IA^3: An Interference Aware Allocation Algorithm for Multicore Hard Real-Time Systems

Marco Paolieri
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
e-mail: marco.paolieri@bsc.es

Eduardo Quiñones
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
e-mail: eduardo.quinones@bsc.es

Francisco J. Cazorla
CSIC-IIIA and BSC
Barcelona, Spain
e-mail: francisco.cazorla@bsc.es

Robert I. Davis
University of York
York, UK
e-mail: robdavis@cs.york.ac.uk

Mateo Valero
Universitat Politècnica de Catalunya and BSC
Barcelona, Spain
e-mail: mateo@ac.upc.edu

***Abstract*—In multicore processors, the execution environment is defined as the environment in which tasks run and it is determined by the hardware resources they get and the workload with which they are executed. Thus, different execution environments lead to different inter-task interferences accessing shared hardware resources due to conflicts with the other co-running tasks, making the WCET estimation of a task dependent on the execution environment in which it runs. Despite such dependency, current partitioned scheduling approaches use a single WCET estimation per task: typically the highest for all execution environments in which a task runs.**

In this paper we introduce IA^3: an interference-aware allocation algorithm that considers not a single WCET estimation but a set of WCET estimations per task. IA^3 is based on two novel concepts: the WCET-matrix and the WCET-sensitivity. The former associates every WCET estimation with its corresponding execution environment. The latter measures the impact of changing the execution environment on the WCET estimation. This allows IA^3 to reduce the number of resources required to schedule a given taskset.

In particular, our results show that in a four-core processor considering tasksets with a total utilization of 2.9, IA^3 is able to schedule 70% of the tasksets using 3-cores while a classical partitioned approach with a First-Fit Decreasing heuristic is able to schedule only 5% of the tasksets using 3-cores.

I. Introduction

The increasing demand for new functionality in current and future hard real-time embedded systems is driving an increment in the performance requirements of embedded processors [1]. Multicore processors are being considered as an effective solution for embedded systems due to their low cost and good performance-per-watt ratio, while maintaining a relatively simple design inside each core. In addition, multicore processors allow scheduling of *mixed criticality workloads*, i.e. workloads composed of safety and non-safety critical applications[1], inside the same processor, maximizing the hardware utilization and so reducing cost, size, weight and power consumption.

[1]In this paper we use the terms *application*, *thread* and *task* interchangeably.

Unfortunately, the use of multicore processors in hard real-time systems is limited by the fact that it is much harder to perform Worst Case Execution Time (WCET) analysis than for single-core processors even when using non-preemptive scheduling as considered in this paper. This is due to *inter-task interferences*[2] generated by tasks running in different cores. Inter-task interferences appear when two or more tasks that share a hardware resource try to access it at the same time, so arbitration is required to select which task is granted access to such a shared resource, potentially delaying the execution time of the others. This makes the WCET of a task dependent on the set of inter-task interferences introduced by the co-running tasks that have the highest impact on the execution time.

The set of inter-task interferences that affect the execution of a task is determined by the *execution environment* in which the task runs. The execution environment is defined as the environment in which the task is executed; in a multicore processor, it depends on both the workload and the hardware configuration. Regarding *workload*, the scheduling/allocation algorithm determines the tasks that will execute simultaneously, defining the patterns of access to shared resources. Regarding the hardware configuration, in some architectures [2], [3], the hardware provides the software with some 'levers' to configure the arbitration policy of buses, the amount of cache assigned to each task, etc. Hence, in such architectures (like [2]), the execution environment of a task depends on the workload with which the task runs and on the hardware configuration. Therefore, the execution of a *Hard Real-time Task* (HRT) under different execution environments leads to different inter-task interference scenarios, and hence different WCETs for the task.

This phenomenon is illustrated by Figure 1. This figure

[2]Note that inter-task interferences may also be generated by multiple tasks running in the same core, e.g. in preemptive scheduling schemes when tasks share the cache. This is not considered in this paper which assumes non-preemptive scheduling. Inter-task interference is thus only caused by tasks running on other cores.

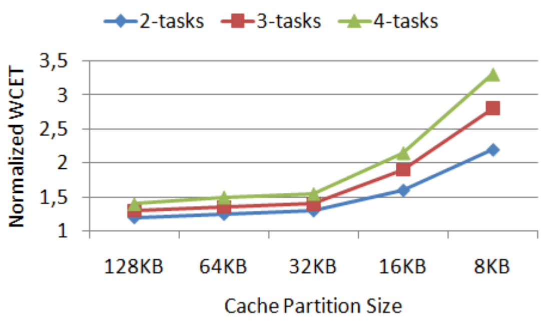


Fig. 1. Normalized WCET estimations of *aifftr* under fifteen execution environments, defined by the number of simultaneous HRTs and the cache partition size assigned to each task, taking as a baseline the WCET estimation computed assuming no inter-task interferences, i.e. running in isolation.

shows the WCET estimation of a memory-intensive EEMBC Automotive benchmark (*aiifft*) when it runs in a 4-core architecture with a private cache partition assigned to each core and a shared memory controller. Thus, inter-task interferences will appear when multiple tasks simultaneously access the memory controller. All WCETs are normalized to the WCET estimation of *aiifft* computed in isolation (i.e. assuming no inter-task interferences). In this architecture we execute *aiifft* under 15 execution environments: we run it simultaneously with 1, 2 and 3 instances of an HRT benchmark called *opponent*, assigning to *aiifft* 5 different cache partitions(from 128 KB to 8 KB). The opponent is a synthetic benchmark that continuously accesses the memory, generating a lot of inter-task interference. For each execution environment we compute a WCET estimation of *aifft* using RapiTime, a measurement-based worst-case execution time analysis tool[4]. We observe that the WCET estimation is affected by both the hardware configuration (the cache partition assigned), and the workload with which *aifft* runs (the number of opponents running simultaneously): It increases as the number of simultaneous opponents (and so inter-tasks interferences) increases and as the cache partition size is reduced because cache misses then occur more frequently.

Despite the impact that the execution environment has on the WCET estimation, most approaches to hard real-time task scheduling consider only a single WCET value for each HRT: the highest among all execution environments in which the HRT can run. Two notable exceptions to this are the works of Kato et al. [5] and Jain et.al. [6], which consider multi-case execution times. The research presented in this paper differs in that our approach focuses on the WCET estimations of each HRT considering only the parameters that determine the execution environment in which a task runs, rather than on estimations of the execution time variations due to details of the various tasks when scheduling them in a simultaneous multi-threading environment. Hence our approach enables system composition based on information about individual components (i.e. applications or tasks) whereas previous work does not.

In this paper we propose IA^3: an off-line *Interference-Aware Allocation Algorithm* that uses a set of WCET estimations corresponding to all the execution environments in which this task may run. IA^3, which is based on the *first fit decreasing heuristic* [7], introduces two novel concepts: the *WCET matrix* and the *WCET-sensitivity*.

The WCET-matrix is an n-dimensional vector space per HRT, where each dimension represents the parameters that determine different execution environments that may affect the WCET of the HRT. Thus, each execution environment has a WCET estimation associated with it. For example, in Figure 1, the WCET-matrix of the *aiifft* task is a 2-dimensional vector space that considers the cache partition size and the number of HRTs in the workload in which *aifft* runs. The WCET-sensitivity complements the WCET-matrix by making the IA^3 algorithm aware of the impact of changing the execution environment on the WCET estimation. The WCET-sensitivity is derived from, and represents the variation across, the WCET-matrix.

The abstraction provided by the WCET-matrix and the WCET-sensitivity allows IA^3 to generate a very efficient partition that reduces the amount of resources (e.g. number of cores, cache partition size assigned to each core, etc.) required to schedule a given taskset. Reducing the number of processors and hence the weight of the system is of significant benefit in many embedded systems, particularly those in avionics and space systems. Moreover, in mixed-criticality workloads, reducing the resources assigned to HRTs, allows the *Non Hard Real-time Tasks* (NRTs) to use more hardware resources and hence increase their quality of service (Note that the architecture is such that NRTs cannot affect the WCET of HRTs).

