ENERGY AND POWER EVALUATION OF PARALLEL DEVS SIMULATIONS ON MULTICORE ARCHITECTURES

Guillermo G. Trabes

Department of Systems and Computer Engineering
Carleton University
and Universidad Nacional de San Luis
Ottawa, ON K1S 5B6, CANADA

Veronica Gil-Costa

Universidad Nacional de San Luis
and CCT CONICET San Luis
Ejército de Los Andes 950
San Luis, D5700, ARGENTINA

Gabriel A. Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By
Ottawa, ON K1S 5B6, CANADA

ABSTRACT

We propose an energy and power evaluation for the execution of Discrete Event System Specification (DEVS) simulations in parallel shared memory architectures. Our approach evaluates how time, energy, and power scales when we increase the number of CPU-cores on multicore architectures. To perform our evaluation, we present the results of a set of experiments using a synthetic benchmark and a real-world scenario. The results obtained show how executing in parallel accelerates the execution up to 3.03 times, while reducing the energy used by up to 44.7%. However, executing in parallel increases the power consumption by up to 67.5%.

Keywords: DEVS, parallel simulation, energy efficiency, power efficiency.

1 INTRODUCTION

Energy consumption is an important problem on many parallel computing applications. In high-performance computing, energy consumption is a major cost associated with operating large data centers and supercomputers. Furthermore, energy consumption is a limiting factor to scale computing platforms further. In addition, there are environmental reasons. In the past decade, the field of *Green Computing* was created. The objective is to make high-performance computing more efficient and effective with minimum or no impact on the environment (Li and Zhou 2011).

Although this is an important topic, and it is considered to be one of the main challenges in computational simulation (Fujimoto et al. 2017), very few research has been conducted in the area. The ideas proposed in the literature mostly focus on evaluating key characteristics and activities of the simulation execution (Biswas and Fujimoto 2016), such as synchronization (Biswas and Fujimoto 2017; Biswas and Fujimoto 2018), data use (Neal, Fujimoto, and Hunter 2016; Neal and Fujimoto 2017), and the management of the future event list (Neal and Fujimoto 2018). However, as mentioned in (Fujimoto 2020), research on this subject is still in its infancy and we need further efforts for its development.

Discrete Event System Specification (DEVS) (Zeigler, Praehofer, and Kim 2000) is a mathematical formalism that allows to reason about modeling and simulation in a hierarchical and composable way. DEVS modular structure allows to execute simultaneous events in parallel to accelerate simulations (Zeigler 2017). However, when we need to execute DEVS simulations on high-performance architectures, a new concern arises: how much energy do computer architectures use to perform them? With these issues in mind, this research presents an empirical evaluation for energy use and power consumption of DEVS simulations executed in parallel on multicore architectures. We evaluate how much time, energy, and power the DEVS simulations spend when executing in parallel using different number of CPU-cores. We performed our analysis on a recently proposed parallel algorithm for DEVS simulations, which has been implemented as an extension of Cadmium (Belloli et al. 2019), a DEVS simulator. Our results show how executing in parallel accelerates the execution up to 3.03 times, while reducing the energy used by the processors and the main memory by up to 44.7%. However, executing in parallel increases the power consumption by up to 67.5%.

The rest of the paper is organized as follows. Section 2 provides a literature review on DEVS simulations and energy measurements. On Section 3 we present the parallel algorithm we used in our experiments. Following, on Section 4 we show the energy experimental evaluation. Finally, on Section 5 we present the conclusions and future work.

2 BACKGROUND AND RELATED WORK

Our research focuses on evaluating time, energy, and power consumption for parallel execution of DEVS simulations. DEVS is a mathematical formalism that provides a theoretical framework to think about modeling using a hierarchical, modular approach (Zeigler, Praehofer, and Kim 2000). In DEVS, atomic models are used to define behavior and coupled models are used to specify the structure of the system under study. In DEVS, there is a clear separation between model and simulation: models are described using a formal notation, and simulation algorithms are defined for executing any legitimate DEVS model. DEVS models are legitimate if they undergo a finite number of events in a finite interval of time. DEVS was extended by the Parallel DEVS (PDEVS) formalism (Chow and Zeigler 1994) which allows to handle simultaneous events. In PDEVS the main simulation algorithm is known as the PDEVS abstract simulator (Chow, Zeigler, and Kim 1994).

Given a PDEVS model, the abstract simulator creates a data structure to mimic its behavior and execute simulations. The structure consists of three types of components: simulators, coordinators, and root-coordinator. Each atomic model is associated with a simulator and each coupled model is associated with a coordinator. One root-coordinator is placed at the root of the structure hierarchy. Once the abstract simulation structure is created, there is a mechanism to execute simulations, known as the PDEVS simulation protocol. This algorithm works by exchanging messages between the components. At the top of this hierarchy, the root-coordinator starts a sequence of simulation steps by communicating to the top-most coordinator. Coordinators at the different levels exchange messages with their parent coordinator and with the components one level below. The messages disseminate through this tree structure until reaching simulators at the leaves of the tree. The details for this algorithm can be found in (Chow, Zeigler, and Kim 1994; Nutaro 2019). This protocol was proven to be correct, and it is implemented in many simulators. However, one problem that the hierarchical message passing across several levels on the tree might involve a large overhead, increasing execution time in the coordination of components at the different levels. To improve this hierarchical communication overhead, a flattening algorithm was proposed in the DEVS literature (Kim et al. 2000; Glinsky and Wainer 2002). The idea of this algorithm is to transform the hierarchical PDEVS structures into flat data structures where all intermediate coordinators are eliminated. The resulting structure has only one root-coordinator; one flat coordinator (which holds all the information about couplings among simulators) and a set of simulators.

