GPU-based Parallel Computing of Energy Consumption in Wireless Sensor Networks

Massinissa Lounis^{1,2}, Ahcène Bounceur^{1,2}, Arezki Laga^{1}, Bernard Pottier^{1}
^{1}Lab-STICC Laboratory
University of Brest, France
^{2}LIMED Laboratory
University Abderrahmane Mira of Bejaia, Algiers
Email: Massinissa.Lounis@univ-bejaia.dz

***Abstract*—The lifetime of a wireless sensor network is the most important design parameter to take into account. Given the autonomous nature of the sensor nodes, this period is mainly related to their energy consumption. Hence, the high interest to evaluate through accurate and rapid simulations the energy consumption for this kind of networks. However, in the case of a network with several thousand nodes, the simulation time can be very slow and even impossible in some cases. In this paper, we present a new model for a parallel computing of energy consumption in wireless sensor networks. This model is combined with a discrete event simulation in a multi-agent environment and implemented on GPU architecture. The results show that the proposed model provides simulation times significantly faster than those obtained by the sequential model for large networks and for long simulations. This improvement is more significant if the processing on each node is very time consuming. Finally, the proposed model has been fully integrated and validated into the *CupCarbon* simulator[1].**

I. Introduction

Wireless sensor networks (WSN) are ad hoc networks with limited resources. Sensor nodes are deployed in different places generally inaccessible and could collaboratively monitor physical or environmental conditions such as temperature, pollution, vibration, etc. To assist the design and study of a wireless sensor network, several simulation tools have been proposed. There are a large number of simulation and modeling tools for WSNs, the authors of reference [1] present a survey of 36 simulators and emulators. The simulation can recreate a real physical complex scenario by executing a program on a computer or network while emulation aims to substitute software by hardware.

The simulators of WSNs based on their energy models can be classified into two main groups, single-node simulators and network simulators. Simulators that work at the level of single sensor nodes are, for example, PowerTOSSIM [2] and Avrora [3]. PowerTOSSIM is an extension of TOSSIM, which is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a sensor node, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. PowerTOSSIM estimate the number of CPU cycles executed by each node. It includes a detailed model of hardware energy consumption based on the Mica2 sensor node platform [4]. Avrora uses an event queue that enables efficient instruction-level simulation of microcontroller programs and allows the hidden parallelism in fine grained sensor network simulations. Landsiedel et al [5] have added a highly accurate energy model to Avrora, enabling power profiling and lifetime prediction of sensor networks.

The second class represents simulators at the network level. They are completely independent from the sensor node hardware. This means that the estimations produced by these simulators depend strongly on the specific resource consumption models used in a simulation. In most simulations, CPU usage is not considered at all. Examples of such simulators are: OMNeT++ [6], SENSE [7] and NS-2 [8]. Authors in [9] propose an energy model that can be integrated in OMNeT++. This model distinguishes the different power consumption rates in each radio state and it explicitly considers the necessary transition energy. A simple CPU model have been built to estimate the energy consumption for computationally intensive operations. In SENSE, the power component is responsible for power management. It includes, a SimplePower component, which can operate on any of 5 modes: TRANSMIT, RECEIVE, IDLE, SLEEP, and OFF. Four parameters specify the energy consumption rate under each of the first modes, while in the OFF mode there is no energy consumption. The power component accepts control from networking components. In responses to the control signal, it can switch from one mode to another. NS-2 implements a simple energy model, and in this model, the energy consumed in receiving and sending data, listening to communication channel, and idle could be parameterized.

In the present work, we introduce a new model combining a discrete events simulation model in a multi-agent environment and GPU-based parallel simulation to compute the energy consumption in wireless sensor network.

This paper is organized as follows, In section II *CupCarbon* will be presented. Section III will describe some basic notion on graphics processing units (GPUs). Sequential and parallel simulation models are presented in Section IV and V. comparison results of the energy consumption of sensor nodes obtained with sequential and parallel model will be presented

[1]This project is supported by the French Agence Nationale de la Recherche ANR - PERSEPTEUR - REF: ANR-14-CE24-0017-01.

in Section VI. Finally, Section VII presents the conclusion.