We illustrate the concept of the IA^3 by using the multicore architecture proposed in [2], [8], which has two main features: (1) the hardware allows the software to configure the size of cache partition assigned to each running task; (2) the inter-task interferences that a HRT may suffer depend only on the *number* of HRTs running simultaneously and not on the *particular* HRTs (see Section II-B for further details).

The rest of the paper is organized as follows: Section II lists some definitions and notational conventions we use in this paper. Section III describes our proposal. In Section IV and V, the test methodology and the results of our experiments are covered. In Section VI additional considerations are exposed. Finally, Section VII lists the related work and conclusions are addressed into Section VIII.

II. Background

A. Definitions

This paper focuses on the *allocation problem* in homogeneous multicore processors, considering a *partitioned* scheduling approach in which once a task has been assigned to one core it is not allowed to migrate. Moreover, we consider a *sporadic task model*, in which the arrival times of jobs of the same task are separated by a minimum inter-arrival time, referred to as the task's period. The arrival times of jobs of different tasks are assumed to be independent. At any point in time a job can execute on at most one core and, at most m HRTs can run simultaneously in an m-core processor.

In general, an allocation algorithm assigns a *taskset* τ composed of n independent tasks $(\tau_0, ...\tau_n)$ to a set of m identical cores $(s_1, .., s_m)$. Each task, τ_i, is characterized by a WCET estimation C_i, a period P_i, and a hard deadline D_i, assuming *implicit deadlines*, i.e. $D_i = P_i$. The utilization of a task, u_i, is defined as $\frac{C_i}{P_i}$ and it ranges between $0 \leq u_i \leq 1$. The utilization of a taskset, u_{sum}, is defined as $\sum_{\tau_i \in \tau} u_i$. The static partition generated by the allocator assigns a subset of tasks $\gamma_k \subseteq \tau$ to core s_k with a *cumulative utilization* $u^k_{sum} = \sum_{\tau_i \in \gamma_k} u_i \leq 1$. However, finding an optimal allocation is an NP-hard problem in the strong sense [9] and so non-optimal solutions derived from the use of bin-packing heuristics are typically used [10], [11], [12].

In a partitioned scheduling scheme, once tasks are allocated to cores, an on-line *uniprocessor* scheduling algorithm is used. In this paper we consider non-preemptive EDF scheduling. A γ_k is *schedulable* using a non-preemptive EDF scheduling algorithm if the following conditions [13] are satisfied: (1) $u^k_{sum} \leq 1$ and (2) $\forall \tau_i \in \gamma_k \mid \forall L : P_1 < L \leq P_i, L \geq C_i + \sum_{j=1}^{i-1} \lfloor \frac{L-1}{P_j} \rfloor C_j$.

However, any other non-preemptive uniprocessor scheduling algorithm can be used. Note, we consider non-preemptive scheduling as this is used in many commercial systems, particularly those in aerospace applications. Non-preemptive scheduling significantly reduces the difficulty involved in obtaining valid estimates of the WCET, as the use of preemption can introduce new inter-tasks interferences due to the cold start when the task is resumed. This is part of our future work.

B. Multicore Architecture

This paper considers the analyzable multicore processor described in [2], [8] as the target processor in which tasksets are allocated. Concretely, we consider a four-core processor in which each core has a private data and instruction L1 cache connected to a partitioned L2 cache through a shared bus. The L2 cache is loaded from a JEDEC-compliant DDR2 SDRAM memory system. Figure 2 shows the complete architecture.

Under this architecture, the execution environment in which a given HRT can run is identified by two parameters: (1) the cache partition size assigned to each core and (2) the number of HRTs running simultaneously at any point in time. To do so, the architecture ensures, by design, that the maximum delay a request to a shared hardware resource can suffer due to inter-tasks interferences is bounded by a pre-computed *Upper Bound Delay* (UBD) and such UBD depends on the number of co-running HRTs, but not the number of co-running NRTs. In other words, the architecture ensures that requests cannot be delayed longer than UBD cycles, regardless of behavior of the other simultaneous tasks running with a given HRT [2], [8].

In this architecture, there are two sources of inter-task interferences that can increase the WCET estimation of HRTs: *bus* and *memory interferences*. The former appears when two requests from different cores attempt to access the L2 cache

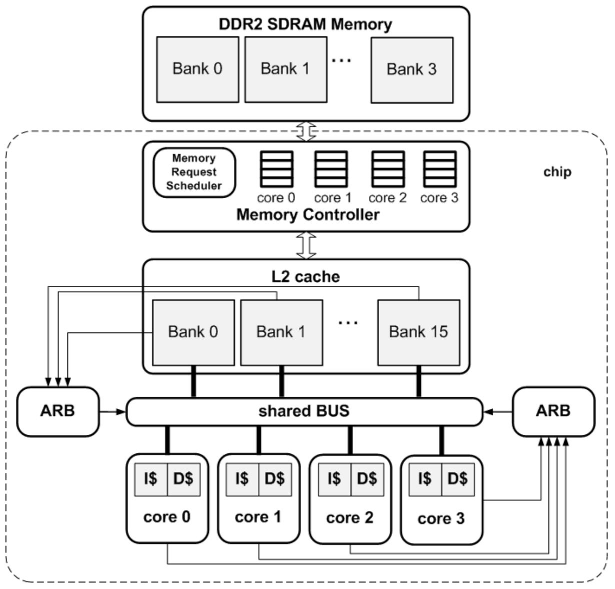


Fig. 2. Multicore processor architecture considered

at the same time. The latter appears when two requests from different cores attempt to access the SDRAM memory device at the same time.

In order to make the analysis of inter-task interferences independent of other simultaneous HRTs, the arbitration component of both resources, i.e. the bus arbiter and the memory controller, implement a *round robin* arbitration policy with a private request queue per core (see Figure 2) and prioritize HRTs over NRTs. By doing so, the UBD a request can suffer due to interferences is determined by the worst access latency of that resource and the number of simultaneous HRTs (No_{HRT}), i.e. the number of round robin slots. For this, the UBD can be defined as a function of No_{HRT}, i.e. $UBD(No_{HRT})$. For example, in case of the bus, the $UBD_{bus} = No_{HRT} \cdot L_{bus} - 1$ [2], L_{bus} being the bus transmission latency: In the worst case scenario, a request must wait until all other HRTs have been served. Similarly, in case of the memory controller, the $UBD_{MC} = No_{HRT} \cdot t_{ILWORST} - 1$ [8], $t_{ILWORST}$ being the longest possible issue latency, which depends on the specific timing constraints determined by the DDR2-SDRAM memory device used [14]. Please note that in both formulas (UBD_{MC} and UBD_{bus}) the impact that NRTs may have on HRTs is considered.

Then, in order to consider the $UBD(No_{HRT})$ in the computation of the WCET estimation, every request that access to the bus and the memory controller is delayed by UBD_{bus} and UBD_{MC} cycles respectively, so the HRT is subject to the worst-case delay that it can suffer due to interferences. Hence, the WCET estimates obtained considering the $UBD(No_{HRT})$ are the largest values that could occur [2], [15]. It is important to remark that different $UBD(No_{HRT})$ (varying the No_{HRT}) lead to different execution environments, and so different WCET estimates, making them independent of the specific details of the tasks that are running at the same time.

Finally, in order not to be affected by cache interferences (bank and storage inter-task interferences[2]), the architecture also allows the software to set dynamically the amount of private cache partition given to each core. This technique, called *bankization*, assigns to each core a private subset of the total number of banks that no other core can use.

Therefore, under this architecture, there are as many execution environments as possible cache partitions multiplied by the number of simultaneous HRTs, upper bounded by the total cache size and the number of cores respectively.

III. Our Proposal

This section presents the main contribution of this paper: IA^3, an off-line interference-aware allocation algorithm for multicore processors that, based on the *WCET-matrix* and the *WCET-sensitivity* concepts, generates a more efficient allocation compared to other partitioned schemes.

