# PERFORMANCE OPTIMIZATION OF GAIA OPERATIONAL SIMULATOR USING PCOF & PDES METHODOLOGIES

**Workshop on Simulation for European Space Programmes (SESP)**
**24-26 March 2015**

**ESA-ESTEC, Noordwijk, The Netherlands**

Sami Hammal[(1)], Vagelis Alifragkis[(1)], Nikolaos-Antonios Livanos[(1)],
Anthony Walsh[(2)], Vemund Reggestad[(2)]

*[(1)]EMTech*
*44, Kifisias Ave. Marousi, 151 25, Athens, Greece*
sami.hammal@emtech.gr, vagelis.alifragkis@emtech.gr, nikolaos.livanos@emtech.gr

*[(2)]European Space Agency ESA/ESOC*
*Robert-Bosch-Str. 5, D-64293 Darmstadt*
anthony.walsh@esa.int, vemund.reggestad@esa.int

## ABSTRACT

A common contemporaneity in spacecraft simulations is the aim to achieve the best possible performance, accuracy and fidelity regarding model's execution. An approach to meet these demands is to incorporate computer hardware equipped with multi-core CPUs, capable of executing extremely heavy computational work using several processes and threads. A farther approach is to adapt the simulation environment to using multi-core technology according to a certain framework by incorporating appropriate software tools and methodologies, with vision to improve the system's capacities with features promising enhanced usability and execution performance. Moreover, simulator development is carried out based on incremental deliveries; the advantage of having appropriate tools for performance analysis provides the ability to detect performance issues early during development, which constitutes a significant benefit regarding project cost and schedule. As part of the "Enhancement of SIMSAT Simulation Engine (ESSE)" ESA/ESOC's project, an innovative Conservative-Parallel Discrete Event Simulation (C-PDES) scheduler for SIMSAT along with a Performance Control and Optimization Framework (PCOF) were realized. The current paper presents the successful attempt to utilize the C-PDES engine and the PCOF in the GAIA operation simulator, aiming to achieve performance evaluation and parallelization.

## INTRODUCTION

Satellite simulators at ESA/ESOC are built-up on top of the SIMULUS infrastructure, [1], [2], which comprises of a run-time framework (SIMSAT), software emulators and a set of reusable generic models. Recent Operational Simulator developments are based on a Reference Architecture (REFA), [3], [4]. REFA and all future operational simulators that will be based on REFA are compliant with the SMP2 standard, [5]. SMP2 embraces the ideas of component-based design and Model Driven Architecture (MDA) as promoted by the Object Management Group (OMG) and is based on the open standards of UML and XML. The SMP2 specification provides standardized interfaces between the simulation models and the simulation run-time environment for common simulation services as well as a number of mechanisms to support inter-model communication.

Over the years, efforts have been made to support concurrency and parallelization in simulation environments. Initially, a concurrent scheduler [11] was developed for the SIMSAT environment, which utilized O/S threads in order to execute certain predefined simulation events in parallel. To support this novel scheduler, a study was undertaken in 2003 [12] that utilized the concurrent features of the SIMSAT scheduler in order to execute parts of the Rosetta's operational simulator concurrently. Another effort was a realization of a distributed master/site daemon architecture in SIMSAT, which allowed a single simulation to be run across multiple operational nodes, thus leading to parallel configuration where the master daemon was the main "core" of the simulation while the sites achieved synchronization based on the Master's clock. The most notable effort to utilize this infrastructure was the SWARM operational simulator [13], which was a cluster of identical spacecrafts, each one operating in different SIMSAT Site node. While all of the above are notable efforts that paved the way for distributed execution and parallelization of spacecraft simulations, there were several issues that made them difficult and counter-intuitive. The Concurrent Scheduler used a predefined number of parallel entities and each

event parallelization required re-compilation of the simulator. The Master/Site architecture, while it offered large scale parallelization, it failed to deliver sufficient synchronization mechanisms since the synchronization was predefined and could be enforced solely by the Master daemon. The most important caveat, however, was that both failed to define a well-structured methodology to lead to efficient parallelization of the simulator, which had as a result a difficult to use infrastructure, as parallelization efforts could greatly diminish the simulation's fidelity.