II. CupCarbon simulator

CupCarbon [10][11] is discret event multi-agent wireless sensor networks simulator based on geolocation. It allows designing and simulating networks on *OpenStreetMap*, a geographical map. In order to do this, *CupCarbon* provides a set of easy to manipulate and configurable objects. The use of multi-agent systems optimizes the simulation time by parallelizing the different objects and events. *CupCarbon* is composed of three main blocks: the multi-agent environment simulator, the online wireless sensor network (wsn) simulator and the offline wsn simulator (*SimBox*). The multi-agent environment simulator can simulate objects such as sensor nodes and their settings (eg: variations of range), the movement of mobiles (eg: car, bus, flying objects [12], etc.), environmental phenomena (fire, gas, etc.).

III. GPU Architecture

A GPU architecture is generally composed of several graphic processors ranging from 2 to 32 in some powerful graphic cards. They also have different types of memory (global, constant, and local). Operations on the global memory can be read/write from all threads but each thread has only access to his local memory, while access to the constant memory is on read only.

A GPU is composed of several hundreds of cores. They contain a read/write memory shared by the threads of the same block. Figure 1 shows that $3/4$ of the GPU area is dedicated to computing units (Blue), while in CPU only $1/8$ of the area is dedicated to the computing.

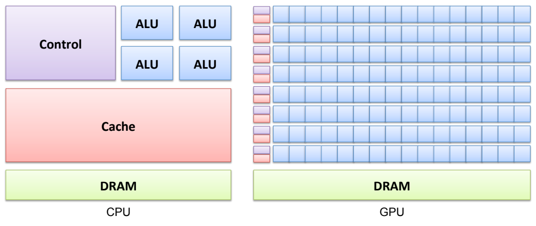


Fig. 1. Difference Between GPU and CPU Architecture

GPU architecture belongs to the family of SIMD parallel architectures *(Single Instruction Multiple Data)*. In other words, in GPU architecture, all threads execute the same instructions. In wireless sensor networks, the sensor nodes have different behaviors. If we translate their behavior in a classical algorithmic model, we will have different instructions for each sensor node.

IV. Sequential Model for Energy Consumption Computing

This section presents a sequential simulation model to compute the energy consumption of sensor nodes in a static or mobile networks. The current version of this model doesn't take into account packets sending and their routing. It only takes into account packets size and sensor nodes waiting times. We will proceed in two principal steps. The first step applies our model to a static networks, the second step shows how mobility was integrated in this model.

A. Case of static network

To illustrate the discrete event simulation algorithm of *SimBox*, we introduce some terminologies to facilitate the comprehension of this algorithm. To do so, we will focus on the chart of Figure 2.

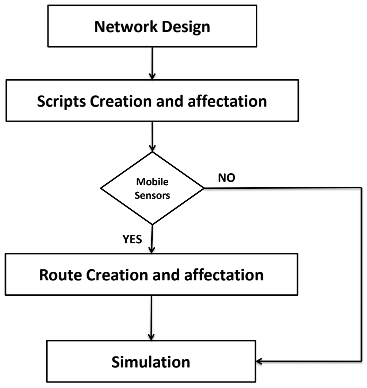

Network Design
Scripts Creation and affectation
Mobile Sensors
NO
YES
Route Creation and affectation
Simulation

Fig. 2. *CupCarbon* simulation flow.

The first step is to design the network to know the different communication links between the different sensor nodes.

These links are described by matrix $A_{n,n}$ as follows:

$$A = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n0} & a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

where a_{ij} is defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if sensor node } i \text{ communicates with sensor node } j, \\ 0 & \text{otherwise.} \end{cases}$$

Note that a sensor communicating with another sensor means that it locates in its radio range.

The second step in the chart of Figure 2 is to create communication scripts and assign them to each sensor node. In a script we have two types of instructions; the first type is `SEND x` which specifies that the sensor node must send a packet of x bytes, the second instruction is `DELAY y` which specifies that the sensor node should not do anything for y milliseconds. Figure 3 shows an example of a communication script where the arguments of the $SEND$ are specified in bytes and those of the $DELAY$ in milliseconds.