A. WCET-matrix and WCET-sensitivity

Current scheduling approaches typically consider only a single WCET value per task to perform the allocation, which usually corresponds to the highest WCET estimation in all execution environments in which the HRT task can run [6]. Instead, in this paper we propose an allocation algorithm that considers a set of WCET-values, each corresponding to a different execution environment in which the task can run.

Definition 1. *Given a HRT, the WCET-matrix is the collection of WCET estimations of this HRT when it runs on a processor under different execution environments.*

Thus, each WCET estimation inside the WCET-matrix is identified by the execution environment parameters that result in that WCET. For example the number of simultaneous HRTs that run in a multicore processor, and the size of the cache partition assigned to each task.

Note, not all HRTs are affected in the same way due to inter-task interferences. For example, when comparing the WCET estimation of a memory intensive HRT that requires a lot of accesses to main memory in an execution environment with two, three or four HRTs running simultaneously, its WCET estimations will vary more than the WCET estimations of a CPU bounded HRT that does not access the main memory.

Definition 2. *Given a HRT, the WCET-sensitivity measures the WCET estimation variation among the different execution environments that comprise the WCET-matrix.*

Thus, a HRT with low WCET-sensitivity means that the variation of the WCET estimations under different execution environments is small, while a HRT with high WCET-sensitivity means that different execution environments in which the task is executed make the WCET change significantly.

Therefore, by considering the WCET-matrix and the WCET sensitivity, the allocation algorithm can be aware of how the execution environment impacts on the WCET estimation of each task, thus enabling a more efficient allocation to be found. Current partitioning allocation approaches, e.g. first-fit decreasing, prioritize tasks by sorting them based on their utilization, so tasks with higher utilization are allocated first. Instead, WCET sensitivity enables partitioning based on the execution environment required, such that tasks with higher resource demand are allocated first.

B. Re-defining Terminology and Notations

The introduction of the WCET-matrix and the WCET-sensitivity makes it necessary to extend the terminology and notations provided in Section II.

Our interference-aware allocation algorithm not only assigns a taskset τ composed of n independent tasks $(\tau_1, ...\tau_n)$ to a set of m identical cores $(s_1, .., s_m)$, but also selects the execution environment in which each task will run based on the WCET-matrix and the WCET-sensitivity. For that, we define configuration φ as the set of tuples $(< e_1, a_1 >, ..., < e_n, a_n >)$, such that each task $t_i \in \tau$ has an associated tuple $< e_i, a_i >$ that specifies the execution environment e_i in which t_i runs, and the core a_i to which t_i is allocated. It is important to remark that, in a partitioned scheduling approach such as the one considered in this paper, tasks are statically assigned to different cores, forcing each task to be executed exclusively on one core. Therefore, all tasks that have been assigned to a given core are forced to have the same execution environment.

The relationship between the WCET estimation and the execution environment can be defined as an n-dimensional vector space $M \subseteq \mathbb{N}^{z+1}$ of $n = z + 1$ dimensions, in which the first z dimensions identify all possible parameters that define an execution environment and the $(z + 1)$-th dimension identifies all possible WCET estimations of a HRT under the different execution environments. A vector $\hat{v} =< v_1, ..., v_{z+1} >$ associates each WCET-estimation (v_{z+1}) with its corresponding execution environment $(v_1, ..., v_z)$. The WCET-matrix of each task $\tau_i \in \tau$ is a vector sub-space $WM \subseteq M$ that corresponds to the set of $\hat{v}$ in which this task can run. We require that all WCET-matrices are such that: Given a task τ_i and two vectors $\hat{v}_1$ and $\hat{v}_2$ both from the same task, if the execution environment of $\hat{v}_1$ requires less resources than $\hat{v}_2$, the associated WCET estimation of $\hat{v}_1$ is equal to or higher than the associated WCET estimation of $\hat{v}_2$.

Similarly, the variation that the WCET estimation has among all different execution environments can be defined as a n-dimensional vector space $S \subseteq \mathbb{Z}^{z+1}$. A vector $\hat{w}$, defined as $\hat{v}_1 - \hat{v}_2$ where $\hat{v}_1$ and $\hat{v}_2 \in M$, measure the WCET estimation variation when changing the execution environment from the one defined in $\hat{v}_2$ to the one defined in $\hat{v}_1$. Hence, the WCET-sensitivity of each task $\tau_i \in \tau$ is a vector sub-space $WS \subseteq S$ that corresponds to the set of $\hat{w} = \hat{v}_1 - \hat{v}_2$ in which $\hat{v}_1$ and $\hat{v}_2$ belong to τ_i and where they represent all the different execution environments.

It is important to remark that configuration φ must contain a valid set of execution environments. If we consider the previous example in which different cache partitions can be assigned to each core, the size of each cache partition in the configuration φ must be equal to or less than the total cache size. Moreover, the maximum number of simultaneous HRTs considered must be equal to or less than the total number of cores. In this case we say that configuration φ is *valid*.

C. IA^3: Interference-Aware Allocation Algorithm

The main objective of IA^3 is to determine a valid set of φ to schedule a taskset τ that minimizes the resources assigned to HRTs and hence enables the rest of the resources to be assigned to NRTs, thus maximizing the hardware utilization and taking full advantage of multicore processors.

The IA^3 is composed of two phases: the *common partitioned* phase, in which the same execution environment is considered in all cores, and the *WCET-sensitivity* phase, in which different execution environments are assigned to different cores. Figure 3 shows the pseudo-code implementation of IA^3 considering the multicore processor described in [2], [8] and summarized in Section II-B, in which the execution environment is identify by: (1) the number of HRTs that run simultaneously (No_{HRT}) and (2) the size of the cache partition assigned to each core (p). Thus, the WCET-matrix of a task is a vector sub-space of $z = 2$ dimensions with $m \cdot x$ vectors (where x is number of cache partition sizes, and m is the number of cores), such that given a $\hat{v} =< k, p, c >$ it associates the WCET estimation c with the cache partition p and the number of cores k used by HRTs. Moreover, given two different vectors $\hat{v_1} =< k_1, p_1, c_1 >$ and $\hat{v_2} =< k_2, p_2, c_2 >$, the WCET-sensitivity is defined as $\hat{w} =< k_1 - k_2, p_1 - p_2, c_1 - c_2 >$, where $c_1 - c_2$ is the WCET estimation variation changing the execution environment from $< k_2, p_2 >$ to $< k_1, p_1 >$. We note that IA^3 can be generalized for additional dimensions (Section VI considers execution environments identified by different parameters).

Concretely, IA^3 takes as input the taskset τ, the total number of cores m and the different cache partition sizes $cache\_partitions$. As output it provides a list of valid configurations ϕ that schedule τ.

The IA^3 iterates over all $z = 2$ dimensions of the WCET-matrix, i.e. over No_{HRT} (line 2) and p (line 7), using a *depth-first search* approach [16] in increasing order, i.e. from the execution environment in the WCET-matrix that leads to the lowest WCET estimation for each task ($No_{HRT} = 1$ and p_1 = biggest size of cache partition), to the one that leads to the highest WCET estimation ($No_{HRT} = m$ and p_x which refers to the smallest size of cache partition). Note this ordering is used because we are interested in solutions that use the least processing capacity, thus maximizing the available spare capacity for NRTs. At the beginning of every new iteration of the outermost loop (line 2), No_{HRT} determines the number of cores available (n_av_cores) for the HRTs (line 3) and a working copy (τ') of the original taskset (τ) is set (line 4). Note that the selected starting point, i.e. the one that leads to the lowest WCET estimation, aims to determine the minimum number of processors that could possibly result in a solution, and then refine the solution from there.

