# SIMULATION FOR MULTIPROCESSOR REAL-TIME SCHEDULING EVALUATION

**Richard Urunuela, Anne-Marie Déplanche, Yvon Trinquet**

IRCCyN, UMR CNRS 6597
University of Nantes
1 rue de la Noë, BP 92101, 44321 Nantes cedex 3, France

{*richard.urunuela—anne-marie.deplanche—yvon.trinquet*}*@irccyn.ec-nantes.fr()*

## Abstract

The increasing complexity of the hardware multiprocessor architectures as well as of the real-time applications they support makes very difficult even impossible to apply the theoretical real-time multiprocessor scheduling results currently available. Thus, so as to be able to evaluate and compare real-time multiprocessor scheduling strategies on their schedulability performance as well as energy efficiency, we have preferred a simulation approach and are developing an open and flexible multiprocessor scheduling simulation and evaluation platform called STORM (”Simulation TOol for Real-time Multiprocessor scheduling”). This paper presents the simulator on which STORM relies and that is able to simulate accurately the behaviour of those (hardware and software) elements that act upon the performances of such systems. An example is given to illustrate such a performance evaluation.This work has been supported by the french Agence Nationale de la Recherche through the PHERMA project (Contract ANR ANR-06-ARFU06-003). See http://pherma.irccyn.ec-nantes.fr.

**Keywords: Real-time systems, Multiprocessor global scheduling, Simulation**

## Presenting Author’s Biography

Urunuela Richard is a research engineer of the IRCCyN laboratory in the ”Real-Time Systems” group. Since 2003 he is interested in the design of real-time systems and more particularly: operating systems, power management for such systems, and real-time scheduling. At present, he is in charge of the implementation of STORM, a Simulation TOol for Real time Multiprocessor scheduling. He is also focusing on the development of engineering tools for helping to the design of power management policies. Previously he worked around scheduling and power management in the OBASCO research group at the Ecole des Mines of Nantes.

## 1 Introduction

Multiprocessor architectures are becoming more and more attractive. From single-core designs with heat and thermal problems, chip makers are now shifting to multicore technologies. While making them decreasingly expensive, they are making very powerful platforms available. Such platforms are desirable for embedded system applications with high computational workloads such as robot controls, image processing, or streaming audio. The additional characteristic of such applications is to meet some real-time constraints; consequently they require predictable performance guarantees from the underlying operating system. This has led to renewed interest in multiprocessor real-time scheduling: scheduling algorithms together with schedulability analysis. So many theoretical results are available but mainly based on simple hardware and software architecture models [1].

These results are difficult to exploit if the assumptions are not satisfied which is the case for modern hardware and software architectures (processor heterogeneity, cache management, inter-processor communications, complex software behaviours, system overheads, etc.). Furthermore, due to the specific application fields where energy efficiency is critical (battery-based embedded systems), power management techniques (Dynamic Voltage and Frequency Scaling or Dynamic Power Management) are inescapable and must be taken into account.

Indeed, in such a context, it is very difficult to evaluate and compare scheduling algorithms on their schedulability performance as well as energy efficiency. So we think that a more global approach is needed in order to explore the adequacy between scheduling policies and (hardware and software) architectures. It is not for us a matter of implementing scheduling algorithms on true multiprocessor platform(s) but rather of designing an open and flexible framework able to simulate accurately the behaviour of those (hardware and software) elements that act upon the performances of such systems. The tool we are working on is called STORM which stands for ”Simulation TOol for Real-time Multiprocessor scheduling” [2]. This multiprocessor scheduling simulation and evaluation platform is intended to: i) use as input the specifications of the hardware and software architectures together with the scheduling policy; ii) simulate the system behaviour using all the characteristics (task execution time, processor functioning conditions, etc.) in order to obtain the chronological track of all the ”events” that occurred at run-time; iii) compute various real-time metrics in order to analyse the system behaviour and performances from various point of views. For the time being, engineering efforts on STORM have concerned mainly its simulator component since it is the retaining element of the platform. For a given ”problem” i.e. a software application that has to run on a (multiprocessor) hardware architecture, this simulator is able to ”play its execution” over a specified time interval while taking into account the requirements of tasks, the characteristics and functioning conditions of hardware components and the scheduling rules. The original need to develop STORM came from the works of the PHERMA research project (PHERMA - ”Parallel Heterogeneous Energy efficient Real-time Multiprocessor Architecture” - is an ANR (French National Research Agency) project (see http://pherma.irccyn.ec-nantes.fr). This project addresses multiprocessor architectures for which one needs to evaluate the performances in term of energy efficiency for specific hardware and software architectures.

