Estimation of Energy Consumption through Parallel Computing in Wireless Sensor Networks

Article *in* Journal of Ambient Intelligence and Humanized Computing · August 2017

DOI: 10.1007/s12652-017-0582-5

CITATIONS
4

READS
637

4 authors:

Massinissa Lounis
Université de Bretagne Occidentale
13 PUBLICATIONS **72** CITATIONS
SEE PROFILE

Ahcène Bounceur
Université de Bretagne Occidentale
159 PUBLICATIONS **981** CITATIONS
SEE PROFILE

Reinhardt Euler
Université de Bretagne Occidentale
139 PUBLICATIONS **492** CITATIONS
SEE PROFILE

B. Pottier
Université de Bretagne Occidentale
129 PUBLICATIONS **436** CITATIONS
SEE PROFILE

Noname manuscript No.
(will be inserted by the editor)

Estimation of Energy Consumption through Parallel Computing in Wireless Sensor Networks

Massinissa Lounis · Ahcène Bounceur · Reinhardt Euler · Bernard Pottier

Received: date / Accepted: date

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 can be very slow and even impossible. In this paper, we present a new model for computing the energy consumption in wireless sensor networks in parallel. The model uses discrete event simulation implemented on a massively parallel GPU architecture. The results show that the proposed model provides simulation times significantly shorter than those obtained with the sequential model for large networks and for long simulations. This improvement is even more significant if the processing on each node is very time consuming. Finally, the proposed model has been fully integrated and validated on the *CupCarbon simulator*.

Keywords WSN Simulation · GPU Simulation · Energy Consumption Model · Discrete Event Simulation · CupCarbon

1 Introduction

Wireless sensor networks (WSN) are ad hoc networks with limited resources. Sensor nodes are deployed in different and often inaccessible places in the aim to collaboratively monitor physical or environmental conditions such as temperature, pollution, vibration, etc. Saving energy is one of the most important challenge in different fields (Kiani et al, 2014), more specifically in wireless sensor networks because they are autonomous networks and generally powered by a battery.

Massinissa Lounis
LIMED laboratory, Department of computer science, University of Bejaia, Algeria

Massnissa Lounis · Ahcène Bounceur · Reinhardt Euler · Bernard Pottier
Lab-STICC UMR CNRS 6285, Université de Bretagne Occidentale, 20 Avenue Victor Le Gorgeu, 29238 Brest, France
E-mail: massinissa.lounis@univ-bejaia.dz

To assist the design for the development of their application and protocols (Chen et al, 2016), several simulation and modeling tools have been proposed. The authors of reference (Musznicki and Zwierzykowski, 2012) present a survey of 36 simulators and emulators. Simulation can recreate a real physical scenario by executing a program on a computer or a network while emulation aims to substitute software by hardware.

The simulators of WSNs that are based on 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 (Levis et al, 2003) and Avrora (Titzer et al, 2005). 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 within the TOSSIM framework, which runs on a PC. This allows to debug, test, and analyze algorithms in a controlled and repeatable environment. PowerTOSSIM estimates 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 (Polastre, 2003). Avrora uses an event queue that enables efficient instruction-level simulation of microcontroller programs and allows a hidden parallelism in fine-grained sensor network simulations. Landsiedel et al. (Landsiedel et al, 2005) 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. Which 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++ (Varga, 2001), SENSE (Chen et al, 2005) and NS-2 (The University of Southern California, 2016; Issariyakul and Hossain, 2008). Authors in (Feeney and Willkomm, 2010) propose an energy model that can be integrated into 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 has been built to estimate the energy consumption for computationally intensive operations. In SENSE, the power component is responsible for power management. It includes a simple power component, which can operate in 5 modes: TRANSMIT, RECEIVE, IDLE, SLEEP, and OFF. Four parameters specify the energy consumption rate in each of the first four modes, while in the OFF mode there is no energy consumption. The power component accepts control from networking components. In response to the control signal, it can switch from one mode to another.

NS-2 implements a simple energy model in which the energy consumed in receiving and sending data, listening to the communication channel, or being idle could be parameterized. J-Sim (Sobeih et al, 2005) uses a power model for a sensor node which incorporates both the energy-producing components (battery) and the energy-consuming components (CPU and radio). The sensor function model is dependent on the power model. For example, the energy incurred in handling a received data packet is dictated by the CPU model, and the energy incurred in sending and/or receiving data packets (Malik et al, 2011) is dictated by the radio model. In SensorSim (Park et al, 2000), the power model consists of a single energy provider and multiple energy consumers. The battery is the energy provider with a finite amount of energy storage. The energy consumers are the radio module,

the CPU, and other various sensor devices including the geophone, the infrared detector, the microphone, etc. QualNet (Vir et al, 2013) offers a communication-oriented model to estimate the energy of reception and transmission. Energy is dissipated in active mode when the radio transmits or receives a packet, in sleep or idle modes of the transceivers, and for the transition among states. Further simulators and energy models have been presented in (Nayyar and Singh, 2015; Korkalainen et al, 2009; Singh et al, 2008).

In the present work, we introduce a new model based on discrete event simulation and implemented on a massively parallel GPU architecture to compute the energy consumption in wireless sensor networks.

This paper is organized as follows. Section 2 introduces the *CupCarbon* simulator, some basic notions on graphics processing units (GPUs) and explains the communication process between sensors. Sequential and parallel simulation models are presented in Section 3 and Section 4. Comparative results of the energy consumption of sensor nodes obtained with sequential and parallel models will be presented in Section 5 and Section 6 concludes the paper.

2 Background

In this section we will present the CupCarbon simulator, the GPU architecture and the communication processes between two sensor nodes. The parallel algorithm proposed in Section 4, used to calculate the energy consumption, is integrated into the CupCarbon simulator in order to deal with the scalability. It is implemented on a massively parallel GPU (Graphics Processing Unit) architecture. Explaining this architecture is necessary to understand the parallel model presented in this paper. To achieve this, the second part of this section will introduce the notion of GPU and its general architecture. Finally, the third part of this section presents the communication process (sending and receiving) between two sensor nodes. This part is essential to show how to calculate the time of sending and receiving data packets and their corresponding energy consumption.