During the *common partitioning* phase, the IA^3 tries to allocate the taskset τ' using n_av_cores cores, assigning to

IA^3
Input τ: taskset
m: total number of cores,
$cache\_partitions$: cache partition sizes
Output $list_\varphi$: list of valid φ

```
1  list_φ := $\emptyset$;
2  for $No_{HRT}$ in [1:$m$]
3    n_av_cores:=$No_{HRT}$;
4    $\tau'$ := $\tau$;
5    $tmplist_\varphi$ := $\emptyset$;
6    $\varphi_{wcstmp}$, $\varphi_{ffdtmp}$ := $\emptyset$;
7    for each $p_j$ in $cache\_partitions\_sizes$ (decreasing order)
8      // Common Partition Phase
9      <$\varphi_{ffdtmp}$,$\emptyset$> := ffd($\tau'$,n_av_cores,$No_{HRT}$,$p_j$,$\emptyset$);
10     if (schedulable($\tau$,$\varphi_{ffdtmp} \cup \varphi_{wcstmp}$))
11       if (valid($\varphi_{ffdtmp} \cup \varphi_{wcstmp}$)
12         add-to-list($\varphi_{ffdtmp} \cup \varphi_{wcstmp}$,$tmplist_\varphi$);
13       endif
14     else
15       if (p = max($cache\_partitions$))
16         break;
17       endif
18       // WCET sensitivity Phase
19       wst := compute-sensitivity($\tau'$,$No_{HRT}$,$p_j$,$p_{j-1}$);
20       < $\varphi_{1c}$,$\tau'$ > := ffd($\tau'$,1,$No_{HRT}$,$p_{j-1}$,wst);
21       $\varphi_{wcstmp}$ := $\varphi_{1c} \cup \varphi_{wcstmp}$;
22       n_av_cores:= n_av_cores − 1;
23       <$\varphi_{ffdtmp}$,$\emptyset$> := ffd($\tau'$,n_av_cores,$No_{HRT}$,$p_j$,$\emptyset$);
24       if (schedulable($\tau$, $\varphi_{ffdtmp} \cup \varphi_{wcstmp}$) )
25         if (valid($\varphi_{ffdtmp} \cup \varphi_{wcstmp}$)
26           add-to-list($\varphi_{ffdtmp} \cup \varphi_{wcstmp}$,$tmplist_\varphi$);
27         endif
28       else
29         break;
30       endif
31     endif
32   endfor
33   $\varphi_{min}$:=find-minimum-φ($tmplist_\varphi$),
34   add-to-list($\varphi_{min}$, list_φ);
35 endfor
36 return list_φ;
```

Fig. 3. Pseudo-code implementation of the IA^3 considering the multicore architecture presented in Section II-B with $z = 2$.

each core the execution environment defined by No_{HRT} and p, and generating configuration φ_{ffdtmp}. To do so, the IA^3 applies a first-fit decreasing heuristic[7] (ffd function) using the WCET-estimations c_i such that $\hat{v}_i =< k_i, p_i, c_i > \in WM$ (the WCET-matrix), $k_i = No_{HRT}$ and $p_i = p_j$ (line 9)[3]. The last parameter of the ffd function (in this case $\emptyset$) defines the criteria used to sort tasks before allocating them: $\emptyset$ indicates to use the WCET estimations. Then, if the resulting φ_{ffdtmp} (joined with the result of previous *WCET sensitivity* phases) can schedule the complete taskset τ (line 10) and it is a valid configuration (line 11), i.e. the amount of resources required by τ do not exceed the actual amount of resources available in the processor, it is stored (line 12) .

Let us assume that the execution environment e_2 (defined by No_{HRT} and p_j) cannot schedule τ while e_1 (defined by No_{HRT} and p_{j-1}) can. Note that the size of the cache partition considered in e_1 (p_{j-1}) is bigger than the one considered in e_2 (p_j). In this case, the IA^3 starts the *WCET-sensitivity*

[3]$\hat{v}_i$ exists as all execution environments considered by IA^3 have a corresponding $\hat{v}_i$ per task

phase (line 18), in which it identifies those tasks whose WCET estimations suffer a higher impact when changing the execution environment from e_1 to e_2. To do so, the IA^3 computes the WCET-sensitivity $\hat{w} = \hat{v}_1 - \hat{v}_2$ for all tasks in τ', such that the first 2 dimensions of $\hat{v}_1$ and $\hat{v}_2$ are equal to e_1 and e_2 respectively, and it sorts all the tasks by the 3rd dimension of $\hat{w}$ (line 19). The task order[4] is stored in wst. Note that, since the amount of resources assigned to e_2 is less than e_1, the 3rd dimension of $\hat{w}$ will be always bigger than or equal to 0.

Then, the subset of tasks $\gamma \in \tau'$ with higher sensitivity are allocated to one core, fixing e_1 as the execution environment of that core. To do so, the ffd function uses the WCET estimations c_i such that $\hat{v}_i =< k_i, p_i, c_i >\in WM$, $k_i = No_{HRT}$ and $p_i = p_{j-1}$ and the task order defined by wst (line 20). As a result the ffd function provides the partial configuration φ_{1c}, which only contains the set of tuples $< e_i, a_i >$ assigned to γ, being $e_i = e_1$, and a new taskset τ' that replaces the previous one, such that for all $t_i \in \tau'$, $t_i \notin \gamma_k$ and $t_i \in \tau$. This new φ_{1c} is then added to other allocations that have been computed during previous WCET sensitivity phases (φ_{wcstmp})(line 21) and the number of available cores is reduced (line 22). By doing so we expect to assign the tasks γ with the highest WCET-sensitivity to a core with an execution environment e_1, so the new taskset τ' may be scheduled with e_2 or less resources on the remaining cores.

Finally, the ffd function is called with the remaining tasks contained in the new τ', the current available cores and the execution environment e_2, resulting in a new partial configuration φ_{ffdtmp} (line 23). Then, it is checked whether the complete configuration made up of φ_{ffdtmp} and φ_{wcstmp} can schedule the complete taskset τ (line 24). If the unified φ is valid and it can schedule τ, it is added to the $tmplist_\varphi$ (line 25). If τ is not schedulable, the cache partition exploration finishes (line 29).

Note that the IA^3 uses two different first fit decreasing sorting criteria: one uses the WCET estimation and it does not modify τ' (lines 9 and 23 with $\emptyset$ in the input and output), and one that uses the WCET sensitivity and generates a new τ' (line 20).

For a given No_{HRT} IA^3 generates at most one valid configuration. That is, fixing No_{HRT}, the algorithm selects the configuration, if there is one, that minimizes the total cache used. This is performed at the end of every iteration of the p_j loop, by calling the function $find-minimum-\varphi$ (line 33) that, given the list of φs stored during the exploration of cache partition sizes p_j, selects the one that requires the smallest amount of cache. However, different number of cores (No_{HRT}) can lead to multiple configurations (φs), so the same τ can be scheduled using different execution environments and different task-to-core mappings (line 34). Then, the embedded system designer can select the proper solution according to the system constraints.

[4]We also checked ordering tasks using the $sensitivity/period$ rather than just $sensitivity$ criteria, expecting that tasks with a large decrease in utilization by virtue of having more cache, could be assigned on a core with a larger cache partition. However, both heuristics have very similar results.

It is important to remark that the IA^3 explores invalid configurations, i.e. with more resources than the ones available in the processor, in order to reach a valid one. For example, it can be the case that the IA^3 explained above explores a configuration in which the sum of the cache partition sizes assigned to each core exceeds the total cache size. Then, the goal of the WCET sensitivity phase is to identify those tasks that require a higher cache partition and group them into a core, such that the remaining tasks may be assigned to cores with a smaller cache partition. By doing this a valid configuration in which the sum of the cache partition sizes does not exceed the total cache size can be reached.

D. Example: Applying IA^3 to a four-core processor with partitioned cache

This section explains all the steps of the IA^3 implementation described in Section III-C considering the following input parameters: a taskset τ, $m = 4$ and $cache\_partitions = [64, 32, 16, 8]$.

