SAE INTERNATIONAL

Performance Comparison of Real-Time and General-Purpose Operating Systems in Parallel Physical Simulation with High Computational Cost

2014-01-0200
Published 04/01/2014

Carlos Garre and Domenico Mundo
Universita della Calabria

Marco Gubitosa and Alessandro Toso
LMS International

CITATION: Garre, C., Mundo, D., Gubitosa, M., and Toso, A., "Performance Comparison of Real-Time and General-Purpose Operating Systems in Parallel Physical Simulation with High Computational Cost," SAE Technical Paper 2014-01-0200, 2014, doi:10.4271/2014-01-0200.

Abstract

Real-time simulation is a valuable tool in the design and test of vehicles and vehicle parts, mainly when interfacing with hardware modules working at a given rate, as in hardware-in-the-loop testing. Real-time operating-systems (RTOS) are designed for minimizing the latency of critical operations such as interrupt dispatch, task switch or inter-process communication (IPC). General-purpose operating-systems (GPOS), instead, are designed for maximizing throughput in heavy-load systems. In complex simulations where the amount of work to do in one step is high, achieving real-time depends not only in the latency of the event starting the step, but also on the capacity of the system for computing one step in the available time. While it is demonstrated that RTOS present lower latencies than GPOS, the choice is not clear when maximizing throughput is also critical.

In this paper, the performance of RTOS and GPOS running complex real-time simulations is compared, focusing on the computation of large simulation steps. GNU/Linux has been chosen as GPOS. A RTOS is chosen with a micro-benchmark comparing the major choices of Linux-based RTOS. Once chosen the systems, the simulation of a tire model is used as application case for benchmarking, comparing within 52470 different configurations (with different number of elements and threads). The benchmark measures which configurations miss a single deadline, and demonstrates that even in simulations with a high number of elements and large time steps, RTOS are a better choice due mainly to the latency associated to IPC and task switch when the simulation is parallelized.

Introduction

Real-time Operating Systems (RTOS) are often used in the automotive industry in hardware-in-the-loop (HIL) testing, where the ECU of some subsystem of the vehicle is tested against computer-controlled conditions, i.e. a physical simulation of the vehicle. The ECU typically works at a specific rate, and the computer running the HIL must be able to communicate with the ECU at this rate. This means that the physical simulation must compute and provide feedback of one time step in a time constrained by the ECU rate. The computation of the time step does not need to use all the available time, and it is a common practice [1] to compute the time step in less time and then wait or sleep until the period is completed and a new input from the ECU is received. This wait time is a safe backup for guaranteeing that the worst case latencies of the system cannot yield to a missed deadline, i.e. not giving feedback to the ECU at the proper rate. The complexity of the physical simulation scenarios that modern computers can achieve is growing more and more and nowadays it is possible to run interactive simulations of highly complex systems [2,3]. Having a more complex model means that the time required for computing each step may be higher, and then the bottleneck for achieving real-time depends not only in the latency of the system when attending the inputs of the ECU, but also on the time required for doing a high number of computations with a significant amount of data.

While RTOS are designed with a focus on maximizing time determinism (minimizing latencies of basic operations such as task switch or interrupt dispatching), general-purpose operating systems (GPOS) are designed with a focus on maximizing throughput, i.e. accomplishing more work in less time. For this reason, GPOS are more suitable for tasks where instead of

wanting results at a specific moment we want as many results as possible in less time. These tasks, known as *best-effort* or *real-fast* tasks, require a lot of computation power and a lot of resources (such as memory), as in the case of computing a time step of a complex physical model. Unfortunately, the mechanisms used for maximizing throughput, such as fair scheduling [4] or memory paging [5], typically break determinism and thus a GPOS cannot give guarantees on when a specific task will start and finish.

In this paper, we analyze the scalability of a physical simulation running in a GPOS and in a RTOS. The idea is to find the system that better responds to the increasing complexity of the simulation while guaranteeing that the simulation is always accomplished at a specific rate. The performance of the system is analyzed in terms of hard real-time [6], meaning that instead of measuring the difference between the desired rate and the achieved rate, we want to measure the binary condition: the system can work at this rate or not. This is the condition desired in HIL testing, where losing the synchronization with the ECU may cause a system failure and more in an embedded system where the consequence can be critical with the loss of human lives. The present work is composed of two main sections. First, we design and run a benchmark for choosing one RTOS from the wide number of choices currently available. We focus only on free and open-source solutions, avoiding any commercial influence, providing a general solution independent of the available budget and with the potential of adapting the solution to any specific needs thanks to the availability of the source code. The benchmark for choosing the RTOS will be explained on section *Micro-benchmark of real-time systems*, and will be referred throughout the paper as the *RT micro-benchmark*.

Once having the RTOS with the best real-time performance from the test set, the next step is to compare its performance in a physical simulation scenario with the performance of a GPOS. For the GPOS, GNU/Linux has been chosen as the obvious example of free and open source GPOS, and because the three choices of RTOS in the benchmark were all Linux-based. This second benchmark focuses only on the computation of the simulation steps, because the superior timing of RTOS with respect to GPOS when attending external interrupts is demonstrated in other works on benchmarking [7,8]. For this reason, the scenario for the benchmark consists in a physical simulation loop running at a specific rate isolated from the intervention of external interrupts. The simulation of a simple tire model was designed specifically for the purpose of this benchmark. The model is easily scalable in the number of elements to simulate (masses and springs) and in the level of parallelization of the solver (number of concurrent threads). The simulation is run in each system (the RTOS and the GPOS) with different configurations (different number of elements and different levels of concurrency), and the number of missed deadlines (steps that were not solved in a specific maximum period) is measured. The details of this benchmark are explained in section *Comparison of RTOS and GPOS* and will be referred throughout the paper as the *RTOS/GPOS benchmark*.