2.1 CupCarbon Simulator

CupCarbon (Mehdi et al, 2014; Bounceur and Lounis, 2016; Lounis et al, 2015) is a Smart City and Internet of Things Wireless Sensor Network (SCI-WSN) simulator. Its objective is to design, visualize, debug and validate distributed algorithms for monitoring, environmental data collection, etc. and to create environmental scenarios such as fires, gas, mobiles, and generally within educational and scientific projects. CupCarbon offers two simulation environments. The first simulation environment enables the design of mobility scenarios and the generation of natural events such as fires and gas (Mehdi et al, 2014) as well as the simulation of mobiles such as vehicles and flying objects (e.g. UAVs, insects, etc.) (Lounis et al, 2014). The second simulation environment represents a discrete event simulation of wireless sensor networks which takes into account the scenario designed on the basis of the first environment.

Networks can be designed and prototyped by an ergonomic and easy to use interface using the OpenStreetMap (OSM) framework to deploy sensors directly

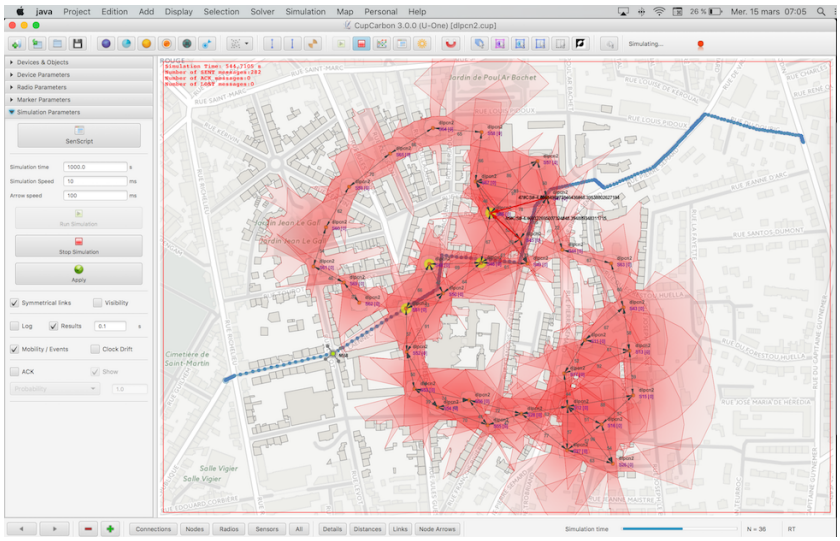

Fig. 1: The main user interface of the simulator *CupCarbon*

on the map. Figure 1 presents the main interface of the simulator. It includes a script called SenScript (Bounceur and Lounis, 2016) which allows to program and configure each sensor node individually. CupCarbon offers the possibility to simulate algorithms and scenarios in several steps. The energy consumption can be calculated and displayed as a function of the simulation time. This allows to clarify the structure, feasibility and realistic implementation of a network before its real deployment.

CupCarbon was designed with a modular structure to simplify replacement and customization of specific parts of the simulator. The modular structure provides two very important abilities. First, the structure simplifies customizing the target architecture, it is very easy to experiment with part of the architecture while keeping the rest unchanged. This makes CupCarbon useful for simulating specific wireless sensor networks. Second, the modular structure promotes multiple implementations of a given module, which allows users to switch between very accurate and very fast versions. The main modules of the CupCarbon simulator are:

1. IHM Box (contains the main program and the graphical user interface)
2. Map Box (contains the digital model of a city)
3. SenScript Box (for the sensor node programming)
4. Devices Box (contains the sensor nodes, radio models, mobiles, etc.)
5. Signal Box (contains the radio propagation and radio interference models)
6. SimBox (for the energy consumption calculation)
7. WisenSimulation Box (contains the discrete event simulator)
8. CodeGen Box (to generate hardware platform codes)

Figure 2 shows the interaction between the different modules of CupCarbon and the SimBox module, which contains the sequential and the parallel models for computing the energy consumption proposed in this work.

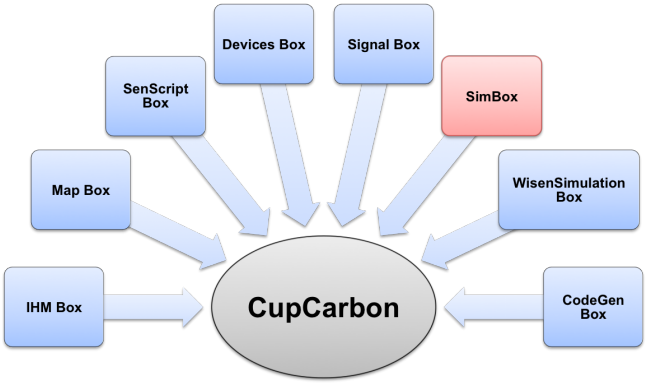

Fig. 2: CupCarbon modules.

2.2 GPU Architecture

A GPU architecture is generally composed of several graphics processors ranging from 2 to 32 in some powerful graphics 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 3 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 computing.

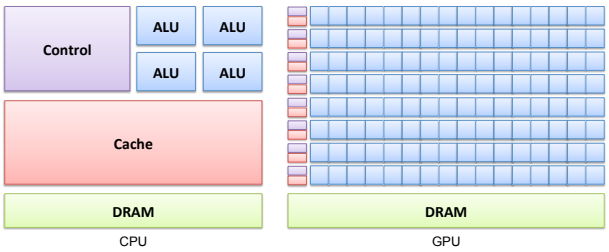

Fig. 3: Difference between CPU and GPU Architectures

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

2.3 Sensor Communication

A sensor node is composed of four parts: a microcontroller, sensors, a battery and a communication module. In simulation, the developers should take account of this composition because it is essential for representing the realistic sensor behaviors. The energy consumption of sensor communication presents the biggest part of resource consumption. Therefore, we aim to explain the energy depletion for sensor communication in this section. Figure 3 illustrates the simulation of sending a message from sensor S1 to sensor S2.