Let us assume that the first iteration of the common partition phase (No_{HRT} = 1, $p = 64$) results in a φ_{ffdtmp} that cannot schedule τ (line 10) (no WCET sensitivity phase has been executed, so φ_{wctmp} is empty). Then, since $p_1 = 64$, i.e. the biggest cache partition size (line 15), IA^3 breaks out of the p_j loop (line 16). The same conditions are verified in the next iteration (No_{HRT} = 2, p_1 = 64). Instead, with No_{HRT} = 3 and p_1 = 64, τ is schedulable (line 10), but the resultant φ_{ffdtmp} is not stored because it is not valid (line 11) (φ_{ffdtmp} assigns 64 KB to each core but the total available cache size is 64 KB). The same happens in the next iteration (No_{HRT} = 3 and p_2 = 32). Note that in these two cases the IA^3 is exploring invalid configurations. Instead, when performing the common partition phase with No_{HRT} = 3 and p_3 = 16, τ is not schedulable with the resultant φ_{ffdtmp}. In this case, the condition stated in line 15 does not hold, so the WCET sensitivity phase starts.

In the *WCET-sensitivity* phase, the WCET-sensitivity vectors $\hat{w}$ are computed (line 19), such that No_{HRT} = 3, p_3 = 16, and p_2 = 32, storing the sort criteria in wst. Table I shows an example of the computation of the WCET-sensitivity of the three highest sensitive tasks in τ', please note that the first z dimensions of $\hat{w}$ are not considered. Then, the IA^3, using a first-fit decreasing heuristic, allocates the tasks with the highest WCET-sensitivity to a single core, and it fixes No_{HRT} = 3, p_2 = 32 as the execution environment of this core (line 20). This allocation is stored in φ_{1c} and the set of tasks that have not been allocated are stored in τ'. Then, φ_{1c} is joined with the results of the previous WCET sensitivity phases, φ_{wcstmp} (which in this case is empty), and the number of available cores is reduced to 2. Finally, a first fit decreasing algorithm is applied on the remaining 2 cores, with No_{HRT} = 3, p_3 = 16 and the new taskset τ'. The resultant φ_{ffdtmp} is joined with φ_{wcstmp}, i.e. considering 3 cores, with respectively 32, 16 and 16 KB of cache partition. Let us assume that τ is schedulable.

TABLE I
WCET-SENSITIVITY COMPUTATION ($\hat{w}$) OF 3 TASKS CONSIDERING $k_1 = 3, k_2 = 3, p_1 = 16$ AND $p_2 = 32$. ORDERED BY THE 3-RD DIMENSION.

	$\hat{v}_1$	$\hat{v}_2$	$\hat{w}$
task$_6$	$< 3, 16, 132 >$	$< 3, 32, 101 >$	$< -, -, 31 >$
task$_2$	$< 3, 16, 100 >$	$< 3, 32, 75 >$	$< -, -, 25 >$
task$_5$	$< 3, 16, 256 >$	$< 3, 32, 235 >$	$< -, -, 21 >$

Thus, since the new φ is valid, it is added to $tmplist_\varphi$ (line 26).

In the next iteration (No_{HRT} = 3 and $p_4 = 8$) the common partition phase is started again, but considering 2 available cores and the new τ' (line 9). In this case, the resultant φ_{ffdtmp} is joined with the previous φ_{wcstmp} (line 10), i.e. considering 3 cores, with respectively 32, 8 and 8 KB of cache partitions. Let us assume, τ is schedulable and the resultant φ is valid, so it is added to the list $tmplist_\varphi$. However, in the next iteration (No_{HRT} = 3 and $p_5 = 4$) the common partition phase results in a φ that cannot schedule τ so the WCET sensitivity phase starts again. In the WCET sensitivity phase, the resultant joined φ_{ffdtmp} and φ_{wcstmp} is 3 cores, with respectively 32, 8 and 4 KB of cache partition, which cannot schedule τ so IA^3 breaks out of the p loop (line 29).

Once the cache size dimension p_j has been fully explored, the function $find-minimum-\varphi$ finds, among all the stored φ (3 cores, with respectively 32, 16 and 16 KB of cache partition and 3 cores, with respectively 32, 8 and 8 KB of cache partition), the one with the smallest amount of cache used, i.e. 3 cores, with respectively 32, 8 and 8 KB of cache partition. Thus, in order to reach this valid configuration, the IA^3 has explored invalid ones, i.e. assigning to each core 32 KB. Then, a new iteration starts with $No_{HRT} = 4$. The same steps are carried out, but they are omitted here due to lack of space.

It is important to remark that this implementation can lead to multiple allocation solutions. That is, the same τ can be scheduled, for example, using a configuration with three cores and cache partitions equal to 32, 8, 8 KB, and with a configuration with four cores but with less cache, allowing the embedded system designer to select the best solution according to the system constraints.

IV. Test Methodology

Evaluating the effectiveness of the IA^3 requires a methodology of generating tasksets. However, in order to perform an effective evaluation of our proposal, it is important to consider tasks with different levels of WCET-sensitivity. To that end, we first characterized the WCET-sensitivity of the EEMBC Autobench [17] benchmark suite under different execution environments considering the multicore architecture introduced in section II. This characterization follows the same methodology introduced in [2], [8].

A. Characterizing the EEMBC Autobench

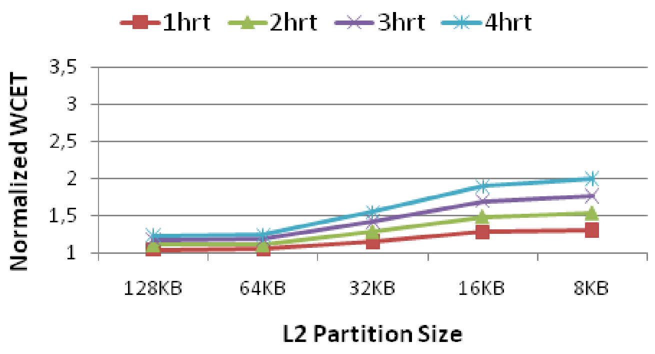
The EEMBC Autobench is a well known benchmark suite formed by sixteen applications widely used in both industry and academia, that reflect the current real-world demands of embedded systems. However, the data memory footprint of these benchmarks is, at most, 32 KB. Hence, in order to be representative of future hard-real time applications requirements [1], we have increased their memory requirements without modifying any instruction inside the source code, by enlarging the data set structures they use. The new data memory footprint of some of the EEMBC benchmarks are: aifftr01/aiifft01 (64 KB), aifirf01 (72 KB), pntrch01 (62 KB), ttsprk01 (104 KB), tblook01 (70 KB), matrix01 (91 KB).

In order to characterize their WCET-sensitivity, we computed the WCET estimation of each EEMBC benchmark considering that each request to a shared resource is delayed by UBD cycles (see Section II-B), under 20 different execution environments, as a result of assigning different cache partition sizes (128 KB, 64 KB, 32 KB, 16 KB and 8 KB) and varying the number of HRTs that access simultaneously the shared bus and the memory controller, i.e. different $UBDs$ corresponding to No_{HRT} from 1 HRT to 4 HRTs. Hence, the WCET-matrix of each benchmark results in a 2-dimensional vector space indexed by the cache partition size and the number of simultaneous HRTs.