Despite the improvements on the PDEVS simulation protocol, as simulations became more complex and time consuming, techniques were proposed to accelerate them in parallel computers. The main methods

proposed to execute DEVS simulations in parallel used the concept of Logical Processes (LPs). LPs work as simulation entities, which do not share any state variables, and interact with each other through timestamped event messages (Fujimoto 1990). The leading contributions in this topic were made in the Proceedings of the Parallel and Distributed Simulation (PADS) conferences between 1990 and 2019 (Fujimoto et al. 2017). The reader can find an introduction to this field in (Fujimoto 1999). The major challenge in this technique is being able to produce the same results as in a sequential execution. Synchronization among these LPs is violated when one of the LPs receives an event that is older than the current clock time of the recipient LP. Such violation is referred to as causality error. To deal with causality errors, different synchronization techniques have been proposed. There have been efforts on conservative approaches (Jafer and Wainer 2011), which strictly avoid causality errors; and on optimistic approaches (Nutaro 2009; Liu and Wainer 2012), which allow causality errors but provide mechanisms to correct the state of the simulations. The problem with LP approaches in DEVS simulations is that only some conservative techniques have been proven to exactly represent the behavior of the DEVS sequential simulator (Zeigler 2017). This feature distinguishes DEVS from the many other simulation methodologies that use the LP approaches, the behavior of the sequential DEVS simulator cannot be reproduced in all scenarios.

A different approach proposed in (Zeigler 2017) enables the execution of simultaneous events in parallel. The main idea behind this approach is to execute in parallel the independent simultaneous events occurring in the PDEVS simulation protocol. The idea is to apply parallel execution in two situations: when multiple output functions must be computed at the same time; and when multiple transition functions must be computed at the same time. Furthermore, in (Zeigler 2017) a theoretical analysis was made with this idea, concluding that speedups are related to specific model characteristics, such as the level of activity, defined as the probability of execute a transition function on each component. This theoretical analysis concludes that this approach can achieve 60% of the best possible parallel execution, using this simple parallel implementation. This idea was evaluated in (Lanuza, Trabes, and Wainer 2020), where the authors show a practical implementation in a simulator. However, the speedup obtained was limited and did not scale when increasing the number of threads. In this work we use an algorithm that extends this idea, providing additional parallelism.

As we mentioned, in this work we propose an energy evaluation on the execution of DEVS simulations. There are two important metrics related to the energy consumption on computer architectures. The first one is the energy used by the hardware to execute a computer program. The general definition for energy is “the capacity for doing work”, and it is measured in Joule units (Fujimoto 2020). The second metric is called “Power”, which is the amount of energy consumed per unit of time, with one watt referring to the expenditure of one joule of energy per second. The energy consumed by a computer program depends on the hardware power use and the execution time (Fujimoto 2020). Minimizing execution time does not guarantee minimizing energy consumption, even though these metrics are related. Executing with more computing resources may reduce time but it may increase both energy and power consumption. Similarly, minimizing power is not equal to minimizing energy consumption. Executing on less computing resources may reduce power, but it will lead to longer execution times that may increase the total amount of energy required to complete the computations. Therefore, to analyze energy efficiency on computer programs we must consider all these related variables: execution time, energy use and power consumption.

To measure execution time there are several libraries available, some of them as tools implemented in operating systems. However, to measure energy consumption in the execution of programs, only some exist and very few have been proven to perform accurate measures. The main tool to measure energy and power consumed on CPUs is Running Average Power Limit (RAPL) (David et al 2010.; Hähnel et al. 2012). RAPL is a tool designed by Intel that allows to monitor energy and power consumption across different domains of the CPU chip, attached DRAM, and on-chip GPU with promising accuracy (Khan et al. 2018). We can see an example of the execution of the RAPL in Figure 1. Here we can see the information we can obtain. For each multiprocessor the tool reports a PACKAGE value, in this example packages 0 and 1. For each package, the energy is measured for the processors (PP_ENERGY), main memory

(DRAM_ENERGY) and the multicore and main memory combined (PACKAGE_ENERGY). In this work we use as the measurement for our evaluation the PACKAGE_ENERGY, as it provides the most complete information about the energy used on an application by a processor chip.

```
Energy measurements:
PACKAGE_ENERGY:PACKAGE0              176.450363J (42.9W)
PACKAGE_ENERGY:PACKAGE1               75.812454J (18.4W)
DRAM_ENERGY:PACKAGE0                  11.450363J  (2.9W)
DRAM_ENERGY:PACKAGE1                   8.450363J  (2.0W)
PP0_ENERGY:PACKAGE0                  118.029236J (28.7W)
PP0_ENERGY:PACKAGE1                   16.759064J  (4.1W)
```

Figure 1: RAPL measurement example.

