Cunetsim: A GPU based simulation testbed for Large Scale Mobile Networks

Ben Romdhanne Bilel
Eurecom
2229, route des Cretes
06904 Sophia Antipolis, France
benromdh@eurecom.fr

Nikaein Navid
Eurecom
2229, route des Cretes
06904 Sophia Antipolis, France
nikaeinn@eurecom.fr

***Abstract*—Most of the existing packet-level simulation tools are designed to perform experiments modeling small to medium scale networks. The main reason of this limitation is the amount of available computation power and memory in CPU-based simulation environments. To enable efficient packet-level simulation for large scale scenarios, we introduce a CPU-GPU co-simulation framework where synchronization and experiment design are performed in CPU and node's logical processes are executed in parallel in GPU according to the master/worker model. The framework is developed using the Compute-Unified Device Architecture (CUDA) API and denoted as Cunetsim, CUDA network simulator. In this work, we study the node mobility and connectivity as they are among the most time consuming task when large scale networks are simulated. Simulation results show that Cunetsim runtime remains stable and that it achieves significantly lower runtime than existing approaches when computing mobility and connectivity with no degradation in the accuracy of the results. Further, the connectivity is achieved up to 870 times faster than Sinalgo, which presents the best performances know until now.**

***Index Terms*—Large Scale, Simulation, Evaluation, GPGPU**

I. Introduction

Packet-level simulators are usually based on a discrete events paradigm where sequences of events are generated and processed. In general, such events represent mobility and connectivity evaluation, protocol operations, and in/out packets processing. The time complexity of a simulation is then proportional to the network size and the number of packets scheduled to be processed, which represent two main bottlenecks when targeting scalability. To enable efficient and scalable simulations, traditional approaches involve parallel nodes execution environments with minimal inter-process communication overhead to improve performances when total number of nodes and packets increase significantly [1]. To deal with large scale simulations, CPU parallelism through distributed simulation is the most investigated method [2]. However, this approach introduces a significant overhead due to the synchronization among different processes and machines and requires sophisticated and expensive simulation infrastructure [3].

Furthermore, mobile networks require periodic nodes and link updates, which needs to access global information such as nodes' position across multiple machines causing distributed simulations to be inefficient in terms of runtime. Sinalgo is an example of a mono-process simulator that is capable of simulating up to one hundred thousand nodes [4]. It only focuses on the verification of network algorithms, and provides a message passing view of the network by abstracting the underlying layers. However, performances degrades when the number of nodes increases as it requires the global view of the network prior to each iteration.

This work presents an efficient and scalable CPU-GPU [5] co-simulation framework for testing and validating network algorithms, denoted as Cunetsim, CUDA [6] Network Simulator. As opposed to previous works, Cunetsim is designed to provide an independent parallel execution environment for logical processes of a node. Nodes communicate with each other only through the message passing based on the buffer exchange, thus avoiding the usage of any global knowledge. Furthermore, it exploits the master/worker model for CPU-GPU co-simulation and provides CPU-based conservative synchronization.

The remainder of the paper is organized as follows. Section II presents exiting solutions for scalable networks simulation. Section III briefly presents the concept of Cunetsim and its design features. Preliminary results are given in section IV followed by concluding remarks and future directions in section V.

II. State of the Art

A. Scalability in networks simulation

The scalability and efficiency problems was approached by several works previously [1], [7], [8], where authors demonstrated the impact of the number of nodes and packets. To enable efficient and scalable simulations, CPU-based parallel execution is the traditional approach [2]. There are two dominants axes in the realization of parallel discrete events simulation systems: the native and federated parallel systems.

Native parallel systems are those where the software was custom designed for particular parallel simulation architecture which provides, at a minimum, services for communication and synchronization. Examples of parallel network simulators using this approach include GloMoSim [9], TeD [10] and SSFNet [11] among others. Even if such simulators are generally efficient, they introduce a compatibility problem and

requires a significant amount of time and effort to create a usable system.

Federated systems evolve interconnecting simulators. These simulations may include multiple copies of the same one (modeling different portions of the network), or entirely different simulators. The individual simulators that are to be linked may be sequential or parallel. The federated approach offers the benefits of model and software reuse, and provides the potential of rapid parallelization of existing sequential simulators. It also offers the ability to exploit models and software from different simulators in one system [12]–[14]. However, setting such testbeds implies the use of complicate and expensive infrastructure and sophisticate coordination work.

