Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 145863, 7 pages
http://dx.doi.org/10.1155/2013/145863

Research Article

An Efficient WSN Simulator for GPU-Based Node Performance

An Na Kang,[1] Hyun-Woo Kim,[1] Leonard Barolli,[2] and Young-Sik Jeong[1]

[1] *Department of Multimedia Engineering, Dongguk University, 30 Pildongro 1 Gil, Jung-Gu, Seoul 100-715, Republic of Korea*
[2] *Department of Information and Communication Engineering, Fukuoka Institute of Technology (FIT), 3-30-1 Wajiro-Higashi, Higashi-Ku, Fukuoka, Japan*

Correspondence should be addressed to Young-Sik Jeong; ysjeong@dongguk.edu

Received 14 August 2013; Accepted 13 September 2013

Academic Editor: Ken Choi

In wireless sensor network, when these sensors are wrongly placed in an observation region, they can quickly run out of batteries or be disconnected. These incidents may result in huge losses in terms of sensing data from numerous sensors and their costs. For this reason, a number of simulators have been developed as tools for effective design and verification before the actual arrangement of sensors. While a number of simulators have been developed, simulation results can be fairly limited and the execution speed can be markedly slow depending on the function of each simulator. In this regard, to improve the performance of existing simulators, this research aimed to develop a parallel calculation simulator for independent sensor (PCSIS) that enables users to selectively use the GPU mode and, based on this mode, enables parallel and independent operations by matching GPU with many cores in order to resolve the slowdown of the execution speed when numerous sensor nodes are used for simulations. The PCSIS supports the GPU mode in an environment that allows the operation of compute unified device architecture (CUDA) and performs the parallel simulation calculation of multiple sensors using the mode within a short period of time.

1. Introduction

Today, wireless sensor networks (WSNs) are utilized by their integration into various fields in the real world. The data collected by sensing via WSNs are used in various service areas such as individual research, national projects, energy saving, luxury automobile systems, important social infrastructure (e.g., electricity, water), manufacturing, communication systems, weapon systems, robots with distributed processing on multiple computers, transportation control, and elderly people [1–6].

A number of factors should be considered to establish effective WSNs. Basic questions include which type of sensors to select for a target sensing region, which protocol to select for communication, how to arrange sensors, how many sensors to be used, how to decide the density of sensors, and how much budget to arrange for network establishment. Therefore, topology configuration is not an easy task. For this reason, various tools have been developed to arrange sensors and design and verify communication protocols. These include various types of simulators such as GloMoSim [7], ATEMU [8], NS2 [9], TOSSIM [10], AVRORA [11], SWANS [12], and SENSE [13].

Despite the development of a number of tools, their processing speed for large or small networks is not fast enough. In addition, as simulations are performed based on the information of simulator-dependent sensor nodes, substantially limited results are obtained. In addition, while the functional aspect of simulators is important, their execution speed cannot be neglected. Sensors in a simulator are based on independent operations, and thus, one sensor uses one thread. For this reason, an increase in the number of nodes significantly slows down the simulation speed [1, 6].

To overcome a marked slowdown of the execution speed according to an increase in the number of nodes set to run simulations in a WSN simulator, this paper intended to develop a PCSIS that offers the function of using many GPU cores. The PCSIS enables parallel operations by applying a node to each thread of the GPU at the ratio of 1 : 1. Using this sensor, sequential calculations of a large scale of sensor nodes can be processed simultaneously. Therefore, users are allowed to perform more accurate and speedy simulations.

This paper is organized as follows. Section 2 explains the operating mode of existing WSN simulators and briefly introduces CUDA to use GPUs. Section 3 explains the use of a GPU to improve the performance of a PCSIS proposed by this study. Section 4 introduces the design of the PCSIS. Section 5 presents a comparative explanation on the construction of the PCSIS and resulting improvements in the performance of existing simulators. Section 6 finally presents a summary and future research tasks.

2. Related Works

This section reviews CUDA to use the operating mode of existing WSN simulators that have been developed by relevant studies and GPUs.

2.1. Existing WSN Simulators. A number of tools have been developed as WSN simulators including GloMoSim [7], ATEMU [8], NS2 [9], TOSSIM [10], AVRORA [11], SWANS [12], and SENSE [13]. Table 1 examines the operating modes of the above simulators.