3 THE PARALLEL CADMIUM SIMULATOR

To perform our empirical evaluation, the first step was to define a parallel algorithm to execute DEVS simulations. To do so, we introduced a parallel algorithm that was implemented as an extension of the Cadmium simulator. Cadmium (Belloli et. al 2019) is an open source DEVS simulator written in C++ originally designed to execute a single-threaded sequential algorithm described in (Vicino et al. 2017). A first parallel version for the Cadmium simulator was presented in (Lanuza, Trabes, and Wainer 2019), where the algorithm used the ideas proposed in (Zeigler 2017). In this approach only the output and state transition functions executed in parallel on shared memory parallel architectures. The empirical results in (Lanuza, Trabes, and Wainer 2019) show only a 20% maximum improvement in performance using several cores. It is worth noting than an energy evaluation was presented for this algorithm in (Trabes, Gil-Costa, and Wainer 2020). However, these results are constrained by the limited speedup we can obtain with this approach.

In this work we use an algorithm that extends the idea presented in (Zeigler 2017) by providing additional parallelism. This algorithm flattens the DEVS simulation structures and then executes simulations in parallel with the OpenMP multithreading library. This version was also added as an extension of Cadmium. In this approach, all tasks in the PDEVS simulation protocol execute in parallel. All details for this algorithm were presented in (Trabes, Wainer, and Gil-Costa in press). In Figure 2 we can see the basic abstract structure used by this simulator. Following the DEVS flattened abstract simulator, every simulation using this algorithm has the same basic structure. A ROOT-COORDINATOR component is at the top of the tree. It also uses one FLAT-COORDINATOR and one or more SIMULATOR components at the leaves of the tree. Each SIMULATOR is associated with an ATOMIC_MODEL component, and the FLAT-COORDINATOR is associated with the unique COUPLE_MODEL of the flattened model. All details for the components in Cadmium can be found in (Vicino et. al 2017; Belloli et. al 2019).

Let us analyze how the parallel algorithm works. The first task is the initialization of the components. First, ROOT-COORDINATOR calls the *initialize* function on FLAT-COORDINATOR with the initial time as parameter. Then, FLAT-COORDINATOR calls the *initialize* function on each SIMULATOR. On this algorithm, these functions execute in parallel by assigning them to different threads. Each thread is responsible for executing the function on a subset of the SIMULATORs. Each SIMULATOR calculates the time advance function on its associated atomic model and returns the time for its next event to FLAT-COORDINATOR. Then FLAT-COORDINATOR calculates the minimum time for the next event from the next event's times received from all SIMULATORs. This is implemented in parallel through a parallel reduction. A parallel reduction is a type of operation used to reduce the elements of an array into a single result. In this function, each thread calculates a partial minimum result on a subset of times and saves it on a private copy of the variable. Once they finish, the partial results are compared to obtain global minimum. After this, FLAT-COORDINATOR returns the time for the next event to ROOT-COORDINATOR. Once this task completes, the threads perform a synchronization.

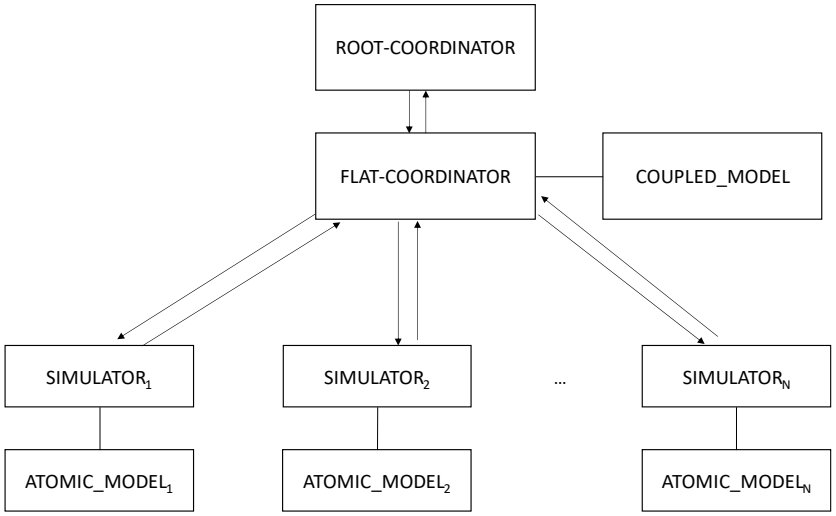


Figure 2: Flattened abstract simulator structure.

After the initializations step, a simulation loop starts. The next task repeats until the simulation finishes. First, there is the "execute outputs" task. To do this, the ROOT-COORDINATOR calls *execute_output_functions* on the FLAT-COORDINATOR. Then, the *execute_output_functions* routine calls the *execute_outputs_function* on every SIMULATOR subcomponent in parallel. In this case, each thread is responsible for executing on a subset of the SIMULATORs. Inside *execute_output_function*, each SIMULATOR checks if the time for its next event is equal to the next time in the simulation. If this is the case, then SIMULATOR executes its output function on its corresponding atomic model and stores it in its outbox. These functions are independent tasks and therefore can execute in parallel. Finally, the threads perform a synchronization step. The following task is to route the messages in the simulation structure. To do this, the ROOT-COORDINATOR calls the *route_messages* routine on the FLAT-COORDINATOR. This function routes all messages among atomic models in parallel. Here, each thread is responsible for routing messages to a subset of the atomic models. To route messages to an atomic model, its corresponding SIMULATOR checks on every component connected to the input of atomic model if the output bag is not empty. If this is not the case, then it inserts that output bag to the input bag of the component. These tasks are independent and therefore can be executed in parallel. Like in the other tasks in this algorithm, the threads perform a synchronization step when they finish this task. The algorithm continues running the "execute transitions" task. To do this, the ROOT-COORDINATOR calls the *execute_transition_functions* on the FLAT-COORDINATOR. Then, the *execute_transition_functions* routine calls the *execute_transition* function on every SIMULATOR subcomponent in parallel. To do this, each thread is responsible for executing on a subset of the SIMULATORs. The *execute_transition* function performs two checks. First, it checks if the component is imminent (i. e. if the time for the next event in the component is equal to the next simulation time). Then, it checks whether its *inbox* is empty or not. According to the results of these checks, we define which actions to execute on its corresponding atomic model:

- If the component is imminent and the *inbox* is empty, the component executes the internal transition function.
- If the component is imminent and the *inbox* is not empty, the component executes the confluent transition function (there are two simultaneous events colliding).
- If the component is not imminent and the *inbox* is not empty, the component executes the external transition function.
- In any other case, the component does not execute a state transition.

Finally, the last task in the simulation loop is to obtain the task for the next event. To do this, the ROOT-COORDINATOR calls the *next_time* function on the FLAT-COORDINATOR. This function obtains the *next_time* for each SIMULATOR subcomponent and compares the values obtained to determine the

minimum of them. Like in the initialization task, this is implemented in parallel through a parallel reduction. Each thread calculates a partial minimum result on a subset of times and saves it on a private copy of the variable. Once they finish, the partial results are compared to obtain the minimum of all of them. After this task completes, the threads perform a synchronization step.

As we can see, the simulation protocol is based on the interaction of the components. Each task starts at the ROOT-COORDINATOR and involves calling functions in parallel along the simulation structure. It follows the PDEVS simulation protocol which can execute DEVS models correctly. In addition, after each task, the threads perform a synchronization step to make sure that each task completes before starting a new one. This definition allows additional parallelism than the algorithms presented in (Zeigler 2017).

Our contribution in this work is to conduct empirical evaluation for this algorithm. Our aim is to analyze how the protocol performs in terms of execution time as well as consumption of energy and power by the shared memory architecture. The methodology we use provides a method for evaluating how this algorithm scales when we use more computing resources. To do this, we assign the threads on the algorithm to different CPU-cores on the computer architectures. This way, we can analyze how this algorithm scales on the different metrics. The following sections will discuss the results obtained in this study.

4 EVALUATING PARALLEL DEVS PROTOCOLS: PERFORMANCE AND ENERGY

In this section we present an experimental evaluation of the parallel DEVS algorithm presented in the previous section and executed on shared memory architectures. The goal is to analyze time, energy, and power performance. The experimental setup we used for our evaluation consists of two independent models built with the Cell-DEVS formalism (Wainer 2009). The first model is a synthetic benchmark. The model is built as a 2D cell space with a one-digit value as the state of each cell. Each cell uses a Moore neighborhood with nine neighbors including itself. The rules the cells evaluate are simple: they start with a zero value in their state variable and change this state from zero to one. Then, they change back to zero, and this is repeated until the end of a predefined simulation time. All cells change their state on every simulation activation. We can see an example of this model in Figure 3. As we can see, the main effort in executing simulations for this model is in communicating the state value to the neighbors.

The second experiment uses an epidemiological model to simulate epidemic spreading. This model divides the population into three classes of individuals: those that are Susceptible, Infected, and Recovered from the disease (SIR). Each cell represents a subset of the population and the state for each cell stands for the portion of SIR individuals in the cell. Following the Cell-DEVS formalism (Wainer 2009), only those cells that are about to change their state will transmit messages to their neighbors, therefore not all of them will execute in every step (which is the case in the benchmark presented in Figure 3: in that model all the cells are active on each step and transmit their state changes to the neighbors). In Figure 4 we can see how this model evolves, the center cell is initialized with infected population, and the disease spreads from there. The infected population in each cell is represented in gray scale. A darker color indicates a higher population percentage infected. All the details for this model can be found in (White at al. 2007, Cárdenas et. al. 2021). The main effort in this model is the equation each cell must compute to update its state.

To evaluate the algorithm on several scenario sizes for these models, we used cell spaces ranging from 100,000 to 1,000,000 cells. All scenarios were executed for 500 simulation steps. We chose these problem dimensions and number of steps to evaluate model scenarios that require large amounts of time and energy to execute, and therefore may benefit with acceleration. In addition, we replicated all experiments 30 times and calculated the average times. All scenarios were executed for 500 simulation steps, and we replicated the results on each scenario 30 times and calculated the average times. We designed our experiments to evaluate this problem on a range of cell space sizes and executing several simulation steps. To perform our evaluation, we use a modern multicore architecture: a server computer equipped with two octa-core Xeon E5-2609V4, giving a total of 16 cores, and 24 GB of RAM memory.