To address the aforementioned issues, an ESA/ESOC TRP study "Linux & Multi-Core Processor Technology for Simulators - LMCPTS" (2009-2011) was conducted, targeting to determine the prerequisites, as well as augmentations, required for introducing parallelism in such simulation environments. The study presented the Parallel DES (PDES) and the related Conservative and Optimistic approaches, [6]. In order to support the conservative PDES scheduling approach, as well as perform critical optimizations of the execution time required by specific software parts, a Performance Optimization Framework (POF) was proposed. The latter described a set of tools developed to monitor the performance of the simulation environments from different viewpoints and allowed the detection of parallelization possibilities [7]. Moreover, efforts were made in order to perform an analysis regarding performance evaluation of existing operational simulators. The procedure that was followed along with the results are presented in [9], which was a first iteration of utilization of the POF tools.

Following this study, the ESA/ESOC/ESTEC project "Enhancement of SIMSAT Simulation Engine" (ESSE) (2012-2014) was undertaken that involved implementation of fully functional Conservative PDES (C-PDES) engine, as well as a new version of the PCOF tools. Major requirements of the C-PDES engine were to support backward compatibility with the existing scheduler (concurrent scheduler), multiple O/S thread and processes, configurable synchronization mechanisms and optimized communication protocols. Major requirement for the PCOF tools was to provide a useful environment for spacecraft operational simulators to acquire and analyze performance, thus enabling performance control and optimization during the development lifecycle of the simulators [8].

The following paragraphs briefly describe our attempt to utilize the C-PDES engine and the PCOF methodology and tools within a real operational simulation, namely GAIA. In section 2 the paper refers to the SIMULUS infrastructure and introduces the reader to C-PDES and the PCOF. Our attempt to introduce the aforementioned tools within GAIA along with the results of our attempt are presented in section 3. Finally, section 4 concludes the work done and proposes some possible future steps towards parallelization and optimization.

## PERFORMANCE IN OPERATIONAL SIMULATORS

Design and development of Spacecraft operational simulators based on SIMULUS infrastructure is an intricate procedure, often involving several software engineers and different development teams. In most cases, an incremental-delivery software development lifecycle scheme is utilized, with each delivery satisfying more features, functionalities and requirements, thus providing more complicated software packages and code lines. Most of the time, this practice may introduce unforeseeable changes regarding the system's performance, and under certain circumstances this may present certain risk during the development lifecycle procedure. On the other hand, spacecraft operational simulators cannot take advantage of possible parallelization of workload between different sub-systems, due to the fact that most of spacecraft's models are tightly coupled with the emulator of the central processor, executing the real On Board Software (OBSW). Even if certain parts of the system's models are completely independent, being able to be executed in parallel, the current version of SIMSAT simulation kernel does not support an efficient manner to allow synchronized parallel execution.

In order to address the aforementioned issues, the need of a Performance Control & Optimization Framework (PCOF) and a related knowledge base is identified for SIMULUS based operational simulators. This shall target to the improvement of software quality, performance and development costs, while in the same time shall reduce the development risks. The following paragraphs introduce concepts and methodologies of PCOF applied in GAIA operational simulator.

### Introducing the Conservative-Parallel Discrete Event Scheduler (C-PDES)

The proposed C-PDES component was based on the widely known concepts and theory of Parallel Discrete Event Simulation (PDES), which is the parallel extension of the common Discrete Event Simulation (DES) [10]. Concerning the engineering field of spacecraft operational simulators, all the functionalities of the simulator are executed according to time scheduled events. Several of these are based on a cyclic, i.e. periodic, nature, while others are spontaneously initiated by functionalities executed within these cyclic events, or external triggers. The principal question is whether it

is possible to parallelize these events, and how this can be realized. Fig. 1 pictures such a case, with several parallelized time horizons. All time horizons are synchronized between each other, using specific "Time-Windows" with the first horizon running the major part of the system. Time-Windows define an available time period within which an event is allowed to run in relation to another. The definition of the "Time-Windows" is the "Conservative" information that is required for parallel execution. These are determined after extended analysis and evaluation regarding the time dependency of the events, in order to avoid causality errors. The time-horizons in the Fig. 1 are executed in different operating system threads, and in case of multi-core processors, these can be allocated in different cores; thus being executed in parallel. All these parallel threads must be continuously synchronized with the real-time clock, in case there is a need for real-time execution of the system.