In addition, a research [14], which aimed for the speedier performance of AVRORA simulators, used a super computer. Java-based AVRORA sensor nodes and Java threads are configured at the ratio of 1 : 1. The results of an experiment that matched them with CPUs showed that an increase in the number of CPUs does not improve simulations. This is because context switches frequently occur as the number of threads exceeds the number of CPUs. In other words, this phenomenon occurs as the processing capacity for synchronization surpasses the improved processing speed of CPUs.

We propose a PCSIS that enables the fast simulation of threads with a maximum number of GPU cords by using a GPU that basically has more cores than a CPU. The PCSIS is a Java-based simulator that uses JCUDA (Java bindings for CUDA) to support the GPU.

2.2. CUDA for GPU. CUDA is the technology that implements parallel computing using GPUs, which was launched by NVDIA in November 2006. In an early stage, the company's CUDA was developed based on the C language but has evolved into CUDA 5.5 after continuous updates. This technology supports various standard programming languages such as C, C++, Java, Fortran, and Python. Its advantages include processing a large volume of parallel calculations and utilizing its full functions simply through the operation of NVIDIA's built-in devices. In addition, software, utilities, and example documents are provided free of charge [15].

CUDA is a powerful technology in terms of enabling parallel programming using GPUs. Programs supported by CUDA may differ slightly depending on the graphic device launched by NVIDIA. However, currently launched devices are upgraded from the previous versions and therefore can operate even formerly written programs. While programmers expect performance improvement through parallel teaks using CUDA, they might encounter lower levels of performance. Therefore, programmers should have the overall knowledge and understanding of CUDA. In addition, nvcc (NVIDIA C Compiler) should be installed to execute CUDA.

A general data flow for using CUDA is as follows.

(i) Allocate a memory necessary for tasks using CUDA in graphic cards.

(ii) Duplicate host data into device memory in the allocated memory (host refers to the RAM run in the CPU, and device refers to the RAM run in the GPU).

(iii) Perform calculations and tasks by summoning GPU kernels.

(iv) Duplicate the processing results from device to host to use them in the host (in case of graphic tasks, direct outputs can also be produced according to the interworking for types of use).

(v) The memory that was initially allocated to the graphic cards is reclaimed.

It should be noted that in performing parallel tasks for only calculations, improvement in the performance can be expected when minimizing the portion of duplications from host to device or from device to host and instead increasing the portion of parallel tasks.

As this study is focused on calculations through parallel tasks, it provides a solution to perform the area that requires a number of calculations for each sensor in the GPU.

3. GPU Functions on PCSIS

The PCSIS proposed, in this paper, was developed based on Java and provides the GPU mode to improve performance in existing WSN simulators.

Certain considerations are required for using the GPU mode as follows.

(i) First, as the PCSIS uses GPUs, an examination should be carried out to check whether GPUs can be used in the simulator.

(ii) An essential examination on operability is to check whether the graphic cards support CUDA.

(iii) Next, once an examination is finished on whether parallel thread execution (PTX) files can be run, the GPU mode is finally activated.

(iv) If the GPU is operable, selective GPU mode is activated. Otherwise, it is inactivated.

The number of nodes necessary for using the GPU mode in the PCSIS depends on graphic cards. Basically, to yield maximum efficiency in the parallel run of each node, each thread in the GPU should not be matched with more than one sensor node. In other words, each thread should perform calculations for a maximum of one sensor node. In addition, back collisions, registers, and local, shared, and global memories should be taken into account.

4. Design of PCSIS

In this paper, the PCSIS proposed is largely divided into the user interface, target area manager, interaction broker,

Table 1: Comparison with exiting WSN simulators.