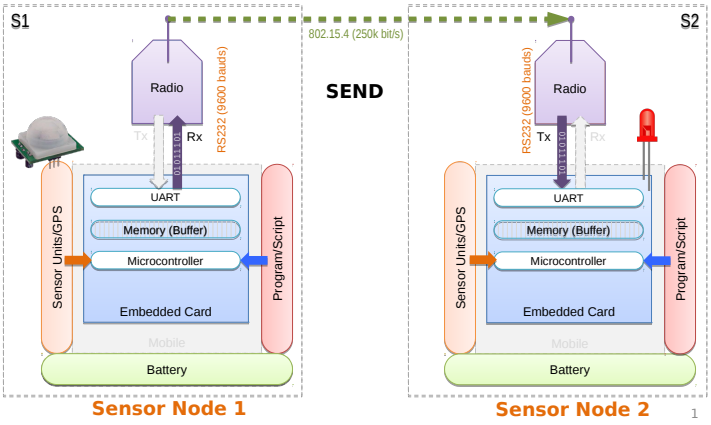

Fig. 4: Communication Process

The send command is executed in two steps. In the first step, it writes the data in the buffer of the sensor node. In this case, the duration is equal to the time required to write data in the buffer.

The send command sends the data using the radio module in the second step, where the duration is related to the communication speed, which is generally 250 kbit/s in wireless sensor networks using ZigBee standards. The same holds for a reception command: in a first step the message is received with the radio module and thereafter it will be sent to the microcontroller.

In some cases, the communication takes more time, due to the channel constraint. All sensors use the same channel and only one can send at a given time. The other nodes must wait for a channel to be free to send. Figure 5 shows an example of such a communication.

As shown in Figure 5, at time *t1* sensor S1 tries to send the message *A*. The channel being busy, it must wait until time *t2* to send. In other words, instead of a communication cost of *d* time units, these will be *d + (t2 - t1)* time units.

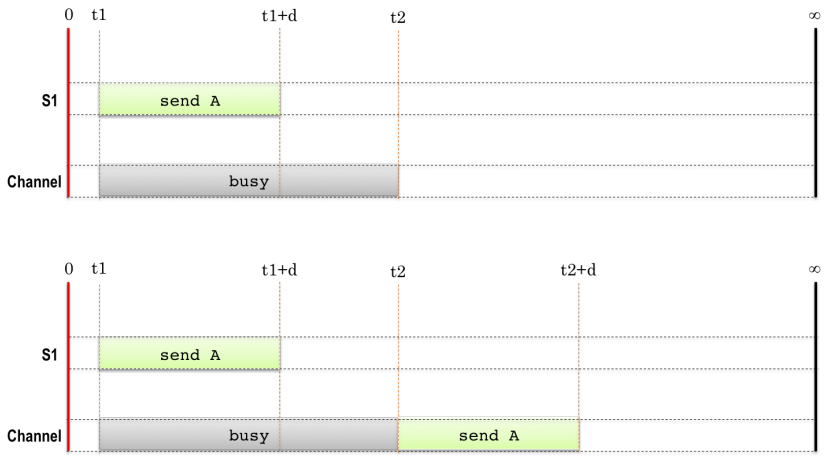

Fig. 5: Example of Communication (send message)

3 A Sequential Model for Computing Energy Consumption

This section presents a sequential simulation model to compute the energy consumption of sensor nodes in static or mobile networks. The current version of this model does not take into account the sending of packets nor their routing. It only accounts for size of packets and waiting times at sensor nodes. We will proceed in two main steps. The first step shows how to apply our model to a static network, the second step how mobility can be integrated into this model.

3.1 The Case of a Static Network

To illustrate the discrete event simulation algorithm of *SimBox*, we will make use of the flow chart given in Figure 6 and introduce some terminology.

The first step is to describe for a given network the communication links between the different sensor nodes.

These links can be represented by a matrix $A \in \{0,1\}^{n^2}$ as follows:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

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

Note, that two communicating sensors have to be located within each other's radio range.

The second step in the chart of Figure 6 is to create communication scripts and assign them to each sensor node. In a script we have two types of instructions: *SEND* x which specifies that the sensor node must send a packet of x bytes, and *DELAY* y which specifies that the sensor node should not do anything for

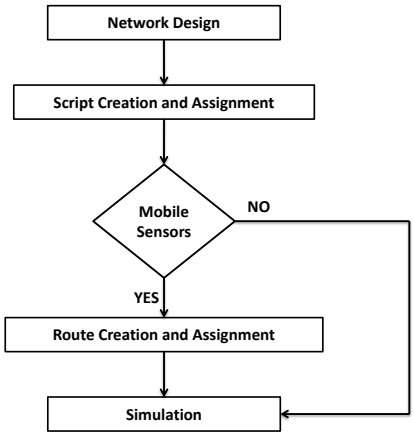

Fig. 6: *CupCarbon* Simulation Flow.

y milliseconds. Figure 7 shows an example of a communication script where the arguments of the *SEND* instruction are specified in bytes (or characters) and those of the *DELAY* instruction in milliseconds.

```
SEND hello
DELAY 1000
SEND bye
DELAY 1000
```

Fig. 7: Example of a Sensor Communication Script.

The events of the two instructions $SEND$ and $DELAY$ have different units. The first is the number of bytes, which corresponds to the number of the characters of the message to send, and the second is the time to wait in seconds. In discrete event simulation, the most important parameter is the time that each instruction takes. Therefore, for each instruction, we need to use its required time to be accomplished. For the $DELAY$ instruction, the problem does not arise because its parameter is given in time. Hence, for the instruction $SEND$, its parameter must be transformed from bytes to seconds. This time represents the duration required to send all the bytes, which depends on the data rate of the used radio standard. For instance, in the ZigBee standard, the data rate is equal to $250,000$ bits per second (or 31250 bytes/second). As an example, if we want to send a message "hello", which has 5 characters (or 5 bytes), then we need $5/31250 = 0,16$ *milliseconds*.

To present the sequential version of the algorithm calculating the energy consumption, which is based mainly on these instructions, let us define the following variables:

– Variable q_i represents the index of the instruction to be executed by sensor node i.