Even if parallel and distributed simulators have crossed a scalability boundary, they introduce a significant overhead due to the synchronization among different processes and/or machines , it requires also sophisticated and expensive infrastructure. This overhead may increase drastically in mobile environment if the network topology and machine mapping is not dynamically managed.. Moreover, famous solutions listed above did not address the emerging network challenges as it was done by centralized simulator as NS2/NS3 [15]. Among the rare exceptions who have studied the subject, SwimNet [16] is a simulation environment designed for wireless Network, nevertheless, this solution did not address the fundamental of mobility problem and its architecture is achieved around a master process which may be the bottleneck in large scale experiments. A different approach was introduced by sinalgo which proposes a simulation framework for testing and validating network algorithms, focused on the verification of network algorithms. Ignoring the physical layer. It presents an important scalability level as it is able to simulate until one hundred thousand nodes. However it remains a centralized solution where large scale still the bottleneck. The scalability decreases with the number of generated packets.

The major challenge of wireless mobile network simulation resides in the connectivity, which needs a global view of the simulated space thus; it needs also all nodes positions. In fact the connectivity information is the unique method allowing the systems to know which node is able to communicate with which one. Centralized (mono process) network simulator achieves this task via the help of the god object as NS3 do or via a clever management of the space as it is done on sinalgo. But this remains a bottleneck because the computing time increases exponentially as a function of the number of node.

In some recent approaches, the Graphics Processing Unit (GPU) is used to offload certain CPU-intensive computing tasks such as channel modeling [17]. Recent studies of GPUs highlights the credibility of the general-purpose computation GPGPU [5], [18] which encourages us to propose an innovate approach where the GPU is the central element of the simulation, not a co-processor. The newer graphics cards implement massively parallel architecture comprising several hundreds of scalar processors [19]. The limit of one thousand in a unique card is already reached. In GPGPU programming concept, a single instruction is executed across all the processors in a data flow. All these processors can communicate using the shared memory space; The GPU provides several memory levels with different speed, size and access right. The new design delivers impressive computational power, which is made possible by the management of numerous threads on fly along with high memory bandwidth. There are two actors in GPUs manufacturing: AMD and NVIDIA. Actually NVIDIA propose a specific GPGPU API: the Compute-Unified Device Architecture (CUDA) which has a mature ecosystem including development, debugging and compiling tools.

III. The Design of Cunetsim

The Cunetsim specification targets a new solution which is able to achieve a large scale simulation of wireless mobile network with high performances and low cost. Multi-platform distribution capability must be respected and the conditions of scientific experimentation must be guaranteed. Moreover, it must be intuitive and easy to use, simplifying the procedure of adding a new protocol or functionality and reducing the learning time. To deal with such requirements, we designed Cunetsim based on Three models: (A) node, where each function/service of a node is performed on an independent logical process, (B) simulation, where the master/worker parallel discrete event model is used between CPU and GPU within the same OS context and (C)Data, where the flow is the basic model to increase the throughput.

A. Node Model

A node is modeled as a stack of independent logical processes (LP) each of which performs a specific functionality/service on behalf of a node. Nodes communicate with each other only through the message passing (i.e. restrictive communication). Only buffers are exchanged between nodes to avoid global knowledge and centralized information.

In Cunetsim, each node contains four ordered LPs: mobility (MOB), connectivity (CON), protocol (PROTO), and packet (PKT). Each LP has its own container (i.e. GPU kernel) and executed in parallel. For each node, the container will be periodically launched and select corresponding data and model applicable to each LP based on the kernel unique identifier (KID). Nodes are logically located in a geometric area called cell used by MOB and CON processes to determine the network topology. The MOB calculates the movement in a space following the mobility model and moving dynamics (e.g. min and max speed) of a node. It implements several mobility and boundary policy models. The current version supports two mobility models: random way point and random walk, and three boundary policy models: annulment of excess, sliding on the boundaries, and bouncing on the boundaries [20]. The CON determines network topology over time following the connectivity model and nodes' position. It counts the number of nodes in each cell and updates the nodes' ID and positions to build the network connectivity according to a set of models including channel, interference, power, transmission models. Currently, only two simple channel models are supported:

unit disk graph and quasi unit disk graph. The PROTO describes network nodes and their operation and algorithm. It implements methods that are called when a node sends and receives a message (i.e. protocol data unit). The current version implements different broadcasting schemes. The PKT encapsulates protocol messages in packets, which contains meta information to perform the message delivery, and provides all the actions associated with the packet processing when nodes communicate with each other.