Simulator	Function and operating mode
GloMoSim	(i) This is a simulator developed for large-scale simulation environments, and it uses the Parsec language to perform parallel simulations. (ii) The Parsec language was selected for the parallel tasks of each sensor. However, many sensor nodes that should be established in an actual environment with complete parallel tasks cannot be realized due to the limitation of parallel tasks in the CPU.
ATEMU	(i) This the first command-based simulator developed based on the C language. (ii) This can ensure cycle accuracy and set parameters for mutual different systems. (iii) This does not offer the GUI mode and produce outputs based on texts. As its sensor nodes perform sequential simulations, the execution speed is significantly low.
NS2	(i) This is a discrete event simulator that has a modular mode. (ii) Many occasions require the interaction between a network and application programs. In this respect, this simulator lacks application program models. (iii) Network animator (Nam) exists to support the GUI of NS2. However, this simulator stores event command files and reads them whenever the files are required. Therefore, it is ineffective by excluding simultaneity.
TOSSIM	(i) This runs the simulation of TinyOS, which is the OS of motes used in a sensor network. (ii) This simulation enables the inference of actual movements and the analysis of hardware-based influences. (iii) This cannot perform simulations in areas other than TinyOS and does not have cycle accuracy, an important factor for code debugging and functional verification. (iv) This has TinyViz, which was developed based on Java by providing a GUI for movements. However, it does not fully show the fluid movement of dynamic sensors.
AVRORA	(i) This is a Java-based simulator that enables the simultaneous simulations of multiple sensor nodes. (ii) In running simulations, each sensor node activates each thread. (iii) As this does not provide a GUI environment, users cannot quickly identify its operational status. Moreover, its CPU-dependent thread operating mode makes it difficult to obtain prompt simulation results.
SWANS	(i) Users can define models using a Java-based simulator. (ii) This graphically shows interactions and areas for the communication of networks. (iii) This can define various conditions of each sensor in terms of temperature, humidity, and movement. (iv) While each thread is logically judged as if run independently, the actual execution mode is CPU-dependent. Thus, this excludes parallel runs for a number of nodes.
SENSE	(i) This simulator was developed based on C++ in 2004. (ii) This lacks configurations for various WSNs and does not support visualization when using only SENSE. Therefore, it is incapable of the swift identification of simulations. (iii) While this has G-Sense as a visualization tool for SENSE, it does not properly process the dynamic movement of sensor nodes and is rather focused on results. (iv) Basically, this has a slow execution speed and thus requires improvement.

map manager, map controller, node manager, coordinate converter, and viewer based on functional terms. The user interface receives inputs from the user regarding the basic setting of sensor nodes and whether the GPU mode will be used or not. The target area manager manages sensing target regions established by the user. The interaction broker plays the role of connecting node and mapping values, which are set and input by the user, to the system. The map manager applies and manages the data of topographic information and performs calculations that apply mapping values input by the user. The node manager performs tasks by either using the CPU only or using the GPU installed to improve performance according to the information of model selection received from the user. The coordinate converter plays the role of processing data to send the operating conditions of simulations to the view in order to show them to the user. Finally, the viewer shows the conditions of simulations in the PCSIS to the user in a visual form. Figure 1 presents the architecture about the overall functions of the PCSIS.

The *user interface component* is further segmented into the map interface, node interface, and OP mode. The map interface is the place that receives the geography markup language (GML) that can be mapped on actual topography, which is input by the user. It can also set information on the location of preferred target regions. The node interface consists of range control, which can set basic sensor information, such as the sensing range, communication range, and supersonic wave range, and configuration on whether the status of range control should be shown to the viewer or not, which includes the sensing range view (SR-V), communication range view (CR-V), supersonic wave range (SWR-V), node trace line view (NTL-V), and node connection view (NC-V). The OP mode makes self-judgment on whether the JCUDA is operable to use the GPU. If the interface is operable, the OP mode provides two types of modes, which are base and GPU modes, to enable the user to selectively operate it.

The *target area manager component* reads actual topographic data via the GML importer of the map manager in order to provide more expanded tests using the actual data and set the area of target regions that require observations in relation to the analyzed GML documents.

The *interaction broker component* plays the role of a broker that analyzes the user-input basic setting of operating modes and sensors and map control messages and then

