Increasing Performance of ESA Operational Spacecraft Simulators

Vemund Reggestad[1], Marta Pantoquilho[2], Daniel Werner[3]

Nikolaos-A.I.Livanos [4], Pantelis Antoniou [5]

[1, 2, 3] ESA/ESOC, Mission Data Systems Division, Robert Bosch Strasse 5, Darmstadt, Germany

Email: [1] *Vemund.Reggestad@esa.in*, [2] *Marta.Pantoquilho@esa.int*, [3] *Daniel.Werner@esa.int*

[4,5] EMTech, Vizantiou 58, Papagou, Athens, Greece

Email: [4] *nikolaos.livanos@emtech.gr*, [5] *pantelis.antoniou@emtech.gr*

ABSTRACT

An Operational Spacecraft Simulator, hereafter referred to simply as Simulator, is a complex software system consisting of high fidelity models of the real Spacecraft (S/C) and its ground segment interfaces. Its main objective is to provide a representative simulation of the S/C behaviour, which is achieved by having an emulator of the on-board processor that runs the actual on-board software. At ESA a new Simulator is developed for each mission making extensive reuse of the ESA generic simulator infrastructure software called SIMULUS.

Due to its complexity, a Simulator normally has several hundreds of technical requirements that must be satisfied. Among these, usually performance requirements take a secondary role during the development phase and are normally only taken into account if the simulator's performance becomes an issue. By this time, however, it is often too expensive to address the problem as there are several design decisions that were made, which bring other benefits, but are detrimental on performance.

In order to detect and understand performance bottlenecks early in the development of Simulators and improve the performance, the Mission Data Systems Division at the European Space Operations Centre (ESOC) carried out a project with the main objective to integrate into SIMULUS a set of Performance Indicator Tools (PIT). The PIT are developed as plug-ins which interface to the components of the SIMULUS infrastructure and support real-time measurement and evaluation of sub-systems performance.

Since the Simulator development is carried out based on incremental (or agile-like) deliveries, the availability of appropriate tools for performance analysis makes it possible to detect performance issues early during the development. This brings significant advantages in terms of project cost and schedule.

This paper presents and analyses the performance evaluation results gathered with the PIT of existing Simulators from the Cryosat2, Hershel and Planck and GAIA missions. It describes the design and usage of the PIT and its advantages and expected benefits on the development of new Simulators based on the SIMULUS 5 infrastructure software.

INTRODUCTION

Real-time execution of a simulation is the main concern and constraint. However, there are specific cases where multiple times faster than real time execution could be very valuable. For instance, flight dynamics simulations could take advantage of very fast execution, as the operational scenarios and the associated S/C manoeuvres under evaluation and validation often require several days of real-time simulation. On the other hand, future missions expect more accurate simulation models, more complicated subsystems, as well as the requirement to emulate other parts of the Data Handling System, such as hot redundant processors, co-processors, etc. As a result, the current performance of the simulation infrastructure, specially the execution of kernel modules used to run the S/Cs' models, may be inadequate without further improvements and updates.

Furthermore, there is a strong incentive to re-use code among simulators, for common subsystems (e.g. radio frequency systems, star trackers, gyros, antennas), in order to reduce cost and implementation time. This means that unoptimized code may have a life span of several missions, thus increasing the need to produce highly performing code or to optimize reused or reusable code for the benefit of current and future missions.

The SIMULUS Infrastructure

Satellite simulators at ESA/ESOC are built-up on top of the SIMULUS infrastructure, [1], [2]. SIMULUS is composed by a run-time framework (SIMSAT), software emulators, a set of reusable generic models, and a set of Ground Station equipment models. Recent Simulator developments are also based on a Reference Architecture (REFA), [3],[4], which has been introduced in order to identify, define, develop and integrate a reference operational S/C simulator architecture. It allows achieving shorter, therefore more costing efficient, simulator development cycles by means of heavy reuse while producing higher quality outputs.

REFA and all simulators based on REFA are compliant with the SMP2 standard, [5]. 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.

A key module inside an operational simulator is the On-Board Software (OBSW) emulation, [6]. The concept is to utilize the real mission's OBSW in the core simulation while other parts of the S/C are being modelled in a functional manner. This enhances the overall realism of the simulation, as well as enables further verification and validation of the OBSW. The execution of the real mission's OBSW is supported by a processor software emulator. The emulation mechanism is the most computational and intensive one within the simulation environment.