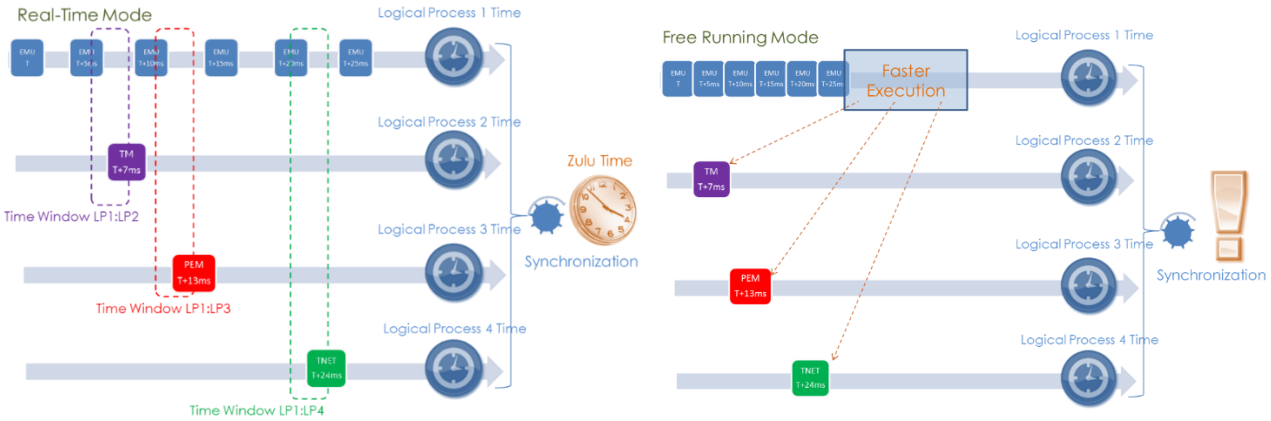


Fig. 1. Synchronized PDES operation for Real-Time & Free Running execution

The C-PDES scheduler for SIMSAT, is a software component providing such parallelization opportunity in SIMULUS based spacecraft operational simulators. It could be considered as a new parallelized "heart" of the simulator, supporting concurrent execution of the functionalities, based on a well-defined timing synchronization methodology. The proposed C-PDES engine for SIMSAT is based on a hybrid implementation using both operating system processes and threads.

The following components are included in the PDES engine:

- The Logical Process Group (LPG) which is depicted as an OS process and is the container of all the other components. Each SIMSAT daemon owns a single LPG.
- The Logical Process (LP) is depicted as an OS Thread that resides within an LPG and is the main execution unit of the PDES engine, as it is responsible for executing the simulation's events. Multiple LPs can reside within a single LPG.
- The Scheduling Scheme Table (SST) is responsible for collecting all the necessary conservative information based on an XML file and then distributing it to the rest of the PDES engine as needed.
- The Synchronizer (SYN) is responsible for handling and applying the synchronization schemes to be used during the execution of the simulation. New synchronization schemes can be defined via the use of a provided interface.
- The Event Scheduler (ES) is responsible for routing the different events of the simulation to the appropriate LPGs and LPs. It is also responsible for locating possible causality errors before they can actually occur as it is defined by the conservative dogma.
- The Communication Manager (CM) is responsible for handling all the communication between different LPGs. It uses communication protocols for different situations, while new ones can be developed via a provided interface.

As already described, the C-PDES engine is based on the conservative information provided to it. To that end, the SST component holds and distributes this information as needed. However, it should be noted that gathering this information requires prior analysis of the system in order to determine parallelization possibilities along with the necessary constraints required for it to work correctly. In an attempt to address this issue the PCOF tools were developed and are described below.

**Introducing the Performance Control & Optimization Framework (PCOF)**

The proposed PCOF for spacecraft operational simulators is considered as an independent development and control process, conducted more or less in parallel with the simulator's development. The principal objectives of the proposed PCOF are the following:

- *Control performance*, by means of placing performance requirements and be able to track system's performance, throughout the entire development lifecycle.
- *Analyze performance*, by means of providing tools, procedures and guidelines to acquire information related to performance, investigate and evaluate findings and generate crystal-clear reports, incorporating the facts, the proof and the path towards performance improvement.
- *Optimize performance*, by introducing improvements inside the architecture and the implemented code, so as to achieve faster execution. The optimization focuses on two different approaches; one targeting parallelization and another focusing on code inefficiencies and poor implementation.
- *Improve the PCOF Knowledge base*. This is an information database, containing experience of performance optimization applied in past. Assuming that performance optimization in SIMULUS based spacecraft operational simulators is a rather complicated activity, reusability of optimization techniques and enhancements is considered of utmost importance.

