular, with the default parameters in the edge distribution matrix, we can calculate the value of $k \approx 0.37$.

Figure 8 (e) and Figure 8 (g) indicate that the optimized algorithmic adaptation strategy works very well on the scale-free network, both on the GPGPU and CPU. The experimental results also demonstrate that the optimized vertex-oriented processing with the A-Loop is better than the edge-oriented processing with the E-Loop with regard to execution time. Furthermore, according to the experimental results, on Fermi GPGPU with the A-Loop, we can achieve 24.78x, 2.12x, 6.78x speedup compared to that of the C-Loop, T-Loop, and E-Loop. In the CPU serial simulation, the A-Loop obtains 10.59x, 1.60x, 6.16x speedup compared to that of the C-Loop, T-Loop, and E-Loop.

4. Experiments and Results

In the last section, we have shown that the proposed optimized algorithmic adaptation strategies perform well in practice. In this section, we demonstrate more experimental results on scalable networks in order to further evaluate our proposed method. As shown in Figure 9, we conduct a set of simulations on random networks and scale-free networks with size of 1M4M, 1M16M, 1M64M, and 1M256M. Due to the GPU memory limits (maximum 3GBs in Fermi C2050 GPGPU), the vertex-oriented processing approach is only feasible on 1M4M, 1M16M, 1M64M

networks but the edge-oriented processing approach is feasible on all the test cases. According to the performance results, it is obvious that we can achieve the best simulation performance with the optimized algorithmic adaptation strategy both on random and scale-free networks. Compared to the CPU serial processing environment, we can further obtain significant performance acceleration on the Fermi GPGPU.

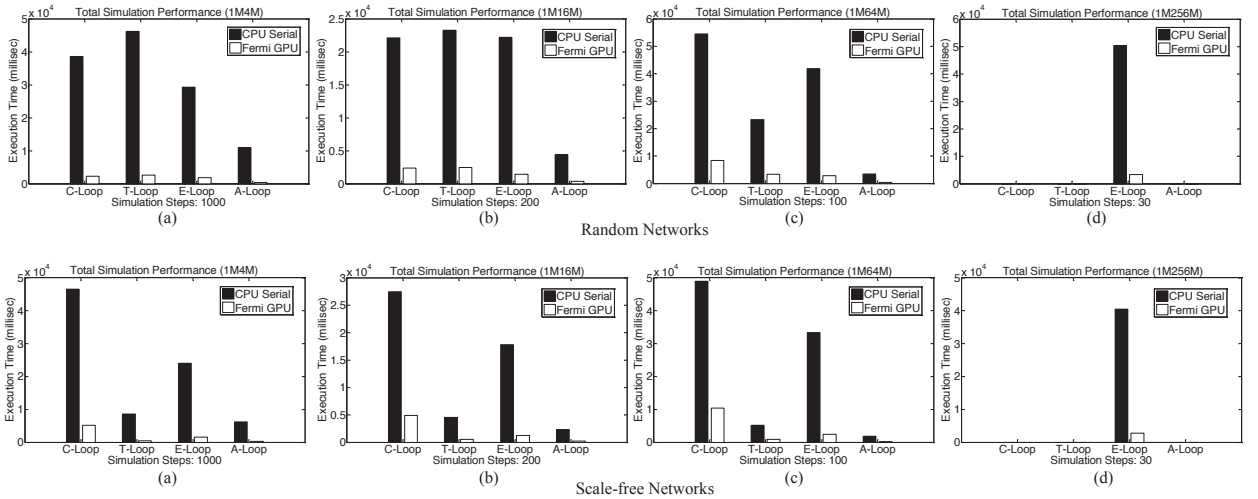


Figure 9: Performance Summary of Different Algorithms

The simulations over networks with large contact degree converge faster than those networks with relatively small contact degree. We also note that the larger the contact degree of the network, the better the performance of the T-Loop. According to the algorithmic analysis, the T-Loop has excellent performance when most of the nodes have changed to active status. We therefore observe that the T-Loop achieves better simulation performance with the increased contact degree, both on random and scale-free networks. Furthermore, the T-Loop shows better performance on scale-free networks than that on random networks since the information spreads much faster on scale-free networks. As a summary, we should choose the optimized A-Loop for performance improvement if the vertex-oriented structure is feasible in practice given the memory constraints. Otherwise, we should choose the simulation with the E-Loop for better spatial scalability.

Given the excellent data parallelism, we can use the OpenMP [12] library for multi-threaded parallel processing to accelerate the simulation performance on the multi-core CPU. Using the baseline of CPU serial processing performance, Figure 10 illustrates the parallel speedup of the E-Loop and A-Loop on Fermi GPGPU and Quad-core CPU (with OMP = 4). With the E-Loop, Fermi GPGPU achieves an average 14.89x speedup as opposed to 3.61x speedup in the CPU multi-threaded parallel processing environment. With the A-Loop, Fermi GPGPU achieves an average 15.08x speedup, and CPU multi-threading obtains an average 2.74x speedup (in Figure 10, the 'R_' denotes Random networks, and the 'S_' denotes Scale-free networks).

