# Accounting for preemption and migration costs in the calculation of hard real-time cyclic executives for MPSoCs

Laura E. Rubio-Anguiano[1,2], José Luis Briz [2] and Antonio Ramírez-Treviño[1]

***Abstract*— This work introduces a methodology to consider preemption and migration overhead in hard real-time cyclic executives on multicore architectures. The approach performs two iterative stages. The first stage takes a cyclic executive, from which the number and timing of all preemptions and migrations for every task is known. Then, it includes this overhead by updating the worst-case execution time (WCET) of the tasks. The second stage calculates a new cyclic executive considering the new WCET of tasks. The stages iterate until the preemption and migration overhead keeps constant.**

***Index Terms*— Real-Time, embedded systems, cyclic executive, scheduling, clustering algorithms, multiprocessors, worst case execution time**

## I. Introduction

Current automotive and aerospace systems encompass a large and increasing number of electronic control units (ECUs), and ECU consolidation is a desirable objective to reduce size, weight, power and cost. The usage of multicore systems on chip (MPSoCs) helps to consolidate components into single miniaturized solutions. They can also provide major improvements for some demanding applications such as engine controllers, real-time (RT) image processing or advance driving assistance (ADA) systems. However, ensuring the accomplishment of hard RT (HRT) and safety constraints on multicores can easily lead to overprovisioning, requiring more cores, and more powerful, than the ones actually demanded. The latter problem comes out from a variety of sources, with two of them standing out: worst-case execution time (WCET) estimation for each task, and task scheduling.

Algorithms for RT multiprocessor scheduling can be broadly classified into partitioned, global, and hybrid schemes (semi-partitioned, clustered) [1], [2]. Partitioned schedulers [3], [4] statically allocate tasks to processors. They can leverage well-known RT uniprocessor schedulers such as RM, which limits CPU utilization to a $69.3\%$ in uniprocessors [5], or EDF which is optimal on uniprocessors. Unfortunately, a partitioned approach on multiprocessors decreases utilization to 50% [6] under a sufficient schedulability condition.

Global schedulers allocate tasks to any CPU. They are mostly preemptive and allow task migration among CPUs. A well-known example is *gEDF*, which guarantees soft real-time (SRT) schedulability for implicit-deadline task sets [7] but is not HRT optimal [8], [9]. Aiming to achieve maximum CPU utilization, *fluid* global schedulers leverage the theoretical principle of instantly sharing all CPUs among all active jobs. In this vein, *pfair* algorithms [10] achieve HRT optimal schedulability for implicit-deadline tasks sets, but they incur in an unfeasible number of context switches. Approaches like *deadline partitioning* can notably reduce such overhead [11]. Howbeit, global schedulers are considered too complex to be practical, whereas hybrid approaches combined with ad-hoc heuristics can achieve similar optimality [12].

Meanwhile, the industry remains understandably conservative with the implementation of these algorithms in critical systems, where the use of static scheduling is pervasive [13], [14]. Thus, in ARINC 653 [15] a *cyclic executive* (CE) is a deterministic scheme of repeated execution of a series of *minor frames*. Each minor frame (MIF) define a sequence of jobs, that execute within the frame. The collection of minor frames is referred as a *major frame* (MAF). CEs are usually implemented as tables with invocation times for each job. A CE offers two significant advantages: its *predictability* and *low run-time overhead*.

Recent research shows that we can get best of two worlds by calculating a predictable, low-overhead CE while obtaining the optimal utilization provided by global or clustered schemes [2], [13]. However, there remain two non-negligible problems to solve. One of them is WCET estimation on multiprocessors, and it is common to every RT multiprocessor scheduler. The other one is the inclusion of context-switch overheads; it is common to any preemptive scheduling scheme used for calculating a CE, and is just the problem we address and solve in this paper.

WCET can be estimated using probabilistic and statistical methods [16], or by calculating the longest possible path in the instruction execution flow of a task as far as tasks do not share resources. In the latter case, the WCET must account for safe time bounds over shared resource such as the memory hierarchy and its interconnection on a multicore system. Conservative approaches multiply the time cost of a memory reference (instruction or data) by the number of cores, yielding safe but unrealistic upper bounds which are hard to narrow [17]. The problem exacerbates when the scheduling scheme allows task instances (jobs) to migrate among cores.

