CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Conference Paper

A Closer Look into the AER Model

Cláudio Maia
Luis Nogueira
Luis Miguel Pinho
Daniel Gracia Pérez

CISTER-TR-160701

A Closer Look into the AER Model

Cláudio Maia, Luis Nogueira, Luis Miguel Pinho, Daniel Gracia Pérez

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Commercial-of-the-shelf based multi-core systems present timing anomalies that cannot be ignored by the real-time systems community due to their unpredictable behaviour. These timing anomalies, often caused by applications' uncontrolled accesses to shared resources such as the components in the memory hierarchy or in the I/O subsystem, introduce interference that may lead to deadline misses if the problem is neglected.The Acquisition Execution Restitution (AER) execution model was previously proposed to circumvent this problem and, therefore, mitigate inter-task interference. In this model, applications decouple communication (acquisition and restitution phases) from the actual execution in a way that at most one acquisition or restitution phase is in execution at any instant of time while the execution phase of different tasks can progress in parallel on multiple cores. Thus, keeping each task's derived worst-case execution time closer to the one measured in isolation. In this paper, we study the AER execution model and compare it against a global Earliest Deadline First (EDF) approach where interferences are considered. Our results show that a priority assignment heuristic which assigns the priorities based on the tasks' periods dominates all the other proposed heuristics and that due to interference it can also schedule task sets which are not schedulable by using the global EDF approach.

A Closer Look into the AER Model

Cláudio Maia, Luis Nogueira, Luis Miguel Pinho
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto
Email: {crrm, lmn, lmp}@isep.ipp.pt

Daniel Gracia Pérez
Thales Research & Technology, Palaiseau, France
Email: daniel.gracia-perez@thalesgroup.com

***Abstract*—Commercial-of-the-shelf based multi-core systems present timing anomalies that cannot be ignored by the real-time systems community due to their unpredictable behaviour. These timing anomalies, often caused by applications' uncontrolled accesses to shared resources such as the components in the memory hierarchy or in the I/O subsystem, introduce interference that may lead to deadline misses if the problem is neglected.**

The Acquisition Execution Restitution (AER) execution model was previously proposed to circumvent this problem and, therefore, mitigate inter-task interference. In this model, applications decouple communication (acquisition and restitution phases) from the actual execution in a way that at most one acquisition or restitution phase is in execution at any instant of time while the execution phase of different tasks can progress in parallel on multiple cores. Thus, keeping each task's derived worst-case execution time closer to the one measured in isolation.

In this paper, we study the AER execution model and compare it against a global Earliest Deadline First (EDF) approach where interferences are considered. Our results show that a priority assignment heuristic which assigns the priorities based on the tasks' periods dominates all the other proposed heuristics and that due to interference it can also schedule task sets which are not schedulable by using the global EDF approach.

I. Introduction

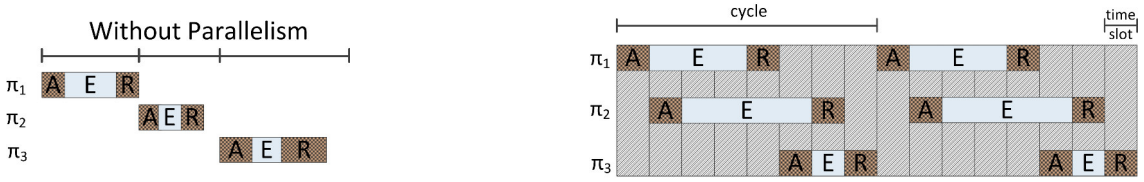
Maximizing application performance is of major importance in the safety-critical domain. Software providers always want to improve their products, and to keep up with the increasing complexity of applications new platforms are always under consideration. In the past, the increase in frequency seen in single-core processors was enough to accommodate new software features. However, the recent shift seen in hardware, where current commercial-of-the-shelf (COTS) architectures no longer include single-core processors but instead incorporate multi-core processors in their designs, made providers think about other issues. Namely, the increase in performance that could be attainable when using multi-cores or even the fact that long term support is required at the hardware level for some of their products (due to the obsolescence of single-core platforms).

Another important feature about using multi-core architectures is the possibility of harnessing application parallelism with an expected increase in application performance. However, parallelism comes at the cost of decreasing the determinism and predictability of the system, requirements of utmost importance in the safety-critical domain. More precisely, these multi-core COTS-based architectures are designed for average-case performance and resources such as memory components (i.e., main memory and cache), peripheral devices and buses are shared among the cores. Applications running concurrently in these systems compete for shared resources and, if care is not taken, they may introduce interference between the accesses which may lead to timing deviations from the worst-case execution times (WCET) computed in isolation. A good example is the contention that occurs at all levels of the memory hierarchy of a traditional multi-core system. That is, without deterministic arbitration mechanisms to handle the memory requests, functionally independent tasks executing in different cores may interfere between each other whenever they need to fetch data from the main memory simultaneously.