Several tools have been developed to support the targets of the aforementioned methodology. These are:
- External Profiling Tools (EPT), is using 3rd party COTS profiling tools, such as OProfile and Linux kernel metrics acquiring information regarding Linux kernel from the /proc file system.
- Internal Performance Metrics (IPM), aims at measuring metrics existing inside the models of a spacecraft assembly; in other words measurements related with REFA utilization in case of specific model implementation.
- Simsat Performance Metrics (SPM), aims at measuring metrics and characteristics related to SIMSAT environment (i.e. Scheduler, kernel, speed factor, script code execution, etc.).
- Event Reporting System (ERS), aims at acquiring information about event and function call executions occurred inside REFA-based simulators during runtime using Graphviz-based graphs that represent elements' execution paths.
- Reporting (RPT), aims at providing the user with an easy and fast way of visualizing the results of the other performance measurement tools. It utilizes Open Office in order to display spreadsheets and graphs.

These tools are independent to each other and as already mentioned, target their functionality to different aspects of the simulation process. Their general operation is depicted in Fig. 2. Notice the whole process is considered to be iterative to achieve the best possible results.

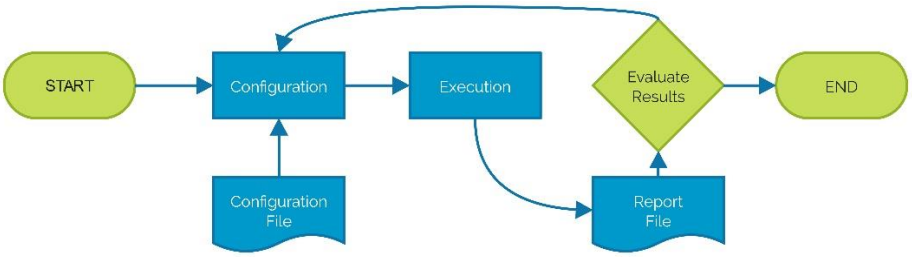


Fig. 2. PCOF Basic Workflow

Initially the tool is configured via XML files that describe its operation's parameters. Example parameters are the tool's operation duration, which metrics should be included, etc. After the configuration step the tool begins each operation and displays the results in real time. Results are also logged to appropriate output files. After runtime is complete, the logged metrics are visualized via the RPT tool to allow the user to examine them in total and acquire a complete image of the simulation's operation. In order for this concept to be supported the tools offer appropriate facilities to the end users such as published fields and methods, tagging of reports and integration with COTS tools.

## DEMONSTRATION IN GAIA OPERATIONAL SIMULATOR

C-PDES and PCOF utilization required several steps to be conducted. At first, the PCOF tools were integrated within the simulation environment. Furthermore, the default SIMSAT scheduler was replaced by the C-PDES scheduler. After both components were fully integrated, attempts were made to acquire an estimation regarding the GAIA operational simulator in order to evaluate its operation. After the evaluation, the C-PDES engine was utilized in order to parallelize parts of the simulator. Meanwhile, the PCOF tools were used to investigate, analyze and subsequently improve a potential performance bottleneck that required code optimization. The present attempt is considered a preliminary application case, and the GAIA (D2 release) has been used as the "case" study.

The principal objectives of the PCOF for the GAIA operational simulator are the following:

- Demonstrate how to control performance, by means of placing performance requirements and being able to track system's performance. The latter is achieved by defining benchmarking procedures that can easily be applied any time during development, in order to validate system execution against specified performance margins.
- Demonstrate how to analyze performance, by means of applying the tools, procedures and guidelines, to acquire information related to performance, investigate and evaluate findings and generate reports incorporating the facts, the proof and the path towards performance improvement.
- Demonstrate how to optimize performance, by introducing certain improvements inside system's architecture and implemented code, so as to achieve faster execution. The optimization focuses on two different approaches; one targeting parallelization and another focusing on code inefficiencies and poor implementation.

