Towards a massively-parallel version of the SimSEE

Raúl Marichal
Facultad de Ingeniería (INCO)
Universidad de la República
Montevideo, Uruguay
rmarichal@fing.edu.uy

Damián Vallejo
Facultad de Ingeniería (IIE)
Universidad de la República
Montevideo, Uruguay
dvallejo@fing.edu.uy

Ernesto Dufrechou
Facultad de Ingeniería (INCO)
Universidad de la República
Montevideo, Uruguay
edufrechou@fing.edu.uy

Pablo Ezzatti
Facultad de Ingeniería (INCO)
Universidad de la República
Montevideo, Uruguay
pezzatti@fing.edu.uy

***Abstract*—The SimSEE is a simulation software used/designed to aid the decision-making in the electric energy generation market. It is based on Stochastic Dynamic Programming technique and allows to simulate the contribution of several energy sources, such as hydro-electric, solar, thermal or wind energy, to a specific electrical network. Uruguay's electric generation system has considerably grown and diversified in the past decades. This evolution implies potentially more complex scenarios and also motivates a more precise modeling of some electric sources. Therefore, the computational cost of the simulations is also expected to rise and the use of HPC techniques becomes mandatory. In this work we study the performance bottlenecks in the SimSEE tool. Additionally, and considering the previously mentioned results, we design a parallelization strategy that enables its acceleration using massively-parallel devices such as GPUs.**

***Index Terms*—Coarse-grained parallelism, electric energy generation, Stochastic Dynamic Programming**

I. INTRODUCTION

The SimSEE (Electric Power Systems Simulator) is a software tool that allows for customized simulations of an electric energy generation system. Its principal purpose is to aid in the decision-making process both in the long term (investment planning) and in the short term (system operation and market simulation) [1]. It was developed at the Universidad de la República, by engineers of the Instituto de Ingeniería Eléctrica (IIE) of the Facultad de Ingeniería (FING), between the years 2006 and 2008, keeping a constant evolution since then [2], [3]. The software is multiplatform (currently for Windows and Linux) and is developed entirely in the Pascal programming language. Additionally, it is publicly available and open-source. One of its most heavy users nowadays is ADME, an Uruguayan public agency in charge of managing the electricity generation market in this country.

The radical change experienced by the Uruguayan electricity generation matrix in the last decade, incorporating an important capacity of wind and solar generation sources, poses significant challenges for predicting demand and generation capacity in the short and medium-term to optimize the use of the different sources available. For example, it is of utmost importance to represent the electrical system with higher accuracy. In other words, it is necessary to adapt to the increase in the number of actors and complexity of the system and, at the same time, maintain the levels of precision.

To calculate the system's cost function, the SimSEE performs a discretization of the state space and time domain. Then, it makes a backward sweep in the time domain, minimizing the operating cost of each step (which equals the cost of the step plus the future cost of the arrival state). The computational cost of this operation is proportional to the Cartesian product of the state variables of the problem, so adding more of them or increasing their discretization can make the problem impossible to solve, in a reasonable time, using conventional tools. In this context, the need to adapt the SimSEE to the possibilities offered by current parallel computing platforms is evident.

Graphics Processing Units, or GPUs, are coprocessors initially designed for image rendering. They mainly aim to relieve the CPU load in applications with a high demand for these types of operations, such as video games or simulations with a high 3D content, among other things. In recent years, several programming languages were developed to allow using GPUs to compute general-purpose problems such as simulations and numerical models that require great computing power and are massively parallelizable [4]. The most remarkable example in languages is CUDA, specially tailored for `Nvidia` GPUs.

In this work, we firstly identify the performance bottlenecks in the SimSEE tool. Later. we study how to adapt the algorithms used in SimSEE to take advantage of the computing power of massively parallel processors. These algorithmic changes can enable the simulation of larger and more refined scenarios while keeping the tool simple regarding the coding of new models.

The rest of the paper is structured as follows. In Section II we briefly review the SimSEE simulator. Next, in Section III, we experimentally evaluate the baseline version of this tool. After that, and considering the previously obtained results, we design a new variant for the SimSEE that is described in Section IV. Finally, Section V presents the concluding remarks and an outline of future research.