All simulation models, including the emulator, are executed by the SIMSAT simulation environment. SIMSAT is composed of several modules providing all kernel functionalities required to support run-time simulation. The kernel includes a scheduling and execution engine that is based on a typical Discrete Event Simulation concept [10]. Events are generated inside the execution of simulation models and are stamped with specific simulation time. Events are linked with function call-backs that

correspond to simulation models' functionalities. In turn, these may schedule new events and so on. The simulation's kernel scheduler serializes all event execution and keeps the timing of the overall execution using a global clock mechanism.

Targeting Performance Optimization

When considering performance optimization for SIMULUS infrastructure there are three major directions to consider: parallelization, code optimization, and emulator optimization & enhancement.

In order to achieve parallelization the SIMSAT simulation engine could be extended to support parallel execution of events, i.e. be enhanced with a Parallel Discrete Event Simulation (PDES) scheduling and execution engine, as described in [11] and [15]. Consequently, the models interconnections could be explored in order to analyse the induced events and the related function call-back, and focus on the possibility to separate independent functionalities or even define the degree of independency between them.

The heart of every operational simulator is the emulation of the On Board Software (OBSW). This is a major issue when trying to parallelize the overall execution and becomes especially important when there are several processors to be emulated. Any kind of optimization related to speeding-up the emulation engine is expected to greatly improve overall performance.

Last, but not least, performance optimization may be targeted directly to specific model optimization by means of introducing code modifications, rearrangements and code optimization. However, implementations of simulators include several modules and models and the overall size of the code is considered rather large. Obviously, the availability of tools and utilities able to analyse performance and direct developers and assembly integrators to specific models which degrade performance is essential.

Performance optimization of SIMULUS based simulators is, therefore, a multi-path procedure, requiring the simulation infrastructure to provide the fertile ground and the tools to cultivate performance.

In this paper we will focus on some of those tools, the Performance Indicator Tools (PIT) and the respective code optimization methodology, which can guide the investigator towards the performance profiling of the overall software. For more information on the Emulator performance enhancement and respective technologies, as well as the implementation of a PDES in SIMULUS, please check the reference section of this paper, namely [14] and [15].

PERFORMANCE INDICATOR TOOLS (PIT)

The Performance Indicator Tools (PIT), included in the SIMSAT infrastructure, [15], support real-time measurement and evaluation of the performance of sub-systems and parts of the overall software arrangement. The PIT are designed and implemented as a separate infrastructure library providing the opportunity to build a performance perspective of a general mission simulation project in future uses. The PIT are implemented as a shared library object and is interfaced inside the tree in the SIMSAT environment through the configuration of an XML file. The object includes variables and member functions that are interfaced with the SIMSAT kernel using the CORBA interface.

The PIT are subdivided into four benchmarking categories (SPM, IPM, EPT, ERS), and a presentation tool (OOML), explained below.

PIT-ERS – Event Reporting System

The PIT-ERS is a tool targeting to acquisition of information related with events and function calls generated in SIMSAT kernel by execution of SMP2 simulation models. The tool tracks all the function calls related with the activated event and generates an analytical call graph of the simulation system. This allows an exhaustive investigation and analysis of the generated graphs related with events and function calls which can be used to manually explore parallelization.

PIT-ERS output files are compatible with the GraphViz [12] COTS, in order to provide an easy and user-friendly method to visualize the graphs.

PIT-EPT – External Profiling Tools

The PIT-EPT is responsible for measuring several quantities regarding the overall SIMULUS infrastructure using 3rd Party External Profiling Tools, such as the O-profile, [13], and the Linux kernel. Principal operation of PIT-EPT is to provide a fast and easy way to perform CPU profiling of all simulation infrastructure components.

PIT-EPT processes the generated statistical information and performs accumulations according to user defined categories. The categorized results are continuously updated inside the PIT-ERS tree structure in the SIMSAT environment, and can also be stored in a logging file for post processing.