The cost of preemptions and migrations is usually neglected or assumed part of the WCET [11], [18]. This is a major problem because task sets which are schedulable under a given global scheduler may become not schedulable when accounting for such overhead. Authors from [19] provide an

[1]CINVESTAV-IPN Unidad Guadalajara, Av. del Bosque 1145, CP 45019, Zapopan, Jalisco, Mexico

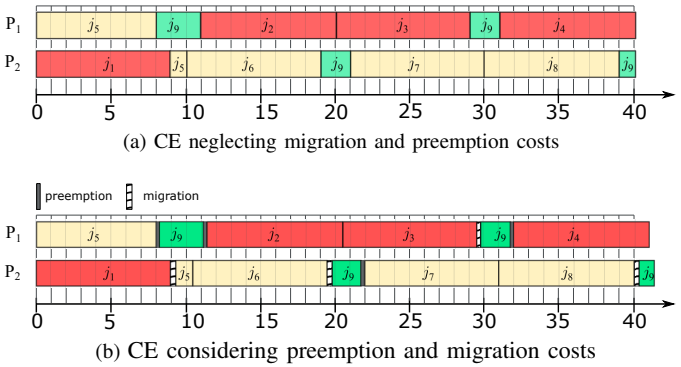
[2]DIIS/I3A Univ. de Zaragoza, María de Luna 1 - 50018 Zaragoza, España
E-mail adresses:laura.rubio@cinvestav.mx (L.Rubio), briz@unizar.es (J.L. Briz) and antonio.ramirezt@cinvestav.mx (A.Ramírez)

in-depth study on practical issues regarding semi-partitioned algorithms. They report that ignoring overheads when assigning tasks may cause overutilization. Thus, a task set that exhausts the capacity of the processors would be unschedulable. They recommend to add a worst-case bound for the overheads to prevent this problem.

There is a context switch whenever a job is released or terminates. The number of such *compulsory* context switches is just the number of jobs per tasks, and is independent on the scheduler, which can differ in CPU allocation of course. Thus, it is trivial to account for the cost of such context switches either with non-preemptive or preemptive schedulers, because we can add it beforehand to the WCET. However, preemptive schedulers introduce new context switches /migrations which largely vary with the scheduler. Accounting for such preemption costs is not trivial. We can either consider upper bounds or run into a causality dilemma, because we only know the precise number of preemptions and migrations after calculating the CE. In priority-driven scheduling algorithms, such as RM or EDF, we can find an upper bound for the number of preemptions, which is strictly less than the number of jobs that are being scheduled; this result also holds for the number of job migrations among CPUs [20]. In global scheduling policies, the upper bound for preemptions and for job migrations per minor frame are, respectively, the number of tasks minus one, and the number of CPU minus one [11]. Nevertheless, these numbers may be very pessimistic, worsening the WCET overestimation problem.

Another approach is to calculate the CE considering the WCET of tasks obviating any overhead, then compute the overhead, add it to the CE, and increase the frequency to augment system capacity accordingly. However, this naive approach leads to missing task deadlines as we can see in the following example.

**Motivational example** Fig. 1a shows a CE for the set of tasks in Table I, where preemption and migration costs are neglected. Illustratively, Fig. 1b highlights the four preemptions and four migrations, adding their computing time to the CE such that the overall execution overflows the hyperperiod. Now, assume a worst-case preemption (migration) cost *pcost* (*mcost*) of 10 and 20 cycles, respectively. Let's first compute the associated utilization overhead on the system $u_{overhead}$ and then *update* the frequency accordingly. The increment of the utilization system rate can be calculate as $u' = \frac{u_{overhead}}{U_{tasks}}$ such that $u_{overhead}$ is the utilization of the overhead up to the hyperperiod, $u_{overhead} = \frac{4p_{cost}+4m_{cost}}{Hf}$. If we increase the frequency by this amount to achieve a 100% utilization, we can expect that the added overhead now fits in the system up to the hyperperiod. The frequency increment is $\frac{3f}{2000f} = \frac{3}{2000}$, i.e. the new frequency is $f = 1001.5$. However, this approach leads to deadline misses. Both $\tau_1$ and $\tau_2$ should complete their first job by time=10. On processor $P_2$, the first job of $\tau_1$ ($j_1$) completes its WCET (9000 cycles), afterwards the first job of $\tau_2$ ($j_5$), previously allocated



(a) CE neglecting migration and preemption costs

(b) CE considering preemption and migration costs

Fig. 1: Motivational example. CE of task set on Tab. I. $j_1$ to $j_4$ are $\tau_1$ jobs, $j_5$ to $j_8$ are $\tau_2$ jobs, and $j_9$ is the only job of $\tau_3$. The black thin line represents a job boundary, the black bar indicates a preemption and the hatched bar indicates a migration.