**Benchmark evaluation**

The principal objective of the Benchmarking evaluation is to design and develop profiling procedures for the operational simulator under investigation in order to acquire an initial, "rough", performance profiling. This procedure is identified as CPU & Performance Profiling. Using very indicative operations of the simulator, such as the idle mode of the spacecraft, repeatedly transmitting TM or receiving TC to/from the ground segment, or even repeatedly transactions between the main processor and payloads/subsystems, the External Profiling Tools (EPT) are iteratively configured in a way to correlate CPU consumption with certain parts of the simulation. The configuration deals with a "Categorization" of CPU loads, by means of defining a set of Load Categories and trying to accumulate relative measurements done by the tool. The procedure is, more or less, intuitive and usually is performed by an experienced Spacecraft Operational Simulator designer/developer. The principal target of this process is to obtain a sense of how the CPU load is distributed inside the overall software, thus offering a starting point for further concentration of specific parts. Fig. 3 depicts some example benchmarking results.

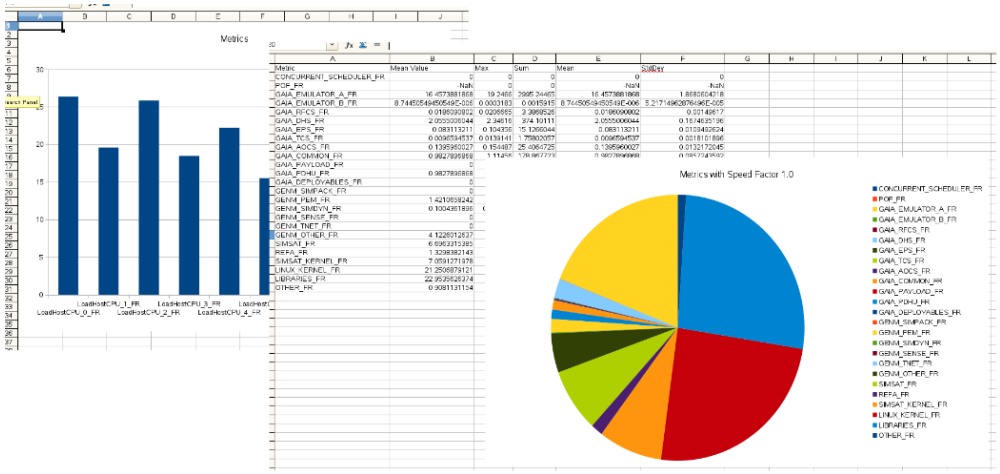


Fig. 3. Benchmarking results for GAIA operational simulator running in IDLE mode

**Parallelization**

The PCOF Parallelization, involves the methodology utilized to parallelize a simulation. It is recognized that in order to effectively parallelize a simulation using the C-PDES engine, it is necessary to investigate the simulator beforehand in order to identify the parallelization possibilities present in the simulator. This leads to the definition of a parallelization methodology that utilizes the PCOF tools in order to produce the required results. The Parallelization approach heavily relies on the Event Reporting System (ERS) tool.

In order to see if we can execute various events in parallel, we analyze the system by gathering information regarding all events and function calls occurred within execution, so we can make suggestions on which events can be parallelized. Basically, we used ERS to generate event and function call graphs regarding the GAIA operational simulator. Fig. 4 demonstrates such a graph. Solid arrows indicate event calls and dotted-line arrows represent event schedules. Within each ellipse-shaped circle that represents an event call, there is statistical information regarding its execution time.