## 2 Motivation of our work

In comparison with the impressive number of publications and results on real-time scheduling since more than 30 years, few tools are available to apply them. On the one hand, some commercial tools exist for the complete design and analysis of real-time systems. Obviously they are not freely available, as examples: RapidRMA$^{TM}$ (from Tri-Pacific Software Inc.) or TimeWiz (from TimeSys Corp.) On the other hand, open source or free products are available coming from the academic community. Some of them are the result of PhD studies that do not overstep the prototype state without maintenance support. As an example we can cite [3] or XRMA [4]. Others are more substantial but do not offer enough flexibility so as to be extended or adapted for new requirements. Some examples are: YASA (”Yet Another Scheduling Analyser”) [5], MAST (”Modeling and Analysis Suite for real-Time applications”) [6, 7], Cheddar [8, 9], or TORSCHE [10, 11]. That is why we decided to implement a new tool with the following main design specifications:

- it addresses specifically multiprocessor architectures (composed of homogeneous or heterogeneous processors); monoprocessor architectures being the simplest case;
- it has to be able to take into account: i) the features of hardware architecture: multicore design, multiprocessor architecture with shared memory, distributed architecture with communication network, memory architecture (L1 and L2 caches, banked memory); ii) the features concerning energy consumption for the processors and memories, in particular the capabilities for DPM (Dynamic Power Management) and DVFS (Dynamic Voltage and Frequency Scaling);
- it is a free, flexible, portable and open tool: i) ”free” because the STORM software is freeware under Creative Commons License (subsequently it will be available as an open source); ii) ”flexible” means the possibility to program and to add easily simulation entities (such as scheduling policies) through well-defined APIs; iii) ”portable” means the possibility to run it on various OS thanks to the Java programming language; iiii) ”open” means that the input/output data are formatted using xml;
- lastly, it is intended to provide a simulation language to drive experiments (statistical studies or

domain explorations) via a simulation controller. Upstream from the simulator, this controller computes the inputs, runs the simulations, and downstream computes the required metrics from simulation results.

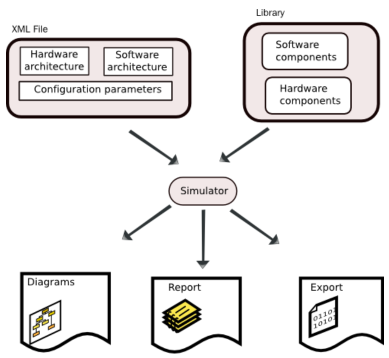

Fig. 1 The STORM simulator.

## 3 The STORM simulator

As shown by Figure 1, the STORM simulator needs the specification of the studied ”problem” as input. By problem, we means a software real-time application, i.e. a set of tasks with execution requirements, that has to run on a multiprocessor hardware architecture. It is described in a XML input file in which specific tags and attributes have to be used, and where references to predefined components (available in attached Java class libraries) are made. Thus, for such a given problem, the simulator is able to ”play its execution” over a specified time interval while taking into account the requirements of tasks, the characteristics and functioning conditions of hardware components and the scheduling rules, and with the highest timing faithfulness. As a result, a lot of simulation outputs can be computed: either user readable in the form of diagrams or reports, or machine readable intended for a subsequent analysis tool. The user’s interactions with STORM go through a user-friendly graphical interface which is composed of command and display windows. The figures given after in the Example section are screen-shots of this GUI.