on $P_1$, resumes and completes its remaining 1000 cycles. Considering the corresponding migration cost, the execution ends at $\frac{9000+20+1000}{1001.5} = 10.005$, missing $\tau_2$ deadline at 10.

| **Tasks** | $\tau_1$ | $\tau_2$ | $\tau_3$ | $U_{tot}$ | 2 |
|---|---|---|---|---|---|
| $cc_i$ | 9000 | 9000 | 8000 | | |
| $d_i$ | 10 | 10 | 40 | | |
| $u_i$ | 0.9 | 0.9 | 0.8 | | |

TABLE I: Task set, WCET ($cc_i$) given in cycles and relative deadlines ($d_i$) in time units. Utilization value $u_i$ is computed assuming a frequency $f = 1000$

**This research** The main contribution of this work is an iterative methodology to compute a CE on a multicore processor considering the overhead costs due to preemptions and migrations ensuring no deadline misses. The proposed algorithm *AdWCET* starts by computing a CE upon the WCET obtained from static code analysis. From this CE the number and temporal localization of all preemptions and migrations are known for every job and core, then the WCET of each task is adjusted by adding up the corresponding overhead. The stages iterate until the preemption and migration overhead keeps constant, thus concluding the adjustment.

**Novelty** As far as we know, this paper introduces the first method to improve the calculation of CEs considering the migration preemption costs, providing a more realistic path to obtain predictable HRT CEs with optimal utilization and minimum overhead on a multicore architecture.The proposed methodology can be applied to any multiprocessor RT preemptive scheduler ( [13]) but in this work we consider implicit-deadline HRT task set and leverage CAlECS [2]. This hybrid RT scheduler optimizes processor utilization at the lowest possible energy to comply thermal and temporal constraints, obtaining a CE with very few preemptions and migrations, but obviates the aggregated overhead. The algorithm tries to find fully partitioned clusters, and resorts to a *global scheduler* to complete the scheduling when required.

**Organization** The next section defines the system and

problem. Section III provides an overview of CAlECS. The methodology to adjust the WCET is presented on Section IV. Experimental results are shown on Section V. Finally, some conclusions are drawn on Section VI.

## II. Problem and System definition

We assume an independent periodic task set [5] to be scheduled upon an homogeneous $m$-processor platform.

*Definition 1:* Let $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ be a set of $n$ independent periodic tasks under hard real time (HRT) constraints. Each task is identified by the $3-tuple$ $\tau_i = (cc_i, d_i, \omega_i)$, where $cc_i$ is the worst-case execution time in processor cycles (WCET) which takes to complete any instance of the task (*job*), $\omega_i$ its period and $d_i$ its deadline ($d_i = \omega_i$). We only consider $\omega_i = d_i$ (*implicit deadlines*) in this work and therefore we define a task as the pair $\tau_i = (cc_i, \omega_i)$.

According to this definition we generally use $\omega$ (the period) throughout the paper, referring to $d$ (the deadline) when in the context of *deadline partitioning*.

*Definition 2:* Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of $m$ identical processors, which can work at different, but discrete clock frequencies $f \in F = \{f_1, \ldots, f_{max}\}$.

A task $\tau_i$ executed on a processor $P_j$ at its maximum frequency $f_{max}$, requires $c_i = \frac{cc_i}{f_{max}}$ processor time at every $\omega_i$ interval, such that its utilization is calculated as: $u_i = \frac{cc_i}{\omega_i f_{max}}$. A task set with implicit deadlines is feasible if the following (sufficient) condition holds [10]:

$$U = \sum_{i=1}^{n} \frac{cc_i}{\omega_i f_{max}} \leq m \tag{1}$$

$U$ is the system utilization and $m$ is the number of processors.

*Definition 3:* The hyperperiod ($H$) of task set $\mathcal{T}$ is the least common multiple (lcm) of the periods of all the tasks in $\mathcal{T}$ and the quantum is the greatest common divisor (gcd) of the periods of all tasks in $\mathcal{T}$.

A job is any instance of a periodic task $\tau_i$. Therefore, a periodic task can produce an infinite number of jobs.

*Definition 4:* Let $\mathcal{J} = \{j_1, \ldots, j_x\}$ denote all the jobs generated by task set $\mathcal{T}$ that have their arrival times and deadlines within $H$, such that each $\tau_i$ generates $\frac{H}{\omega_i}$ jobs.