To standardize the units of a communication script, we choose to use the bits. For time we can use seconds. Therefore, the transmission of bytes must be converted into bits. Thus, x will be transformed into $(x \times 8)$ and y to $(y \times f/1000)$, where f is the frequency of the radio communication modules in bit/second, which must be the same for each module. For ZigBee modules (2.4 GHz) this speed is equal to $250k$

```
SEND 125
DELAY 100
SEND 50
DELAY 1000
```

(a)

```
SEND 1000
DELAY 960
SEND 400
DELAY 9600
```

(b)

Fig. 3. Example of sensor communication script: (a) not standardized and (b) standardized.

bits/second. Now we consider the variables x and y transformed (in bits). The script in Figure 3 standardizes all arguments as it is shown in Figure 3(b). To simplify the illustration of our algorithm we have chosen to use as communication frequency 9600 bits/second instead of $250k$ bits/second.

We use the term *duration* to describe the value of the instruction argument knowing that communication scripts are standardized. This means that, the duration of an operation $SENDx$ is equal to x bits and the duration of an operation $DELAYy$ is equal to y milliseconds.

We define the set of variables used by the algorithm as follows:

- The variable q_i represents the index of the instruction to be executed by the sensor node i.
- The variables $opType_{i,q_i}$ and $opArg_{i,qi}$ represent respectively types ($SEND$ or $DELAY$) and the value of the number instruction argument q_i to run by the sensor i.
- The variable $energy_i$ represents the energy of the sensor node i.
- The boolean variable $dead_i$ indicates if a sensor node is dead ($dead_i = 1$) or not ($dead_i = 1$).

The function $f(v)$ takes the value 1 if v is equal to a SEND operation, 0 if v is equal to a DELAY operation. It is given as follows:

$$f(v) = \begin{cases} 1 & \text{if } v = SEND, \\ 0 & \text{if } v = DELAY. \end{cases}$$

We define also few constants. Constant m represents the number of instruction in the sensor node i script. To simplify the presentation, we assume that this constant is the same for each sensor node. The constant e_{0_i} represents the initial energy of the sensor node i. Constants E_{T_x} et E_{R_x} represent respectively the energy of transmission and reception.

We consider as event in a given iteration the duration of an operation that each sensor node must consume (by send or delay), and which must be the same for each sensor. It is given by the minimum of each current sensor node duration. The different steps of the simulation algorithm are summarized in Figure 4.

If we consider a network with n sensor nodes, these statements are detailed as follows:

- Step 1: Initialization (For every sensor i)
  - Affect its initial energy: $energy_i = e_{0_i}$.
  - Prepare it for the first instruction execution: $q_i = 0$.
  - Declare it as an alive sensor node: $dead_i = 0$.
  - His current event is the argument of the first instruction: $event_i = opArg_{i,0}$ (the duration of the current operation).
- Step 2: Compute the current event ($currEvent$)

$$currEvent = \min\{event_i, i = 1, ..., n\}.$$

- Step 3: Compute the energy consumption of the sending actions (for each sensor i)
  If ($opType_{i,q_i} = SEND$) then:

$$energy_i = energy_i - currEvent \times E_{T_x}.$$

- Step 4: Compute the energy consumption of the receiving actions (for each sensor i)

$$energy_i = energy_i - (currEvent \times E_{R_x} \times c),$$

  where

$$c = \sum_{j=1, j\neq i}^{n} a_{ij} \cdot f(opType_{j,q_j}) \cdot (1 - dead_j).$$

- Step 5: Execute the current event (for each sensor node i)

$$event_i = event_i - currEvent.$$

- Step 6: Test if there are dead sensor nodes (for each sensor i)
  - if ($energy_i = 0$) then: $dead_i = 1$ and $event_i = e_{0_i}$.
  - if all sensor nodes are dead: Stop simulation.
- Step 7: Go to the next instruction (for each sensor node i)
  - If ($event_i = 0$) then:

$$q_i = (q_i + 1) \; mod \; m$$

    and

$$event_i = opArg_{i,q_i},$$

    where mod denotes the modulo, it allows to run the script in a loop.