- 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.
- 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 ($dead_i = 0$). A dead sensor means that its battery is empty.

$$\begin{cases} dead_i = 1 \text{ if } \ energy_i = 0 \\ dead_i = 0 \text{ if } \ energy_i > 0 \end{cases}$$

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 also define a few constants. Constant m represents the number of instructions 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 is the same for each sensor. It is given by the minimum of each current sensor node duration. The different states of the simulation algorithm are summarized in Figure 8.

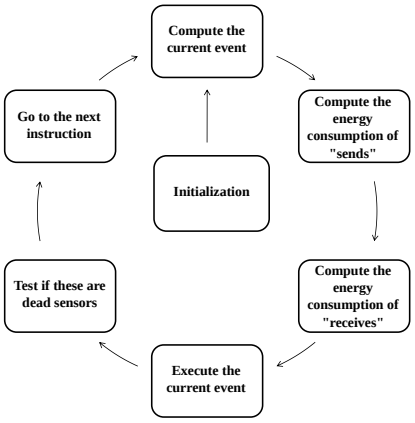

Fig. 8: States of the Discrete Event Simulation Algorithm

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

- Step 1: Initialization (for each sensor node 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 node 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 node i)

$$energy_i = \max\{energy_i - (currEvent \times E_{R_x} \times c), 0\},$$

  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 node 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 operation, which allows to run the script in a loop.

The algorithmic form of these states is given by Algorithm 1.

3.2 The Case of a Mobile Network

We now suppose that the nodes can be mobile. Thus, there will be two types of possible events. Either the node communicates (executes send or delay instructions) or the node moves. This will change the previous algorithm in case the current event is a move. The energy consumption must be computed after changing the position of the sensor node. This generates an updating of the different communication links between the sensor nodes in the matrix A. By adding a moving state of the sensor nodes to states of the algorithm presented in Figure 10, we get a

```
Data: n, m, e0[n], iterMax, a[n][n], opType[n][m], opArg[n][m]
for i=1 to n do
    energy[i] = e0[i];
    q[i] = 0;
    dead[i] = 0;
    event[i] = opArg[i][0];
end
for iter=1 to iterMax do
    currEvent = min{event[i], i=1,...,n};
    for i=1 to n do
        if f(opType[q[i]]) == SEND then
            energy[i] = energy[i] - currEvent;
        end
        c = 0;
        for (j=1, j ≠ i) to n do
            c = c + a[i][j] * f(opType[j][q[j]]) * (1 - dead[j]);
        end
        energy[i] = energy[i] - (currEvent * c);
        event[i] = event[i] - currEvent;
    end
    stop = false;
    for i=1 to n do
        if energy[i] == 0 then
            dead[i] = 1;
            event[i] = e0[i];
        else
            stop = true;
        end
    end
    if stop == true then
        STOP Simulation;
    end
    for i=1 to n do
        if event[i] == 0 then
            q[i] = (q[i]+1) mod m;
            event[i] = opArg[i][q[i]];
        end
    end
end
```

Algorithm 1: Algorithm for Discrete Event Simulation of a Static Wireless Sensor Network.

new simulation algorithm taking into account the mobility. This new situation is presented by the state graph of Figure 9.

We now describe 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 sensor node i, the variable $event_{2_i}$ will represent the movements of a sensor i. The variable $currEvent$ 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 sensor node i to go from point p_{i-1} to point p_i. By taking account of the mobility, Algorithm 1 becomes Algorithm 2.

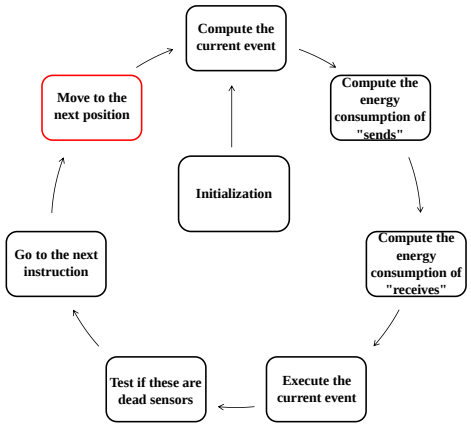

Fig. 9: States of the Simulation Algorithm for a Mobile Network .

4 A Parallel Model for the computation of Energy Consumption

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

As we will see below, the proposed model corresponds to the discrete event simulation algorithm transformed from an algorithmic model into a model based on matrix multiplication. 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 is sending a packet ($v_{i,q_i} = 1$) or not ($v_{i,q_i} = 0$). We also set $a_{ii} = 1$ for $i = 1, .., n$, wich indicates that a sensor node is linked to itself. This allows to assume that there will be only receiving actions and no sending actions, since we present a sensor node that sends a packet as a node that receives a packet from itself. This also allows to gain one step in the algorithm and to make only one access instead of two in the GPU. We may illustrate this procedure as follows. Consider a network of 4 sensor nodes. Sensor node 1 sends a packet ($v_{1,q_1} = 1$) and sensor node 3, which is its neighbor ($a_{13} = 1$) also sends a packet ($v_{3,q_3} = 1$). Sensor node 1 will consume the energy associated with the sending of a packet, and it will also consume the energy associated with the reception of a packet sent by sensor node 3. If we assume that the reception energy is equal to the sending energy, sensor node 1 will therefore consume $c_1 = 2 \times e$, where e is the energy associated with the reception or transmission of a packet. Assume now that sensor node 2 is not a neighbor of sensor node 1 ($a_{12} = 0$) and that it sends a packet ($v_{2,q_2} = 1$). Then sensor node 1's consumption will not be affected by this send action because sensor node 2 does not communicate with sensor node 1. Assume also, that sensor node 4 is connected to sensor node 1 ($a_{14} = 1$) but it does not send anything ($v_{4,q_4} = 0$). The computation of sensor node 1's energy consumption can be obtained by the following formula.

```
Data: n, m1, m2, iterMax, e0[n], a[n][n], opType[n][m1], opArg[n][m1], gps[n][m2]
for i=1 to n do
    energy[i] = e0[i];
    q[i] = 0;
    p[i] = 0;
    dead[i] = 0;
    event1[i] = opArg[i][0];
    event2[i] = gps[i][0];
end
for iter=1 to iterMax do
    Move each mobile sensor to his next point;
    Update the matrix a[n][n];
    currEvent = min{event1[i],event2[i], i=1,...,n};
    for i=1 to n do
        if f(opType[q[i]]) == SEND then
            energy[i] = energy[i] - currEvent;
        end
        c = 0;
        for (j=1, j ≠ i) to n do
            c = c + a[i][j] * f(opType[j][q[j]]) * (1 - dead[j]);
        end
        energy[i] = energy[i] - (currEvent*c);
        event1[i] = event1[i] - currEvent;
        event2[i] = event2[i] - currEvent;
    end
    stop = false;
    for i=1 to n do
        if energy[i] == 0 then
            dead[i] = 1;
            event1[i] = e0[i];
        else
            stop = true;
        end
    end
    if stop == true then
        STOP Simulation;
    end
    for i=1 to n do
        if event1[i] == 0 then
            q[i] = (q[i]+1) mod m1;
            event1[i] = opArg[i][q[i]];
        end
        if event2[i] == 0 then
            p[i] = (p[i]+1) mod m2;
            event2[i] = gps[i][p[i]];
        end
    end
end
```