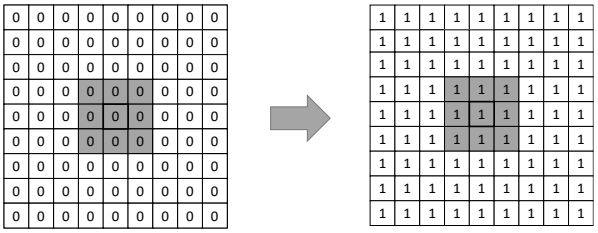


Figure 3: Synthetic benchmark model example.

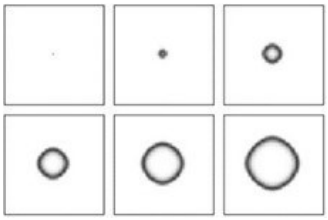
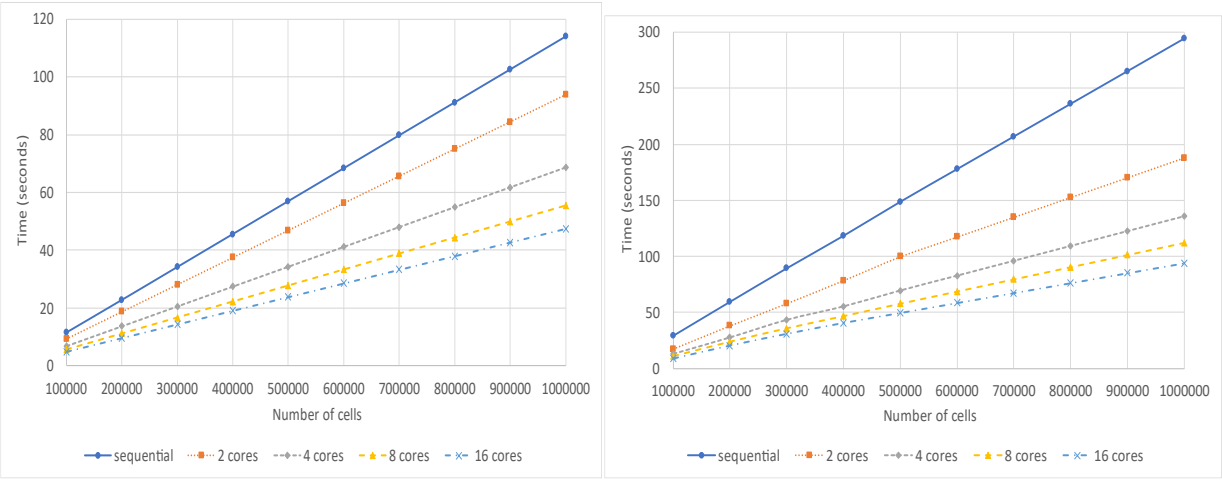


Figure 4: SIR Problem execution example (White et al. 2007).

With the experimental setup defined we conducted our experiments. First, we evaluated the execution time obtained on the sequential algorithm along with the parallel version. The results show the times obtained by increasing the number of cells and the number of CPU-cores used. In Figure 5(a), we can see the time results for the synthetic benchmark. We can see that time increases when we increase the number of cells for every number of CPU-cores used. This is expected because when the number of cells increases, simulations require more computations and longer execution time. In addition, we can see that the parallel algorithm scales and time decreases when we increase the number of CPU-cores used. With more CPU-cores, it performs more computations in parallel. Next, we conducted the second set of experiments on the SIR model. In Figure 5(b), we can see the results obtained for the sequential version, and for the parallel version using different number of CPU-cores. Like in the previous example, on this model we can also see that time increases when the size of the model increases for every number of CPU-cores used. Once again, this is because when the number of cells increases, simulations require more computations to execute the simulations. In addition, here we can also see that the parallel version scales, time decreases when we increase the number of CPU-cores used. This is because of the additional parallel computations performed when using more CPU-cores. Table 1 presents a summary of the average speedup obtained with the different number of CPU-cores used in both problems. We can see the speedups when we add more CPU-cores and therefore the best results we obtain are using all sixteen cores in the architecture. Also, it is worth noticing that for every number of cores, the speedup is higher on the SIR problem than the speedup obtained with the synthetic benchmark. This is because the SIR problem is more computationally intensive and therefore there are more computations to perform in parallel. We must note than even though the communication and synchronization overhead of executing in parallel does not allow us to achieve ideal speedup, our algorithm improves performance by accelerating the simulations several times. In addition, these results improve previous results presented in (Lanuza, Trabes, and Wainer 2019).

The next metric we evaluated is the amount of energy used by the multicore architecture to execute the simulations. The measurements presented in this section are for the multicore processors combined with the energy used by the main memory (RAM) used in the system. In Figure 6(a), we can see the energy results for the synthetic problem. The first observation is that the energy consumption increases when we increase the number of cells. This is expected because when the number of cells increases, simulations require more computations and therefore more energy. In addition, we can see that energy decreases when we increase the number of CPU-cores used on the architecture. These are interesting results considering that we are using more computing resources in the architectures to execute the simulations. However, since the execution time is reduced when we use more cores, this results in less energy used overall. Next, in Figure 6(b) we can see the energy results for the SIR problem. As in the case for the synthetic benchmark

we can see that energy increases when we increase the number of cells. Once again, this is expected because when the number of cells increases, simulations require more computations and therefore more energy. In addition, here we can also see that energy decreases when we increase the number of cores used on the architecture. This is because of the acceleration we obtain using more cores, which leads to less energy used to execute the simulations.