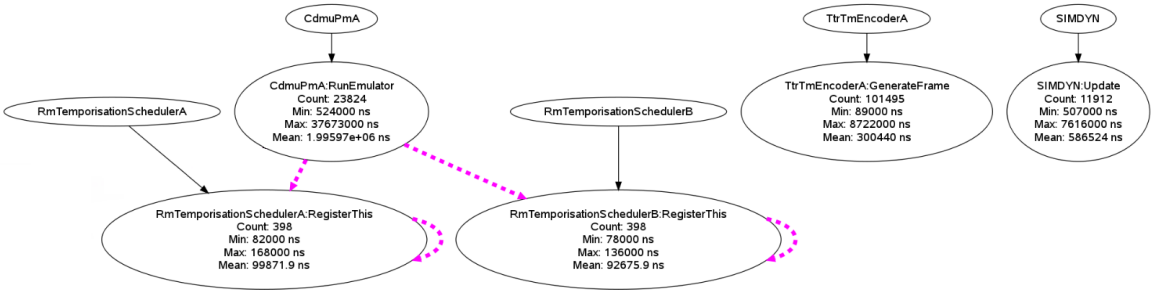


Fig. 4. Event calls graph

It is important to highlight that the current example is used to outline the general process of parallelization. In order to correctly identify a parallelization point, sufficient care must be used. When an event is identified as "independent" it is also important to browse the code in order to investigate data usage from other sources as well as memory management. This is very important in order to ensure that no data corruption or memory leaks occur during event execution. The optimal scenario is that the model code utilizes thread safety mechanisms. Such an approach is considered obligatory in the development of new simulators that intend to utilize the features offered by the PDES engine. After having identified that an event can be parallelized, it is added to the PDES engine via configuration XML files. Specifically, there is the SST.xml which is the Scheduling Scheme Table that is used to define which events shall be executed in parallel. Furthermore, there are two more XML files that define the synchronization policy for the parallelized events. These files are specific to each synchronization policy available.

In an effort to demonstrate parallelization in the GAIA simulator, two distinct steps were used. Firstly, after the required analysis, the GenerateFrame event that is generated by the Telemetry Module, was parallelized. After the GenerateFrame event is set to a different Logical Process the Simulation is ran again according to the previous conditions. As the bit-rate increases, the speed factor clearly drops for the current scheduler while it remains the same for the C-PDES engine. After a certain value, however, the Logical Process of the Telemetry Load offers more load than the first and as a result the speed factor drops sharply and starts following the same pattern as the sequential execution, as shown in Fig. 5 (left half).

The second test concerns the parallelization of several Generic Models (GENM). Specifically the Thermal Network (TNET), the Simulation Dynamics (SIMDYN) and the TM/TC Encoding/Decoding (SIMPACK) have been moved to different LPs. After the GENM models have been parallelized, a significant increase of the Speed Factor can be observed when using the PDES scheduler in Free Running mode, as shown in Fig. 5 (right half).

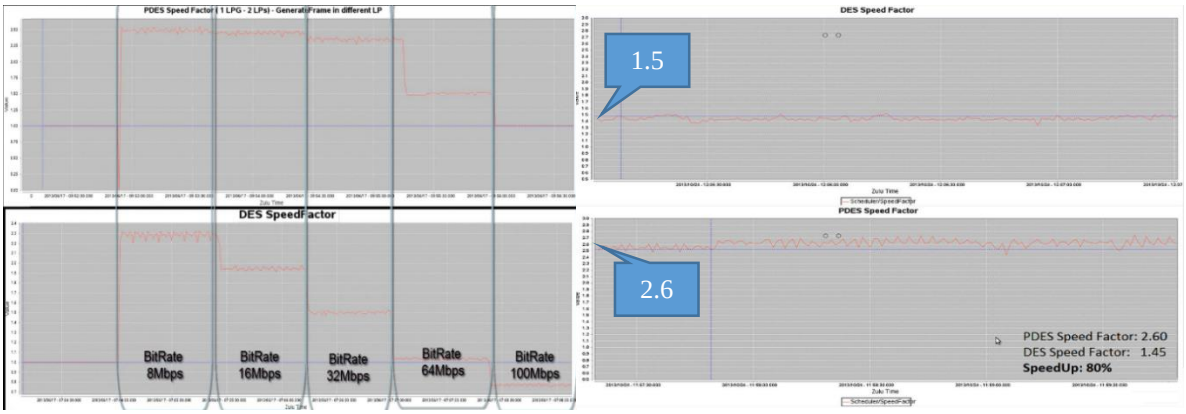


Fig. 5. Parallelization performance gain for GenerateFrame (left) and GENM (right)

**Code optimization**

The Code Optimization approach, is designed to be used independently of the Parallelization approach. It utilizes the PCOF tools, specifically the ERS and the IPM, to help detect possible poor implementation that involves an interactive procedure performed aiming at targeted Code Optimization. In each iteration a certain section of code is examined in order to be optimized. Firstly, the determination of an operational task/scenario is required, the performance metric used for comparisons, along with the PCOF tool suitable to measure the selected/defined metric. During this procedure the