Related Work

The first step when starting the design of a benchmark is to clearly define what characteristic we want to measure for establishing the comparison between the systems in the test set. Best-effort performance can be measured in different ways depending on the type of calculations which are predominant in the system. For integer operations, the Dhrystone benchmark is the basic reference, while for floating point operations the Whetstone benchmark is the reference instead [9]. When talking about real-time performance, there are some basic references such as Rhealstone [10] or Hartstone [11]. Both are complementary because they focus on different aspects of performance. Rhealstone fits in the description of a micro-benchmark [12], what means that it focuses on obtaining latency measures of atomic operations. This provides a good overview of the rt-compliance of the system and gives a unique score (the Rhealstone value) to each system, but does not give too much insight on the performance of real applications. Hartstone, instead, consists of a synthetic benchmark [13] for real-time performance. The design of a synthetic benchmark starts with a statistical analysis of the behavior of typical applications. The synthetic benchmark is then built over a stochastic model resembling the most important characteristics of real software (depending on what we want to measure) but isolating them from other variables or from uncertainty. This makes the synthetic benchmark a very repeatable experiment but is still too generic for understanding precisely how the system will perform with the final real application. Benchmarking directly with final applications is less prone to repeatability and hard to implement when the final application is not fully implemented (or implemented only for one system, requiring a big effort of portability to the different systems in the test set), but provides the more close to reality measure of performance of each system for a single specific application. There exist a lot of previous works on benchmarks for specific applications, such as air defense [14], unmanned vehicles control [15] or air traffic collision detection [16]. To the best of our knowledge, there are no specific benchmarks for RT performance of vehicles or vehicle parts simulation.

The present work starts with a micro-benchmark for establishing a quantitative comparison between different choices of RTOS (the RT micro-benchmark). There is previous work on benchmarking of RTOS specifically focused on embedded systems [17,18,19], but our work focuses on RTOS designed for large applications running in general purpose computers. Since this type of systems evolves at a very fast rate, the results obtained in previous works [6,7,14,15,16,19] may not apply to current systems and an updated benchmark is needed. For more complete surveys on real-time benchmarking the reader may refer to [6] and [20]. Although some authors have compared some of the systems in our test set [7,19], none of them have compared the three of them and their results may be outdated due to the fast evolution of operating systems.

The second part of this paper, the RTOS/GPOS benchmark, compares the performance of the RTOS (that was chosen with the micro-benchmark) with that of a GPOS when running a

real-time physical simulation of a vehicle tire, where the characteristics of the simulation put it in the middle between a real-time and a best-effort task. There is previous work on integrating real-time and best-effort tasks on real-time systems [21], on improving best-effort performance of real-time systems [22] and even operating systems designed for optimizing the tradeoff between best-effort and real-time performance [23]. All these works assume that a task can have either real-time or best-effort nature, but none of them consider hybrid tasks where both performance criteria are relevant. In [7] the performance of a RTOS and a GPOS achieving some specific tasks (controlling fuel injection for an industrial engine and building a Linux kernel) is compared, providing very interesting clues on how to choose between a RTOS or a GPOS depending on the application. However, the application cases presented are clearly biased to real-time (in the case of the fuel injection) or to best-effort (in the case of kernel build) without showing any case of an application where a real tradeoff is needed. Indeed, they touch the surface of the challenge analyzed in this paper when stating that “obtaining the best possible real-time response usually requires that the real-time system be run at low utilization”. In this work, we present an application case where the system is highly stressed with a big workload maximizing the use of resources (CPU and memory) but where hard real-time must be still accomplished.

Micro-Benchmark of Realtime Operating Systems

In this section we describe how we chose the most representative case of RTOS for later comparison with the GPOS. We start with the requisite of comparing only free and open-source solutions, for avoiding the influence of any commercial interests and because these systems are more affordable and provide versatile solutions that can be adapted to almost any system (thanks to the availability of the source code). There exist different choices of free open-source GPOS, such as FreeBSD, OpenBSD or GNU/Linux. GNU/Linux was chosen because it can be considered to be the most representative of this type of systems and because most open source RTOS are based on this platform. In the following subsections we will first show a selection of a few candidates for a representative RTOS, and then we will present the results of a micro-benchmark designed for choosing the candidate with the best real-time performance. Far from being an exhaustive benchmark on real-time, the RT micro-benchmark provides a clue for easy selection of the RTOS with the lowest latencies when performing the operations that will be used in the application case that will be presented in the RTOS/GPOS benchmark.

Setup of the Experiment

The different RTOS to be compared with the RT micro-benchmark were chosen according to the following criteria:

- All of them are currently active projects with a clear long-term future.
- All of them are open source, free and Linux-based.
- Must support Intel x86-64 CMP architecture (because of the machine used for running the benchmark).
- Must be oriented for general purpose applications rather than for small embedded systems (more suitable for complex physical simulation).