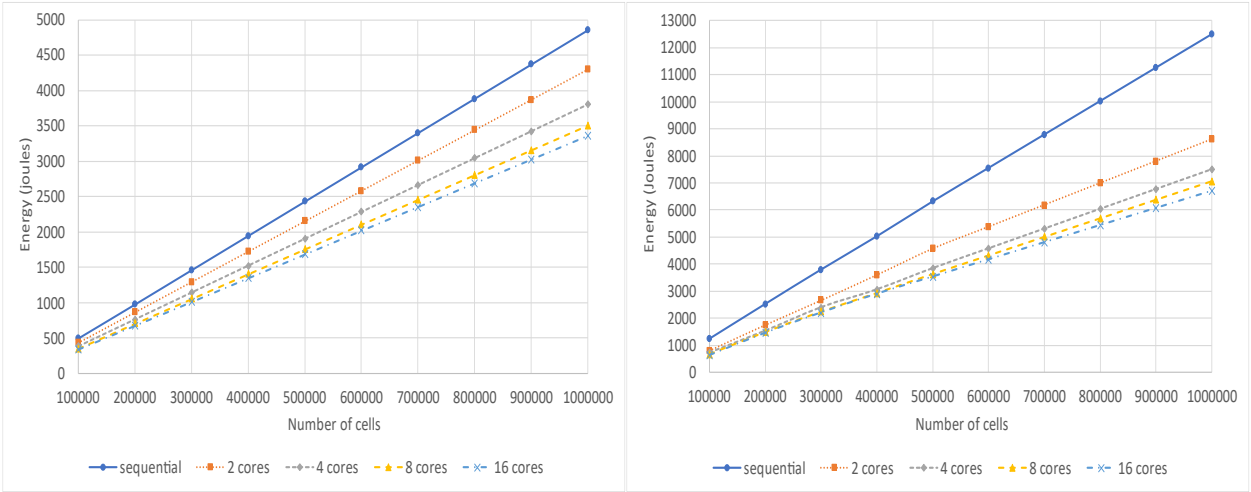


(a) Synthetic benchmark (b) SIR

Figure 5: Execution time.

Table 1: Average speedup summary.

Number of cores	Synthetic Benchmark	SIR
2	1.2	1.54
4	1.66	2.14
8	2.05	2.58
16	2.4	3.03



(a) Synthetic benchmark (b) SIR

Figure 6: Energy results.

Table 2 shows a summary of the average energy use on the parallel algorithm compared to the sequential version. As we can see, the energy use decreases when we increase the number of CPU-cores used for both problems. We can also see that the energy is reduced more on the SIR problem than on the synthetic benchmark problem, for every number of cores, this is related to the higher speedup obtained on this problem. Higher acceleration on the execution leads to less energy used to perform the simulations.

Finally, we analyzed the power consumed by the multicore architecture when executing the simulations. In this case, the results were the same in both problems. In Figure 7 we can see the average results obtained. We can see that power consumption increases when we increase the number of CPU-cores used. This is expected as we are using more computation resources. We should note that executing with two times the number of CPU-cores does not increase two times the power used. Power grows but at a smaller proportion than the number of cores. In addition, as expected, the minimum power we obtained is on the sequential version and the maximum power consumption we obtained is using all sixteen cores in the architecture. Table 3 shows the power consumption compared in the parallel version compared with the sequential version. These results show how much power increases when we increase the number of CPU-cores used compared with the sequential version. When using 2 cores, power increments 7.73%, and as we use more cores it grows higher to 30.15%, 48.34% and 67.50%, with 4,8 and 16 cores respectively. Using more CPU-cores not only includes using more computation circuits, but also more registers and cache memories in the architecture. In addition, the architectures need more effort to synchronize them and maintain the cache coherence. This is the drawback of using more CPU-cores. However, as we saw, the benefit of accelerating the executions leads to less amount of energy used.

Table 2: Average energy use compared to the sequential version.

Number of cores	Synthetic Benchmark	SIR
2	-11.31%	-30.15%
4	-21.63%	-39.32%
8	-27.72%	-42.49%
16	-30.75%	-44.73%

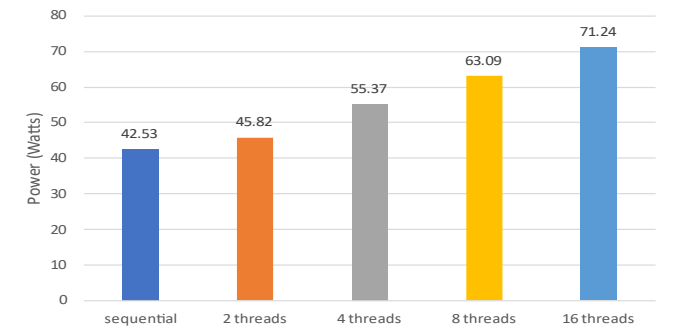


Figure 7: Power results using different number of CPU-cores.

Table 3: Average power consumption compared with sequential version.