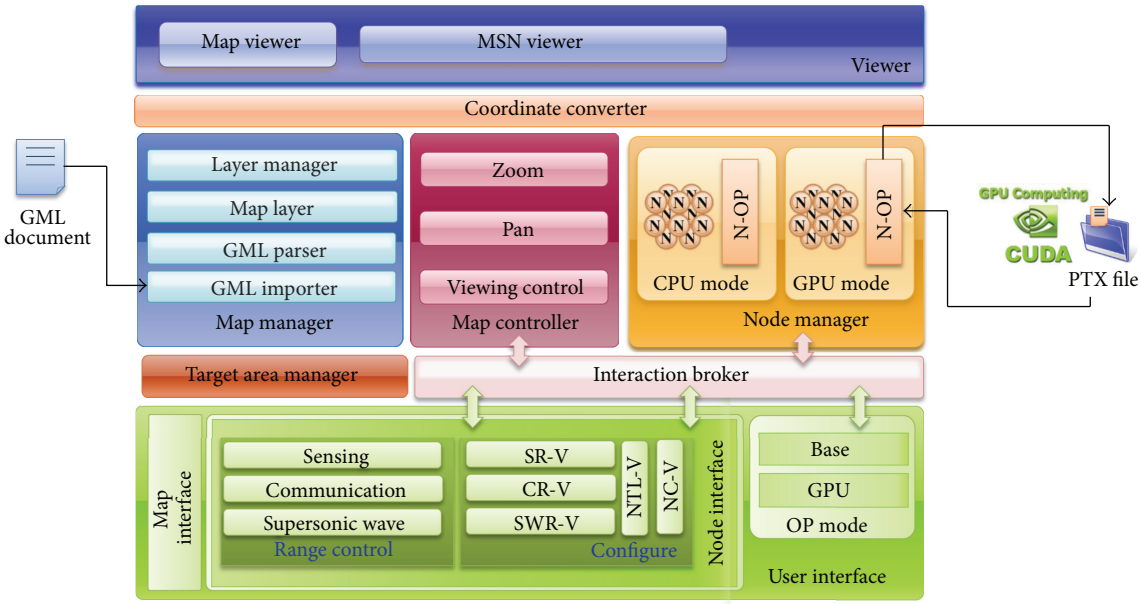

Figure 1: Architecture of PCSIS for high performance of WSN Simulator.

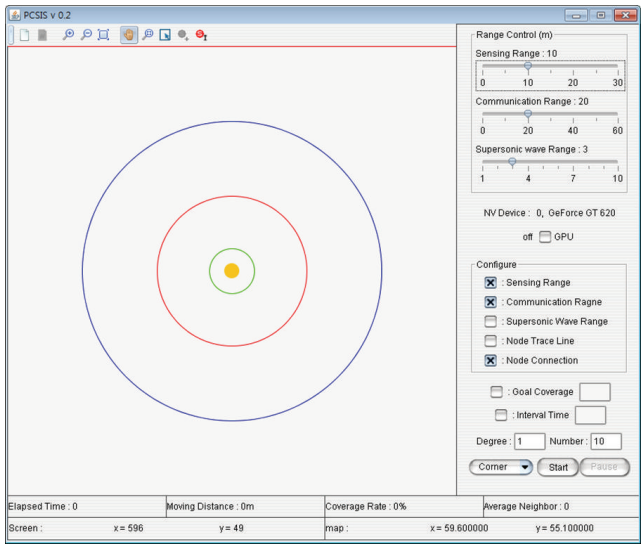

Figure 2: The initial execution status of PCSIS.

sends the analysis results to the map controller and the node manager.

The *map manager component* plays the role of applying managing GML documents that can be mapped on actual topography. In detail, this component consists of the following subcomponents: the GLM importer, which is necessary for adding GLM documents, which are selected by the user using the map interface of the user interface, to the PCSIS, the GML parser for analyzing the added GML documents to apply them to the PCSIS, the map layer for producing map objects by judging the presence of obstacles according to the objects of GML topographic data and then sending them to the layer manager, and the layer manager that provides and manages the polygons (e.g., building), polylines (e.g., road, street, and track), and texts (e.g., building name, road name) of topographic data received from the map layer according to the user's selection.

The *map controller component* performs expansion, contraction, area expansion, and selective movement for maps managed by the layer manager based on values defined and sent from the user. Such functions can be activated if the relevant data are received by the user's input via the map interface of the user interface. The corresponding results are displayed to the user by outputting them into the viewer using the coordinate converter.