B. Case of a Mobile Network

We consider in this part that nodes can be mobile. Thus, there will be two types of possible events. Either the node communicates (executes send or delay insructions) or the node moves. That will change the previous algorithm when the current event is a move. The energy consumption must be computed after changing the position of the sensor node. This generates the updating of the different communication links between the sensor nodes in the matrix A. By adding a moving state of the sensor nodes to the simulation algorithm stats presented in Figure 4, we get a new simulation algorithm taking into account the mobility.

We describe therefore two additional variables $event_{1_i}$ and $event_{2_i}$. The first variable $event_{1_i}$ replaces the variable $event_i$ defined above. In other words, if the event is send or wait for a sensor node i, The event $event_{2_i}$ will be the movements of a sensor i. The current event $currEvent$ is defined as follows:

$$currEvent = \min\{event_{1_i}, event_{2_i}, i = 1, ..., n\}.$$

We also define the variable gps_{i,p_i} which represents the time taken by the sensor node i to go from point p_{i-1} to point p_i.

V. Parallel Model for Energy Consumption Computing

The basic idea of the parallel energy computing model of sensor nodes is to transpose the difference in behavior between nodes on the data. We get then an algorithmic model which consists of a set of instructions running on multiple sets of data. This corresponds perfectly to the SIMD architecture.

As we will see it below, the proposed model corresponds to the discrete event simulation algorithm changed from algorithmic model to a model based on matrix multiplications. In the following, we assume that the sensor nodes only use two types of operations, those that consume energy (SEND) and those which do not consume energy (DELAY). We define a Boolean vector v_{i,q_i} for each sensor node to determine if in a given step, the sensor node i sends ($v_{i,q_i} = 1$) or not ($v_{i,q_i} = 0$) a packet. We set the matrix A diagonal to 1 ($a_{ii} = 1$ for $i = 1, .., n$). This defines that a sensor node is linked to itself. This will allow us to assume that we will have only receiving actions and not sending actions, because we present a sensor node that send a packet as node that receive a packet from itself. This procedure will allow us to win a step on the algorithm and to make only one access instead of two in the GPU. We illustrate this procedure as follows. Consider a network of 4 sensor nodes. The sensor node 1 sends a packet ($v_{1,q_1} = 1$) and the sensor node 3 which is its neighbor ($a_{13} = 1$) sends also a packet ($v_{3,q_3} = 1$). So the sensor node 1 will consume the energy associated to the packet sending and it will also consume the energy associated to the reception of the packet sent by the sensor node 3. If we assume that the reception energy is equal to the sending energy, the sensor node 1 therefore will consume $c_1 = 2 \times e$ where e is the energy associated to the reception or transmission of a packet. Assume now that the sensor node 2 is not a neighbor of sensor node 1($a_{12} = 0$) and it sends a packet ($v_{2,q_2} = 1$), the sensor node 1 consumption will not be affected by this send action because the sensor node 2 does not communicate with sensor node 1. Assume also that the sensor node 4 is connected to sensor node 1 ($a_{14} = 1$) but it sends nothing ($v_{4,q_4} = 0$). The computation of the sensor node 1 energy consumption can be obtained by the following formula.

$$c_1 = \sum_{j=1}^{4} a_{1j} \cdot v_j$$

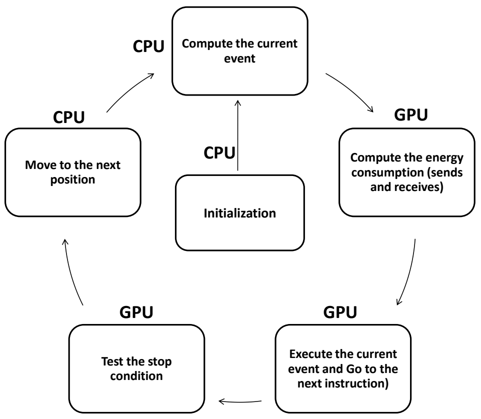
$$c_1 = a_{11}v_{1,q_1} + a_{12}v_{2,q_2} + a_{13}v_{3,q_3} + a_{14}v_{4,q_4}$$

$$c_1 = 1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 0 = 2.$$

The generic formula of consumption (of reception and transmission) of a sensor node i which takes into account its status (dead or alive) is given as follow:

$$c_i = \sum_{j=1}^{n} a_{ij} \cdot v_{j,q_j} \cdot (1 - dead_i).$$

The advantage of this method is that it allows to parallelize the computation of energy consumption of each sensor i by lunching it on threads (each comutation c_i is lunched in a separated thread).