One possible solution to minimize the contention in the shared resources is to decouple the accesses to memory from the actual application execution and guarantee that these accesses are performed exclusively in isolation. This can easily be ensured by using a time division multiple access (TDMA) based protocol that enforces timing isolation among the tasks in the system. This method is well-known in the avionics domain as standards such as ARINC-653 [1] mandate that resource partitioning should be used so that applications execute temporally and spatially isolated and therefore do not affect applications executing in other partitions (fault containment). The Acquisition Execution Restitution (AER) model proposed in [2] and depicted in Figure 1b follows this approach in order to increase the predictability of applications executing in COTS-based platforms.

In the AER model tasks are divided into three distinct phases, namely **Acquisition** (A) and **Restitution** (R) which are communication phases (i.e., phases in which accesses to the memory are performed) and a single **Execution** phase (E) which is a computation-only phase. This model is an evolution from a model with no parallelism, where tasks execute sequentially in each TDMA slot according to a predefined schedule (as depicted in Figure 1a), to a model that considers the parallel execution of tasks. More precisely, this model introduces two advantages: (1) it is an interference-free model and (2) it supports parallel task execution. First, memory contention related issues (such as interference) are avoided by enforcing that communication phases execute exclusively by having a single communication phase (A or R) executing at any time instant. Second, by decoupling communication phases from the execution phase it is possible to exploit parallelism as execution phases of different tasks can now execute in parallel with any other phase of any other task executing in the system.

978-1-5090-1314-2/16$31.00



(a) Sequential model using TDMA time slots

(b) AER model

Fig. 1: Execution Models

In this paper we empirically explore the model proposed in [2]. To the best of our knowledge there is no worst-case response-time schedulability test that allows one to test the schedulability of a task set composed of tasks following the AER model for multi-core systems, as proposed in [2]. The authors in [3] solve this problem by transforming it into a constraint programming problem and obtaining a completely static solution for an instance of the problem. While such work is important, the complexity of the problem still remains intractable. Without a worst-case response-time schedulability test one has to test all the possible schedules to infer any information about the schedulability details of any task set. Therefore, to overcome this limitation we have implemented a simulator to study the behaviour of this type of tasks with regards to a number of metrics such as average response-time, maximum response-time, among others. Moreover, we present different priority assignment heuristics in order to infer the implications of using different ordering schemes in the scheduling of AER tasks and we compare them against a global Earliest Deadline First (EDF) approach where interference is considered in order to simulate the behaviour of these tasks in a COTS system.

Contributions. The contribution of this work is threefold: (1) we present a tool that allows us to study the behavior of AER tasks in multi-core systems; (2) opposed to a static approach to solve the schedulability problem of AER tasks we provide different priority assignment heuristics to schedule task sets composed by AER tasks; (3) we compare these heuristics against a global EDF approach that considers inter-task interference.

Paper organization. The rest of this paper is organized as follows. Section II presents the related work. Section III describes the model of computation used throughout the paper. Section IV details the proposed heuristics and some important aspects about our simulator. Section V presents the configuration settings used to collect the data. Section VI reports on the simulation results from experiments conducted on synthetic task sets. Finally, Section VII concludes the paper and presents the perspectives.

II. Related Work

Deterministic architectures (MERASA [4], PRET [5]) consisting of hardware mechanisms to control inter-core interference have already been proposed in the past. Nevertheless, most of this type of solutions target specific hardware platforms and cannot be applied/found in general COTS platforms leading the stakeholders to study solutions that can be applied at the application level such as the AER model.

The AER model [2] is a generalization of the PRedictable Execution Model (PREM) [6]. PREM considers a single core environment where tasks are divided into predictable intervals: memory phases and execution phases. Memory phases access main memory and deal with the needed cache operations while in the execution phases a task avoids accessing the main memory and therefore minimizing any possible interference.

In [3] the authors present an interference-free model by proposing a constraint programming based solution for the AER model executing in multi-core environments. In that work, the authors show that executing tasks in a multi-core environment leads to increases in the WCET measured in isolation of up to 3x the value in isolation. A similar observation was made in [7] where the authors evaluate the effects of having multiple applications of different criticality levels executing in a multi-core platform. More precisely, the authors observed a maximum slowdown of 5.1x in application execution when multiple cores access network and memory concurrently. Both of these results show that special care must be taken when executing safety-critical applications in multi-core platforms due to the increase in WCET as a result of interference related to concurrent accesses to shared resources.

In [8] the authors introduce the concept of Deterministic Platform Software (DPS) with the objective of identifying pieces of software that can be used to enforce usage domains for which interference levels are known and acceptable. In the paper, AER model is identified as a Deterministic Execution Model as it ensures that interference due to concurrent accesses is avoided. Moreover, in that paper the authors also propose comparison criteria to evaluate how the identified DPS solutions are relevant for industrial application.

Other software oriented solutions exist with the objective of solving the interference problems due to shared resources in multi-core systems. While these models are not a direct application of the AER model, they somehow relate to the approach that is followed by AER. For instance, in [9] the author proposes a model in which critical and non-critical applications execute in the same multi-core platform. However, whenever a critical application is executing only the core that is executing it is allowed to run while the others remain forcefully idle. The main limitation of this model arises when several critical applications execute all together in the platform thus making all the cores but one idle most of the time.