We want to point out that the simulation is a discrete time one, i.e. the overall simulation interval is cut into a sequence of unitary slots [0,1], [1,2],..., [t,t+1], etc., and simulation moves forward at each instant 0, 1, ...t, t+1, etc. Thus, at t, the next simulation state (for instant t+1) is computed from the current one (of instant t) and the pending simulation events at time t. Moreover only the control behaviour of the software application elements is captured by the simulation. It means that no applicative programs run for the tasks, nor true data exchanges between tasks exist, but only task execution times and synchronizations are taken into account while simulating.

In this paper, we chose to overview the internal architecture of the simulator and to present an example so as to illustrate the kind of performance evaluation that can be conducted using STORM. Together with the STORM software (release 3.0), examples and (user and designer) guides about STORM are freely available at http://storm.rts-software.org.

### 3.1 The functional architecture

The internal architecture of the STORM simulator is composed of a set of entities built around a simulation kernel (see Figure 2).

Software (respectively hardware) entities stand for the tasks and data (respectively processors) that compose the software (respectively hardware) architecture for which the simulation is conducted. There are as many task, data link and processor entities as specified in the XML input file, and they are automatically instantiated from the library components. Those libraries provide a large panel of task behaviours: recurrent, periodic or aperiodic, with or without activation recording, with or without deadline abort. It’s also the case for the data links that enable quite complex synchronization relations between tasks. For the time, two processor types are available: a basic one and another with functioning modes and DVFS capabilities.

The system entities represent some run-time and operating components that are present in real-time multiprocessor platforms.They are the task list manager, the scheduler and the memory manager. The task manager manages task activations and synchronizations in accordance with the specifications of the software architecture. Thus, at each simulation slot, depending on the current state, it may interact with the simulation kernel so as to release or block some task(s). The scheduler entity is in charge of allocating the processor(s) to the ready tasks depending on the scheduling strategy it implements. Its type is specified in the XML input file and can be chosen among the numerous real-time global event-driven and time-driven multiprocessor scheduling algorithms that are available in library. Moreover, thanks to the flexibility of STORM, there is no limitation for introducing new scheduling policies as it will be shown in the final paper. Up to now, the memory manager has not be developed. Its part should be to carry out the behaviour of the elements of the physical memory architecture (memory access bus, memory bank) that may impact the execution with timing and energy aspects.

The simulation kernel is at the heart of STORM and has been thought as simplest as possible. In consequence, it provides a very few basic services. Due to the discrete approach for the simulation process, one of them is a time service so as to manage simulation time and to indicate the spending of time to the simulation entities. Another kernel service is a watchdog service. It enables any entity to ask the kernel to arm a watchdog mechanism so that the specified function of the specified entity will be called when the specified delay expires, as illustrated on Figure 3. Lastly, the kernel offers a kernel message service. For some state change or simulation event occurrence, an entity has to inform the kernel by asking it to register a pre-identified kernel message.

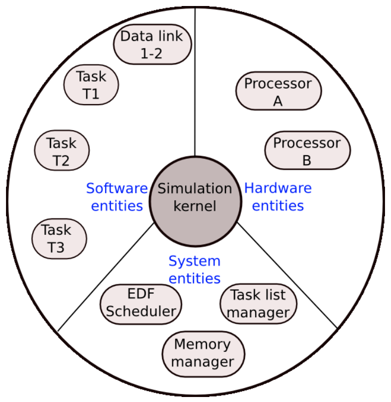

Fig. 2 The simulator architecture.

The kernel is then in charge of processing such a message. Depending of the context, it may lead it to send or not some other information to some other entities. For example (see Figure 4), each time one job of the (periodic) task T1 completes, its STORM entity informs the kernel about its completion (thanks to the well identified BLOCK message). Obviously such an indication has to be known to the scheduler entity. Moreover since task T1 produces some data, the kernel has to inform as well the data link entity.