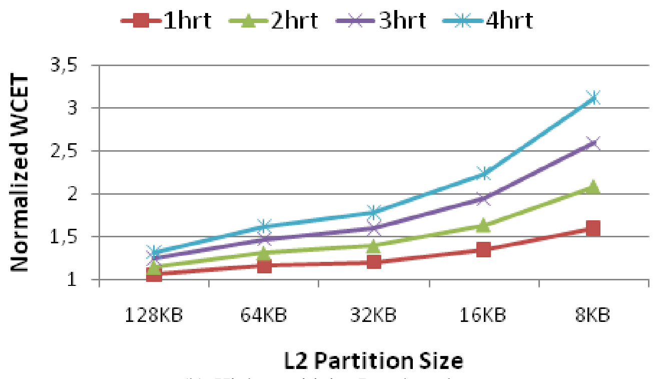
All WCET estimations were computed using RapiTime [4], a commercial tool that estimates the WCET using a measurement-based technique. Moreover, in order to model in detail the multicore architecture presented in Section II, we used an in-house cycle-accurate, execution-driven simulator [2], [8]. Our simulator is compatible with Tricore [18], an Instruction Set Architecture (ISA) designed by Infineon Technologies. We paid special attention to the simulator correctness, extensively validating it through a wide range of tests. Finally, the simulator also introduces the *WCET computation mode* [2]. When obtaining the WCET estimation of a HRT, the processor is set to this new mode and the HRT under analysis is run in isolation. Under this mode, the bus arbiter and the memory controller artificially delay every request by UBD cycles. Once the WCET analysis of all the HRTs is completed the processor is set to *Standard Execution Mode* in which no artificial delay is introduced.

Once the WCET-matrix of each benchmark had been generated, we analyzed the results according to WCET-sensitivity, i.e. how the WCET estimations vary across the different execution environments, creating 3 sensitivity groups called *High, Medium* and *Low Sensitivity*. Each group, whose WCET-matrix is the average of the WCET-matrices of all tasks that form the group, is formed by the following EEMBC benchmarks: High (*aifftr*,*aiifft*,*cacheb*), Medium (*aifirf*, *iirflt*, *matrix*, *pntrch*), Low (*a2time*, *basefp*, *bitmnp*, *canrdr*, *idctrn*, *puwmod*, *rspeed*, *tblook*, *ttsprk*).

Figure 4 [2], [8] shows the normalized WCET-estimation of medium and high sensitivity groups under the twenty different execution environments, taking as a baseline the WCET estimation of the corresponding group assuming no inter-task interferences (it can be considered as the graphical representation of the WCET-matrix of each group). We observe that, while the high sensitivity group reacts rapidly to any



(a) Medium-sensitivity Benchmarks



(b) High-sensitivity Benchmarks

Fig. 4. Normalized WCET estimation increment of two EEMBC WCET-sensitivity groups: *High* and *Medium*, under twenty different execution environments. This characterization follows the same methodology introduced in [2], [8].

TABLE II
THE WCET-SENSITIVITY RANGE OF EACH SENSITIVITY GROUP CONSIDERED IN THE GENERATION OF THE TASKS' WCET-MATRIX

	Cache Partition	Number of simultaneous HRT
High Sensitivity	0.10 - 0.25	0.10 - 0.50
Medium Sensitivity	0.07 - 0.14	0.05 - 0.18
Low Sensitivity	0.00 - 0.03	0.00 - 0.01

change, the medium sensitivity group has a smooth increment of the WCET estimation. The low sensitivity group, not shown due to space constraints, is almost unaffected by variations in the execution environment.

For each sensitivity group, we identified the WCET-sensitivity between two adjacent execution environments of the WCET-matrix. To do so, starting from the execution environment with 1 HRT and 128KB of cache, we compute the WCET-sensitivity as we reduce the cache size, and increase the number of simultaneous HRTs. Table II shows the range of the WCET-sensitivity when the cache size and the number of simultaneous HRTs changes in each group.

B. Randomly-Generated Tasksets

We randomly-generated tasksets based on the WCET-sensitivity ranges shown in Table II: Starting from an *initial*

TABLE III
PERCENTAGE OF TYPE OF TASKS IN THE TASKSET CONSIDERING THE INITIAL WCET CLASS AND THE SENSITIVITY GROUP

	High u (30%).	Low u (70%)
High Sensitivity (20%)	$20\% \times 30\% = 6\%$	$20\% \times 70\% = 14\%$
Medium Sensitivity (30%)	$30\% \times 30\% = 9\%$	$30\% \times 70\% = 21\%$
Low Sensitivity (50%)	$50\% \times 30\% = 15\%$	$50\% \times 70\% = 42\%$

WCET (corresponding to an execution environment with 32KB of cache partition and 1 HRT) we generate a complete WCET-matrix by applying the variations among the different execution environments that resembles the WCET-matrix of a given sensitivity group. In order not to generate identical WCET-matrices, we use a uniform random-generator that considers the maximum and the minimum variance of the corresponding groups.

We assume two different classes of initial WCETs: a *High Utilization* class, corresponding to a task with an utilization ranging between 0.3 and 0.6; and a *Low Utilization* class, with an utilization ranging between 0.1 and 0.3. The initial WCET is also generated using an uniform random-generator that considers the corresponding utilization ranges. For these simple experiments, we are interested only in the effect of different execution environments on the partitioning. We therefore fixed all of the task periods to have the same value (and the deadlines equal to the periods).

Finally, in order to generate tasksets composed of tasks with different requirements, we assume that 30% of the tasks belong to the high utilization class and 70% belong to the low utilization class. Moreover, from the characterization of the EEMBC Automotive benchmark suite we assume that 20% of the tasks in the taskset have a high sensitivity, 30% have a medium and 50% have a low one. Note that, each task can belong to a high or a low utilization class, and its WCET-matrix can have a WCET-sensitivity that resembles a high, medium or low sensitivity group. Table III shows the percentage of tasks considering the sensitivity group and the initial WCET. The total utilization of the generated tasksets, i.e. the target U_t, is computed using the WCET value of each task when running in the initial execution environment.

V. RESULTS

This section evaluates the IA^3. To do so, we have generated 10 series of 10,000 tasksets, each composed of 10 tasks, using the methodology explained in Section IV. Concretely, for each series, we have fixed a target utilization U_t (i.e. u_{sum} computed considering the initial WCET of each task in the taskset t), ranging from 2.9 to 3.9 with an increment step of 0.1. To do so, we used the simple naive and unbiased method described in [19], [20]: we generate 9 out of 10 random WCET matrices in high and low utilization ranges and then check if the remaining task with utilization equal to $U_t - u_{sum}$ belongs to the low utilization class; if not we discard the taskset and start the process again. Moreover, we also consider a real taskset composed of the sixteen EEMBC Autobench benchmarks (described in Section IV), ranging its

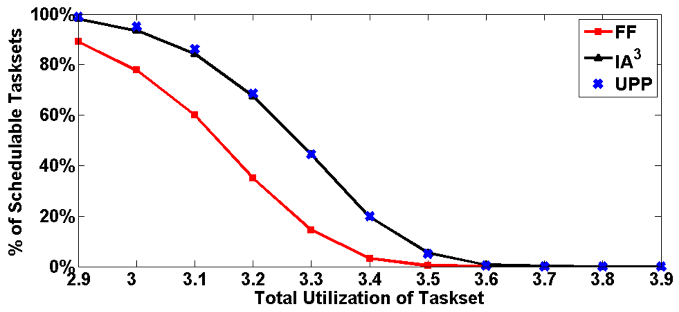


Fig. 5. Percentage of schedulable tasksets when applying *FF, UPP* and *IA^3* in a 4-core processor, ranging the total utilization from 2.9 to 3.9 with an increment step of 0.1

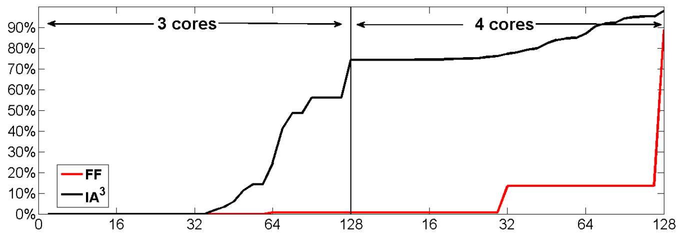


Fig. 6. 10,000 Taskset with Total Utilization 2.9

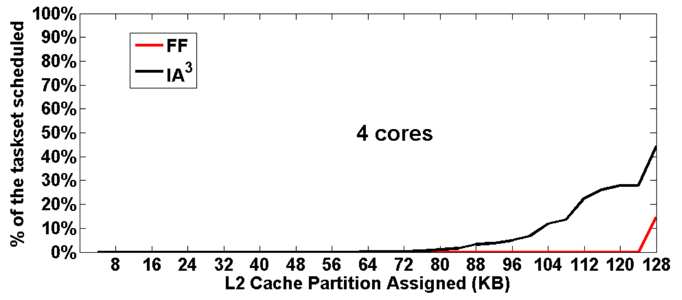


Fig. 7. 10,000 Taskset with Total Utilization 3.3

total utilization from 2.7 to 3.3 with an increment step of 0.1.