Algorithm 2: Algorithm for Discrete Event Simulation of a Mobile Wireless Sensor Network.

$$c_1 = (\sum_{j=1}^{4} a_{1j} \cdot v_j) \times e.$$

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

$$c_1 = (1 \times 1 + 0 \times 1 + 1 \times 1 + 1 \times 0) \times e = 2 \times e.$$

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

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

The advantage of this method is that it allows to parallelize the computation of energy consumption of each sensor i by launching it on threads (each computation c_i is launched on a separate thread).

Figure 10 shows the modified states of the algorithm described in Figure 8 by differentiating parts that run on the CPU from those running on GPU. The corresponding procedure is given by Algorithm 3.

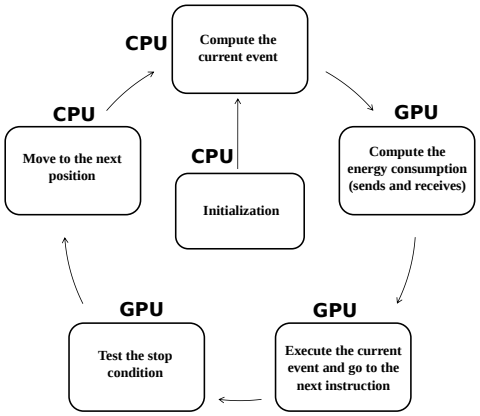

Fig. 10: The states of Discrete Event Simulation Algorithm

5 A Case Study and Discussion

We have implemented the parallel part of the proposed model using OpenCL (Angulo et al, 2015; Khronos Group, 2015; Pizzolante et al, 2014) and CUDA (Compute Unified Device Architecture) (Sanders and Kandrot, 2010) on a NVIDIA GeForce graphic card. The best results are obtained with CUDA, and we will therefore limit our presentation to these. The developer can use the computing power of a graphics card with some operations intended to be processed by the GPU instead of the CPU. However, this last one is always necessary to coordinate the CPU and the GPU work. The GPU is thus considered as a massively parallel processor well adapted to the processing of parallel algorithms. An operation executed on the GPU is also called a *kernel*.

The execution of a CUDA program can be described as follows:

- Initially, the program is run by the CPU.
- A kernel is invoked, it will be executed on the GPU.

```
Data: n, m1, m2, iterMax, e0[n], a[n][n], v[n][m1], opArg[n][m1], gps[n][m2]
for i=1 to n do
    energy[i] = e0[i];
    q[i] = 0;
    p[i] = 0;
    dead[i] = 0;
    event1[i] = opArg[i][0];
    event2[i] = gps[i][0];
end
for iter=1 to iterMax do
    Move each mobile sensor node to its next position;
    Update the matrix a[n][n];
    currEvent = min{event1[i],event2[i], i=1,...,n};
    for idx=1 to n (IN PARALLEL) do
        c = 0;
        for j=1 to n do
            c = c + a[idx][j] * v[j][q[j]] * (1 - dead[j]);
        end
        energy[idx] = energy[idx] - (currEvent*c);
        event1[idx] = event1[idx] - currEvent;
        event2[idx] = event2[idx] - currEvent;
    end
    stop = false;
    for idx=1 to n (IN PARALLEL) do
        if event1[idx] == 0 then
            q[idx] = (q[idx]+1) mod m1;
            event1[idx] = opArg[idx][q[i]];
        end
        if energy[idx] <= 0 then
            dead[idx] = 1;
            event1[idx] = e0[i];
        end
    end
    for idx=1 to n (IN PARALLEL) do
        if dead[idx] == 1 then
            stop = true;
        end
    end
    if stop == true then
        STOP Simulation;
    end
    for i=1 to n do
        if event2[i] == 0 then
            p[i] = (p[i]+1) mod m2;
            event2[i] = gps[i][p[i]];
        end
    end
end
```

Algorithm 3: GPU-based Algorithm for Discrete Event Simulation of a Mobile Wireless Sensor Network.

– A large number of threads are generated and executed in parallel on the GPU.

CUDA allows replication of the hardware architecture specifications of a GPU (NVIDIA GeForce GTX 480 in our case) at a software level and manages the communication between CPU and GPU. This allows 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 their 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 in three dimensions if he makes an operation on volumes.

5.1 A Case Study

To illustrate the algorithm, we follow the steps illustrated in Figure 6 by considering a static network. We start with the network design. As an example, we take a network of three sensor nodes (S1, S2 and S3) which are linearly positioned as shown in Figure 11. The corresponding matrix A is given as follows:

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

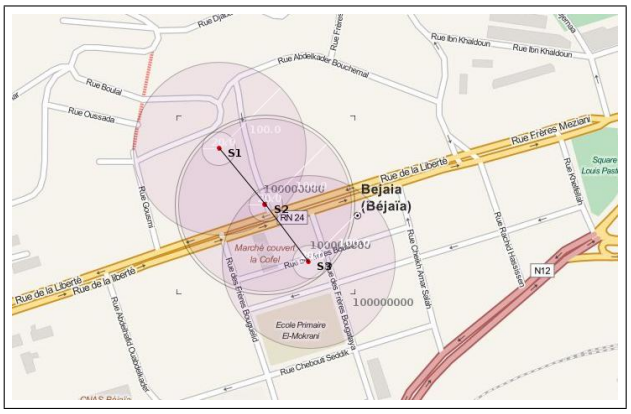

Fig. 11: Example of a Wireless Sensor Network in *CupCarbon*.