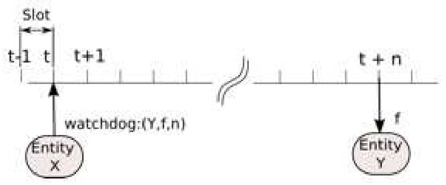

Fig. 3 The watchdog service.

In summary, the kernel acts as a ”conductor”: at each slot, it processes all the pending kernel messages the simulation has produced up to now and in accordance with the rules it implements, it switches them towards the appropriate simulation entities. Of course those interactions between entities that do not require the control of the kernel are allowed to take place in a direct way.

### 3.2 The software architecture

The STORM simulator is written in Java programming language which makes it independent of any execution platform. Figure 5 gives a simplified view of the current UML class diagram of STORM

In accordance with Figure 5, we can find there, on the one side, the $SimulationKernel$ final class that implements the kernel of the simulator, and on the other side, a set of classes ($Task$, $Data$, $Processor$, $Scheduler$, $TaskListManager$, etc.) and their subclasses that model the various simulation entities (some of them correspond to the value that has to be given to the className attributes in the input xml file).

All of the subclasses inherit from the superclass

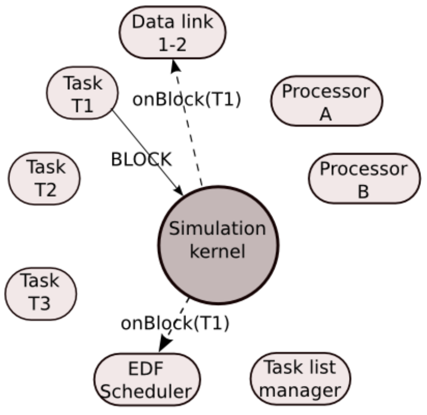

Fig. 4 The kernel message service.

$Entity$. It contains the declaration of the methods involved in the implementation of the different services. Depending on the subclasses and the behaviour they must exhibit, those methods may need to be overrided. If the paper is accepted this part can be extended or not depending on the reviewers’s remarks.

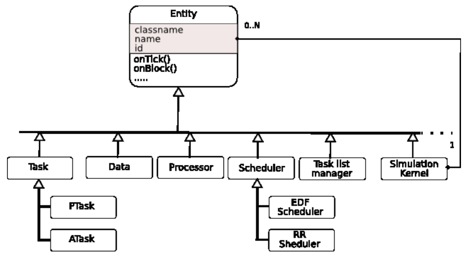

Fig. 5 The UML class diagram.

## 4 Example

In this section we show the use of STORM for the in-depth analysis of a problem. The problem is complex enough because it is about finding the solution that gives the best result (best throughput and best QoS) while consuming the minimum of energy. The example is inspired of a more complex problem: an H264 decoder (image coding / decoding).

### 4.1 The problem

The experiment we have conducted is concerned with the software architecture described in the XML input file and depicted in Figure 6 (this task graph has been gotten from the Graphical User Interface of STORM). It is composed of 8 tasks (referenced ”Task A” to ”Task H”) and 10 data links (shown as top-down transitions between tasks). It is a data-flow graph with precedence constraints. The periodic task ”Task A” generates periodically (every 20ms) some data on which subsequently the tasks ”Task B” to ”Task E” work in parallel. Their outputs (by pair) are required as inputs for the execution of the tasks ”Task F” and ”Task G”. Finally, from the joined outputs of these two previous tasks, ”Task

H" computes some transformation so as to produce the result in the end. This last task is a periodic task because one can thus associate it a deadline that will serve for the measure of QoS (count of missed deadlines). Some parameters of the tasks are given in the Table 1. The tasks "Task A" and "Task H" are periodic (20 ms) and for the experimentations they will be successively of type "PTask_WAM" (With Activation Memory) or type "PTask_NAM" (No Activation Memory). The other tasks are recurrent ones (type "RTask"). The chosen rule for computing the actual execution time of task jobs (specified in the XML input file too) is an uniform random law between the BCET and WCET values for all the tasks. The scheduling algorithm is of type "FP_P_Scheduler" (Fixed Priority full Preemptive) and all the tasks have the same static priority. All the processors are identical and of type "CT11MPCore".