Another solution which also presents restrictions on the access to the bus is the one presented in [10] where the authors propose an approach that restricts access to the buses by the usage of TDMA. The idea behind this solution is that an application accesses memory when it is allowed by the TDMA schedule. If it tries to access memory outside of its assigned TDMA schedule, then it is blocked until its time slot becomes available. Several solutions employ similar mechanisms but what is important to retain is that most multi-core solutions for safety-critical domains apply the principle of exclusivity in order to reduce the interference within the system.

Recently, the authors in [11] propose an operating system design for multi-core COTS platforms that integrates scratchpad memories in their architecture designs. Their approach combines the AER task model with a TDMA bus access schedule on a system with scratchpad memories and DMA support. Our tool allows us to study the AER task model in a more generic way and considering different settings.

III. System Model

We consider a system composed of a set Π of m cores, $\Pi = \{\pi_1, \pi_2, ..., \pi_m\}$. The system executes a task set composed of n periodic tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ where each task τ_i in the set is divided into three distinct phases that execute nonpreemptively: acquisition (A_i), execution (E_i) and restitution (R_i).

During an acquisition phase (A) an application exclusively accesses the main memory to fetch data into the core's local memory (e.g., scratchpad, L1 cache). The exclusive access is required so that this phase does not interfere with any other task in the system. At the end of this phase all data needed by the application is loaded into the core's local memory and the application can start its execution phase (E). In the execution phase (E) a task executes without suffering any interference from any other task in the system as the needed data for execution was previously loaded into the core's local memory and therefore no main memory accesses are performed during this phase. Finally, after the task finishes its execution phase, the restitution phase (R) exclusively accesses main memory so that all data that was modified during execution is stored back into it.

Due to the fact that execution phases do not require any access to main memory, these phases can execute in parallel with any other phases of other tasks executing in the system without interfering with their execution. However, acquisition and restitution phases must exclusively access main memory and therefore *cannot* execute in parallel with any other A or R phases.

Let e_i^a, e_i^e and e_i^r denote the maximum execution time of A_i, E_i and R_i phases of task i, respectively. Then, the worst-case execution time of task i in isolation (without suffering any kind of interference) is given by the sum of the execution times of each phase: $e_i = e_i^a + e_i^e + e_i^r$. It is important to note that for the model to work each phase must finish within its allocated amount of time. This aspect should be enforced so that predictability is not jeopardized.

Each task is characterized further by a period T_i, and a constrained-deadline $D_i \leq T_i$. For schedulability, the workload of each task should be no greater than its corresponding deadline: $C_i \leq D_i$. The *utilization* of task τ_i is given by $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$ and the *total utilization* of the task set τ is $U_\tau \stackrel{\text{def}}{=} \sum_{i=1}^{n} U_i$. Core utilization should not exceed 100% and consequently, the total system utilization should be no greater than the number of cores in the system: $U_\tau \leq m$.

Within a schedule, the release pattern of tasks repeats itself in intervals of time named major cycles, denoted by Γ, which are equal to the least common multiple of all the tasks in the task set.

IV. Proposed Heuristics

To the best of our knowledge there is no method that can be used to evaluate the schedulability of tasks that comply with the AER task model. Without having a schedulability test, one possible way to test the schedulability of a task set is to generate the actual schedule over the hyper-period interval of the task set under consideration and validate if some deadline miss occurs at a particular time instant within this interval. In order to circumvent this limitation and still make it possible to study the behaviour of the AER tasks we have implemented a simulator that generates schedules considering the task model described in Section III.

The authors in [3] formalize the scheduling problem as a constraint programming problem to which they provide as inputs the application timing requirements and the platform model, and as outputs the solver returns a static mapping of tasks to cores. Opposed to the work done by these authors, our simulator considers a dynamic scheduling approach (the next task to execute is selected among the tasks that are in the ready task queue) instead of a static one.

Multi-core scheduling is proven to be a NP-hard problem [12] and therefore without testing all the possible combinations of task orderings, there is no possibility of knowing which ordering leads to a schedule where all tasks are schedulable and the length of the schedule is minimized. Nevertheless, heuristic approaches can be used in order to try to obtain a feasible solution to the problem in a reasonable amount of time.

One of the goals of using our simulator is to study the behaviour of AER tasks when scheduled under different priority assignment rules. This study includes the exploration of different task allocations in order to observe if any rule performs better than the others. The supported assignment rules are the following:

- **Priority**: The priority of each task in the task set is given by the period T in ascending order, that is tasks with smaller periods have higher priorities in a similar fashion to the behaviour of the Rate Monotonic algorithm [13].
- **Minimum Acquisition**: The priority of each task is given by the length of the Acquisition phase of a task where the task with the smaller Acquisition execution value is the one selected for execution.

- **Maximum Acquisition**: Similar to the previous rule but in this case the priority is given to the task with the largest Acquisition value.
- **Minimum Restitution**: This rule considers the tasks according to the size of their Restitution phases where the priority is given to the task with smaller Restitution value among all tasks in the ready queue.
- **Maximum Restitution**: Similar to the previous rule but in this case the priority is given to the task with the largest Restitution value.