After a research on available RTOS [24,25,26], the selected systems were three: RTAI (Real Time Application Interface) [27], Xenomai [28] and Linux patched with Ingo Molnar's RT-Preempt patch [29]. RTAI is a RT extension for the Linux kernel consisting in a hardware abstraction layer based on the ADEOS nano-kernel [30] combined with a very complete API providing an easy and versatile way to program hard real-time applications over Linux. Although Xenomai was once thought to be merged with RTAI in a single project (RTAI/Fusion), it is currently an independent project. Xenomai is also based on ADEOS, but it provides an additional abstraction layer (known as skins) for maximizing portability of applications to Xenomai from a wide range of RTOS, including RTAI or commercial RTOS such as VxWorks, and even a POSIX skin for painless portability of native Linux applications. The Ingo Molnar's RT-Preempt patch is a modification to the Linux kernel adding support for preemption in critical sections of the kernel and converting interrupt handlers in preemptible kernel threads. For each system, the version used in the RT micro-benchmark was the last available at the moment of writing this paper. The versions are:

- RTAI 3.9.1 over Linux kernel 2.6.38
- Xenomai 2.6.2.1 over Linux kernel 3.5.7
- PREEMPT-RT patch over Linux kernel 3.8.4

For a better understanding of the results obtained in RTOS/GPOS benchmark and for establishing a worst-case reference of the latencies, we run the RT micro-benchmark also in the GPOS. Since the results of the RT micro-benchmark present Xenomai as the best candidate of RTOS, the version chosen for the Linux kernel was the same of the chosen RTOS, which is 3.5.7. The machine used for running the benchmark is an Intel i7-3930K (6 HT cores at 3.2 GHz) with 32GB of RAM (DDR3 at 1866 MHz) and an ASUS P9X79-LE/C/SI motherboard. All tests are programmed in C, compiled with GCC 4.4.3 with options *-pipe -O2 - D_REENTRANT* and linked with option *-O1*.

Design of the RT Micro-Benchmark

The RT micro-benchmark is based on the Rhealstone benchmark [10]. There are some generally accepted complaints about the design of Rhealstone [6] being the two most important the lack of worst case measurements (which is the most important measure for hard real-time systems) and the doubtful convenience of providing one single mark to a number of different aspects, being hard to determine the weights of each aspect over the final score. Our benchmark focuses only on the aspects which are explicitly part of our application (task switch time and IPC overhead) but also preemption time has been considered, due to the fact that some kernel services could run in the (potential) idle time of

our application and thus a low preemption latency is a must for ensuring a correct reentry time for the real-time application. Since we focus on hard real-time performance, we measure both average and worst case latencies but, contrary to Rhealstone, we consider the worst case latency as the main criteria. Instead of providing a single score to each system, we extract conclusions from a global analysis of all the results. The following subsections show a brief description of the tests implemented in the RT micro-benchmark with the obtained results.

Task Switch Latency

The first test measures the overhead caused by the scheduler when switching between two tasks of the same (high) priority. The test launches two threads in the same CPU, and each thread runs a loop with a single system call to the scheduler for yielding execution to the other thread. This causes a continuous switch between both threads, and for each switch the latency is measured.

Table 1. Task switch latencies measured in the RT micro-benchmark. The units for all the measures are nanoseconds.

	Average	Worst case
RTAI	715	12060
Xenomai	391	**1422**
Preempt	4027	27085
Linux	1376	13200

The results show that Xenomai has the smallest worst case latency, but all the systems present reasonably good average and worst case latencies in task switch.

Preemption Time

The second test measures the elapsed time from the instant when a high priority task is ready for execution until it actually starts execution. This is one of the most critical aspects of real-time performance, and a priori non RT systems are expected to present worse results than a RTOS. The test launches one low priority thread and one high priority thread. The high priority thread iterates in a loop with a single *sleep* instruction, while the low priority thread loops with a dummy workload. The high priority thread obtains a time measurement on each iteration, and the latency is obtained confronting this time with the expected sleep time. Table 2 shows the result of this test.

Table 2. Latencies measured in the RT micro-benchmark of preemption from a low priority to a high priority task. The units for all the measures are nanoseconds.

	Average	Worst case
RTAI	737	14855
Xenomai	1422	**2922**
Preempt	3763	15986
Linux	33347	$>10^6$

Again, the best performance was observed in Xenomai. As expected, all the RTOS present good average and worst case latencies, while the GPOS completely fails on guaranteeing a correct reentry of the high priority task (with a worst-case latency greater than 1 millisecond).

IPC Overhead

This test measures the latency added by Inter-Process Communication (IPC). There exists a number of different IPC mechanisms, such as semaphores, spin locks, message queues or barriers. The test focuses on the specific case of semaphores because their simplicity allows using them as building blocks for almost any IPC needs and because it will be the only IPC mechanism used in the RTOS/GPOS benchmark. The test launches two threads sharing a variable closed in a *mutex* section controlled by a binary semaphore. Each thread runs a loop locking and unlocking the semaphore, and obtaining measures of the time needed for acquiring the lock.

Table 3. Semaphore shuffle latencies measured in the RT micro-benchmark. The units for all the measures are nanoseconds.

	Average	Worst case
RTAI	1680	12986
Xenomai	162	**1282**
Preempt	1483	19776
Linux	419	12453

Again, Xenomai presents the lowest worst case latency. The other systems present worst case latencies in a higher order of magnitude.

Conclusions of the RT Micro-Benchmark

A summary of the results obtained running the RT micro-benchmark is shown in table 4.

Table 4. Summary of worst case latencies in the three tests of the RT micro-benchmark. The units for all the measures are nanosecond.