We then describe the various communication scripts and we assign them to each sensor node. These scripts are given in Table 1 with units standardized in bits. Note that for simplicity, we have chosen as communication speed 9600 bits/second instead of $250k$ bits/second.

Table 1: Communication Scripts

Script (S1)	Script (S2)	Script (S3)
`SEND 1000` `DELAY 960`	`SEND 2000` `DELAY 960`	`SEND 800` `DELAY 960`

Then we launch the simulation, whose results are given in Table 2.

Table 2: 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

We will use these results to illustrate the progress of the proposed algorithm. 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 3). The status of each sensor node is initialized to alive ($dead_i = 0,\ i = 1, 2, 3$). For simplicity, we set $E_{T_x} = E_{R_x} = 1$.

Table 3: Communication Scripts (Iter=1)

q_1	Script (S1)	q_2	Script (S2)	q_3	Script (S3)
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 arguments of the instruction executed by each sensor node. In our case, these arguments are: 1000, 2000 and 800. The minimum is 800 as shown in bold in the first row of Table 2. This value must be subtracted from each of the sensor events which will give us new event values of, respectively, 200, 1200 and 0. Then, we compute for each sensor node the energy consumption generated by sending 800 bits to other sensor nodes. The sensor node $S1$ for example, sends a packet ($f(opType_{1,q_1}) = f(SEND) = 1$) and it will consume the equivalent of 800 bits. Furthermore, as this sensor node is only linked to sensor $S2$, which also sends a packet ($f(opType_{2,q_2}) = f(SEND) = 1$). Altogether, $S1$ will consume in this step the energy associated with the sending and receiving of 1600 bits. Thus, $(3000 - 1600)$ $S1$ will have only 1400 units of energy. The results obtained for the other sensor nodes are given by row 2 (Iter=2), of the Energy part of Table 2.

Now, we move to the next instructions. Sensor nodes which have event equal to zero will execute their next instructions. In our case, sensor node $S3$ has an event equal to zero and it will thus execute the next instruction (i.e., $q_3 = q_3 + 1 = 0 + 1 = 1$). The result is given in Table 4.

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

Table 4: Communication Scripts (Iter=2)

q_1	Script (S1)	q_2	Script (S2)	q_3	Script (S3)
0→	`SEND 1000` `DELAY 960`	**0**→	`SEND 2000` `DELAY 960`	**1**→	`SEND 800` `DELAY 960`

5.2 Results and Discussions

Sensor nodes are autonomous systems, deployed in the nature and generally in inaccessible areas. They are powered by a battery, which is difficult to replace or recharge. Therefore, increasing the sensor nodes' lifetime is a critical issue (Lalwani et al, 2016; Srivastava and Sudarshan, 2015). Hence, reducing the power consumption is an important goal and the data exchange between nodes (i.e., sending and receiving operations) strongly dominates other consumptions related, for instance, to sensing and processing. As it is specified by (Egea López et al, 2006) and (Riley and Ammar, 2002), conventional simulators do not deal with the scalability. In this section, we analyze the impact of the proposed parallel algorithm on the energy consumption for large networks. For this purpose, we will use the two following simulation metrics:

- **Execution time:** also called wall-clock time, response time, or elapsed time, is the latency to complete a simulation.
- **Acceleration:** is used to compare algorithms in terms of execution time. It is given by the following equation:

$$Acceleration = \frac{\text{Execution time of Algorithm 1}}{\text{Execution time of Algorithm 2}}$$

An acceleration equal to n means that the Algorithm 2 is n times faster than Algorithm 1.

To run the simulations, we have generated a set of wireless sensor networks deployed randomly, using a simple uniform distribution, on an area of $(1000m \times 1000m)$. All the sensor nodes are homogeneous, with a communication range of $100m$. The first network has 1000 sensor nodes, and we repeatedly increase the number of sensor nodes by 1000. The last one has 10000 sensor nodes.

For a comparison, we decided to proceed in two ways. The first consists in fixing the number of simulation iterations and in changing the number of sensor nodes in the network. The second consists in fixing the number of sensor nodes and in changing the number of iterations. Networks are randomly generated. In the first case, we set the number of iterations to 1000 and vary the number of sensor nodes from 1000 to 10000. The resulting graph is shown in Figure 12. We can take as an example the case of a network of 8000 sensor nodes. It is simulated in 5 minutes and 45 seconds on the CPU, but it can be simulated in only 30 seconds on the GPU. The corresponding acceleration graph is given by Figure 13. 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 vary the number of iterations from 1000 to 15000. The resulting graph is shown in Figure 14. As in the first case, the graph shows the simulations performed on the GPU 8

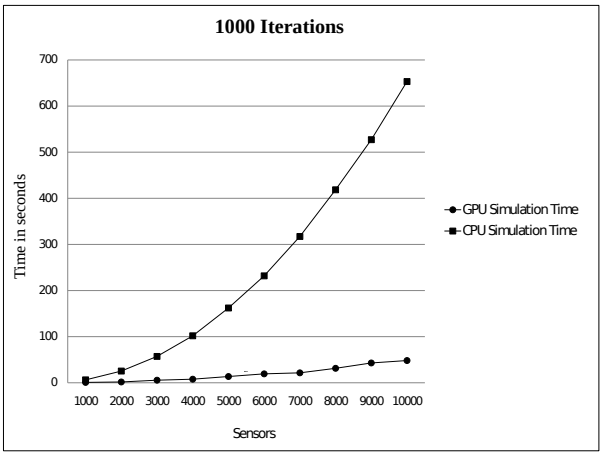

Fig. 12: Simulation Time according to the Number of Sensor Nodes.