CPU
Compute the current event
GPU
Compute the energy consumption (sends and receives)
GPU
Execute the current event and Go to the next instruction)
GPU
Test the stop condition
CPU
Move to the next position
CPU
Initialization

Fig. 4. Discrete event simulation algorithm states

VI. Case Study and Discussion

We have implemented the parallel part of the proposed model using OpenCL and CUDA (Compute Unified Device Architecture) [13] on a NVIDIA GeForce graphic card. As the best results are obtained with CUDA, in this paper we will illustrate only the restults obtained by this one. The developer can use the computing power of a graphic cards with some operations intended to be processed by the GPU instead of the CPU. However, this last one is always necessary to coordinate CPU and GPU work. The GPU is thus seen like a massively parallel processor adapted very well to the processing of parallel algorithms. An operation executed on the GPU is called a kernel.

The execution of a CUDA program is preformed as follows:

- Initially the program is run by the CPU
- A kernel is invoked, it will be executed on GPU.
- A large number of threads are generated and executed in parallel on the GPU

The CUDA allows replication of the hardware architecture specifications of a GPU (NVIDIA GeForce GTX 480 in our case) in a software level and manages the communication between CPU and GPU. This helps to see the GPU as a computing grid formed of one, two or three dimensions of independent computational blocks.
Each of these blocks is decomposed into a matrix of threads with one, two or three dimensions. The developer must arrange the blocks on the grid and determine block dimension and

size according to the characteristics of the application. For example, to multiply two matrices, the developer will choose the blocks in two dimensions, and three dimensions if he makes an operation on volumes.

A. Case Study

To illustrate the algorithm, we follow the steps of Figure 2 by considering a static network. We start with the network design. We have taken the example of a network of three sensor nodes (S0, S1 and S2) linearly positioned. The corresponding A matrix is given as follows:

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Then we describe the various communication scripts and we assign them to each sensor node. These scripts are given in Table I where the units are standardized in bits. Note that for simplicity, we have chosen as communication speed 9600 bits/second instead of $250k$ bits/second

TABLE I
COMMUNICATION SCRIPTS

Script (S0)	Script (S1)	Script (S2)

`SEND 1000`
`DELAY 960`
	`SEND 2000`
	`DELAY 960`
		`SEND 800`
		`DELAY 960`

Then we launch the simulation, the results are given in Table II.

We will use these simulation results to illustrate the progress of the proposed algorithm. First we start with the initialization step. The initial energy of all sensor nodes is set to $e_0 = 3000$. The $q_i,\ i = 1, 2, 3$ are fixed to 0 to execute the first instruction of each script (cf. Table III). The status of each sensor node is initialized to alive ($dead_i = 0,\ i = 1, 2, 3$). For simplicity, we fixed $E_{T_x} = E_{R_x} = 1$.

TABLE III
COMMUNICATION SCRIPTS (ITER=1)

q_0	Script (S0)	q_1	Script (S1)	q_2	Script (S2)

0→	**`SEND 1000`**
	`DELAY 960`
		0→	**`SEND 2000`**
			`DELAY 960`
				0→	**`SEND 800`**
					`DELAY 960`

At iteration 1, we look for the current event ($currEvent$) which represents the minimum of the events of each sensor. These events represent the instruction argument executed by each sensor node. In other words, in our case, the arguments are: 1000, 2000 and 800. The minimum is 800 shown in bold in the first row of Table II. This value must be subtracted from each of the events of each sensor which will give us new events values, respectively, 200, 1200 and 0. Then we compute for each sensor node the energy consumption generated by sending 800 bits by other sensor nodes. For example for the sensor node $S0$, which sends a packet ($f(opType_{0,q_0}) = f(SEND) = 1$), it will consume the equivalent of 800 bits. Furthermore, as this sensor node is only linked to the sensor $S1$ which also sends a packet ($f(opType_{1,q_1}) = f(SEND) = 1$), the sensor node $S0$ will consume the equivalent of 800 bits. So that the sensor node $S0$ will consume in this step the energy linked to the sending and receiving of 1600 bits. Thus, for the sensor node $S0$ it will has only 1400 ($3000 - 1600$) units of its energy. The results obtained for the other sensor nodes are given by the row 2 (Iter=2), Energy part of Table II.