	Task switch	Preemption	IPC
RTAI	12060	14855	12986
Xenomai	**1422**	**2922**	**1282**
Preempt	27085	15986	19776
Linux	13200	$>10^6$	12453

These results clearly present Xenomai as the best candidate. Given the architecture of each system, briefly explained in section *Setup of the experiment*, RTAI was expected to have better performance. The architecture of Xenomai follows the same principles of the RTAI kernel but with an additional abstraction layer that may add some slight overhead. Moreover, our results do not agree with previous work on realtime benchmarking, reporting a better performance of RTAI compared to Xenomai [7]. There are two main reasons for this difference. One, is the different nature of the experiments run by Barbalace et al. They do not run a micro-benchmark, but a benchmark with a final application with three main sources of latency: interrupt handling, rescheduling and datagram

processing. From these sources, only rescheduling is present in our micro-benchmark, because we focus only in the latency sources that will be present in the RTOS/GPOS benchmark. Not only the software side of their experiment is different, but also the hardware. Their setup is basically an embedded system instead of a general-purpose computer. Indeed, they had to do their own port of RTAI to their platform, what means that they were not using an official RTAI version.

The second cause of the different results we obtained compared with the work of Barbalace et al. is the significant difference on versions. Although they do not specify the versions used for each RTOS and for the Linux kernel, even assuming the latest available at the time of his publication, they would have used RTAI 3.5 and Xenomai 2.4. From those versions to the versions we have used, the rates of update have been very different for RTAI and Xenomai. The latest available (at the date of this work) RTAI version (3.9.1) supports Linux kernel up to 2.6.38.8, while Xenomai 2.6.2.1 supports Linux kernel up to 3.5.7. Apart from numbers, an study of the Changelog of both projects show that most of the updates of RTAI were basically for supporting more architectures and for improving tools such as RTAI-Lab, which are great but are not used in our benchmark. Instead, the Changelog of Xenomai shows an important number of improvements directly related with real-time performance, mainly in version 2.5, when a massive rework of some core components was done for improving performance.

Our results are more close to those of [19], which are from a more recent date, although they do not include RTAI on the benchmark.

Comparison of RTOS and GPOS

Once chosen the RTOS (Xenomai) to be compared with the GPOS (Linux), a different benchmark is setup. The RTOS/GPOS benchmark is designed with some of the characteristics typically present in simulations of vehicle dynamics: multiple bodies affected by external and internal forces, integration of positions and velocities, contact constraints, linear elasticity and a multi-threaded parallel solver. It consists in the simulation of a simple tire model at a fixed rate of 100Hz (time step of 0.01 ms). The tire falls due to gravity and deforms under collision with a floor plane. The benchmark measures the hard-real-time performance of both systems (Linux and Xenomai), meaning that each test is passed only when no deadlines are missed in any of the time steps of the whole simulation time.

The setup for the experiment uses the same hardware of the RT micro-benchmark, and the programming language is C as well, with the same compiler settings. Although a demonstrator was built for visualization of the simulation (see Figure 1), the actual benchmark was run without any graphical output.

Figure 1. Final textured render of the tire model simulation.

Tire Model

Mass-spring-damper systems are a typical approach for dynamics simulation of vehicle tires [31]. Our tire model consists in a mass-spring-damper system for resembling the typical characteristics of tire models, but it has not been designed for being an accurate representation of real tire dynamics, but rather for being an easily scalable model for the purpose of the RTOS/GPOS benchmark. The mass-spring-damper system has one mass on the center of the tire and a number of masses on the surface. The number of masses on the surface starts from 4 (one on each edge of the vertical and horizontal axis) and can grow in steps of 4 by subdivision of the four quadrants of the circumference. For each surface mass, there is one bidirectional spring-damper connecting to the center mass, and two bidirectional spring-dampers linking with the adjacent surface masses. This makes the number of springs to be twice the number of surface masses. Since the quality of the simulation is not important for benchmarking purposes, the subdivisions were made with respect to the horizontal axis for simplicity, but a better balanced model could be built by making subdivisions with respect to the circumference arcs. Figure 2 shows an example of two tire models using different resolutions (number of surface masses and springs).

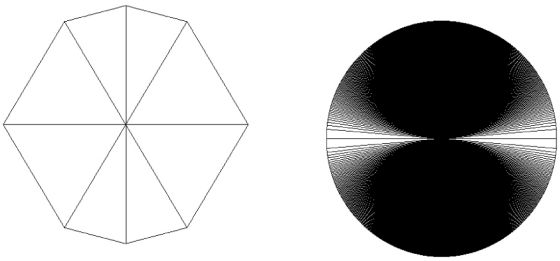

Figure 2. Example of different resolutions of the tire model. Left: 9 nodes, 16 springs. Right: 1201 nodes, 2400 springs.

The mass-spring system is solved using an explicit Euler integrator, because it guarantees that all the time steps will be solved in a fixed number of iterations. A low stiffness had to be used for guaranteeing stability of the simulation with the explicit

solver, but in any case the parameterization of the model has no influence on time measures. Gravity was used as the only force source, and the floor was simulated with a simple geometric constraint. Since collision detection is checked only against the floor level, all the time steps require the same amount of computations for detecting collisions. Using more constraints and/or a more complex collision detection scheme such as bounding volume hierarchies [32] would imply to have significant different costs for computing each time step, what would affect the comparison in terms of time determinism.