The target multicore architecture considered is a 4-core processor with a 128 KB partitioned cache. The architecture allows assigning dynamically to each core a private partition of cache ranging in size from 4 KB to 128 KB.

We also compare IA^3 to an idealized scheme that assumes each core has the same cache partition size, and checks a necessary feasibility condition (labeled *UPP*). In this case we simply check that, given an execution environment with k cores and a cache partition size assigned to each core identified by p, that for all $\hat{v} =< k, p, c >$, the cumulative utilization $\sum_{u \in \hat{v}} \leq k$. If this test fails, then the configuration could not possibly be schedulable as it has a higher utilization that the available processor capacity. *UPP* represents an upper bound on the maximum possible performance of a global scheduling approach with no migration overheads.

Moreover, we also compare IA^3 with a partitioned scheduling algorithm that uses a First-Fit Decreasing Heuristics (labeled as *FF*) [7] that assigns to each core the same execution environment such that the resources used are minimized, i.e. the cache partition assigned and the number of cores. For *FF* we are effectively applying only the *common partitioning* phase of the IA^3.

Figure 5 compares the percentage of schedulable tasksets using IA^3, *FF* and *UPP*, increasing the U_t from 2.9 to 3.9. As expected, our allocation algorithm is able to consistently schedule more tasksets than *FF*, while achieving almost the same ratio of scheduled tasksets as the hypothetical upper bound given by *UPP*. On average, when compared to *FF*, IA^3 is able to schedule 20% more tasksets, with a maximum difference of 32% when considering $U_t = 3.2$ and minimum of 5% when considering $U_t = 3.5$.

However, the benefit of IA^3 is not only the number of schedulable tasksets, but the resources used to schedule them. Figures 6 and 7 show the cumulative distribution function of the resources required to schedule tasksets with a U_t equal to 2.9 and 3.3 respectively, when using IA^3 and FF. Concretely, the figures show the percentage of scheduled tasksets with a certain number of cores (X-axis division on figure 6) and the size of cache partition assigned (in KB) to each core. For example, when considering a total utilization equal to 2.9 (figure 6), the IA^3 is able to schedule more than 70% of the tasksets with only 3 cores, while, in the case of *FF*, less than 5% of the tasksets have been scheduled with 3 cores. With a total utilization of 3.3, both schemes require 4 cores to schedule all tasksets. However, IA^3 is able to schedule more than 30% of the tasksets without requiring the complete cache size, i.e 128 KB. Therefore, IA^3 generates a very efficient partition scheme by assigning different environments to different cores and so reducing the resources required. Reducing the amount of resources assigned to HRTs is particularly beneficial in *mixed-application workloads*, because the resources not used by HRTs can be assigned to NRTs. For example, in Figure 6, 50% of the tasksets have been scheduled using only 3 cores and less than 96 KB of cache partition, so the rest of the resources, i.e. one core and 32 KB of cache partition can be assigned to NRTs. By contrast, in the case of *FF*, only 15% of the tasksets are able to be scheduled with less than 64 KB, but requiring 4 cores, and so giving less resources to NRTs.

Finally, Table IV shows the number of cores and the total cache partition size required to schedule the real taskset composed of the sixteen EEMBC benchmarks when using FF, UPP and IA^3, with utilization U_t scaled from 2.7 to 3.3. IA^3 requires consistently less resources than the other schemes.

VI. Additional Considerations

The IA^3 permits a WCET-matrix of z dimensions, $z \subseteq \mathbb{N}$. However, in this paper we have considered only a simple WCET-matrix of $z = 2$ dimensions that includes the number of simultaneous HRTs and the size of the cache partitions assigned to each core, which improves taskset schedulability when compared to a classical partitioned scheme. However the benefit of making task allocation aware of the impact that different execution environments have on the WCET is potentially much bigger. For example, the WCET-matrix can

TABLE IV

NUMBER OF CORES AND CACHE PARTITION SIZE (IN KB) REQUIRED TO SCHEDULE A TASKSET COMPOSED OF THE 16 EEMBC BENCHMARKS WHEN APPLYING *FF, UPP* AND *IA³*, RANGING THE TOTAL UTILIZATION FROM 2.7 TO 3.3 WITH AND INCREMENT STEP OF 0.1

	FF		UPP		IA^3	
U_t	cores	cache (KB)	cores	cache (KB)	cores	cache (KB)
2.7	4	32	3	48	**3**	**40**
2.8	4	32	4	32	**3**	**52**
2.9	4	32	4	32	**4**	**20**
3.0	4	64	4	32	**4**	**28**
3.1	4	64	4	64	**4**	**32**
3.2	4	64	4	64	**4**	**32**
3.3	*not schedulable*		4	64	**4**	**40**

potentially be applied to heterogeneous multicore processors, in which each core has a different computational power. In this case, an extra dimension is required to measure the WCET of the different tasks under the different cores, Thus, the IA^3 will consider a WCET-matrix of $z = 3$, assigning to the most powerful cores those tasks with the highest WCET-sensitivity.

An important future research line of IA^3 is to consider a preemptive scheduling approach. Some works introduce a factor in the WCET estimation [21], [22], [23] in order to consider the impact of inter-tasks interferences introduced by preemption due to the cold start when the task is resumed. These factors can be introduced in the WCET-matrix to allow IA^3 to use preemption. Similarly, the WCET-matrix can consider other execution environment parameters such as the number of TDMA-slots assigned to a HRT, different frequencies at which the processor can run, etc.

However, it is important to notice that adding additional dimensions to the WCET-matrix increases the computational complexity of IA^3 because the current implementation of the algorithm effectively searches over all combinations of parameters (e.g. number of cores, and cache partition size). The complexity can be reduced by applying a search algorithm optimization, for example, Emberson et. al. [24]. However, as WCET estimates need to be obtained for each task in all the execution environments of the WCET-matrix, in practice the dimension of the matrix will be limited to a small number of important factors such as cache size, number of simultaneous HRTs, computational speed, etc.

VII. RELATED WORK

There are two main strands of research in multiprocessor scheduling [25], reflecting the ways in which tasks are allocated to cores. Partitioned approaches allocate each task to a single core, dividing the problem into one of task allocation (bin-packing) followed by single processor scheduling. In contrast, global approaches allow tasks to migrate from one core to another at run-time. In partitioned scheduling finding an optimal task allocation is an NP-hard problem in the strong sense [9] and so non-optimal solutions derived from the use of bin-packing heuristics are typically used.

Dhall and Liu [11] proposed two heuristic assignment schemes: *rate-monotonic next-fit* and *rate-monotonic first-fit* based on the next-fit and first-fit bin-packing heuristics. In both schemes, tasks were sorted in decreasing order of their periods and assigned to a so-called current core until the schedulability condition was not achieved. Davari and Dhall [7] proposed a variation of the rate-monotonic first-fit, the *first-fit decreasing utilization*, in which tasks were sorted by their utilization. Similarly, Oh and Son [26] proposed a *best-fit decreasing utilization* in which tasks were allocated to the cores that would then have the least remaining utilization. However, all these approaches did not consider the effect that interferences have on the WCET estimation. Jain et al. [6] proposed a scheduler for a two-way simultaneous multithreading processor in which the WCET estimation for each soft real-time task depends on the co-running task. In this case, there is a single hardware configuration so the execution environment of each task only depends on the particular soft real-time task with which it is executed. This approach has not been applied to hard real-time systems.