Now, it remains to move to the next instructions. The sensor nodes which have event equal to zero will execute their next instructions. In our case, the sensor node $S2$ has an event equal to zero. Then, it will execute the next instruction (ie. $q_2 = q_2 + 1 = 0 + 1 = 1$). The result is given in Table IV.

Then we go to iteration 2. In this iteration we will do exactly the same steps while considering new event values (Event). In our case, we take the values of row 2 (Iter=2) in Table II. Note that when the sensor node i energy is equal to zero, this sensor node will die ($dead_i = 1$).

TABLE IV
COMMUNICATION SCRIPTS (ITER=2)

q_0	Script (S0)	q_1	Script (S1)	q_2	Script (S2)

0→	**`SEND 1000`**
	`DELAY 960`
		0→	**`SEND 2000`**
			`DELAY 960`
				
				1→
					`SEND 800`
					`DELAY 960`

B. Results and Discussions

This sections compares the two discrete event algorithms presented above. The first is based on the CPU simulation and the second on the GPU simulation. This comparison validates which algorithm is appropriate to use in a given situation.

To compare these two algorithms, we decided to proceed in two ways. The first consists in fixing the number of simulation iterations and change the number of sensor nodes in the network. The second consists in fixing the number of sensor nodes and change number of iterations. Networks are randomly generated. In the first case, we set the number of iterations to 1000 and varying the number of sensor nodes from 100 to 10000. The resulting graph is shown in Figure 5. We can give as an example the case of a network of 8000 sensor nodes. It is simulated in 5 minutes and 45 seconds on the CPU, however it can be simulated only in 30 seconds on the GPU. The corresponding acceleration graph is given by Figure 6.

Note that for a network of less than 500 sensor nodes, simulation times obtained by each algorithm are almost identical. Consequently it is not necessary to use the GPU.

In the second case, we set the number of sensor nodes to 3000 and varing the number of iterations form 1000 to 15000. The resulting graph is shown in Figure 7. As in the first case, the graph shows that the simulations performed on the GPU are 8 times faster than those running on the CPU. This is confirmed by the acceleration graph given by Figure 8.

VII. CONCLUSION

Most existing wireless sensor network simulators have the disadvantage of being very slow when the number of network

TABLE II
THE SIMULATION RESULTS

Iter	Time	Event (bits) [Die (0/1)]			Energy (units)		
1	0	1000 [0]	2000 [0]	**800** [0]	3000	3000	3000
2	800=0+**800**	**200** [0]	1200 [0]	960 [0]	1400	600	1400
3	1000=800+**200**	960 [0]	1000 [0]	**760** [0]	1000	200	1200
4	1760=1000+**760**	**200** [0]	3000 [1]	800 [0]	240	0	440
5	1960=1760+**200**	1000 [0]	3000 [1]	**600** [0]	240	0	240
6	2560=1960+**600**	3000 [1]	3000 [1]	3000 [1]	0	0	0

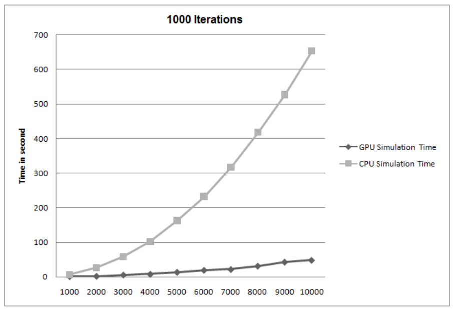


Fig. 5. Simulation time according to the number of sensor nodes.