In order to understand how the above-mentioned rules compare to an approach that is not interference-free we also have added to the simulator an implementation of global Earliest Deadline First (EDF) algorithm. Standard EDF algorithm gives priority to the job that has the earliest absolute deadline among all the jobs in the ready queue. Global (as opposed to partitioned scheduling) in this context means that the scheduling algorithm allows any job to execute in any core of the system.

Our implementation of global EDF works with a copy of an AER task set where, instead of having each task composed of three distinct phases, each task in the set is converted into a traditional task in which all phases are merged together and, in this study, its WCET reduced by a certain amount. The motivation for this reduction in WCET relies on the fact that the AER model requires traditional tasks to be modified in order to accommodate the code necessary for the decoupling of communication phases from execution phases (decoupling the phases allow us to achieve an interference-free deployment as explained previously). Intuitively, by transforming a traditional task into an AER task there is an increase in the WCET of each AER task due to the extra code needed for phase decoupling. Therefore, for fairness in comparison of both task models (AER tasks and traditional tasks) we apply a reduction to the WCET when converting the AER task into a traditional task to be tested under global EDF.

The reduction factor is an input parameter of the simulator and the user can modifiy it to increase or decrease the traditional task execution time under global EDF, dependending on the system under study and/or the task transformation technique applied.

Moreover, we assume that when a task executes under global EDF the task executes in an interference-prone platform. This assumption intends to model task execution in current multi-core COTS platforms where tasks suffer interference whenever concurrent accesses are made to shared resources. To represent the lack of temporal and spatial isolation that exists in COTS platforms and in traditional task models, we artificially slowdown task execution by a slowdown factor under global EDF whenever tasks execute in parallel with other tasks in the system.

There are other important aspects that should be mentioned about the behaviour of our simulator/model. Specifically, the scheduler is omniscient in the sense that it knows the state of every core at a given time instant t when it needs to allocate a task.

For any given priority assignment rule (except global EDF), the next task to execute is selected from the ready queue and it is assigned to the first idle core that the scheduler finds as long as no other core is executing a memory phase (A or R phases). Task phases start their execution as soon as possible and execute non-preemptively until the end of the phase. Each execution phase (E) starts executing as soon as the preceding A phase completes its execution and it may execute in parallel with any other task's phase of other tasks. Moreover, E phases do not suffer any kind of direct interference or blocking[1] in the sense that while an A phase can suffer interference and subsequently an E phase can also suffer interference, an E phase never waits to execute once it becomes ready. Interference and blocking need to be considered in our model not only in a single core but also across all cores in the system due to the non-preemptive execution of phases. For instance, there are cases when a memory phase of a task is ready to execute in a core but it cannot due to a task that is executing a memory phase somewhere in another core of the platform (this can happen either with a higher or lower priority task than the ready task).

Moreover, once a task is assigned to a core it remains executing exclusively on that core until all phases complete their execution. The intuition behind this behaviour lies on the fact that the memory phases fetch/store data from the main memory and once they do it a core can execute without suffer/induce interference on the shared resources. Having a task moving from a core to another would improve system utilization but at the same time would break the principle of avoiding interference and cause additional overheads due to the task migration between cores. The same reasoning applies if preemptions were allowed after a task started its execution.

Our implementation of global EDF considers traditional tasks (as described above) and therefore there are no distinct phases within a task. Moreover, the m higher priority tasks are the ones executing in the platform and as it typically occurs in global EDF, preemptions are allowed due to the arrival of a higher priority task into the system. In our simulator we do not consider any overhead related to such preemptions. Higher priority tasks may interfere with lower priority tasks but a lower priority task may never block a higher priority task.

V. Experimental Settings

Our tool allows us to better study the behaviour of AER tasks with respect to a certain number of metrics while varying several parameters. In this section we describe the most important parameters that the simulator accepts and provide the reasoning behind the values' selection.

As input parameters the simulator accepts the number of cores m, the number of tasks n that the user wants to use per

[1]Interference occurs whenever a phase of task is ready to execute but it cannot execute due to the execution of other higher priority tasks in the system. Blocking occurs whenever a higher priority task is ready to execute but it cannot execute due to the execution of a phase of a lower priority task in the system.

iteration of the simulation, the minimum and maximum values for the execution times of the phases of an AER task and an array with slowdown values that will be applied when testing the global EDF schedulability of a task set, among others.

The task generation in the simulator works as follows. Each (outer) iteration starts with a task set with n tasks and then a new task set is created from the previous one by iteratively adding a new task into it (in an inner iteration). This iterative procedure stops when the new task set is deemed unschedulable by all the heuristics under test including global EDF, point at which a new outer iteration starts again.

The parameters' values that we have used in the generation of the tasks are the following. Each phase has an initial value in the range $[1, 3]$ which is multiplied by a factor in order to test different configurations of phase sizes. The WCET for the AER task is the sum of the individual phases' execution times as presented in Section III. The period of each task is then generated by randomly choosing a factor value in the range $[2, 4]$ which is then multiplied by the generated WCET in order to obtain the task period. The deadline for each task is always equal to the task period.