In addition, we have noted that the parallel speedup of the E-Loop on Fermi GPGPU keeps steady in the experiments. However, the A-Loop's performance on Fermi GPGPU indicates that the parallel speedup decreases on the increased contact degree. This is due to the thread scheduling mechanism and memory access pattern on Nvidia GPUs. That is, 32 GPU threads are encapsulated into a warp. Each warp is considered as the minimal scheduling unit at runtime. Furthermore, the memory access pattern of the Nvidia GPUs is to fetch maximum consecutive 128KB device memory data at once. If the data for a warp is accessed in many separated fetches, we consider such behaviors as uncoalesced memory accesses. As shown in Figure 2, we assign CUDA cores to the vertex array in the vertex-oriented processing approach. If the given network has a small contact degree, the uncoalesced accesses can be reduced when CUDA cores in a warp are concurrently tracing edge information. Our observations show that the Fermi GPGPU provides better parallel speedup on those networks with relatively small contact degree.

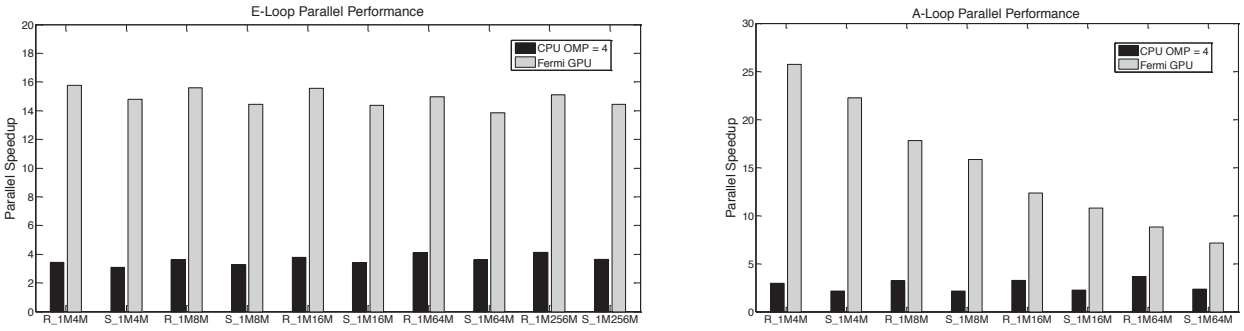


Figure 10: Parallel Speedup of E-Loop and A-Loop

5. Conclusions

In this paper, we have presented optimizations in simulations of information propagation. With an optimized algorithmic adaptation strategy (A-Loop) on the vertex-oriented structure, we can achieve the best performance in execution time. With the edge-oriented processing (E-Loop), we obtain better spatial scalability than that of vertex-oriented processing. Furthermore, we implement our proposed method on multi-core CPU and many-core GPGPU respectively. Compared to the CPU serial execution, we can obtain average 2.74x and 3.61x speedup in the A-Loop and E-Loop respectively on the Quad-core multi-threading environment. Furthermore, an average 15.08x and 14.89x parallel speedup in the A-Loop and E-Loop can be gained on the Fermi GPGPU.

Acknowledgements

Part of this work was conducted as a subcontract to the European Union project DynaNets (www.dynanets.org), EU project no. FET-233847.

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, Nidia "http://www.nvidia.com/object/cuda_home_new.html", 2011
[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] H. W. Hethcote, "The Mathmatics of Infectious Disease", SIAM Review 42, pp. 599-653, 2000.
[8] D. Kempe, J. Kleinberg, E. Tardos, "Maximizing the spread of influence through a social network", In Proceedings of the ninth ACM SIGKDD, pp. 137-146, 2003.
[9] 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
[10] M. E. J. Newman, "Spread of Epidemic Disease on Networks", Physical Review E 66, pp. 016128, 2002.
[11] M. E. J. Newman, "The Structure and Function of Complex Networks", SIAM Review 45, pp. 167-256, 2003.
[12] OpenMP Library, "http://openmp.org/wp/", 2011.
[13] R. Quax, D. A. Bader, P. M. A. Sloot, "SEECN: Simulating Complex System using Dynamic Complex Networks", International Journal for Multiscale Computational Engineering, Vol. 9(2), 2011.
[14] H. Rahmandad, J. Sterman, "Heterogeneity and Network Structure in the Dynamics of Diffusion: Comparing Agent-Based and Differential Equation Models", Management Science, pp. ec1-ec30, 2008.
[15] Wikipedia Scale-free network, "http://en.wikipedia.org/wiki/Scale-free_network", 2011.
[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] S. Wasserman, K. Faust, "Social Network Analysis: Methods and Applications", Cambridge University Press, pp. 1-27, 1994.
[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.