PIT-IPM – Internal Performance Metrics

The principal purpose of PIT-IPM is to benchmark performance of subsystems, such as REFA, the Generic Models, or the specific S/C models, by measuring the respective total throughput in specific subsystem utilization of the S/C. For instance, the number of Tele-commands and Telemetry packets per second, OBSW memory inputs & outputs, Bus transactions, etc. are some examples of quantities that may be used as metrics for performance comparisons. A typical utilization case of the PIT-IPM is to develop stress tests of S/C subsystems or even as-fast-as-possible execution, in order to measure the achieved performance by evaluating and making comparisons among the selected metrics.

PIT-SPM - SIMSAT Performance Metrics

The PIT-SPM tool includes performance metrics and functionalities related to SIMSAT environment (i.e. Scheduler, kernel, object access, script code execution, etc.). The tool includes reporting functionality in order to log the acquired measurements in file, including time reference and tagging messages. The measurements are separated into three main categories: rates, counters and variables.

PIT-OOML - Open Office Macro Library

The PIT - Open Office Macro Library (PIT-OOML) has been developed in order to ease the development of presentations dealing with information acquired by the PIT tools. The library has been developed in StarBasic language that is supported by the Open Office. The end-user is not required to be an expert StarBasic developer, but he must be familiar with basic programming concepts.

APPROACH TO CODE PERFORMANCE OPTIMIZATIONS

The PIT target the examination of specific code (mission specific or infrastructure) parts that perform inefficiently and direct integrators to focus on specific inefficiencies and performance issues. The first step of this approach requires executing the simulation while the PIT-EPT tool is activated. This allows the user to determine which parts of the simulation cause the larger load to the CPU. Notice that this step should be performed repeatedly in order to fine tune the categories in which the PIT-EPT tool distributes the load, i.e. the user must make sure the correct libraries have been assigned to the correct EPT categories.

After determining the system where poor implementation might exist, a benchmark must be defined that will stress-test it. The benchmark will also have to utilize either the PIT-EPT tool or the PIT-ERS tool. By choosing the PIT-EPT, one must redefine its categories in order for them to be specific to the system and be able to analyse each model it contains separately. This allows the user to confine the poor implementation to a specific model. On the other hand, one can use the PIT-ERS to get exact measurements for a specific model. This allows the problem to be narrowed to a specific function due to the detailed statistical output provided in the graphs generated from the PIT-ERS.

The next step of the procedure is executing the benchmark and evaluating the results. Depending on which PIT tool is used in the benchmark, the results will differ and one must act on them in a different manner. Specifically, by using the PIT-EPT tool, no code optimization can be performed but the resulting report will show more specifically in which model(s) the overhead occurs, thus allowing the effective use of the PIT-ERS tool. When the latter is used in the benchmark, a graph of the model(s) in question is expected. Through this graph, one can determine specific functions where overhead occurs and attempt to optimize their implementation in order to improve the overall performance of the simulation.

Finally, after performing a code improvement, the whole procedure should be repeated in order to determine its effect on the overall performance and locate other possible poor implementations. For optimum results, it is expected that the whole process is executed several times. Detailed information on the configuration of the tools is described in [15].

The applicability of the PIT can be introduced in the design and development procedures of simulators in two different stages:

- During Development involving operational simulators that are to be designed and developed according to performance optimization requirements (e.g. future operational simulators).
- During Delivery involving operational simulators that are already developed and performance optimization is required. This can also be done for new simulators during the validation at customer premises.

Fig. 1, extracted from [15], demonstrates a flowchart that describes the overall process of the Code Optimization approach.