Parallelization of the Solver

Solving one time step of the tire model with the explicit integration scheme implies first computing the forces of all springs and then integrating positions and velocities of all masses. Both operations can be parallelized by dividing the elements to simulate in a number of groups, and then launching one thread for each group of elements. In a first attempt, we divided both the computation of spring forces and the integration of masses, but the synchronization between threads for waiting to have other forces computed before integrating nodes was causing an important overhead. We found more efficient to split only the integration of the masses. In this way, all the threads compute all the forces affecting their associated masses, although having each force computed twice (by different threads) and then each thread integrates only a subset of the masses. The detailed algorithm is shown in table 5.

Table 5. Algorithm for parallelization of the tire model, where n is the total number of masses and i the number of threads.

For each thread:	
Assign $\lfloor n/i \rfloor$ masses	
For each simulation step	
→	Read mass positions and velocities from other threads
	Compute spring-damper forces
→	Integrate positions and velocities of assigned masses

The steps marked with an arrow require access to mutual exclusion regions shared by all threads. These consist in one binary semaphore for accessing each of the masses (position and velocity). This algorithm is far from being optimal, but present the typical characteristics of a parallel solver for physical simulation, dividing the elements to simulate in a number of threads and synchronizing all threads using IPC mechanisms (semaphores in this case). All threads are created with the same priority of the parent process, avoiding forced preemption inside the solver. The parent process waits in a *join* to all threads before starting the compilation of benchmark data (number of missed deadlines).

Benchmark Design

In the algorithm in table 5 we see that in the first step each thread is assigned a set of masses, depending on the total number of masses in the model (*n*) and on the number of threads (*i*). The RTOS/GPOS benchmark is composed of a set of tests consisting in simulating the tire model using different values for both *n* and *i*. In this way, the system can be stressed by increasing the number of elements to simulate and by increasing the level of concurrency of the solver. In our parallelization algorithm, having more threads per CPU means having more blocks of *mutex* access with potential waits, so there is no advantage in having a high number of threads per CPU. For this reason, the value of *i* is limited to 36, which means a maximum of 3 threads per HT core. It is outside of the scope of this paper to discuss the best choice of the algorithm and the scalability of different algorithms with respect to the number of threads.

One of the main differences between a GPOS and a RTOS is the task scheduler. Real-time applications usually trust in real-time scheduling policies such as FIFO (First In First Out). FIFO scheduling gives the programmer control on when and which task switches happen, unless a task switch is forced due to preemption of a high priority task or due to any condition requiring the task to wait for some resource (such as a semaphore lock). In the other side, GPOS use scheduling policies designed for optimizing best-effort performance of all the applications that could be concurrently running on the system, and do not trust in the programmer of a single application for control of task switches. In the case of GNU/Linux, the Completely Fair Scheduler (CFS) [4] is designed for providing equal CPU power to all concurrent tasks. Although both systems (Xenomai and Linux) allows the use of both the FIFO and the CFS scheduler, a fair comparison between RTOS and GPOS assumes that each system is using the scheduler for which it is designed, i.e. the FIFO scheduler for Xenomai and the CFS scheduler for Linux. For each system, the benchmark is run using its native scheduler, but results are also presented for FIFO scheduling under Linux, demonstrating that the results are not only influenced by the choice of the scheduling policy.

Benchmark Results

The real-time performance of each system measured with the benchmark is shown in figures 3 and 4 (final page). In the figures, the X axis represents the number of simulated elements (*n*) and the Y axis the number of threads (*i*). The number of missed deadlines for each configuration is coded with colors. For better visualization, the aspect ratio is adjusted and the size of the points is scaled by a factor of 40.

The results have different interpretations depending on the *hardness* of real-time required by our application. In the case of life-safety real-time systems [19], we are interested in the apparition of the first colored dot of each graph. This first dot establish a threshold on the maximum number of elements we can simulate with timing guarantees. In the case of Xenomai (Figure 3), the first dot appears with 2189 elements, while for

Linux Figure 4) it appears with only 681 elements. It means that Linux fails to meet strict hard real-time requirements even simulating a relatively low number of elements.

In the case of an application requiring firm real-time where only a few deadlines can be missed, we should concentrate on the first red dots of each graph. In the case of Linux, the first red dot appears again with 681 elements, while for Xenomai it appears with 3149 elements.

In applications requiring only soft real-time, it is better to analyze the colored regions of the graphs instead of finding isolated dots. The graph of Xenomai (Figure 3) presents a first sparse zone mainly composed of light blue dots, what means that those configurations are good for soft real-time, losing very few deadlines in the worst case. The red region, starting above 5200 elements, show configurations for which the system is not even capable of guaranteeing a good soft real-time performance. In the case of Linux (Figure 4), the first region is much more dense and mainly composed of red dots. In this case, a good soft real-time performance is not guaranteed above 2400 elements. The graph present an intermediate region where red color is less predominant, but this apparent increase on performance is only a proof of the nondeterministic behavior of the GPOS compared to the RTOS; the RTOS give us three clear choices (strict real-time, soft real-time and no real-time), while the RTOS tell us that above 2400 elements we do not know what to expect.

Regarding the number of threads, the best performance for both systems is achieved when using one thread per core. Xenomai seems to exhibit a worse handling of virtual cores (Hyperthreading), since the best performance is obtained up to 6 cores instead of 12 (our machine has 6 HT cores, with a total of 12 virtual cores). Even considering this, when using 12 threads (one thread per virtual core), the first red dot of Linux appears in 3609 elements, while for Xenomai it appears in 5329 elements.