The above described procedure works for the generation of AER tasks. For global EDF tasks (for the sake of clarity let us denote this tasks as traditional tasks), the simulator converts each generated AER task into a traditional task in which the respective AER WCET is reduced by a certain amount defined as an input parameter. In our simulations we have chosen to reduce the WCET of an AER task by 25%. As explained in Section IV, this reduction intends to mimic the increase in task execution time when a traditional task is converted into an AER task due to the addition of the code needed to enforce phase decoupling.

Another important aspect that should be considered is the interference that may occur in a COTS platform due to the usage of shared resources. In [7] the authors observed a maximum slowdown of 5.1x in application execution when multiple devices access network and memory concurrently in a platform of $m = 4$ cores. In the same line of research, the authors in [3] state that the slowdown that a task suffers when moving from a single-core configuration to a multi-core configuration is of 2.7x in a cached-based version of a multi-core platform for $m = 6$. Therefore, in order to emulate this behaviour when scheduling tasks under global EDF (AER model is an interference free model therefore it does not need to be considered under this scenario) we apply a slowdown value to each traditional task that is dependent on the number of tasks that execute in parallel in the same time unit.

Using the values reported in [3] as a reference, we have computed the direct proportion of slowdown values up to $m = 4$. That is, the slowdown value for m cores is given by $S_m = \frac{m \cdot 2.7}{6}$ for $m \geq 2$. Therefore, the input list of slowdown values used is the following $[1, 1, 1.35, 1.8]$ up to $m = 4$. Each value in the list represents the factor of increase in WCET that a traditional task will incur due to interference when it executes in parallel with other tasks in the system in a given time unit. For instance, when a task executes in parallel with two other tasks the increase in its WCET will be 1.35x per time unit. Thus, instead of taking 1 time unit to execute, the considered task takes 1.35 time units due to interference. A special remark must be made concerning the first two values in the list: the first value represents the time in isolation (without any interference); while the second value should be 0.9 after applying the above formula which leads to a speedup in execution instead of a slowdown. Our decision was to round this value to the closest integer[2]. The slowdown value is one of the parameters that we vary in our simulations so that we can evaluate its impact on the schedulability of tasks.

The parameters' values used in the generation are smaller due to the fact that we actually generate the schedules throughout the hyperperiod of the task set in order to test their schedulability. Thus, the larger the values of task periods the larger the hyperperiod will be and therefore the longer the simulator needs to execute in order to obtain the results. We consider that this does not pose any limitation over the collected data as the properties still hold for smaller values, as for instance the relation between the lengths of each phase (e.g., A phase size larger than R phase).

As output, the simulator generates a schedule for each of the heuristics and global EDF, the average response-times and maximum response-times for all the tested task sets, and a graph with the number of schedulable task sets per heuristic per utilization value.

VI. Discussion

In this section we present the outcomes of the simulator and a discussion of the results obtained for the scheduling of randomly generated task sets under the different proposed heuristics described in Section IV.

The simulator allows us to observe the average and maximum response-times of a given task set, as depicted in Figure 2 and Figure 3. In the figures it is possible to see a task set composed of 4 tasks being executed in a platform with $m = 4$ cores and a slowdown value for global EDF of 2 times the values in the slowdown list presented above. The average response-time is computed by summing up the individual values of response-times of each instance and then dividing this value by the number of instances of each task.

While the response-time is a useful metric for understanding the specific properties of a given task set, it is not a good metric for understanding how the heuristics behave, as it is too dependent on the values of the task set. Therefore, another metric that can be used for heuristic comparison is the number of schedulable task sets that each heuristic can schedule from a given number of randomly generated task sets. Therefore, in the experiments presented in Figures 4, 5, 6, 7 , 8, and 9, we evaluated the amount of schedulable task sets (y-axis) per

[2]Our simulator works with discrete time units which means that even though the slowdown values are floating point, after computing the results the values will be rounded to the closest integer value. We believe that this is a design decision that mimics the real systems and that does not affect the analysis of resulting data.

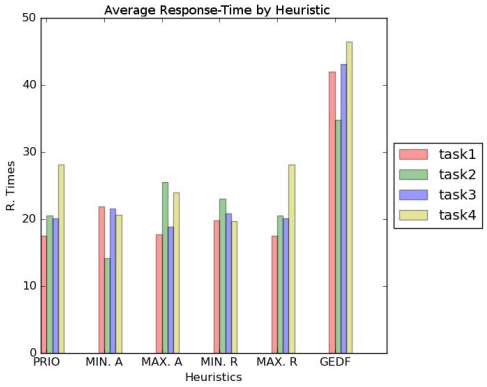


Fig. 2: Average response-time observed in a task set with 4 tasks in a platform with 4 cores and slowdown values of 2x

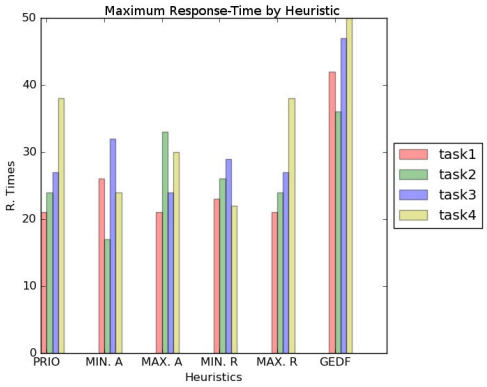


Fig. 3: Maximum response-time observed in a task set with 4 tasks in a platform with 4 cores and slowdown values of 2x

utilization value (x-axis) for 2000 randomly generated outer iterations of the simulator using different settings.