ERS and the IPM tools were used to investigate and resolve a case of code optimization located in one of Gaia's operational simulator models. In general, the operation of IPM and how it is utilized to investigate a possible ambiguity in the system is simple. Basically, after having implemented certain IPM metrics in the model that requires possible investigation a benchmark script is developed and executed in order to get a result. This result represents the behavior of the model before the modification. The next step would be to insert the modification and run the benchmark script again in order to get a second result, which represents the behavior of the model after the modification. Usually, if not always, in code optimization, the improvement involves modifying the source code of the target model, therefore a compilation of the system is required after having applied the modification. After getting the two results comparisons are made in order to resolve to an outcome. Fig. 6 is captured from the graph of the "RunEmulator" event that belongs to the CdmuPmA model, generated by ERS and it represents idle mode execution.

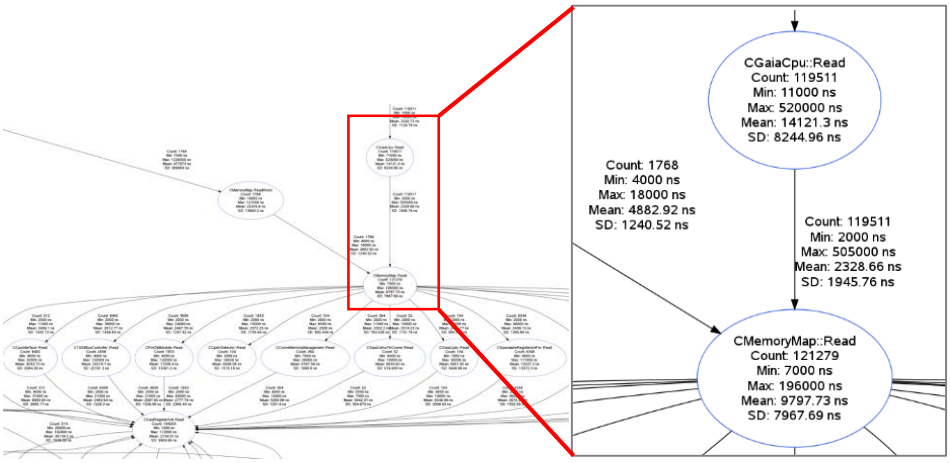

Fig. 6. Function "Read" of the CMemoryMap model

Focusing on function "Read" of model "CMemoryMap" a high value in number of executions can be noticed, indicating a high traffic. Following the path of the caller two models that call this function are shown, "CGaiaCpu" & "CMemoryMap". Taking that into account the location of the IPM metrics be implemented is determined. What's more, model "CGaiaCpu" performs more executions (119511 times) of this function than model "CMemoryMap" (1768 times). Therefore, implementing an IPM metric in "CMemoryMap" model is unnecessary, so only one metric is required to be installed in CGaiaCpu model that will calculate the number of memory reads.

After having examined the source code of operation "Read" inside the CMemoryMap model, a possible poor implementation is located. As determined, every time this function is called it searches for the specific memory range using the address passed as argument in order for the CPU to be able to read from memory using the correct MMIO device, and this is done from the beginning each time. The total execution duration time of this function varies and it depends on the total number of MMIO devices available in the system and how often these devices read from memory. Referring back to the ERS output, we can see that this function is called many times during execution. We modified the code of this function by incorporating a cached variable of memory range that would store the last read memory range so next time the function is called it checks if the memory range is the same. In most cases, the last read memory range is the same, especially when the CPU is reading a buffer from memory, this way we reduce the execution duration of this function.

After having applied the modification, a recompilation is performed and an execution of the test in order to get the results afterward. All results from before and after the modification are shown in Fig. 7. The results depicted represent the rate of execution regarding MMIO memory reads before and after the modification was applied. As shown, an increase in rate value after changing the code is present. Basically, a higher value means that the function was executed more times in a given time, therefore the execution duration of the function was reduced.