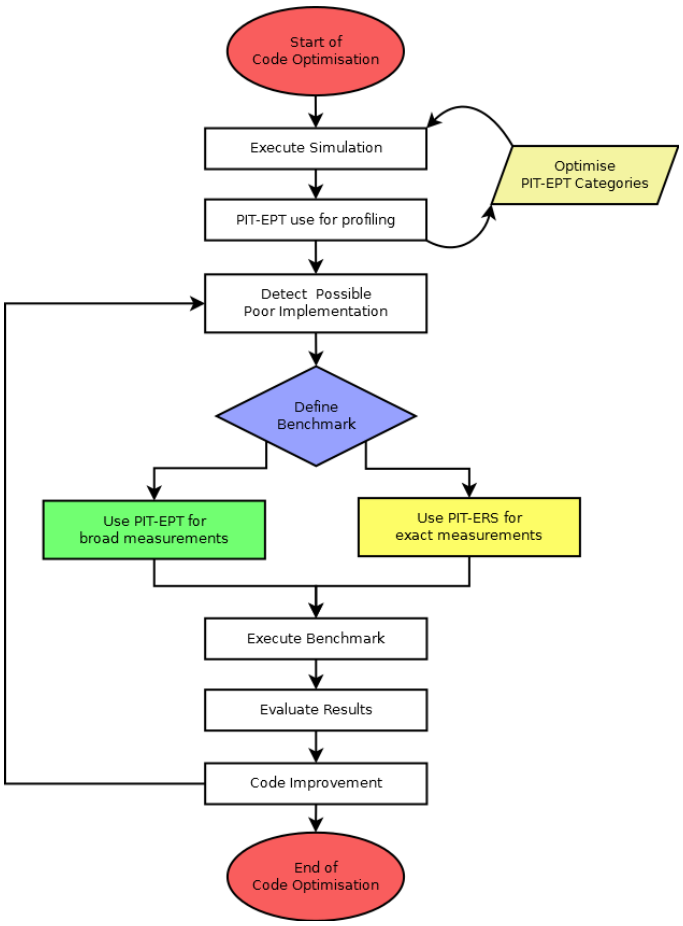

Fig. 1: PIT code optimization data flow [15]

OPTIMIZATION ANALYSIS: SOME RESULTS

A thorough CPU profiling of three different operational simulators (CRYOSAT2, AEOLUS and HERSCHEL/PLANCK) was performed using a prototype version of the PIT, [9]. These results were used as proof of concept and motivation for the development of the current PIT suite and its integration into the SIMULUS package, [15]. With the current version of the tools, the GAIA simulator was analysed and the preliminary results are presented below.

CRYOSAT2, AEOLUS and HERSCHEL/PLANK Performance Analysis Results

The profiling analysis of these three simulators showed that the biggest performance problem, as expected, is the emulator core. From the three of them, Cryosat2 is the one that uses most of the CPU time on the emulator core - with 82% CPU time. Aeolus spent 61% of the total time in the emulator core function, while Herschel and Planck simulator showed an unusual lighter CPU load even with its two different CPU cores emulated simultaneously [7], [8], which can be justified by the disabling of the on-board software memory scrubbing functions on the emulator.

With the Cryosat2 analysis it was observed a high occurrence of function call invocations of non-local functions (located in other shared objects) and a high occurrence of string compare functions, which is an indication of non-optimal implementation looking up a named data structure.

In Aeolus, after the emulator core, is the thermal network simulation component which takes most CPU time since its implementation is $O(n^2)$. In addition it was observed that the thermal model handling is using considerable resources, and that there are considerable string and memory copy functions, used for copying big amounts of data through the layers of the simulation.

On Herschel and Planck simulator an interesting observation was that on a low enough load the SIMULUS infrastructure functions and the TTC streams activity is quite high. The kernel profile is quite uneventful, with most of the time spent in the idle loop and in the Linux kernel scheduler calls, indicating high frequency of context switching. There were also detected evidences of communication using TCP/IP sockets, which can be improved by using a more efficient local CORBA transport.

These initial observations have been further expanded using the PIT and results are available in [8].

GAIA Performance Analysis Results

The preliminary analysis of the GAIA simulator showed that in the default generic configuration, the maximum achievable speed-factor was 3 in free-running mode (running in an Intel Core2Quad@2400Mhz), although GAIA simulator can reach the speed factor of 5 times real-time in a specific configuration used for Flight Dynamics. The reason for the low speed factor is that the software cannot take advantage of more than one core of the CPU, since currently the SIMSAT kernel is based on a Discrete Event Simulation engine and the total setup is tightly coupled with the emulator circular event. As observed in other simulators, the emulator consumes most of the CPU time.

Fig. 2 below depicts the load percentage of the used CPU for the whole GAIA system (includin OS and SIMULUS) while running at the speed factor of 1 (real time). Fig. 3 below depicts the load percentage of the used CPU for several parts of the GAIA simulator. As expected the part that consumes most of the CPU is the Data Handling System part. However, it is noted that this state corresponds to an IDLE state by means of not really running a considerable load for each sub-component of the spacecraft.