Concerning the slowdown values used in these experiments, we have chosen slowdown values of 1.5x and 2x the values in the slowdown list presented above (the slowdown list values become $[1, 1.5, 2.03, 2.7]$ and $[1, 2, 2.7, 3.6]$ for 1.5x and 2x respectively).

For each value of m we varied the task parameters. For $m = 2$ we used two settings: (1) a setting where R phases are larger than A phases (Figure 4) while the E phase is larger than both; (2) a setting where E phases are smaller than both A/R phases (Figure 5). Both with a slowdown value of 1.5x.

For $m = 4$ we used four settings: (1) A phases are larger than R phases while the E phase is larger than both with a slowdown of 1.5x (Figure 6); (2) R phases are larger than A phases while the E phase is larger than both with a slowdown of 1.5x (Figure 8); (3) E phases are smaller when compared to both A/R phases with a slowdown of 1.5x (Figure 7); (4) A phases are larger than R phases while the E phase is larger than both with a slowdown of 2x (Figure 9).

Our goal with these experiments is to observe the behaviour of each of the heuristics when different configurations of phase lengths of an AER task are used; observe how the global EDF approach compares with the proposed heuristics and see how it is affected by interference when the number of cores increases and different slowdown values are applied.

Several observations can be made by looking into the figures. In the figures one can see 6 lines, one for each of the heuristics (namely, priority (PRIO), minimum A (MIN.A), maximum A (MAX.A), minimum R (MIN.R), maximum R (MAX.R), as detailed in section IV) and one for global EDF (G-EDF).

First Observation, among the proposed heuristics the one that uses the period as the priority assignment rule (PRIO in the figures) dominates the other heuristics in terms of schedulable task sets. This can be seen in all of the figures.

Second Observation, when A and R phases are similar in length the heuristics that use the length of A or R phases as a priority criterion (i.e., MIN.A, MIN.R, MAX.A and MAX.R) schedule a similar amount of task sets. This can be seen in Figure 5 and Figure 7 for two and four cores respectively. Obviously, even though the heuristics use different criteria to select the next task to schedule, as the phases have the same length the outcome will be the same for the heuristics that use either the minimum phase (MIN.A and MIN.R) or the maximum phase (MAX.A and MAX.R) lengths.

Third Observation, in all of our experiments the heuristics that use the minimum length of A or R schedule more task sets than the ones that use the maximum length of A or R phases. The most probable reason for this behaviour may be related to the fact that selecting the tasks with larger A or R phases first induces a larger amount of interference/blocking on the other tasks in the ready queue when compared to the heuristics that use the minimum length of A or R phases (recall that ready A or R phases cannot execute until other A or R phases complete their execution). This leads to larger response-times in average and therefore to non-schedulability of task sets when the maximum heuristics are applied due to the response-times being larger than the task's deadline.

Fourth Observation, one can easily see that as the values of slowdown increase (Figure 6 and Figure 9) the number of task sets that are schedulable by global EDF drastically decreases as well as the utilization of the schedulable task sets (in the best case it decreases from 2.1 in the 1.5x slowdown setting to 1.6 in the 2x slowdown setting). Recall that increasing the slowdown means that each task takes longer to execute due to the amount of interference that it suffers when executing in parallel with other tasks. This observation is important because it shows that when the number of devices competing for shared resources increase then the interference increases which leads to larger execution times than what is estimated in isolation and therefore to unschedulable task sets. This effect can also be seen in Figure 2 and Figure 3 either in terms of average and maximum response-times where tasks under global EDF take longer to execute than the proposed heuristics.

Fifth Observation, for a smaller number of cores ($m = 2$) global EDF behaves better than any of the proposed heuristics even when considering a slowdown of 1.5x. Nevertheless, when the number of cores increase global EDF will behave

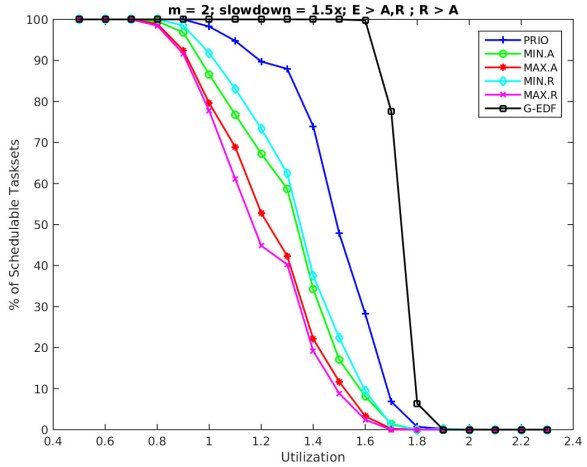


Fig. 4: Number of schedulable task sets per utilization value in a setting of $m = 2$ cores and slowdown values of 1.5x. Task parameters: $A = 1 \cdot \text{random}(1,3), E = 4 \cdot \text{random}(1,3), R = 2 \cdot \text{random}(1,3)$[3]