II. The SimSEE tool

The SimSEE tool is distributed among various programs, which allow the management of information in an orderly manner. In turn, the information of the electrical system configuration is grouped in what is known as a *play room* or simply *room*. A SimSEE room is an input file where the electrical system that will be simulated is described. This description comprises aspects such as the number and type of electrical generators, loads, time frames, generator maintenance schedules, and forecasts generators, among others. The generators of a certain room are called *actors*, while the forecasts are called *sources*. The number of state variables of a room equals the sum of the number of state variables of its *sources* and *actors* . An example of state variables is the one used for depicting the volume of water in hydro generator's reservoirs.

The use of SimSEE involves two main stages: optimization and simulation. In the optimization process, the time frame is swept inversely, solving the minimization problem of the cost of each time step and for each point in the space of the state variables. With this, a function of future cost is built, which depends on time and the state. On the other hand, simulation sweeps time forward and starts from a given initial state. The execution of the simulation requires performing the optimization first to obtain the cost function. Additionally, stochastic processes such as water inflows to reservoirs, wind speed, solar radiation, and generators failures need to be considered during the simulation. For this reason, in optimization as well as in simulation, many chronicles are executed with different realizations of these processes, and the results are generally expressed in terms of the expected value or as distributions or probabilities of exceedance.

In the course of this work, only the simulation process will be considered (it should be noted that optimization variant, from a computational point of view, is only an extension of simulation counterpart). This process can be roughly divided into different sections. An initialization part, where the program reads the room file, creates all the instances, lists, reads the cost function's file created in the optimization process, and prepares everything for the simulation to start. Then, within the actual simulation (the baseline version), two loops are distinguished, the chronicles loop and the time steps loop. The simulator is programmed so that the outer loop iterates through all the chronicles, while the inner loop traverses the time steps of each chronicle. For each time step of the simulation, an optimization problem is solved using the Simplex algorithm. The Simplex method implemented in the SimSEE is based on the work of Rutishauser et al. [5].

III. Experimental evaluation of SimSEE

As described in previous sections, the SimSEE is a large-scale simulation and optimization tool. To improve the performance of this simulator (or advance on the simulation precision), we evaluate, for different case scenarios, the computational cost of the functions and procedures involved in the execution of the tool.

We use the `callgrind/valgrind` profiling tool [6] in order to determine which units (set of functions and procedures) involve more computational cost and also to understand how the data flows through the simulation. The addressed scenarios can be classified into three categories: hourly, daily, and weekly (in other words, short, mid, and long-term). For each of these test cases, we also vary some of the parameters in the *playrooms*. We follow this strategy in order to reach results with enough diversity, reducing the bias in the evaluation that can arise from the particularities of a certain dataset. In this context, we evaluate the most costly operations for each combination of these cases and variations. To depict the complexity of the SimSEE tool, a graphical representation of the functions and procedures according to their execution time for the daily case is shown in Figure 1.

We start by evaluating the short-term (hourly) data set, which includes actors whose state variables are the volume of water in hydroelectric dam reservoirs. As we mentioned earlier, with the goal to explore different scenarios we created, in this case, three *playrooms* varying the water volumes: *dry*, *mid* and *humid*.

Tables I, II and III summarize the obtained results for each of the three scenarios, where each row displays the percentage of execution, CPU cycles, number of invocations and the unit comprising the most computationally expensive procedures.

TABLE I
Hourly dry playroom.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`PASOBUSC...`	5.7	2.902×10^9	4.786×10^5	`USIMPLEX`
`INTERCAMBIAR`	5.5	2.772×10^9	$1,640 \times 10^6$	`USIMPLEX`
`RND`	4.8	2.421×10^9	7.457×10^7	`FDDP`

TABLE II
Hourly mid-humid playroom.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`PASOBUSC...`	6.8	3.612×10^9	5.942×10^5	`USIMPLEX`
`INTERCAMBIAR`	4.9	2.584×10^9	1.611×10^6	`USIMPLEX`
`RND`	4.6	2.421×10^9	7.457×10^7	`FDDP`

TABLE III
Hourly humid playroom.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`EROGADO...`	10.5	6.949×10^9	2.037×10^5	`UHIDRO...`
`VAL_X`	9.0	5.929×10^9	2.044×10^8	`COMPOL`
`PASOBUSC...`	5.1	3.392×10^9	4.977×10^5	`USIMPLEX`
`INTERCAMBIAR`	4.6	3.030×10^9	1.678×10^6	`USIMPLEX`
`RND`	4.6	2.421×10^9	7.457×10^7	`FDDP`

Considering the results in Tables I and II the top CPU-consuming procedure is, in the *dry* and *mid-humid* cases and for the *hourly* simulations, `PASOBUSCARFACTIBLEIGUALDAD4`. This procedure is at the core of the Simplex algorithm computation, and its purpose is to find the best feasible step or permutation in the simplex matrix.