Global scheduling approaches have also considered the effect of the execution environment on WCET estimation. Holman et al. [22] realized that scheduling all processors simultaneously can result in a heavy bus contention at the start of the scheduling quantum due to reloading data into caches. To reduce such bus contention they presented a new global scheduling model that distributes more uniformly the bus traffic by shifting the scheduling decisions of the different tasks. Anderson et al. [23] proposed a new task organization, called *megatasks* as a way to reduce the miss rate in shared caches on multicore platforms. Megatasks artificially inflate their utilization factors in order to consider cache contention. Both techniques have good results in the average case, reducing the bus and cache contention. However, interferences remain uncontrolled, and they are still unknown in the worst case.

Kato et al. [5] considered the use of multiple execution time estimations (EET) using U-Link Scheduling to schedule a taskset in a soft real-time simultaneous multi-threading (SMT) environment. Concretely, they provided the EET of a task in an SMT execution environment based on WCET estimation in a single-threading execution and the execution variation introduced by the details of the other co-scheduled tasks. The research presented in this paper differs from that of Kato et al. in that our approach focuses on the WCET estimations of each HRT considering only the parameters that determine the execution environment in which a task runs, rather than on estimations of the execution time variations due to details of the various tasks when scheduling them in a simultaneous multi-threading environment.

We note that multiple execution times per task have also been considered as part of approaches to improve quality of service on single processors [27].

The work of Pellizzoni et al. [28] considers the effect of contention for access to main memory on WCETs of tasks. This is done by abstracting the accesses of each interfering core into an arrival curve which combined with peripheral traffic gives a delay bound for the task under analysis. Again this means that the analysis is dependent on the individual characteristics of the tasks, rather than just the characteristics of the execution environment.

To the best of our knowledge, IA^3 is the first task allocation algorithm that considers a matrix of WCET estimations for the different execution environments in which tasks can run.

VIII. Conclusions

In this paper we presented IA^3, an new off-line interference-aware allocation algorithm for multicore processors. The IA^3 is based on two novel concepts: the WCET-matrix and the WCET-sensitivity.

The WCET-matrix is a z-dimensional vector space that contains the WCET estimations of each task under different execution environments. Each dimension represents a different execution environment parameter (e.g. number of simultaneous HRTs, size of the cache partition assigned to each core, etc.) that makes the WCET estimation vary. The WCET-sensitivity measures the impact of changing the execution environment, i.e. modifying the parameters that define it, on the WCET estimation. These two concepts allow IA^3 to consider not just a single WCET estimation but a set of WCET-estimations generating a more efficient partitioning.

The target multicore architecture considered in this paper is a four-core processor [2], [8], in which an execution environment can be identified based on the number of simultaneous HRTs and the size of the cache partition assigned, and so having a WCET-matrix of $z = 2$ dimensions.

In our simple experiments, comparing IA^3 with a classical first-fit decreasing partitioning scheme under the target architecture, IA^3 is able to schedule, on average, 20% more tasksets. Moreover, when considering the amount of resources used, IA^3 schedules more than 70% of the tasksets with only 3 cores, while the first-fit partitioning approach is able to schedule only 5% of the tasksets with 3 cores. Finally, using IA^3 50% of the tasksets that are scheduled using 3 cores require less than 96 KB of cache, and so leaving one free core and 32KB of cache to NRTs. Instead, using first-fit only 15% of the tasksets that are scheduled in four cores require less than 64 KB, which does not leave any core to NRTs.

Acknowledgments

This work has been mainly funded by MERASA STREP-FP7 European Project under the grant agreement number 216415. Marco Paolieri is partially supported by the Catalan Ministry for Innovation, Universities and Enterprise of the Catalan Government and European Social Funds. Eduardo Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the grant Juan de la Cierva JCI2009-05455. Robert Davis is partially funded by the EPSRC Tempo project (EP/G055548/1) and the EU ArtistDesign Network of Excellence. Authors would also like to thank Professor Alan Burns and Guillem Bernat for their helpful comments.

References

[1] *MERASA EU-FP7 Project: www.merasa.org*, 2007.

[2] M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero, “Hardware support for WCET analysis of hard real-time multicore systems,” in *ISCA*, 2009.

[3] D. Chiou, P. Jain, S. Devadas, and L. Rudolph, “Dynamic cache partitioning via columnization,” in *DAC*, Los Angeles, CA, USA, 2000. [Online]. Available: citeseer.ist.psu.edu/chiou00dynamic.html

[4] *RapiTime: WCET analysis. User Guide. Rapita Systems. Ltd.*, 2007.

[5] S. Kato and N. Yamasaki, “Extended u-link scheduling to increase the execution efficiency for smt real-time systems,” *Real-Time Computing Systems and Applications, International Workshop on*, vol. 0, pp. 373–377, 2006.

[6] R. Jain, C. J. Hughes, and S. V. Adve, “Soft real- time scheduling on simultaneous multithreaded processors,” in *RTSS*, 2002, p. 134.

[7] S. Davari and S. Dhall, “An on line algorithm for real-time allocation,” in *ICSS*, 1986, pp. 133–141.

[8] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, “An analyzable memory controller for hard real-time CMPs,” in *Embedded System Letter*, 2009.

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

[10] J. Liebeherr, A. Burchard, Y. Oh, and S. H. Son, “New strategies for assigning real-time tasks to multiprocessor systems,” *IEEE Trans. Comput.*, vol. 44, no. 12, pp. 1429–1442, 1995.

[11] S. Dhall and C. L. Liu, “On a real-time scheduling problem,” in *Operation Research*, 1978, pp. 127–140.

[12] Y. Oh and S. H. Son, “Allocating fixed-priority periodic tasks on multiprocessor systems,” *Real-Time Syst.*, vol. 9, no. 3, pp. 207–239, 1995.

[13] K. Jeffay, D. F. Stanat, and C. U. Martel, “On non-preemptive scheduling of periodic and sporadic tasks,” in *RTSS*, 1991, pp. 129–139.

[14] *DDR2 SDRAM specification JESD79-2E*, JEDEC, 2008.

[15] J. W. Lee and K. Asanovic’, “Meterg: Measurement-based end-to-end performance estimation technique in qos-capable multiprocessors,” in *In Proc. of the 12th RTAS*, 2006, pp. 135–147.

[16] T. H. Cormen, C. E. Leiserson, R. L. R. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001.

[17] J. Poovey, *Characterization of the EEMBC Benchmark Suite*, North Carolina State University, 2007.

[18] *Tricore 1. 32-bit Unified Processor Core v1.3*, Infineon, October 2005.

[19] R. Davis and A. Burns, “Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” in *RTSS*, Washington, USA, 2009.

[20] R. Davis and A. Burns, “Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” *Real-Time Systems*, 2010.

[21] J. Staschulat, S. Schliecker, and R. Ernst, “Scheduling analysis of real-time systems with precise modeling of cache related preemption delay,” in *ECRTS*, 2005, pp. 41–48.

[22] P. Holman and J. H. Anderson, “Adapting pfair scheduling for symmetric multiprocessors,” *Journal of Embedded Computing*, vol. 1, no. 4, pp. 543–564, 2005.

[23] J. H. Anderson, J. M. Calndrino, and D. C. UmaMaheswari, “Real-time scheduling on multicore platforms,” in *RTAS*, 2006, pp. 179–190.

[24] P. Emberson and I. Bate, “Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems,” in *RTSS*, 2008, pp. 270–279.

[25] R. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Computing Survey (to appear)*, 2010.

[26] Y. Oh and S. H. Son, “Tight performance bounds of heuristics for a real-time scheduling problem,” Charlottesville, VA, USA, Tech. Rep., 1993.

[27] K. jay Lin, S. Natarajan, and J. Liu, “Imprecise results: Utilizing partial computations in real-time systems,” in *RTSS*, 1987.

[28] R. Pellizzoni, A. Schranzhofery, J.-J. Cheny, M. Caccamo, and L. Thiele, “Worst case delay analysis for memory interference in multicore systems,” in *DATE*, 2010.