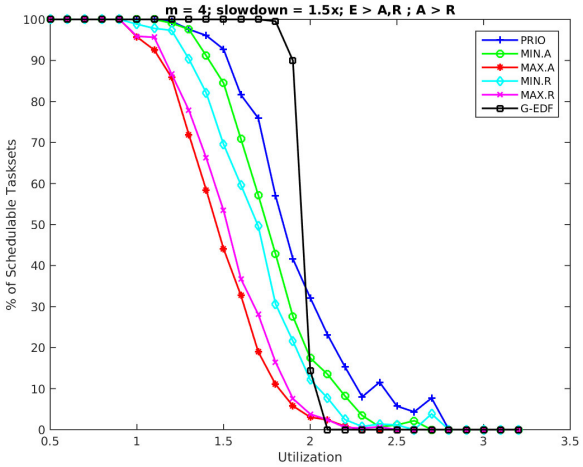


Fig. 6: Number of schedulable task sets per utilization value in a setting of $m = 4$ cores and slowdown values of 1.5x. Task parameters: $A = 2 \cdot \text{random}(1,3), E = 4 \cdot \text{random}(1,3), R = 1 \cdot \text{random}(1,3)$

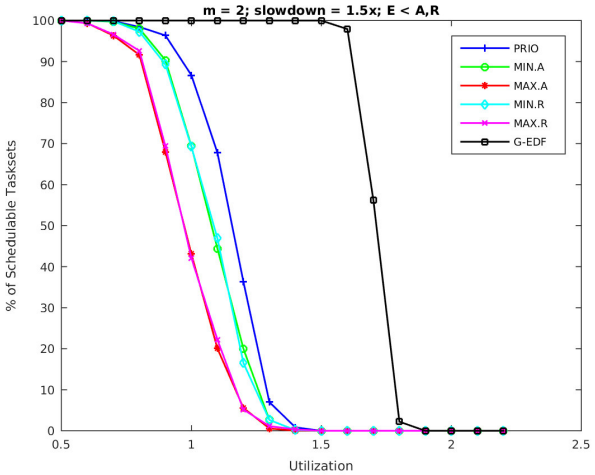


Fig. 5: Number of schedulable task sets per utilization value in a setting of $m = 2$ cores and slowdown values of 1.5x. Task parameters: $A = 2 \cdot \text{random}(1,3), E = 1 \cdot \text{random}(1,3), R = 2 \cdot \text{random}(1,3)$

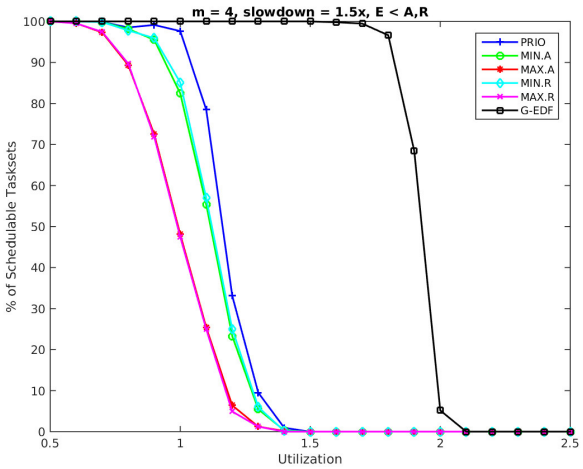


Fig. 7: Number of schedulable task sets per utilization value in a setting of $m = 4$ cores and slowdown values of 1.5x. Task parameters: $A = 2 \cdot \text{random}(1,3), E = 1 \cdot \text{random}(1,3), R = 2 \cdot \text{random}(1,3)$

worst due to the interference increase when tasks execute in parallel as explained in the previous observation.

Sixth Observation, when the execution phases are smaller than memory phases as in Figure 5 and Figure 7 all the proposed heuristics behave poorly because the tasks will not benefit from any parallelism and in fact most of the phases will execute in a sequential way in the majority of the time. It can be seen from both figures that even if the number of cores increase (from $m = 2$ to $m = 4$) the schedulability of the task sets remains nearly the same.

[3] $x \cdot \text{random}(1,3)$ means that a factor x is multiplied by random value generated in the range $[1,3]$

VII. Conclusion

In this paper we empirically explored the AER model by the means of a simulator in order to circumvent the lack of a worst-case response-time schedulability test that allows one to test the schedulability of a task set composed of tasks following the AER model for multi-core systems.