The results show that the RTOS provide better real-time guarantees than the GPOS during the simulation of a time step of a complex simulation. For isolating the time required for computation of the time step, the interrupt dispatch latency was not taken into account. Considering that other works demonstrate that the interrupt dispatch latency is lower for a RTOS than for a GPOS (mainly in the worst case), the combination of all latencies (computation of time step and interrupt dispatch) present globally better real-time performance for the RTOS even in the case of complex simulations with large time steps. The observations regarding the performance dependence on the number of threads are applicable only to the parallelization algorithm used in the benchmark (table 5), and cannot be generalized to other algorithms.

An explanation for the better performance of the RTOS can be found on the latencies measured in the RT micro-benchmark. Furthermore, in the case of the FIFO scheduler, task switch happens only due to semaphore locks, considering that the code of the thread does not include any explicit yield. For the CFS scheduler, instead, a time slice could finish during the computation of the time step and thus non explicit task switches may happen (the time slice of the CFS scheduler was dynamically changing between 2.25 ms and 18 ms). The existence of non explicit task switches in the CFS scheduler and the bigger (worst case) latencies for task switch and for IPC (as reported in the RT micro-benchmark), makes the GPOS require more time for computing the time step in the worst case.

The use of different scheduling policies for each system could be a determinant factor that may be biasing the results toward one of the candidates if the algorithm used for parallelization was better suited for one specific policy. Figure 5 shows the performance of Linux using a FIFO scheduling policy, demonstrating that obtaining RT performance is not just a matter of using a specific scheduling policy on any system. Comparing the performance of Linux using the CFS or the FIFO scheduler, with FIFO policy there are no outliers of missed deadlines with very few masses and it seems to obtain more benefit from high levels of parallelization. The behavior of Linux with FIFO is qualitatively more comparable to that of Xenomai, showing more deterministic regions (one completely blue and other almost completely red). The first missed deadline appears with 2197 elements, which is close to the 2189 elements of Xenomai, but the first red dot with 12 threads (optimal CPU usage) appears with only 3609 elements compared to the 5329 elements of Xenomai.

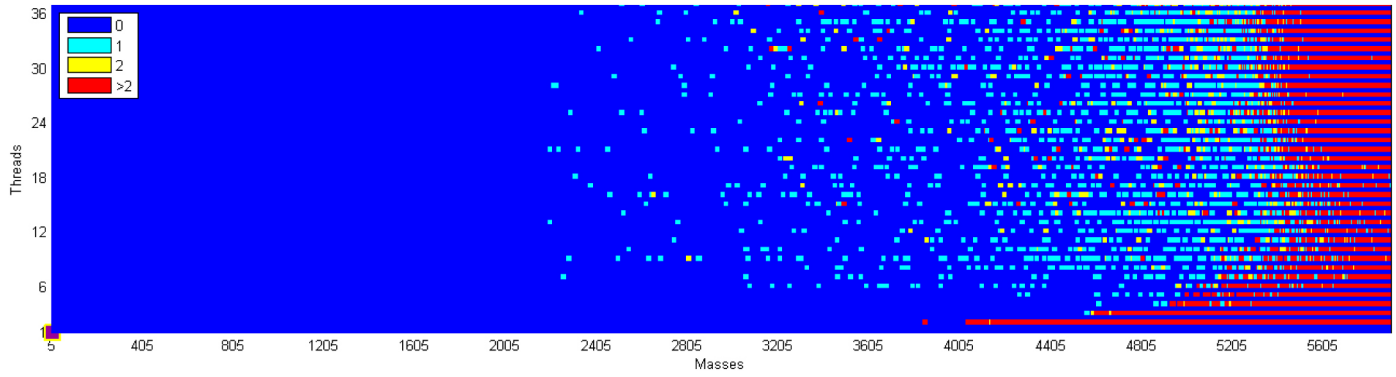

Figure 3. Real-Time performance of Xenomai using FIFO scheduler. The X axis shows the number of elements to simulate and the Y axis the number of concurrent threads. The legend shows the number of missed deadlines.

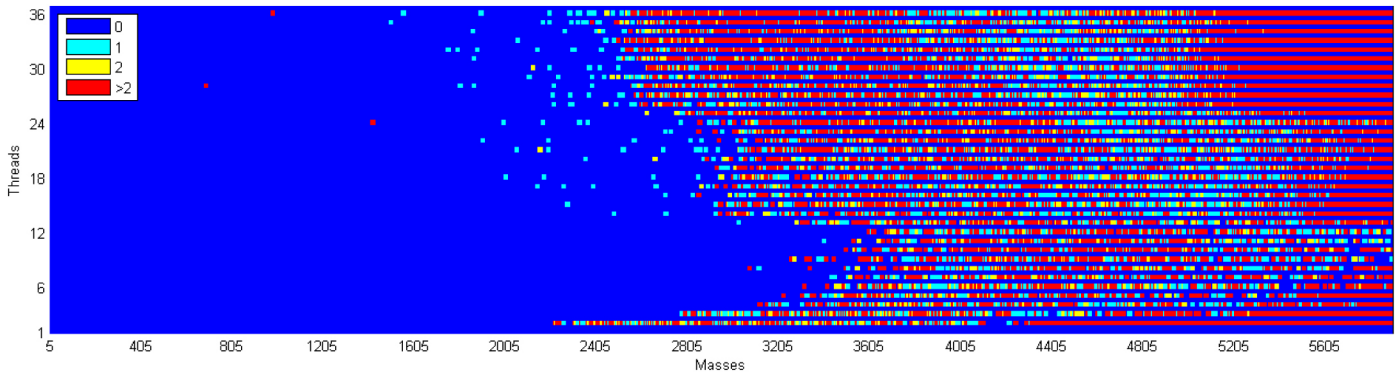

Figure 4. Real-Time performance of GNU/Linux using CFS scheduler. The X axis shows the number of elements to simulate and the Y axis the number of concurrent threads. The legend shows the number of missed deadlines.