The node manager component applies and manages node values input by the user. This is further divided into CPU and GPU modes and operates in one of the two modes using the OP mode of the user interface. In the CPU mode, the PCSIS operates in the same manner as general WSN simulators. In the GPU mode, the node manager component assesses whether the number of nodes designated for activation will be able to perform simulations. If no abnormality is detected, a common arithmetic unit across the nodes is produced into a PTX file. This PTX file is produced if the PCSIS is first run or no PTD files exist internally. If a PTX file already exists, the existing file is reused. The node-operator (N-OP) is the place to calculate the next location of mobile sensor nodes. In the GPU mode, this component converts data to enable parallel computation on the locations of sensor nodes. The converted data have the information necessary for the computation of each sensor node in the Java language.

The *coordinate converter component* plays the role of processing and sending data on the basic information of topography and nodes and the operating conditions of the nodes in a form that can be displayed at the viewer.

The *viewer component* visually presents the processed data delivered via the coordinate converter to the user.

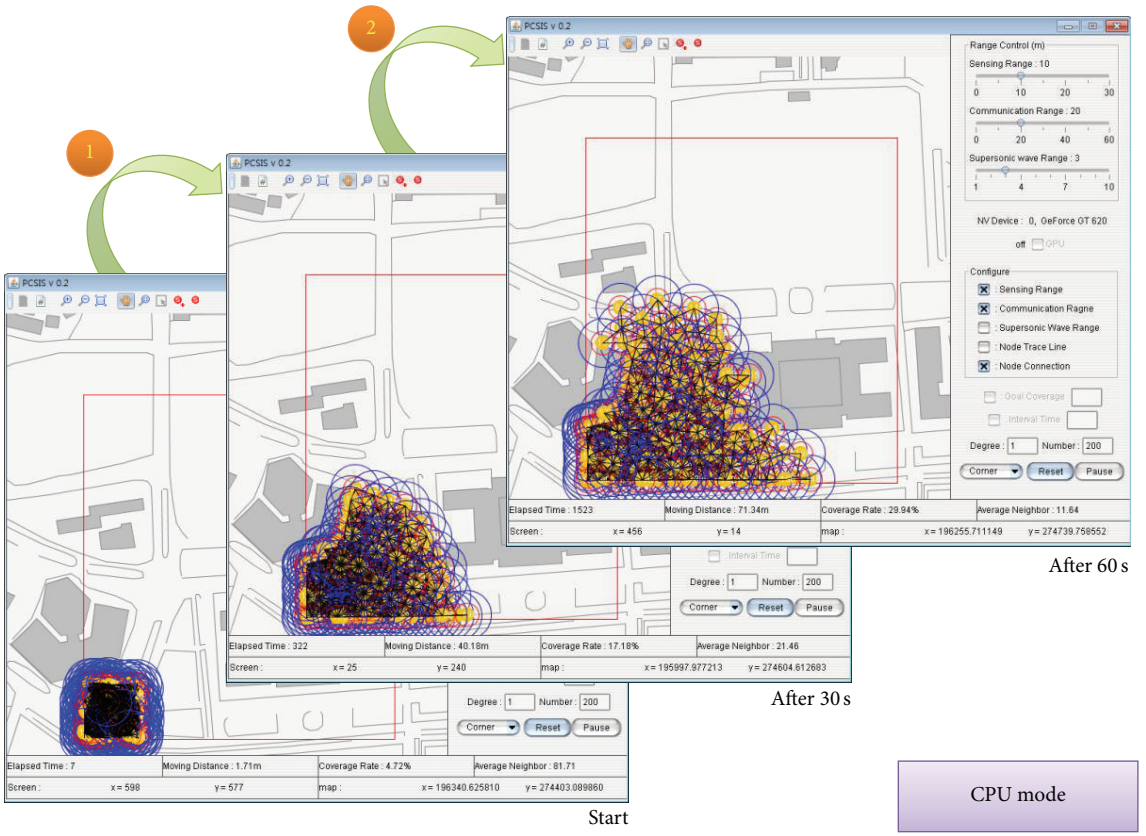

Figure 3: Simulation for 200 sensor nodes in the CPU mode.

The user can view the operating conditions of simulations through the viewer and draw expected problems by assessing and analyzing the conditions.