Our simulator allows us to reason about the behaviour of AER tasks with regards to a number of metrics such as average response-time and maximum response-time and considering different priority assignment heuristics. Moreover, we have compared the allocation heuristics against a global scheduling approach (global EDF) where interference is considered in

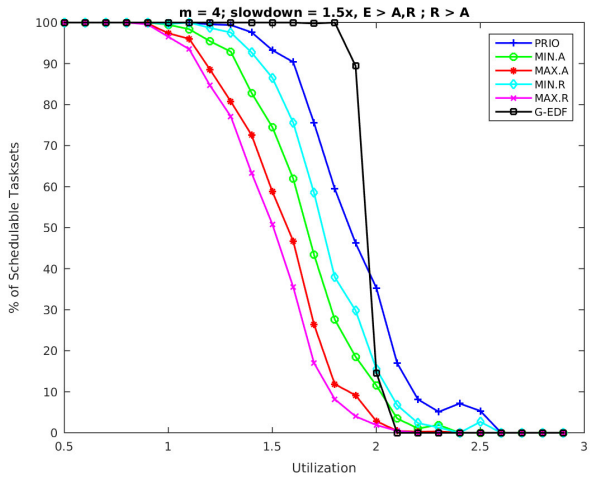


Fig. 8: Number of schedulable task sets per utilization value in a setting of $m = 4$ cores and slowdown values of 1.5x. Task parameters: $A = 1 \cdot \text{random}(1, 3), E = 4 \cdot \text{random}(1, 3), R = 2 \cdot \text{random}(1, 3)$

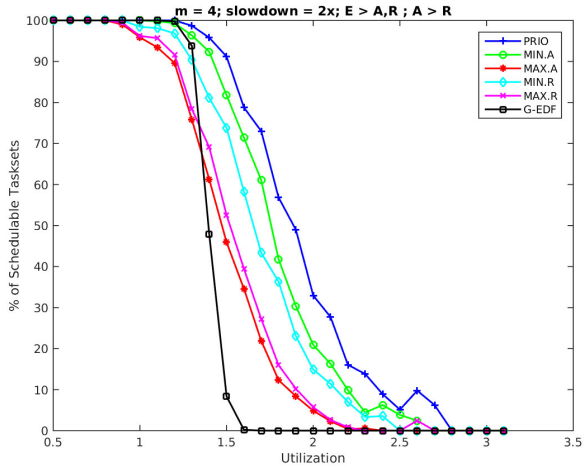


Fig. 9: Number of schedulable task sets per utilization value in a setting of $m = 4$ cores and slowdown values of 2x. Task parameters: $A = 2 \cdot \text{random}(1, 3), E = 4 \cdot \text{random}(1, 3), R = 1 \cdot \text{random}(1, 3)$

order to simulate the behaviour of tasks in a COTS system. Our results show that a priority assignment heuristic which assigns the priorities based on the tasks' periods dominates all the other proposed heuristics and that due to the interference it can also schedule task sets which are not schedulable using global EDF. These observations show that the simple version of the AER model can be useful in today's multi-core architectures, and from a safety point of view this solution is actually probably more viable than some other scheduling approaches that are interference-prone as global EDF.

Concerning the future work, we would like to explore the potential of the simulator to generate new metrics such as average allocation per heuristic and study even further the scalability of the approaches presented.

Acknowledgments

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2); also by the HIPEAC Network of Excellence, under the 7th Framework Programme under the project number ICT-287759; also by FCT/MEC and the ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH / BD / 88834 / 2012.

References

[1] ARINC., "Arinc specification 653: Avionics application software standard interface," 2005.

[2] G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in *Embedded Real Time Software and Systems (ERTS'14)*, Toulouse, France, Feb. 2014.

[3] S. Girbal, D. Gracia Pérez, J. Le Rhun, M. Faugère, C. Pagetti, and G. Durrieu, "A complete tool-chain for an interference-free deployment of avionic applications on multi-core systems," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, Sept 2015, pp. 1–13.

[4] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66–75, 2010.

[5] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '08. New York, NY, USA: ACM, 2008, pp. 137–146. [Online]. Available: http://doi.acm.org/10.1145/1450095.1450117

[6] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, April 2011, pp. 269–279.

[7] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *Dependable Computing Conference (EDCC), 2012 Ninth European*, May 2012, pp. 132–143.

[8] S. Girbal, X. Jean, J. Le Rhun, D. Gracia Pérez, and M. Gatti, "Deterministic platform software for hard real-time systems using multi-core cots," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, Sept 2015, pp. 8D4–1–8D4–15.

[9] S. Fisher, "Certifying Applications in a Multi-Core Environment: The World's First Multi-Core Certification to SIL 4," SYSGO AG, Tech. Rep., 2013.

[10] X. Jean, D. Faura, M. Gatti, L. Pautet, and T. Robert, "Ensuring robust partitioning in multicore platforms for ima systems," in *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, Oct 2012, pp. 7A4–1–7A4–9.

[11] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric os for multi-core embedded systems," in *22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2016)*, Vienna, Austria, April 2016.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[13] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: http://doi.acm.org/10.1145/321738.321743