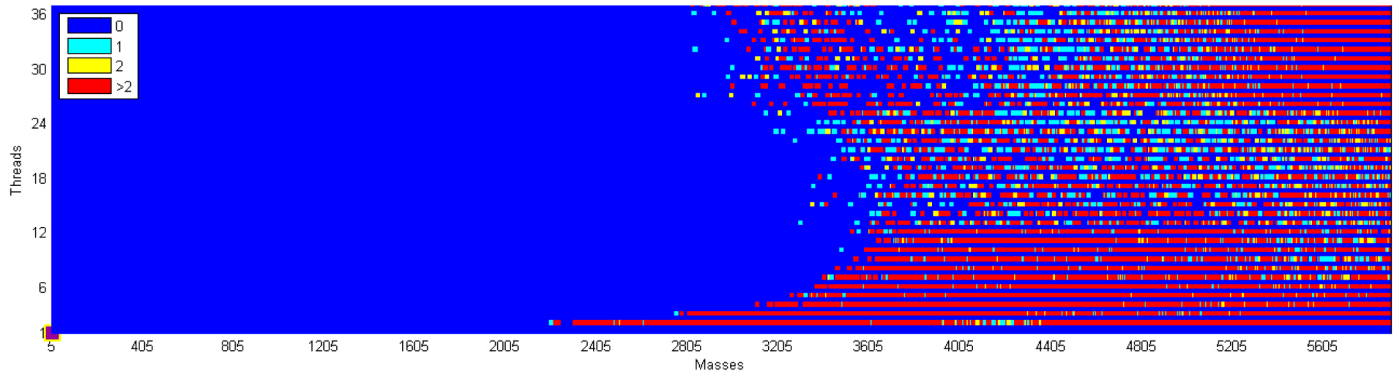

Figure 5. Real-Time performance of GNU/Linux using FIFO scheduler. The X axis shows the number of elements to simulate and the Y axis the number of concurrent threads. The legend shows the number of missed deadlines.

Conclusions

In this paper, the performance of real-time and general-purpose operating systems was compared when running parallelized physical simulations with high computational cost. A representative case of RTOS was chosen from various candidates through a micro-benchmark measuring real-time performance, and then compared with a GPOS in a second benchmark. The second benchmark consisted in the simulation of a scalable tire model with different configurations including different number of bodies to simulate and different number of parallel threads. The first benchmark showed Xenomai as the candidate with the best performance (compared with RTAI and Linux patched with the RT-Preempt patch), while the second benchmark demonstrated that the RTOS present better response times even in the case of simulations with large time steps stressed with a high number of bodies to simulate. Different scheduling policies were tested for the GPOS, demonstrating that using a scheduler oriented toward real-time applications does not guarantee a better performance in a general-purpose system.

The present work presents physical simulation with large time steps and high complexity as an example of a task requiring a trade-off between real-time and best-effort performance. More research on systems running both real-time and best-effort tasks in parallel would be of valuable interest, considering the growing complexity of systems dedicated to physical simulation. Multi-rate co-simulation environments are a good example of applications where both real-time and best-effort tasks may compete for system resources. The presented benchmarking tools could be easily adapted for considering this case of study in the future, as well as other configurations of interest in the automotive industry.

References

1. Fathy, H., Filipi, Z., Hagena, J., Stein, J., “Review of hardware-in-the-loop simulation and its prospects in the automotive area”, Proc. SPIE 6228, Modeling and Simulation for Military Applications, 62280E, 2006, doi:10.1117/12.667794.
2. Garre, C., Otaduy, M.A., “Haptic rendering of objects with rigid and deformable parts”, *Computers & Graphics*, 34(6): 689-697, 2010, doi:10.1016/j.cag.2010.08.006.
3. Prescott, W., Heirman, G., furman, M., De Cuyper, J. et al., “Using High-Fidelity Multibody Vehicle Models in Real-Time Simulations,” SAE Technical Paper 2012-01-0927, 2012, doi:10.4271/2012-01-0927.
4. Pabla, C.S., “Completely Fair Scheduler”, *Linux Journal*, 2009(184): Article No. 4, 2009.
5. Gorman, M., “Understanding the Linux Virtual Memory Manager”, Prentice-Hall, USA, ISBN-10: 0131453483. ISBN-13: 978-0131453487, 2004.
6. Halang, W.A., Gumzej, R., Colnaric, M., and Druzovec, M., “Measuring the Performance of Real-time Systems.”, *Real-time Systems* 18(1): 59-68, 2000, doi:10.1023/A:1008102611034.

7. Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., et al., “Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application”, *IEEE Transactions on Nuclear Science*. 55(1): 435-439, 2008, doi:10.1109/TNS.2007.905231.
8. Regnier, P., Lima, G., and Barreto, L., “Evaluation of interrupt handling timeliness in real-time Linux operating systems”. *ACM SIGOPS Operating Systems Review archive*, 42(6): 52-63 ACM, 2008, doi:10.1145/1453775.1453787.
9. Weicker, R.P., “An overview of common benchmarks” *Computer* 23(12): 65-75, 1990, doi:10.1109/2.62094.
10. Heursch, A., Horstkotte, E., and Rzehak, H., “Preemption concepts, Rhealstone benchmark and scheduler analysis of Linux 2.4”, presented at Real-Time & Embedded Computing Conference, Italy, November 27-28, 2001.
11. Weiderman, N., “Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications”, *Proceedings of the Working Group on Ada Performance Issues*, 10(3): 126-136, 1989, doi:10.1145/322807.322853.
12. Bershad, B.K., Draves, R.P., Forin, A., “Using microbenchmarks to evaluate system performance”, Third Workshop on Workstation Operating Systems, 1992, doi:10.1109/1992.275671.
13. Curnow, H.J., Wichmann, B.A., “A synthetic benchmark”. *The Computer Journal* 19(1):43-49, 1976, doi:10.1093/19.1.43.
14. Welch, L.R., and Shirszi, B.A., “A Dynamic Real-time Benchmark for Assessment of QoS and Resource Management Technology.”, Fifth IEEE Real-Time Technology and Applications Symposium, 1999, 36-45, 1999, doi:10.1109/RTTAS.1999.777659.
15. Fadia, N., Cassé, H., Sainrat, P., Bahsoun, J.P., et al., “PapaBench: a Free Real-Time Benchmark.”, 6th International Workshop on Worst-Case Execution Time Analysis, Germany, 2006, doi:10.4230/OASIcs.WCET.2006.678.
16. Kalibera, T., Hagelberg, J., Pizlo, F., Plsek, A., et al., “CD X: a Family of Real-time Java Benchmarks.” presented at the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems, 41-50, Spain, September 23-25, 2009.
17. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., et al., “MiBench: A Free, Commercially Representative Embedded Benchmark Suite.” presented at 2001 IEEE International Workshop on Workload Characterization, 2001. *WWC-4*, 3-14, 2001, doi:10.1109/WWC.2001.990739.
18. Tan, S., Tran, B., “Survey and Performance Evaluation of Real-time Operating Systems (RTOS) for Small Microcontrollers”, *IEEE Micro*, PP(99):1, 2009, doi:10.1109/MM.2009.56.
19. Brown, J. H., Martin, B., “How Fast Is Fast Enough? Choosing Between Xenomai and Linux for Real-time Applications.” presented at Twelfth Real-Time Linux Workshop, Kenya, October 25 - 27, 2010.
20. Gumzej, R., Halang, A., Springer, W., “Real Time Systems' Quality of service. Introducing Quality of Service Considerations in the Life Cycle of Real-time Systems”, Springer, London, ISBN: 978-1-84882-847-6.
21. Yanyong, Z., Sivasubramaniam, A., “Scheduling Best-effort and Real-time Pipelined Applications on Timeshared Clusters.” presented at the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, Greece, July 4 - 6, 2001.
22. Banachowski, S., Bisson, T., and Brandt, S., “Integrating Best-effort Scheduling into a Real-time System.” presented at 25th IEEE International Real-Time Systems Symposium, Portugal, December 5 - 8, 2004.
23. Dellinger, M., Piyush G., and Binoy R., “ChronOS Linux: a Best-effort Real-time Multiprocessor Linux Kernel.” presented at 48th ACM/EDAC/IEEE Design Automation Conference, USA, June 5 - 9, 2011.
24. Stankovic, J.A., and Rajkumar, R., “Real-time Operating Systems.” *Real-Time Systems*, 28(2-3): 237-253, 2004, doi:10.1023/B:TIME.0000045319.20260.73.
25. Aroca, R., Caurin, G., “A Real Time Operating Systems (RTOS) Comparison.” presented at the 6th Work-shop on Operating Systems, Brazil, 2009.
26. Straumann, T., “Open Source Real Time Operating Systems Overview.”, presented at ICALEPCS 2001 Conference, San Jose, USA, Nov. 2001.
27. Mantegazza, P., Dozio, E.L., Papacharalambous, S., “RTAI: Real Time Application Interface”, *Linux Journal*, 2000(72): Article No. 10, 2000.
28. Gerum, P., “Xenomai-Implementing a RTOS emulation framework on GNU/Linux”, www.xenomai.org, 2004.
29. Dietrich, S.T., and Walker, D., “The Evolution of Real-Time Linux” presented at Seventh Real-Time Linux Workshop, France, November 3 - 4, 2005.
30. Yaghmour, K., “Adaptive Domain Environment for Operating Systems.” www.opersys.com, 2001.
31. Gallrein, A., and Bäcker, M., “CDTire: a tire model for comfort and durability applications.” *Vehicle System Dynamics*, 45(S1): 69-77, 2007.
32. Ming, L., and Gottschalk, S., “Collision Detection Between Geometric Models: A Survey.” presented at IMA Conference on Mathematics of Surfaces, 1:602-608, USA, 1998.

Acknowledgments

We gratefully acknowledge the European Commission for their support of the Marie Curie IAPP project “INTERACTIVE” (Innovative Concept Modelling Techniques for Multi-Attribute Optimization of Active Vehicles), with contract number 285808; see http://www.fp7interactive.eu. We also want to thank Marcos Novalbos, Alvaro Perez and Pilar Rodriguez for their support. Furthermore, we kindly acknowledge IWT Vlaanderen for their support of the ongoing research project “Model Driven Physical Systems Operation - MODRIO”, which is part of the ITEA2 project 11004 “MODRIO” (in turn, supported by the European Commission).

ISSN 0148-7191

http://papers.sae.org/2014-01-0200