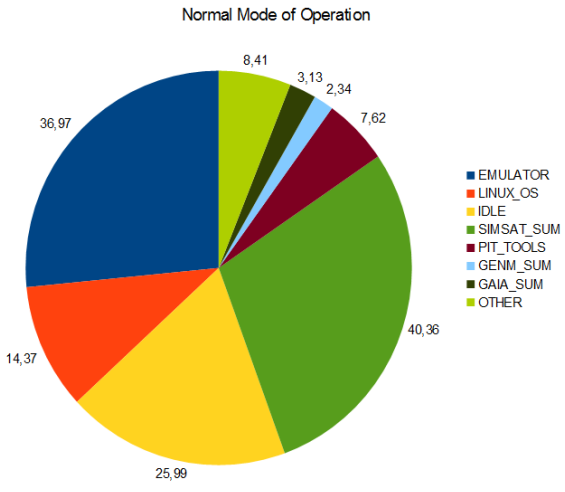

Fig. 2: Overall System Load

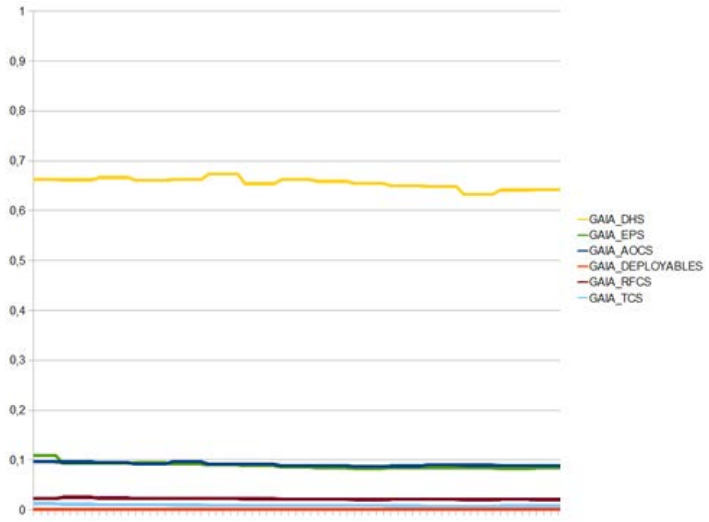

Fig. 3: GAIA Simulation Load

CONCLUSION & FUTURE WORK

This paper presented the Performance Indicator Tools (PIT) and the process for its applicability in the improvement of the code quality and performance of SIMULUS based operational spacecraft simulators. Concrete examples and results, based on a prototype version of the PIT, have been presented in the case of several simulators which support these statements. In addition, the operational version of the PIT was used to retrieve the preliminary findings of the GAIA simulator performance analysis. Since the GAIA simulator is still under development, it could be the first simulator to test the operational version of the PIT and to be able to benefit from the performance optimizations that will be suggested by the application of the tools.

Future missions, which will still start the development of its simulators, will be able to fully benefit from these tools and apply them early during their development phases. These tools will be provided as part of SIMULUS and thus will be made available to all developers and customers using SIMULUS in their products. It is our strong belief that the PIT will be able to significantly improve the performance and code optimization of newly developed SIMULUS based simulators.

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] Ian McGregor, “The Relationship Between Simulation and Emulation”, Proceedings of the 2002 Winter Simulation Conference.

[7] ESA/ESOC DOPS-STU-TN-1002-OPS-GD, “TN1: Simulators Designer Guide for Performance Issues”, 2011

[8] ESA/ESOC DOPS-STU-TN-1003-OPS-GD, “TN2: The Multi Core and Performance Project Report”, 2011.

[9] ESA/ESOC DTOS-SST-TN-0698-TOS-GIC, “Application of a Prototype Concurrent Scheduler Technical Note”, 2003.

[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] R. M. Fujimoto, “Parallel Discrete Event Simulation”, Communications of the ACM, Vol. 33 No. 10.

[12] Graphviz, http://www.graphviz.org/

[13] Oprofile, http://oprofile.sourceforge.net/about/

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

[15] ESA/ESOC DTOS-ESSE-SRS-1001-HSO-GIC, “ESSE - Enhancement of SIMSAT Simulation Engine: Software Requirements Specifications”, 2012-04.