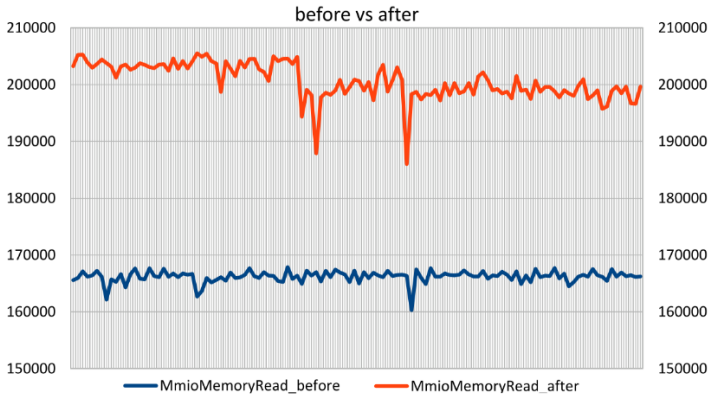

Fig. 7. MmioMemoryRead metric results before and after

## CONCLUSION

All things considered, within the context of the current paper, the utilization of the PDES engine and the PCOF methodologies in the GAIA operational simulator was presented focusing on parallelization and code optimization cases. While the current study was applied on a preliminary release of a real operational simulator, the PCOF was developed with vision to be utilized in parallel with the development cycle of the simulator by providing the development team clear guidelines in order to comply with certain performance requirements. Our vision is focused on using the PCOF and the PDES in a veritable on-going operational simulator project to further prove their capabilities and applicability during the development lifecycle procedure.

As it stands, both the C-PDES engine and the PCOF are considered fully operational. Currently, considerations are being made to incorporate the tools within the SIMULUS infrastructure and to make the C-PDES the Universal Scheduler for SIMSAT. Furthermore, it is desired to utilize the whole PCOF methodology along with the C-PDES engine during the development of a real spacecraft operational simulator in order to formally validate it and fine-tune it. Finally, in order for the tools to be completely supported and integrated within the SIMULUS infrastructure, several additions and modifications in the SMP2 standard are investigated. The latter shall provide appropriate tools to develop thread safe simulation models with parallelization in mind.

## REFERENCES

[1] V. Reggestad, D. Guerrucci, P.P. Emanuelli, D. Verrier (2004). “Simulator Development: the flexible approach applied to Operational Spacecraft Simulators”, *SpaceOps*, May 2004.

[2] Nuno Sebastião (OPS-GIC), Vemund Reggestad (OPS-GDA), David Verrier (OPS-GDS), “ESOC Simulators: Past, Present and Future”, *OPS-G Forum*, 15 Jan 2008.

[3] ESA/ESOC EGOS-SIM-REFA-SUM-1002, “Operational Simulator Development Guide”.

[4] ESA/ESOC EGOS-SIM-REFA-SUM-1003, “General Simulator Developers Guidelines”.

[5] ESA/ESTEC, ECSS-ETM-40-07, “Simulation Modeling Platform”, 2008-07.

[6] V. Reggestad, Nikolaos-A.I.Livanos, Pantelis Antoniou, and Ioannis E. Venetis, “Operational Simulators going parallel: From a dream to a concrete concept”, *In Proc. of the 2011 DASIA*, San Anton, Malta, May 2011.

[7] V. Reggestad, Nikolaos-A.I.Livanos, Pantelis Antoniou, Elias Zois, “Introducing Parallelization & Performance Optimization in SIMULUS Based Operational Simulators”, SimTools2012, March 2012

[8] Nikolaos-A.I.Livanos, I. Angelis, V. Alifragkis, S. Hammal, A. Walsh, V. Reggestad, M. Pantoquilho, “Performance Optimization Framework for Spacecraft Operational Simulators”, *European Ground System Architecture Workshop 2013*, 18-19 June 2013.

[9] V. Reggestad, M. Pantoquilho, D. Werner, Nikolaos-A.I.Livanos, Pantelis Antoniou, “Increasing Performance of ESA Operational Spacecraft Simulators”, *SESP 2012, ESTEC – Noordwijk*, 25-27 September 2012

[10] T. J. Schriber and D. T. Brunner, Inside Discrete Event Simulation Software: How it works and why it matters. In Proceedings of the Winter Simulation Conference, 2008.

[11] DTOS-SST-TN-0696-TOS-GIC, “Concurrent Scheduler Design Technical Note”.

[12] DTOS-SST-TN-0698-TOS-GIC, Application of a Prototype Concurrent Scheduler Technical Note”.

[13] Max Pignède, Jose Morales, Peter Fritzen, John Lewis, “Swarm Constellation Simulator”, *SpaceOps Conference*, April 2010.