On the other hand, in the *humid* test case, Table III, the procedure that concentrates the most runtime is `EROGADOMINIMO_CTRL_CRECIDA`. This procedure is invoked when the water levels in the hydro reservoirs are very close to the state variable upper bound, which is common in the *humid* scenario.

Unlike the hourly playrooms, which only have hydraulic actors, daily step playrooms also include wind and solar

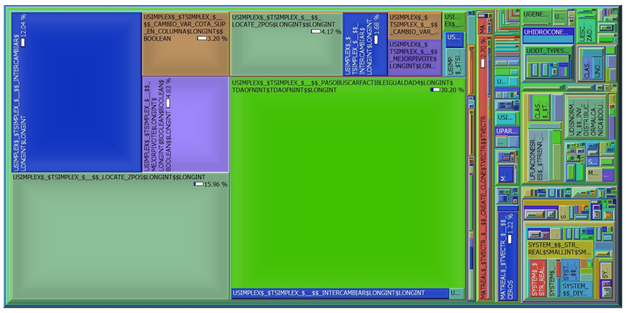

Fig. 1. Graphical representation of the call tree of a daily playroom without modifications. The area of each square is proportional to the runtime of the corresponding procedure.

farms, which do not have any state variable associated.It should be highlighted that this kind of playroom is the most required by SimSEE users [7]. This case is formulated as varying the number of solar and wind actors. Since these actors do not have state variables, they can be grouped into macro-actors with the same power generation. Tables IV, V and VI summarize these experiments.

TABLE IV
DAILY PLAYROOM WITH NO MODIFICATIONS.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`PASOBUSC...`	29.0	3.349×10^{11}	1.348×10^{6}	`USIMPLEX`
`LOCATE_ZPOS`	20.3	2.344×10^{11}	5.991×10^{7}	`USIMPLEX`
`INTERCAMBIAR`	15.4	1.780×10^{11}	8.258×10^{6}	`USIMPLEX`
`MEJORPIVOTE`	6.1	6.991×10^{10}	5.972×10^{7}	`USIMPLEX`
`CAMBIO_V...`	4.1	4.719×10^{10}	5.670×10^{10}	`USIMPLEX`

TABLE V
DAILY PLAYROOM WITH 20% MORE WIND ENERGY PRODUCTION.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`PASOBUSC...`	27.6	3.672×10^{11}	1.348×10^{6}	`USIMPLEX`
`LOCATE_ZPOS`	22.1	2.936×10^{11}	6.765×10^{7}	`USIMPLEX`
`INTERCAMBIAR`	18.2	2.422×10^{11}	9.214×10^{6}	`USIMPLEX`
`MEJORPIVOTE`	5.9	7.783×10^{10}	6.742×10^{7}	`USIMPLEX`
`CAMBIO_V...`	3.5	5.252×10^{10}	6.401×10^{7}	`USIMPLEX`

TABLE VI
DAILY PLAYROOM WITH ONLY ONE MACRO-ACTOR.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`PASOBUSC...`	25.9	1.179×10^{11}	1.160×10^{6}	`USIMPLEX`
`INTERCAMBIAR`	20.6	9.388×10^{10}	8.174×10^{6}	`USIMPLEX`
`LOCATE_ZPOS`	8.6	3.909×10^{10}	2.455×10^{7}	`USIMPLEX`
`MEJORPIVOTE`	6.2	2.819×10^{10}	2.418×10^{7}	`USIMPLEX`
`CREATE_CLONE`	4.5	2.037×10^{10}	1.237×10^{7}	`MATREAL`
`CAMBIO_V...`	3.8	1.730×10^{10}	2.076×10^{9}	`USIMPLEX`

The results presented in previous tables show that there are no essential differences for daily cases in terms of computational cost. Again, it should be noted that `USIMPLEX` is the unit with the most costly operations, particularly the ones associated with the simplex resolution. However, if we measure the impact of grouping all the non-state variable actors for each simulation, as shown in Table VII, it can be appreciated that there is a massive difference between the elapsed time.

Finally, for the *weekly* playrooms, we study the impact on the simulation of considering the occurrence of an issue in the hydraulic generators. For this purpose, we modify the probability of an incident in the range between 0.01 to 0.15. Tables VIII and IX sintetize these experimental evaluations.