B. Simulation Model

The simulation is modeled based on a master/worker simulation model [2], where the master is a CPU process and workers is GPU (and in some cases CPU) threads within the same OS context. Due to this specificity, the management of master and workers interaction can be simplified. The main simulation model is designed around node and LPs. Each node is composed of several LPs and each LP is implemented in a unique process P and executed sequentially (see Figure 1a). The master process resides in CPU context and is responsible for global time synchronization and the simulation coherence, which is achieved through safe timestamps-ordered processing of simulation events within each LP, also known as conservative synchronization [21]. This ensures that an LP does not execute an event until it can guarantee that no event with a smaller timestamps will later be received by that LP.

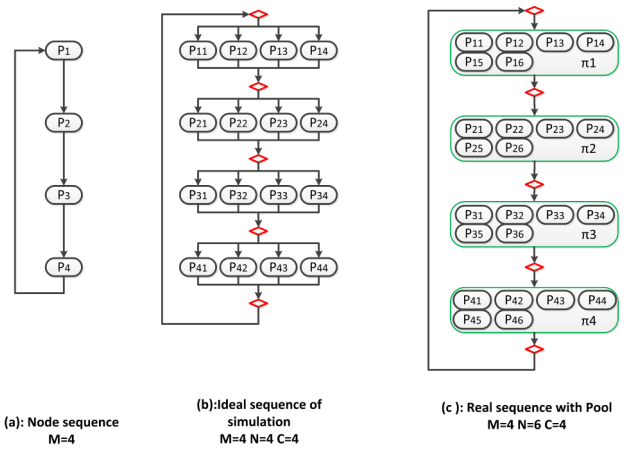

Fig. 1. Node LP sequence and pool

In a general case, each node is composed of M processes and the experiment evolves N nodes. A simulation round is defined by the execution of all process of all nodes one time which means that the total number of processes in one round is $M * N$. In ideal conditions, the N nodes will be executed in parallel and their processes sequentially as shown in (figure 1 b). Now assume that only C cores (or GPU kernels) are available, which implies that only C nodes could be executed in parallel. In this case, the correctness of simulation will be preserved if and only if : $\forall i \in [0, M] \wedge j \in [0, N]$, $P_{i+1,j}$ could not start until all $P_{i,j} \forall j$ ends. To achieve this, a sequence of process pool Π_j is defined incorporating the same process for all nodes (see Figure 1c). For a given Π_j, all P_{ij} processes must ends to assert that Π is achieved. This presents a simple yet efficient implementation of the coherence and consistency paradigm [22].

C. Data Model

Cunetsim data is based on the kernel/flow model [23]. We define several flows where each one presents a specific part of the simulation data. Data is grouped by functionality. Each LP uses one (or more) flows and each node has a specific box with Read/Write rights. One node can access foreign data with read right. Flow model is natively used by graphics application to manage the communication between the GPU and the CPU. We apply a flows loading-offloading mechanism between the GDRAM (limited and non-extensible) and the principal memory (larger and extensible). The master manages flows transfer between the principal memory and the GPU one, such that no LPs will be in famine situation. Conceptually there are two types of objects represented by flows in Cunetsim: The simulated space and simulated nodes.

Cunetsim provides two important services: memory allocation abstraction (MAA) and critical section management (CSM). MAA insures the double allocation of each data flow in both of the RAM and the GDRAM. The synchronization of the two copies of each flow is a manual operation which must be specified by the user. Critical section is a recurrent challenge in case of shared memory between several processes. Software mutual exclusion solutions such as semaphores, mutex and locks are commonly used in CPU context. However, GPU context did not provides such explicit solutions. Conflict situation can be described as following: If two nodes A and B want to send messages to node C, than they must write them into the input buffer of C. if this will happen in parallel the probability that node A erases the message of B is not negligible. Further if the number of concurrent nodes is important, all messages will be lost. CSM provides an abstraction of this problem based on cuda atomic operations.