5. Implementation of PCSIS and Performance Evaluation

5.1. Implementation of PCSIS. The initial execution status of the PCSIS is shown in Figure 2. The tool bar in the upper part of Figure 2 consists of a button to read GML documents that can be mapped on actual topography, a button to expand, reduce, and move the views and select target observation regions, a button to add sensor nodes to an operating simulator, and a button for the matching of coordinates between moving sensor nodes and GML documents. The control view on the right side of Figure 2 shows the menu "range control" that enables the control of sensing ranges, communication, and supersonic wave ranges for sensor nodes. It also provides a checkbox to set whether the GPU mode will be used or not and selective visualization regarding the traces of range and movement for each sensor and the connection between sensors in order to configure the display of simulations during the run of simulations. The setup part for GPU mode in Figure 2 outputs the NV device that displays the identification number of each graphic device and the graphic model name in an operating environment. In addition, it can set the coverage of target regions as the range for the completion of simulations and define the basic number of operating sensor nodes when starting simulations. Figure 2 provides visualization on the variables of elapsed time, moving distance, coverage rate, average neighbor, and coordinates on the screen and actual maps.

Figure 3 shows the screens in which target regions and sensor nodes are set after reading GML documents that can be mapped on actual topography and run a simulation. ① and ② illustrate a sequential flow of views. Each view exhibits the conditions at the start of simulation, after 30 seconds of simulation, and after 60 seconds of simulation.

Figure 4 shows a simulation run by selecting the GPU mode in the same sensor condition as that in Figure 3. ① and ② exhibit a sequential flow of views. Compared with Figure 3, performance differences in the movement of each node can be confirmed with the naked eye.

5.2. Performance Evaluation. This section compares the execution speeds of using only the CPU and using the GPU in the PCSIS proposed by this study. The comparison was performed by examining performance differences in terms of temporal outputs through the computation of the coverage rate with time, while using the same number of sensor nodes within the same target regions. In addition, the status of performance according to the number of nodes was compared by increasing the number by 100 nodes each time.

Figure 5 presents the coverage rates with time when using 100 sensor nodes. The coverage rate is the percentage of the