TABLE VII
ELAPSED TIME (IN SEC.) OF DAILY PLAYROOM SIMULATIONS.

Playroom	**Elapsed time (s)**
Daily with no modifications	158.133
Daily with 20% more wind energy production	178.024
Daily with only one macro-actor	61.713

TABLE VIII
WEEKLY PLAYROOM WITHOUT MODIFICATIONS.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`_INV_DIST...`	12.4	2.788×10^{10}	3.302×10^{7}	`UDISNORMCAN`
`PASOBUSC...`	12.1	2.726×10^{10}	1.030×10^{6}	`USIMPLEX`
`INTERCAMBIAR`	11.2	2.518×10^{10}	4.702×10^{6}	`USIMPLEX`
`LOCATE_ZPOS`	6.1	1.367×10^{10}	1.495×10^{7}	`USIMPLEX`
`MEJORPIVOTE`	2.5	5.532×10^{9}	1.482×10^{7}	`USIMPLEX`
`_CALCULAR_XS`	2.2	5.042×10^{9}	6.450×10^{4}	`UESCLA...`

TABLE IX
WEEKLY PLAYROOM WITH CORRECT FUNCTION PROBABILITY OF 85% OR BREAK PROBABILITY OF 15%.

Procedure	Perc. (%)	CPU Cycles	Calls	Unit
`_INV_DIST...`	12.5	2.788×10^{10}	3.302×10^{7}	`UDISNORMCAN`
`PASOBUSC...`	12.2	2.721×10^{10}	1.031×10^{6}	`USIMPLEX`
`INTERCAMBIAR`	10.8	2.413×10^{10}	4.639×10^{6}	`USIMPLEX`
`LOCATE_ZPOS`	6.1	1.356×10^{10}	1.484×10^{7}	`USIMPLEX`
`MEJORPIVOTE`	2.4	5.461×10^{9}	1.471×10^{7}	`USIMPLEX`
`_CALCULAR_XS`	2.3	5.042×10^{9}	6.450×10^{4}	`UESCLA...`

Results for weekly simulations show that there are not much differences in runtimes where the probability of an issue in a generator is varying. It should be noticed that, again, `USIMPLEX` procedures represent a strong impact into the runtime of the whole simulation, around 30%.

From the obtained results, we can state that the operations–procedures that require a large percentage of the total runtime of these simulations are from `USIMPLEX` unit. This unit is in charge of representing and solving the simplex method. Additionally, the considerable computational cost of the previous experiments is mainly due to the vast number of simplex solutions required and not to the cost of each solution. In other words, the dimension of each problem (simplex resolution) is negligible regarding their solution in modern massively-parallel devices.

IV. DESIGN OF A PARALLEL VERSION OF SIMSEE

After the experimental evaluation stage (previous section), we can conclude that none of the functions or procedures have a computational cost that can represent a critical bottleneck. In other words, none of the procedures can be significantly optimized to improve the total runtime. This situation allows discarding the use of fine-grain parallelism as a promising approach to accelerate the model. Therefore, we propose a different strategy, based on leveraging the coarse-grain parallelism [8] to adapt the execution of SimSEE simulations to massively parallel devices, like GPUs.

Instead of concentrating on optimizing a particular operation by exposing parallelism, the new approach implies modifying the whole computational structure of the simulation. The main idea is to compute many independent trajectories concurrently, executing multiple instances of `resolver` or, equivalently, solving multiple simplex matrices in parallel.

Initially, the simulation can be simplified into two main loops. As mentioned in Section II, the outer loop iterates through the trajectories or chronicles, and for each of these

TABLE X
ESTIMATED TIME OF ONE STEP, FOR EACH STAGE OF THE PARALLEL PROPOSAL FOR DIFFERENT NUMBER OF TRAJECTORIES.

# of Chronicles	Initialize structures	HtoD transf.	DtoH transf.	Total transf.	Total + GPU res. time
1	0.05	0.07	0.07	0.19	2.19
32	1.50	1.39	1.70	4.59	6.59
64	3.01	2.71	3.32	9.04	11.04
128	6.02	5.48	6.70	18.19	20.19
512	24.06	22.23	27.17	73.47	75.47
1024	48.13	43.57	53.25	144.94	146.94

trajectories, there is a loop that solves a simplex matrix of each time step. As each trajectory is independent, we redesign the algorithm to solve for the different trajectories in parallel in each step. Concretely, we parallelize the solution of several simplex matrices, one for each trajectory.