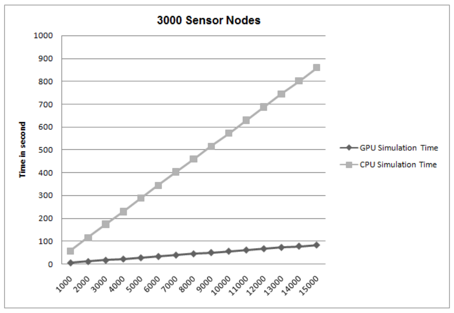


Fig. 7. Simulation time according to the number of iterations.

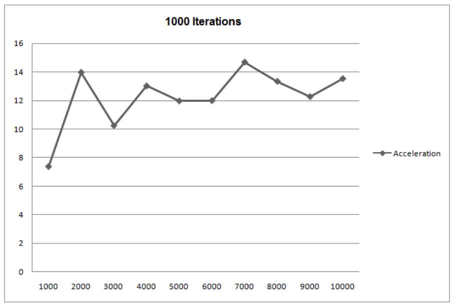


Fig. 6. Acceleration according to the number of sensor nodes

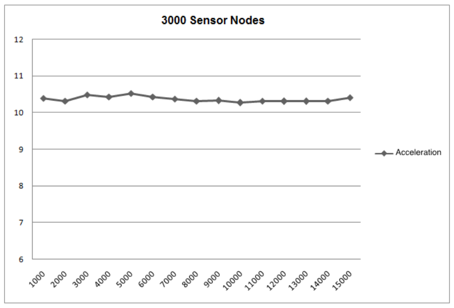


Fig. 8. Acceleration according to the number of iterations

sensor nodes exceeds hundreds. One of the solutions to explore in order to accelerate these simulations is the use of GPUs which are inexpensive parallel architectures. In this work we have proposed a new parallel model to simulate wireless sensor networks using GPU. This network can be static or mobile. The simulation results show the need to use the simulation on the GPU for networks exceeding 1000 sensor nodes where simulations have been accelerated from 6 to 25 times. These accelerations can be greater if the processing on each sensor node is very time consuming. The proposed model has been implemented and validated in the simulator *CupCarbon*, a tool to simulate wireless sensor networks in a multi-agent environment.

REFERENCES

[1] Bartosz Musznicki and Piotr Zwierzykowski. Survey of simulators for wireless sensor networks. *Int J Grid Dist Comput*, 5(3):23–50, 2012.
[2] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.
[3] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. *Information Processing in Sensor Networks, IPSN 2005. Fourth International Symposium*, 2005.
[4] University of california berkeley, mica2 schematics,, 2003.
[5] O. Landsiedel, K. Wehrle, S. Rieche, S. Gotz, and L. Petrak. Accurate prediction of power consumption in sensor networks. *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop*, 2005.
[6] A. Varga. The omnet++ discrete event simulation system. *European Simulation Multiconference (ESM 2001)*, 2001.
[7] Gilbert Chen, Joel Branch, Michael J. Pflug, Lijuan Zhu, , and Boleslaw K. Szymanski. Sense: Awireless sensor network simulator. *Advances in Pervasive Computing and Networking, Springer, New York, NY, 2004, pp. 249-267*, 2004.
[8] NS-2 official website. http://www.isi.edu/nsnam/ns/.
[9] Laura Marie Feeney and Daniel Willkomm. Energy framework: An extensible framework for simulating battery consumption in wireless networks. pages 20:1–20:4, 2010.
[10] Kamal Mehdi, Massinissa Lounis, Ahcène Bounceur, and Tahar Kechadi. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. *In IEEE 7th International Conference on Simulation Tools and Techniques (SIMUTools'14), Lisbon, Portugal*, March 17-19 2014.
[11] http://www.cupcarbon.com. Version 1.0 (bêta). 2014.
[12] M. Lounis, K. Mehdi, and A. Bounceur. A cupcarbon tool for simulating destructive insect movements. *1st IEEE International Conference on Information and Communication Technologies for Disaster Management (ICT-DM'14), Algiers, Algeria*, March 24-25 2014.
[13] Jason Sanders and Edward Kandrot. Cuda by example: An introduction to general-purpose gpu programming. 2010.