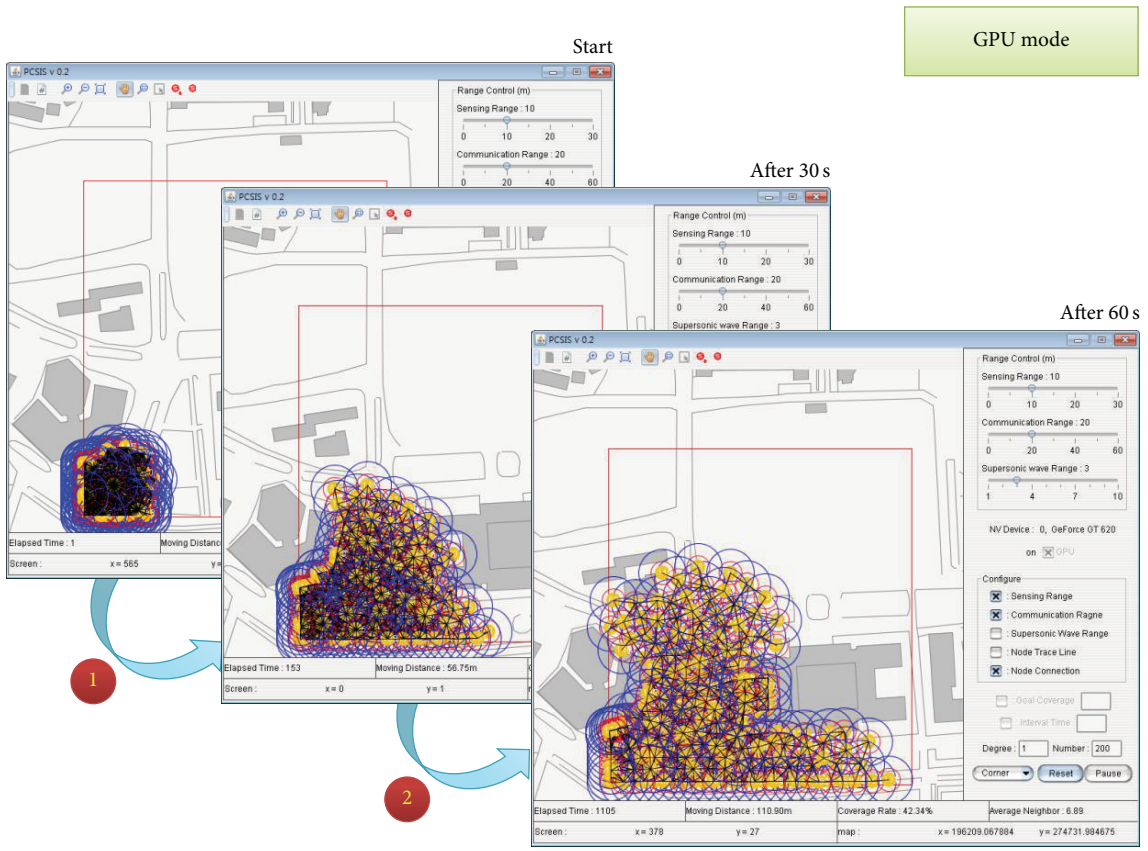

Figure 4: Simulation for 200 sensor nodes in the GPU mode.

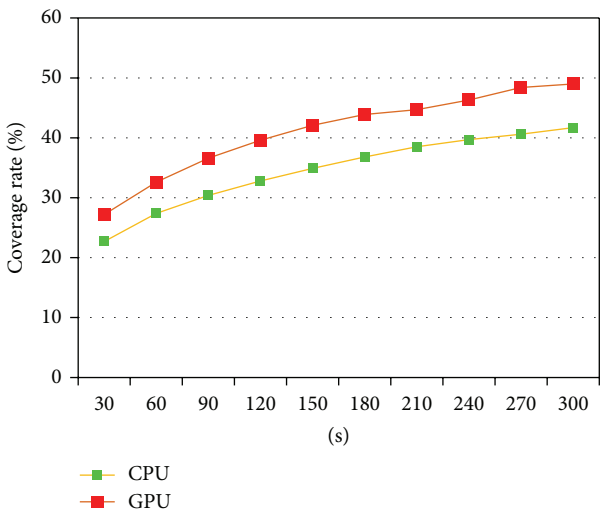

Figure 5: Coverage rates with time in the case of using 100 sensor nodes.

sensing area of sensor nodes to the target sensing region. The graph shows two situations of using the CPU and the GPU according to the calculation method for the next location of each sensor. Figure 5 exhibits that the coverage rates are similar between 60 seconds of using the GPU and 120 seconds of using the CPU.

This indicates that a task performed by using the CPU for 120 seconds can be achieved by using the GPU only for 60 seconds.

Figure 6 presents the results of using 100, 200, and 300 sensor nodes, respectively, by increasing the number of sensors by 100 nodes each time. The figure proves that each increase in the number of nodes resulted in a much greater level of performance improvement when using the GPU than using only the CPU.

However, if the number of nodes exceeds the number of threads that can be run at a time in a graphic device, their operation becomes rather ineffective. A key reason is that an increase in the number of nodes requires a corresponding increase in the data that should be duplicated from host to device, which, in turn, increases the delay time. In addition, the time required for running a number of threads in the CPU and showing the progress on the screen becomes longer than the time for the duplication of data transmitted from host to device. Moreover, an excess number of threads may cause a lot of processing, which subsequently increases the execution time.

6. Conclusion and Future Works

Existing WSN simulators have the disadvantage that an increase in the number of sensor nodes necessary for running simulations markedly slows down the execution speed. In this