Our intent is to evaluate what is the minimum required number of processors of a multiprocessor architecture so as to maximise the output flow (i.e. the number of "Task H"'s outputs relatively to the number of "Task A"'s activations), while minimising its jitter (i.e. the variation on the dates where "Task H"'s outputs are produced) together with minimising the electrical power consumption. Such a software architecture as well as our performance evaluation objective are complex enough to prefer a simulation study rather than a (necessary) approximate theoretical one.

Tab. 1 Task parameters

| Name task | BCET (ms) | WCET (ms) | Task type |
|---|---|---|---|
| Task A | 1 | 3 | PTask... |
| Task B | 2 | 4 | RTask |
| Task C | 2 | 4 | RTask |
| Task D | 2 | 6 | RTask |
| Task E | 2 | 6 | RTask |
| Task F | 2 | 3 | RTask |
| Task G | 2 | 3 | RTask |
| Task H | 4 | 6 | PTask... |

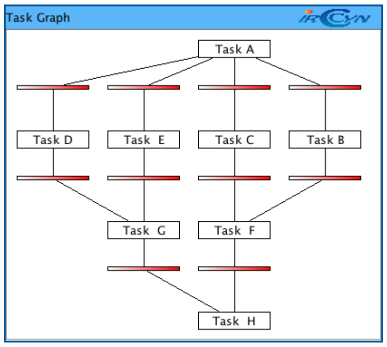

Fig. 6 The task graph of the software architecture.

## 4.2 The experiment and results

So as to determine the minimum number of processors, simulations have been successively conducted with an increasing value from 1 up to 5 processors (the performance metrics do not change beyond 4 processors). So as to conduct a statistical evaluation, 100 simulations (with a total duration equal to 1 second for each) have been achieved. During these simulations all the periodic tasks were of "PTask_WAM type". Then another set of 100 simulations have been conducted, all the periodic tasks being of "PTask_NAM type". In order to have a measure for the QoS (number of missed deadline of "Task H" ) the deadline of this task has been fixed to 25 ms, so 5 ms after its period. The measure for the throughput is the number of executions of "Task H" during the simulation duration: so the maximum value is 50. Table 2 presents the resulting metrics. The values are computed as normalized mean values for the simulation duration.

Tab. 2 Metric evaluation of the problem

| Periodic task type | Number of processors | Power (W) | Missed deadline | Number of executions |
|---|---|---|---|---|
| WAM | 1 | 1 | 45 | 27.6 |
| NAM | 1 | 1 | 39 | 25 |
| WAM | 2 | 1.6 | 23 | 49.5 |
| NAM | 2 | 1.6 | 0.6 | 49 |
| WAM | 3 | 1.8 | 5.3 | 49.8 |
| NAM | 3 | 1.8 | 0.4 | 49 |
| WAM | 4 | 2 | 0 | 50 |
| NAM | 4 | 2 | 0 | 50 |
| WAM | 5 | 2 | 0 | 50 |
| NAM | 5 | 2 | 0 | 50 |

From the analysis of the results of the Table 2 the following conclusions can be stated:

1. whatever the type of task the best results (throughput and QoS) are gotten from 4 processors but the power is then maximal;

2. using the "PTask_WAM" task type, one gets more missed deadlines, therefore the QoS is less good but the throughput is better, from 2 processors;

3. with the configuration using "PTask_NAM" task type and 2 processors, one consumes 25% less than with 4 processors; the throughput is only 2% lower than the maximal throughput; the QoS is good. This solution seems to be a good compromise.