IV. Performances & Results

In this section, we compare the performance of Cunetsim with Sinalgo in terms of runtime for computing nodes mobility and network connectivity as the number of nodes increases. We vary the network size from 256 nodes to 64k. The benchmark scenario is the following: Nodes are initially randomly distributed into a cubic space (edge=1600). The mobility model is the random way point with speed uniformly distributed between 1-5m/s. The maximum transmission range, Rmax, is 100 and the connectivity model is UDG. We run 500 rounds[1] to provides a significant average. Experimentations are done using a simple workstation(i7 940 CPU with 4GB RAM and NVIDIA GTX 460 SE GPU with 1GB GDR5). In order to assess the performance gain and simulation efficiency when GPU is used, Cunetsim is also built for CPU target using the PGI compiler [24].

[1]One round is finished when all nodes achieve their mobility and compute their connectivity set.

A. MOB Performances

Figure 2 presents the average runtime to compute nodes mobility in Sinalgo and Cunetsim (GPU and CPU-only). Even if both of them present a linear behavior, we observe that Cunetsim (GPU) runtime is always lower than Sinalgo one when computing nodes mobility, and that it remains stable as the number of nodes increases. Compared to Sinalgo, Cunetsim is up to 4 times faster in CPU and 74 times in GPU.

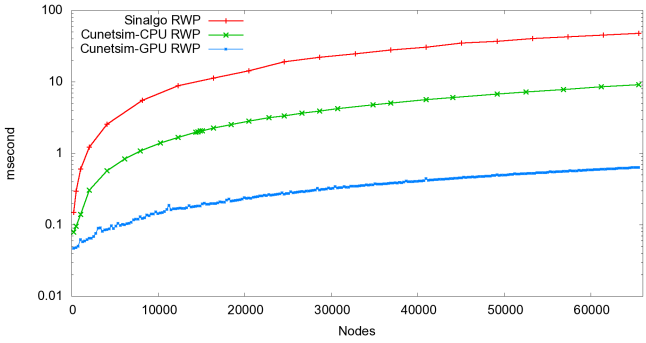

Fig. 2. Execution time for mobility

The random way point mobility implementation is similar and presents an equivalent complexity in sequential execution for both simulators. However, Cunetsim is completely scalable for parallel contexts .Theoretically it must be 4 times faster in quad-cores CPU. Nevertheless, Cunetsim introduces a management overhead with a significant impact in small and medium scales. This is more visible for the CPU version especially for network bellow 1000 nodes, where Cunetsim is two times faster than Sinalgo while it uses four threads. In the other hand, the analysis of Cunetsim on GPU context is more complex due to three major parameters: First a GPU core did not has the same architecture than the CPU one which skews the comparison. Second, the GPU and CPU clocks are different which introduce a speed factor. Third using the GPU implies a memory transfer between the GDRAM and the RAM which presents an additional overhead. To simplify the evaluation we suppose that both of cores are equivalent than we can calculate the relative speed factor: Our CPU is clocked at 3.06 Ghz and our GPU is clocked at 763 MHz. Based on previous assumption, one CPU core is equivalent to four GPU cores. As our GPU includes 336 Cores, we can admit that they are equivalent to 84 CPU cores. Theoretically, Cunetsim must be 84 times faster on the GPU than Sinalgo however we distinguish two phenomena: First, in small scale networks, the GPU is under used while the memory transfer still significant, inducing a reduced gain. Second, in large scale networks, the performance converge to be 74 times faster, which implies that the impact of the memory transfer is not negligible in all cases.