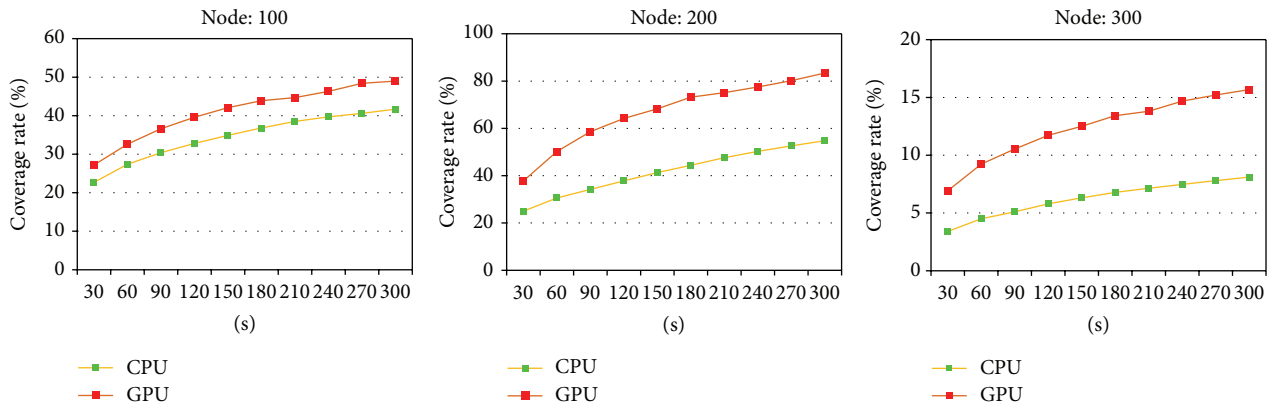

Figure 6: Performances by increasing the number of sensor nodes by 100 nodes.

respect, this study proposed a PCSIS that provides the GPU mode aimed at improving the performance of existing WSN simulators. The function of the GPU mode is the parallel processing of arithmetic units that should be separately calculated for each sensor node by using GPU cores. In addition, the results of increasing the number of nodes and the operating time of the simulator revealed that using the GPU yielded much faster execution speeds than using only the CPU. This demonstrates that the use of GPUs is effective for improving the performance of simulators. A follow-up study is planned to enable the incorporation of modularity into most WSN simulators by applying the parallel run of GPUs. In addition, given that the types of graphic devices increase with time, research will be performed to provide an interface that allows the user to define the number of grids, blocks, and threads per block.

Acknowledgments

This research is supported by the MSIP (Ministry of Science, ICT and Future Planning), Republic of Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-H0301-13-4007) supervised by the NIPA (National IT Industry Promotion Agency).

References

[1] B. Musznicki and P. Zwierzykowski, "Survey of simulators for wireless sensor networks," *International Journal of Grid and Distributed Computing*, vol. 5, no. 3, pp. 23–50, 2012.

[2] J. Chen, M. B. Salim, and M. Matsumoto, "A single mobile target tracking in Voronoi-based clustered wireless sensor network," *Journal of Information Processing Systems*, vol. 6, no. 4, pp. 17–28, 2010.

[3] A. U. Bandaranayake, V. Pandit, and D. P. Agrawal, "Indoor link quality comparison of IEEE 802. 11a channels in a multi-radio Mesh network testbed," *Journal of Information Processing Systems*, vol. 8, no. 1, pp. 1–20, 2012.

[4] S. Silas, K. Ezra, and E. B. Rajsingh, "A novel fault tolerant service selection framework for pervasive computing," *Human-Centric Computing and Information Sciences*, vol. 2, no. 5, pp. 1–14, 2012.

[5] X. Zhou, Y. Ge, X. Chen, Y. Jing, and W. Sun, "A distributed cache based reliable service execution and recovery approach in MANETs," *Journal of Convergence*, vol. 3, no. 1, pp. 5–12, 2012.

[6] Y. S. Jeong, Y. H. Han, J. J. Park, and S. Y. Lee, "MSNS: mobile sensor network simulator for area coverage and obstacle avoidance based on GML," *EURASIP Journal on Wireless Communications and Networking*, vol. 95, no. 1, pp. 1–15, 2012.

[7] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "GlomoSim: a scalable network simulation environment," UCLA CSD Technical Report #990027, UCLA, 1999.

[8] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, "ATEMU: a fine-grained sensor network simulator," in *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (IEEE SECON '04)*, pp. 145–152, October 2004.

[9] "The Network Simulator—ns—2," http://www.isi.edu/nsnam/ns/.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 126–137, November 2003.

[11] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, pp. 477–482, April 2005.

[12] "Java in simulation time/scalable wireless Ad hoc network simulator," http://jist.ece.cornell.edu/.

[13] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and B. K. Szymanski, "SENSE: a wireless sensor network simulator," http://www.ita.cs.rpi.edu/publications/sense-book-chapter.pdf.

[14] H. Joe, S. Y. Yoon, J. Hong, and H. Kim, "The sensor network simulation on the supercomputer," in *Proceedings of the Korea Computer Congress (KCC '11)*, vol. 38, pp. 442–445, 2011.

[15] "NVIDIA CUDA," http://www.nvidia.com/content/global/global.php.