To validate that this is a reasonable strategy, we estimate the cost of initialization, pre-processing, solving (multiple simplex matrices), and post-processing the results. The evaluation can be divided into CPU and GPU estimations. For the CPU original version, we first isolate the resolution step, measuring the cost of a simplex instance. For the current CPU implementation, this cost is $\sim 1ms$. Then, we scale that time to estimate the cost of solving multiple trajectories.

From the GPU side, we need to measure the cost of transferring the data required to compute the corresponding number of trajectories. Additionally, the GPU resolution involves other stages. For example, the initialization stage includes invoking the GPU driver and memory allocation, both only needed once before starting the simulation. Inside the simulation step, a pre-processing phase rearranges the input data, organized in SimSEE's object-oriented data structures, to a more GPU-friendly data layout for the parallel solution of the simplex. Then, the resulting data structures are transferred from the CPU to the GPU memory (through a cudaMemCpy command). Solving the multiple simplex matrices associated with different trajectories on the GPU side in parallel is the main stage of the method, whose cost should be comparable to solving one step in CPU. The last stage, post-processing, consists of transferring the data back to the CPU with the solution and updating the original data structures. These values for different groups of chronicles, with the number of chronicles varying between 1 and 1024, are summarized in Table X. The results show that the time for the data transferences is less than the CPU simplex computations, especially when the number of chronicles grows. For largest scenario, the difference in runtime is in the order of $8\times$ in favour of the new paradigm.

V. CONCLUDING REMARKS AND FUTURE WORK

In this work, we have evaluated the computational performance of SimSEE, a computational tool aimed at assisting in decision-making in the electric energy generation market.

In order to pursue our objectives, we design different realistic test cases, grouped in three different classes: hourly, diary, and weekly scenarios. The experimental evaluation suggests that none of the procedures represent an important bottleneck from a computational point of view. Contrary, the enormous number of invocations of some specified routines configures the main challenge.

Considering the previously described situation, we advance with the design of a parallel version to optimize the SimSEE runtimes. In particular, we depart from the idea of including fine-grain parallelism to focus on a coarse-grain approach. In this line, we preliminarily evaluate and validate the conditions to apply this kind of strategy and complete a design of the data structures and the main workflow of a new parallel variant of the SimSEE tool.

As future work, we identify several lines. Firstly, we need to implement and experimentally evaluate versions of the new module for different parallel hardware platforms, such as traditional multi-core CPUs, ARM multi-core CPUs, and GPUs. Another essential aspect that was not covered in the present effort is leveraging distributed platforms, especially considering the scaling in the dimension of the addressed problems. Finally, it is interesting to develop a procedure to automatically select, considering the targeted problem characteristics, the best version-hardware platform to perform the simulations.

ACKNOWLEDGMENT

The authors of this article were partially financed by project ANII FSE_1_2018_1_153060.

REFERENCES

[1] E. Coppes, C. Tutté, F. Maciel, M. Forets, E. Cornalino, and R. Chaer, "SimSEE Proyecto ANII FSE 2009 18 Mejoras a la plataforma SimSEE," 2012.

[2] G. Flieller and R. Chaer, "Introduction of ensemble based forecasts to the electricity dispatch simulator simsee," in *2020 IEEE PES Transmission & Distribution Conference Latin America (T&D LA)*, pp. 1–6, IEEE.

[3] V. Camacho and R. Chaer, "Hourly model of a combined cycle power plant for simsee," in *2020 IEEE PES Transmission & Distribution Conference and Latin America (T&D LA)*, pp. 1–5, IEEE, 2019.

[4] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[5] H. Rutishauser, M. Gutknecht, W. Gautschi, H. Schwarz, P. Henrici, and P. Läuchli, *Lectures on Numerical Mathematics*. Birkhäuser Boston, 1990.

[6] J. Weidendorfer, "Sequential performance analysis with callgrind and kcachegrind," in *Tools for High Performance Computing*, pp. 93–113, Springer, 2008.

[7] "Usos del SimSEE." https://www.simsee.org/simsee/usos.html. Accessed: 2021-07-31.

[8] J. I. Aliaga, E. Dufrechou, P. Ezzatti, and E. S. Quintana-Ortí, "Accelerating the task/data-parallel version of ILUPACK's BiCG in multi-CPU/GPU configurations," *Parallel Comput.*, vol. 85, pp. 79–87, 2019.