B. CON Performances

The connectivity evaluation of a wireless network using the UDG model is NPC problem [25]. Figure 3 presents the average connectivity runtime. It can be seen that Cunetsim runtime remains stable as the number of nodes increases while that of Sinalgo increases exponentially. In particular, when the number of nodes is close to 50k, the runtime increases drastically. In Cunetsim, we observe that the runtime remains stable by exploiting the high bandwidth of the GPU memory (10 times faster than the RAM) and parallel processing power, which in turn allow Cunetsim to be radically faster than Sinalgo. For 64k nodes, Cunetsim compute the connectivity set of all nodes 870 times faster than Sinalgo. These results

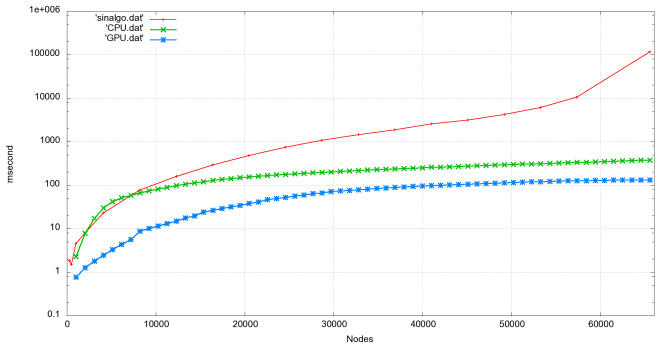

Fig. 3. Execution time for connectivity

confirm that our parallel architecture is natively adapted to multi-cores hardware. Our approach enlarges the scalability borders and reduces significantly the simulation runtime. Moreover, the GPU implementation presents extraordinary results compared to one machine CPU-based solutions.

C. PKT and PROTO Performances

To evaluate the performance of PKT and PROTO LPs regardless the efficiency of MOB and CON, we propose a scenario which models a simple network, where the nodes are arranged in a grid topology as illustrated in Fig. 4. The scenario includes one traffic source which generates 600 uniform packets with 1 second of inter-departure-time. Packets size is fixed to 128 Bytes. All nodes -including the source- relay unseen packets after a delay of 1 second, thus flooding the totality of the network. The delay of 1 second models the propagation delay. Nodes did not provide any packets management services. Transmission and reliability are modeled on the channel using a fixed dropping probability which is identical on all links. The sender is the node with the lowest identity and the receiver is the one with the highest identity. This methodology described above is the was implemented in both Sinalgo and Cunetsim. The minimal simulation time is set to 700 seconds[2]. Figure 5 shows the average measured simulation runtime for each simulator. The CPU version of Cunetsim is the fastest simulator up to 200 nodes and still faster than Sinalgo for all network sizes. The efficiency factor between Cunetsim (GPU version) and Sinalgo is about 11

[2]To allows the deliverance of the last packet -number 599- we extend the simulation time for network larger than 2500 node following this rule: $N = a * a$; $simtuime = max(600 + 2 * a, 700)$. where N is the number of node and a is its root

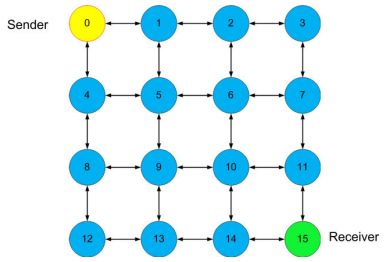

Fig. 4. Grid Network

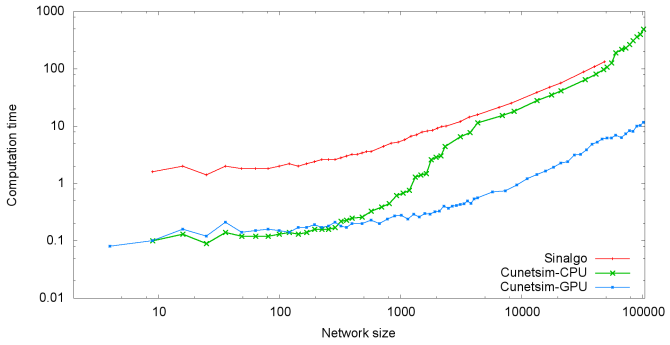

Fig. 5. Execution time for PKT & PROTO LPs

and remained stable until the threshold of 48K nodes where the RAM in our machine becomes insufficient. For small network the CPU version is the fastest because it did not need any memory transfers and use the totality of the CPU resources (four cores). However, for large scale, the GPU version becomes the fastest owe due to its large computing power and its memory bandwidth. Sinalgo is lagging because the scenario involves a large number of small packets which complicate the JAVA garbage Collector work.