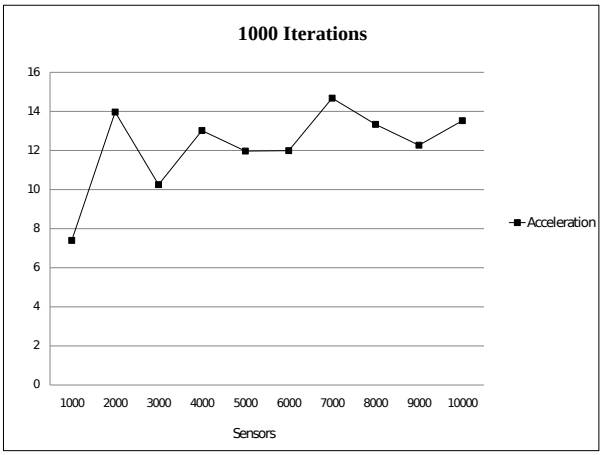

Fig. 13: Acceleration according to the Number of Sensor Nodes.

times faster than those running on the CPU. This is confirmed by the acceleration graph given by Figure 15.

It should be noted that these algorithms are compared based on the computation of energy consumption, which can be done by a simple addition operation. On the other hand and in future versions of the *CupCarbon* simulator, there will be inevitably other parameters to be taken into account for computing energy consumption using realistic models. Also, it is possible to include treatments related to routing (Azharuddin and Jana, 2016), security (Dou et al, 2016), packet management, etc. which will generate an additional computing time. To this end, we may add an instruction to increase the computing time to study the sensitivity of the proposed algorithms with respect to the simulation time. We have done so with an empty loop *for* running 10^5 times. Then we have computed the CPU

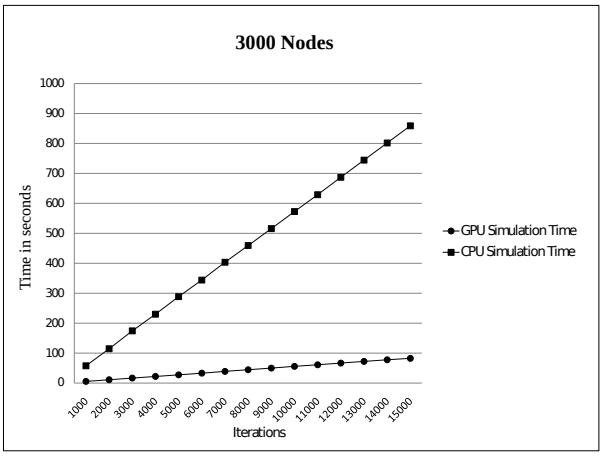

Fig. 14: Simulation Time according to the Number of Iterations.

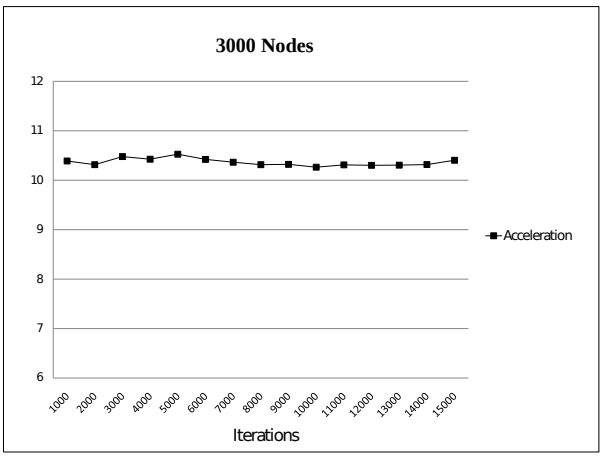

Fig. 15: Acceleration according to the Number of Iterations.

and the GPU simulation time for different network sizes. The results obtained are given in Figure 16.

Finally, we have computed the acceleration (cf. Figure 17). From these graphs we can conclude that the simulation times on the GPU were not affected by the newly added treatment (loop). The simulation time on CPU, however, has increased: we obtained accelerations reaching a factor of 25. This is the case for example, for a network with 3000 sensor nodes. The simulation times on the GPU are 30 times faster than those obtained on the CPU.

Thus, we conclude that for future versions of the simulator *CupCarbon*, simulations on GPUs should be favored, but there will be necessarily other parameters and treatments to be included in the simulations.

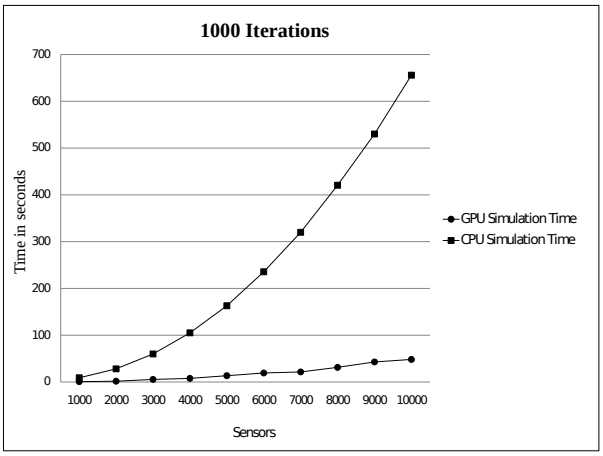

Fig. 16: Simulation Time according to the Number of Iterations.

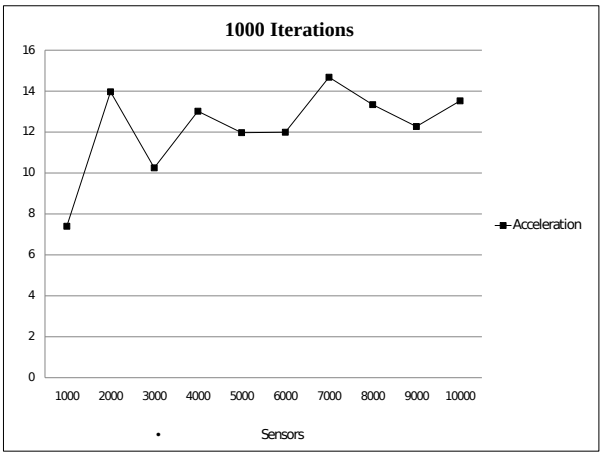

Fig. 17: Acceleration according to the Number of Iterations.