Number of cores	Power increase
2	+7.73%
4	+30.19%
8	+48.34%
16	+67.50%

5 CONCLUSIONS AND FUTURE WORK

In this work we presented an energy and power evaluation for DEVS simulations executed in parallel in shared memory architectures. The results show that time and energy increase when we increase the size of the models, as expected. In addition, both time and energy decrease when we increase the number of CPU-cores. Finally, we can see that power increases when we increase the number of CPU-cores. The best results in time and energy were obtained using all CPU-cores in the architecture evaluated, where we obtained a speedup between 2.4 and 3.03, while reducing the energy between 30.75% and 44.73%. However, to achieve these results power increased by 67.50%. In Table 4 we can see a summary of these results. While scaling our algorithm with more CPU-cores leads to improved execution time, we must also consider the accompanying increase in power consumption. However, despite the increase in power consumption, the acceleration is greater than the increase in power, resulting in less total energy used for simulations As future work we propose to conduct an analysis of the energy consumption of communication, computing and memory. In addition, we plan to perform an analysis of the energy used on the different tasks on the algorithm and to evaluate the use of the Dynamic Voltage Frequency Scaling (DVFS) technique.

Table 4: Results Summary using sixteen cores on the architecture compared with sequential version.

Model	Speedup	Energy	Power
Synthetic benchmark	2.4	-30.75%	+67.50%
SIR	3.03	-44.73%	

REFERENCES

Belloli, L., D. Vicino, C. Ruiz-Martin, and G. Wainer. 2019. “Building DEVS Models with the Cadmium Tool”. In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, pp. 45–59. Piscataway, New Jersey, USA: Institute of Electrical and Electronics Engineers, Inc.

Biswas A., and R. Fujimoto. 2016. “Profiling Energy Consumption in Distributed Simulations”. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '16)*, edited by R. Fujimoto, B. Unger, and C. Carothers. pp. 201-209. New York, NY, USA: Association for Computing Machinery.

Biswas, A., and R. M. Fujimoto. 2017. “Energy Consumption of Synchronization Algorithms in Distributed Simulations”. *Journal of Simulation* vol. 11(3), pp 242–252.

Biswas A., and R. M. Fujimoto. 2018. “Zero Energy Synchronization of Distributed Simulations”. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '18)*, edited by F. Quaglia, A. Pellegrini, and G. K. Theodopoulos. pp. 85-96. New York, NY, USA: Association for Computing Machinery.

Cárdenas, R., K. Henares, K., C. Ruiz-Martín, C., and G. Wainer. 2021. “Cell-DEVS Models for the Spread of COVID-19”. In *Proceedings of the 14th International Conference on Cellular Automata for Research and Industry (ACRI)*, edited by T.M. Gwizdalla, L. Manzoni, G. C. Sirakoulis, S. Bandini, and K. Podlaski. pp. 239-249. Switzerland, Cham: Springer.

Chow, A.C., and B. P. Zeigler. 1994. “Parallel DEVS: a Parallel, Hierarchical, Modular, Modeling Formalism”. In *Proceedings of the 1994 Winter Simulation Conference*, edited by J. D. Tew, M. Manivannan, D. A. Sadowski, and A. F. Seila. pp. 716–722. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chow, A.C., B. P. Zeigler and D. H. Kim. 1994. "Abstract Simulator for the Parallel DEVS Formalism". In *Proceedings of the Fifth Conference on AI, Simulation, and Planning in High Autonomy Systems*, edited by P. A. Fishwick. pp. 157-163. Gainesville, FL, USA: Institute of Electrical and Electronics Engineers, Inc.

David, H., E. Gorbatov, U. R. Hanebutte, R. Khanna and C. Le. 2010. "RAPL: Memory Power Estimation and Capping". In *Proceedings of ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, edited by: V. Oklobdzija, B. Pangle, N. Chang, N. Shanbhag, and C. H. Kim, pp. 189-194. New York, NY, USA: Association for Computer Machinery.

Fujimoto, R. M. 1990. "Parallel Discrete Event Simulation", *Communications of the ACM* vol. 33(10), pp. 30–53.

Fujimoto, R. M. 1999. *Parallel and Distribution Simulation Systems.* 1st. ed. New York, NY, USA: John Wiley & Sons, Inc.

Fujimoto, R. M., R. Bagrodia, R. E. Bryant, K. M. Chand, D. Jefferson, D. Nicol, and B. Unger. 2017. "Parallel Discrete Event Simulation: The Making of a Field". In *Proceedings of the 2017 Winter Simulation Conference,* edited by V. Chan, A, D'Ambrogio, G. Zacharewicz, and N. Mustafee. pp. 262-291. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Fujimoto R. M., C. Carothers, A. Ferscha, D. Jefferson, M. Loper, M. Marathe, and S. J. E. Taylor. 2017. "Computational Challenges in Modeling & Simulation of Complex Systems". In *Proceedings of the 2017 Winter Simulation Conference,* edited by V. Chan, A, D'Ambrogio, G. Zacharewicz, and N. Mustafee. pp. 431-445. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Fujimoto, R. M. 2020. "Power Consumption in Modelling and Simulation". *Journal of Simulation* vol. 14(3), pp. 157-168.

Glinsky, E., and G. Wainer. 2002. "Performance Analysis of Real-Time DEVS Models". In *Proceedings of the 2002 Winter Simulation Conference*, edited by E. Yücesan, C. H. Chen, J. L. Snowdon, and J. M. Charnes. pp. 588-594. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hähnel, M., B. Döbel, M. Völp and H. Härtig. 2012. "Measuring Energy Consumption for Short Code Paths using RAPL". *ACM SIGMETRICS Performance Evaluation Review* vol. 40(3), pp. 13-17.