V. Conclusion & Future Work

New challenges emerge when simulating a large scale mobile network. While network simulation tools are widely used for validation and performance evaluation, their scalability and efficiency remain challenging. Cunetsim aims to unlock the parallel capabilities of the state-of-the-art hardware and software architectures to achieve simulation scalability and efficiency with significantly lower cost. It provides a CPU-GPU co-simulation framework for testing and validating network protocols and algorithms for large scale scenarios. Preliminary results show that the execution time could be radically improved when CPU-GPU parallelism is used. In future work, we will evaluate the execution time of Cunetsim with other existing approach such as NS-3 and extend the results for protocol and packet operations. We also plan to further optimize the parallelism capability of Cunetsim by exploiting multi-GPU hardware as a simulation platform.

VI. Acknowledgments

This paper describes work undertaken in the context of the LOLA and CONECT project. The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n^o248993 and n^o257616.

References

[1] R.M. Fujimoto, K. Perumalla, A. Park, H. Wu, M.H. Ammar, and G.F. Riley. Large-scale network simulation: how big? how fast? In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems. MASCOTS 2003.*

[2] Alfred Park and Richard M. Fujimoto. Parallel discrete event simulation on desktop grid computing infrastructures. *International Journal of Simulation and Process Modelling*, 5(2):157 – 171, 2009.

[3] Alfred Park and Ric Fujimoto. Efficient master/worker parallel discrete event simulation. *2009 ACMIEEESCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 145–152, 2009.

[4] http://disco.ethz.ch/projects/sinalgo/.

[5] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

[6] C. Nvidia. Compute unified device architecture programming guide. *NVIDIA: Santa Clara, CA*, 2011.

[7] George F Riley. Simulation of large scale networks ii: large-scale network simulations with gtnets. 2003.

[8] B.B. Romdhanne, D. Dujovne, and T. Turletti. Efficient and scalable merging algorithms for wireless traces. In *ROADS: Workshop on Real Overlays and Distributed Systems*, 2009.

[9] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on*, pages 154–161. IEEE, 1998.

[10] A.L. Poplawski and D.M. Nicol. Nops: a conservative parallel simulation engine for ted. In *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on*, pages 180–187. IEEE, 1998.

[11] J.H. Cowie, D.M. Nicol, and A.T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50, 1999.

[12] K. Perumalla, R. Fujimoto, T. McLean, and G. Riley. Experiences applying parallel and interoperable network simulation techniques in on-line simulations of military networks. pages 97–104, 2002.

[13] J. Pelkey and G. Riley. Distributed simulation with mpi in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 410–414. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[14] Charles Peri Ken Renard and Jerry Clarke. A performance and scalability evaluation of the ns-3 distributed scheduler. The Workshop on ns-3 (WNS3)), 2012.

[15] http://www.nsnam.org/.

[16] Azzedine Boukerche, Sajal K. Das, and Alessandro Fabbri. Swimnet: A scalable parallel simulation testbed for wireless and mobile networks. *Wireless Networks*.

[17] S. Bai and D.M. Nicol. Acceleration of wireless channel simulation using gpus. In *Wireless Conference (EW), 2010 European*, pages 841–848. IEEE, 2010.

[18] K.S. Perumalla. Discrete-event execution alternatives on general purpose graphical processing units (gpgpus). In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pages 74–81. IEEE Computer Society, 2006.

[19] B. Neelima and P.S. Raghavendra. Recent trends in software and hardware for gpgpu computing: A comprehensive survey. In *Industrial and Information Systems (ICIIS), 2010 International Conference on*, pages 319–324. IEEE, 2010.

[20] A. Boukerche and L. Bononi. Simulation and modeling of wireless, mobile, and ad hoc networks. *Mobile ad hoc networking*.

[21] K.S. Perumalla. Parallel and distributed simulation: traditional techniques and recent advances. In *Proceedings of the 38th conference on Winter simulation*, pages 84–95. Winter Simulation Conference, 2006.

[22] S. De Munck, K. Vanmechelen, and J. Broeckhove. Revisiting conservative time synchronization protocols in parallel and distributed simulation. 2011.

[23] C. Zhang and D. Huang. Study of gpu and cpu collective in matching data flow. *Procedia Engineering*, 15:4068–4072, 2011.

[24] M. Wolfe. Implementing the pgi accelerator model. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 43–50. ACM, 2010.

[25] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(13):165 – 177, 1990.