6 Conclusion

Most existing wireless sensor network simulators have the disadvantage of being very slow when the number of sensor nodes exceeds a few hundred. One of the possible directions to explore these simulators for acceleration could be 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. These networks can be static or mobile. The simulation results show the need to use simulation on GPU for networks exceeding 1000 sensor nodes for which simulations have been accelerated from 6 to 25 times. These accelerations can even be greater if the processing on each sensor node is very time consuming. The model proposed

in this paper has been implemented and validated on the simulator *CupCarbon*, a tool to simulate wireless sensor networks.

Acknowledgment

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

References

Angulo AC, Alvarez RC, Aguilar JO, Castillo JV, Marrufo OP, Atoche AC (2015) A case study of opencl-based parallel programming for low-power remote sensing applications. In: Electrical Engineering, Computing Science and Automatic Control (CCE), 2015 12th International Conference on, IEEE, pp 1–6

Azharuddin M, Jana PK (2016) Pso-based approach for energy-efficient and energy-balanced routing and clustering in wireless sensor networks. Soft Computing pp 1–15

Bounceur A, Lounis M (2016) Cupcarbon: A smart city & iot wsn simulator [online]. http://www.cupcarbon.com [Accessed 01 May 2017]

Chen G, Branch J, Pflug M, Zhu L, Szymanski B (2005) Sense: a wireless sensor network simulator. Advances in pervasive computing and networking pp 249–267

Chen RC, Hsieh CF, Chang WL (2016) Using ambient intelligence to extend network lifetime in wireless sensor networks. Journal of Ambient Intelligence and Humanized Computing 7(6):777–788

Dou Y, Weng J, Ma C, Wei F (2016) Secure and efficient ecc speeding up algorithms for wireless sensor networks. Soft Computing pp 1–9

Egea López E, et al (2006) Simulation scalability issues in wireless sensor networks

Feeney LM, Willkomm D (2010) Energy Framework: An Extensible Framework for Simulating Battery Consumption in Wireless Networks, 10, vol 4, 3rd edn. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium

Issariyakul T, Hossain E (2008) Introduction to Network Simulator NS2, 1st edn. Springer Publishing Company, Incorporated

Kiani SL, Anjum A, Antonopoulos N, Knappmeyer M (2014) Context-aware service utilisation in the clouds and energy conservation. Journal of Ambient Intelligence and Humanized Computing 5(1):111–131

Korkalainen M, Sallinen M, Karkkainen N, Tukeva P (2009) Survey of wireless sensor networks simulation tools for demanding applications. Networking and Services, 2009 ICNS '09 Fifth International Conference on pp 102–106

Lalwani P, Banka H, Kumar C (2016) Bera: a biogeography-based energy saving routing architecture for wireless sensor networks. Soft Computing pp 1–17

Landsiedel O, Wehrle K, Gotz S (2005) Accurate prediction of power consumption in sensor networks. In: Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on, IEEE, pp 37–44

Levis P, Lee N, Welsh M, Culler D (2003) Tossim: Accurate and scalable simulation of entire tinyos applications. In: Proceedings of the 1st international conference on Embedded networked sensor systems, ACM, pp 126–137

Lounis M, Mehdi K, Bounceur A (2014) 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

Lounis M, Bounceur A, Laga A, Pottier B (2015) Gpu-based parallel computing of energy consumption in wireless sensor networks. In: Networks and Communications (EuCNC), 2015 European Conference on, pp 290–295

Malik H, Malik AS, Roy CK (2011) A methodology to optimize query in wireless sensor networks using historical data. Journal of Ambient Intelligence and Humanized Computing 2(3):227

Mehdi K, Lounis M, Bounceur A, Kechadi T (2014) Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. In: Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp 126–131

Musznicki B, Zwierzykowski P (2012) Survey of simulators for wireless sensor networks. International Journal of Grid and Distributed Computing 5(3):23–50

Nayyar A, Singh R (2015) A comprehensive review of simulation tools for wireless sensor networks (wsns). Journal of Wireless Networking and Communications pp 19–47

Park S, Savvides A, Srivastava MB (2000) Sensorsim: A simulation framework for sensor networks. ACM, pp 104–111

Pizzolante R, Castiglione A, Carpentieri B, De Santis A (2014) Parallel low-complexity lossless coding of three-dimensional medical images. In: Network-Based Information Systems (NBiS), 2014 17th International Conference on, IEEE, pp 91–98

Polastre J (2003) Sensor network media access design. EECS Department, University of California Berkeley, USA

Riley G, Ammar M (2002) Simulating large networks: How big is big enough. In: Proceedings of First International Conference on Grand Challenges for Modeling and Simulation, vol 2

Sanders J, Kandrot E (2010) CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley, Pearson Education, Inc. Rights and Contracts Department 501 Boylston Street, Suite 900 Boston, MA 02116 Fax: (617) 671-3447

Singh CP, Vyas OP, Tiwari MK (2008) A survey of simulation in sensor networks. Computational Intelligence for Modelling Control Automation, 2008 International Conference on pp 867–872

Sobeih A, Chen WP, Hou JC, Kung LC, Li N, Lim H, Tyan HY, Zhang H (2005) J-sim: a simulation environment for wireless sensor networks. pp 175–187

Srivastava JR, Sudarshan T (2015) Energy-efficient cache node placement using genetic algorithm in wireless sensor networks. Soft Computing 19(11):3145–3158

Khronos Group (2015) Opencl, parallel computing for heterogeneous devices. https://www.khronos.org/assets/uploads/developers/library/overview/opencl_overview.pdf

The University of Southern California (2016) The network simulator ns-2 [online]. http://www.isi.edu/nsnam/ns/ [Accessed 01 May 2017]

Titzer BL, Lee DK, Palsberg J (2005) Avrora: Scalable sensor network simulation with precise timing. In: Information Processing in Sensor Networks, 2005. IPSN

2005. Fourth International Symposium on, IEEE, pp 477–482
Varga A (2001) The omnet++ discrete event simulation system. In: European Simulation Multiconference, vol 9, pp 1–7
Vir D, Agarwal S, Imam S (2013) Wsn performance evaluation of power consumption analysis of dsr, olsr, lar and fisheye in energy model through qualnet. International Journal of Scientific and Research Publications pp 19–47