Jafer, S. and G. Wainer. 2011. "Conservative Synchronization Methods for Parallel DEVS and Cell-DEVS". In *Proceedings of the 2011 Summer Computer Simulation Conference (SCSC '11)*, edited by P. Kropf, A. Abhari, M. K. Traoré and H. Vakilzadian. pp. 60-67. Vista, CA, USA: Society for Modeling & Simulation International.

Kim K., W. Kang, B. Sagong, and H. Seo. 2000. "Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One". In *Proceedings of the 33rd Annual Simulation Symposium (SS 2000)*, pp. 227-233. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Khan K. N., M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou. 2018. "RAPL in Action: Experiences in Using RAPL for Power Measurements". *ACM Trans. Model. Perform. Eval. Comput. Syst.* vol 3(2). Article 9.

Lanuza, J., G. Trabes, and G. Wainer. 2020. "Parallel Execution of DEVS in Shared-Memory Multicore Architectures". In *Proceedings of the 2020 Spring Simulation Conference*, edited by F. Barros, X. Hu, H. Kavak, and A. del Barrio. pp. 1-11. San Diego, CA, USA, Society of Computer Simulation International.

Li Q. and M. Zhou. 2011. "The Survey and Future Evolution of Green Computing". In *Proceedings of IEEE/ACM International Conference on Green Computing and Communications*, edited by Z. Shao, I. Bate and Y. Sun. pp. 230-233. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Inc.

Liu, Q., and G. Wainer. 2012. "Multicore Acceleration of Discrete Event System Specification Systems". *SIMULATION* vol. 88, pp. 801–831.

Neal S., R. Fujimoto, and M. Hunter. 2016. "Energy consumption of Data Driven Traffic Simulations". In *Proceedings of the 2016 Winter Simulation Conference,* edited by T. M. Roeder, P. I. Frazier, R.

Szechtman, and E. Zhou. pp. 1119-1130. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Neal, S., and R. M. Fujimoto. 2017. “Energy Consumption of HLA Data Distribution Management Approaches”. In *Proceedings of the 2017 Winter Simulation Conference,* edited by V. Chan, A, D’Ambrogio, G. Zacharewicz, and N. Mustafee. pp. 810-820. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Neal, S., and R. M. Fujimoto. 2018. “Power Consumption of Future Event List Implementations in Discrete Event Simulations”. In *Proceedings of Annual Simulation Symposium (ANSS 2018)* edited by E. Frydenlund, S, Shafer, and H. Kavak. Article 9. San Diego, CA, USA, Society of Computer Simulation International.

Nutaro, J. 2009. “On Constructing Optimistic Simulation Algorithms for the Discrete Event System Specification”. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 19, no. 4.

Nutaro, J. 2019. “Chapter 14 - Parallel and Distributed Discrete Event Simulation”. In *Theory of Modeling and Simulation*. 3rd edition, edited by B. P. Zeigler, A. Muzy and E. Kofman. 339 – 372. San Diego, CA, USA: Academic Press.

Trabes, G. G, G. A. Wainer, and V. Gil-Costa. (in press). “A Parallel Algorithm to Accelerate DEVS Simulations in Shared Memory Architectures”. *IEEE Transactions on Parallel and Distributed Systems*.

Trabes, G. G., V. Gil-Costa, and G. A. Wainer. 2020. “Energy Efficiency Evaluation of Parallel Execution of DEVS Models in Multicore Architectures”. In *Proceedings of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing. pp. 2173 - 2184. Institute of Electrical and Electronics Engineers press.

Vicino, D., D. Niyonkuru, G. Wainer, and O. Dalle. 2015. “Sequential PDEVS Architecture”. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (DEVS ’15)*, edited by F. Barros, M. H. Wang, H. Prähofer, and X. Hu. pp. 165-172. San Diego, CA, USA: Society for Computer Simulation International.

Wainer, G. 2009. *Discrete-Event Modeling and Simulation : A Practitioner’s Approach*. Boca Raton, FL, USA: Taylor & Francis.

White, S. H., A. M. Del Rey, and G. R. Sánchez. 2007. “Modeling Epidemics Using Cellular Automata”. *Applied Mathematics and Computation* vol. 186(1). pp. 193–202.

Zeigler, B., H. Praehofer and T. G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, San Diego, CA: Academic Press.

Zeigler, B., 2017. "Using the Parallel DEVS Protocol for General Robust Simulation with Near Optimal Performance". *Computing in Science & Engineering* vol. 19(03). pp. 68-77.

AUTHOR BIOGRAPHIES

GUILLERMO G. TRABES is a Ph.D. student in Electrical and Computer Engineering (Carleton University) and Computer Science (Universidad Nacional de San Luis). His email address is guillermotrabes@sce.carleton.ca.

VERONICA GIL COSTA is a former researcher at Yahoo! Labs Santiago hosted by the University of Chile. She is currently an associate professor at the University of San Luis and researcher at the National Research Council (CONICET) of Argentina. Her email address is gvcosta@unsl.edu.ar.

GABRIEL A. WAINER is Professor at the Department of Systems and Computer Engineering at Carleton University. He is a Fellow of the Society for Modeling and Simulation International (SCS). His email address is gwainer@sce.carleton.ca.