If necessary it is possible to look finely at the behaviour of the tasks using a Gantt diagram. As an example the Figure 7 (such diagrams are accessible through STORM's GUI) gotten with "PTask_NAM" task type and 2 processors shows the deadline failure of task H during the first period. This missed deadline occurs because it is not possible to have a high degree of parallelism during this period.

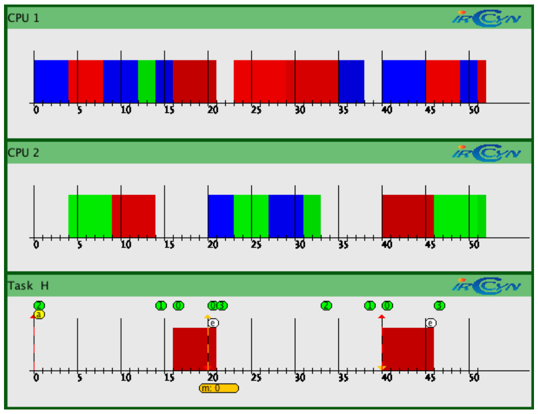

Fig. 7 An example of missed deadline with ”PTask_NAM” task type

## 5 Conclusion

The intent of our work is in the end to evaluate and compare schedulability performance and energy efficiency of scheduling algorithms and to investigate their adequacy with regard to the hardware and software architecture features of the considered systems. Presently, the STORM simulator is able to simulate accurately the execution of a set of tasks over a multiprocessor system. Tasks may exhibit various behaviors and be independent or not. Various scheduling strategies are supported too. As a result, the simulator is not only able to state about the schedulability of the studied system but also to characterize its behavior with some measurements for further analysis. At the present time, our work on STORM is concerned with:

- The energy and power aspects: i) we are expanding the hardware component library with processors supporting DVFS and DPM; ii) we are studying how to introduce banked memory patterns; iii) we are integrating the corresponding energy/power management strategies as new system entities;

- The memory architecture modelling: since memory time access can act significantly upon the global performances of a multiprocessor system, we are investigating how to model the behaviour of (different levels of) cache and external memory and in particular the overhead induced by their management;

- The evaluation process automation: we are defining a simulation language to drive the experiments. Thus the user would be able to describe a ”domain” for the simulation inputs and to specify the metrics to be measured as a result. Then the tool would automatically build a statistically representative set of compliant test cases, run their simulations, and output the statistical results.

## 6 References

[1] J.Y-T Leung. *Handbook of scheduling:Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC, New York, 2004.

[2] STORM, May 2009. http://storm.rts-software.org/.

[3] J. Goossens and Ch. Hernalsteen. A tool for statistical analysis of hard real-time scheduling algorithms. In *31st Annual Simulation Symposium , Boston (USA), IEEE Computer Society Press*, pages 58–65, April 1998.

[4] William Henderson, David Kendall, and Adrian Robson. Improving the accuracy of scheduling analysis applied to distributed systems. In *Journal of Real-Time Systems n* $^o$1, volume 20, pages 5–25, January 2001.

[5] YASA, May 2009. http://yasa.e-technik.uni-rostock.de/.

[6] MAST, May 2009. http://mast.unican.es/.

[7] M. Gonzlez Harbour, J. J. Gutirrez Garca, J. C. Palencia Gutirrez, and J. M. Drake Moyano. MAST: Modeling and analysis suite for real time applications. In *13th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 125–134, Delft (Netherlands), 2001.

[8] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGADA Ada Letters*, volume 24, pages 1–8, December 2004.

[9] CHEDDAR, May 2009. http://beru.univ-brest.fr/ singhoff/cheddar/.

[10] P. Šůcha, M. Kutil, M. Sojka, and Z. Hanzálek. Torsche scheduling toolbox for matlab. In *IEEE Computer Aided Control Systems Design Symposium (CACSD’06)*, pages 1181–1186, Munich, Germany, October 2006.

[11] TORSCHE, May 2009. http://rtime.felk.cvut.cz/scheduling-toolbox/.