We also declare the mapping $L : \mathcal{J} \rightarrow \mathcal{T}$ where $j_x \in \mathcal{J}$ and $\tau_i \in \mathcal{T}$, such that $L$ relates a job to its task.

*Definition 5:* A cyclic executive (CE) is a schedule determined prior to run-time. It describes a sequence of jobs to be executed on a fixed period of time called *minor cycle*. The complete execution of the task set is defined in several *minor cycles*, which are then grouped in a *major cycle*, also called *major frame*.

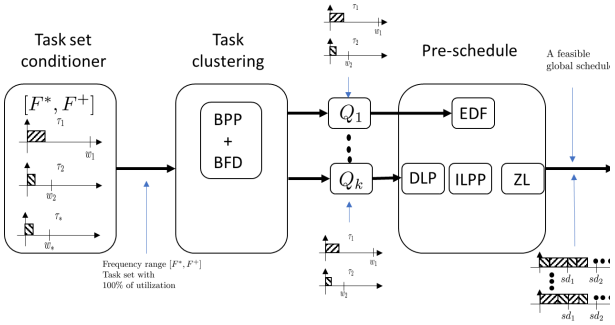



Fig. 2: CAlECS offline modules used to compute a cyclic executive

On [13], the *major cycle* is selected as the hyperperiod and the *minor cycle* as the quantum. Formally, the problem addressed in this work is stated as follows.

**Problem** *1:* Given the sets $\mathcal{T}$ and $\mathcal{P}$ of tasks and CPUs, respectively, the problem consists in designing a CE that accounts for the overhead of both migrations and preemptions.

## III. Thermal-aware cyclic executive with optimal utilization

AdWCET can virtually use any appropriate scheduler to compute a CE at each iteration. This section summarizes CAlECS [2], the scheduler we leverage in this paper because of its properties (Sec. I). Fig. 2 shows CALEC's modules. The *task set conditioner* receives the task set $\mathcal{T}$ and processor set $\mathcal{P}$, and outputs a subset $\mathcal{F} \subseteq F$ of frequencies such that the task set is feasible, thermal safe and energy efficient [2]. Then, the *task clustering* module tries to partition the task set into subsets ($Q_k$) that could fit in clusters of sizes $s_j \leq m$. If the cluster has size equal to one, then it is scheduled using EDF [21], otherwise the cluster is scheduled using AlECS [22], which is an strategy that encompasses a deadline partitioned scheme, a linear programming problem and a zero-laxity policy.

The individual schedules per each cluster are then put together into a single CE within the same major frame. For every CE under CAlECS, the *major cycle* is selected as the hyperperiod $H$ of the task set. The *minor cycles* are selected as the time intervals defined from the set $SD$ of deadlines of the task set. The example in Fig. 1 entails $SD = \{0, 10, 20, 30, 40\}$; therefore the set of *minor cycles* results in $\{[0, 10), [10, 20), [20, 30), [30, 40)\}$.

## IV. The AdWECT algorithm

At every iteration, the input to AdWCET is a CE (computed with CAlECS in this paper). Then AdWCET updates the WCET according to this CE. When the WCET stabilize, the algorithm finishes. Otherwise a new iteration of AdWCET is executed. We assume that the WCET of each task used to calculate the first CE is obtained from static tools and accounts for memory conflicts but not for scheduling overheads. Alg. 1 formalizes the steps, which we outline as follows:

| **Tasks** | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $U_{tot}$ | 2 |
|---|---|---|---|---|---|---|
| $cc_i$ | 3000 | 4000 | 5000 | 4000 | | |
| $d_i$ | 4 | 6 | 12 | 24 | | |

TABLE II: Task set in the example. WCET ($cc_i$) given in cycles, relative deadlines ($d_i$) in time units.

1) Adjust WCET at every task with the new task overhead. For the first iteration this overhead is zero.
2) Check if the *adjusted* task set is still schedulable with the current frequency. Otherwise increase the frequency to make the task set schedulable again.
3) Compute a CE and the appropriate frequency for the adjusted task set.
4) Count the number of preemptions and migrations per job on the resulting schedule.
5) Given a preemption cost ($p_{cost}$) and a migration cost ($m_{cost}$), compute the overhead of each job ($j_x$) as $p_{cost}$ times its number of preemptions ($pre_{j_x}$) plus the $m_{cost}$ times its number of migrations ($mig_{j_x}$):

$$overhead_{j_x} = (p_{cost} \times pre_{j_x}) + (m_{cost} \times mig_{j_x}) \quad (2)$$

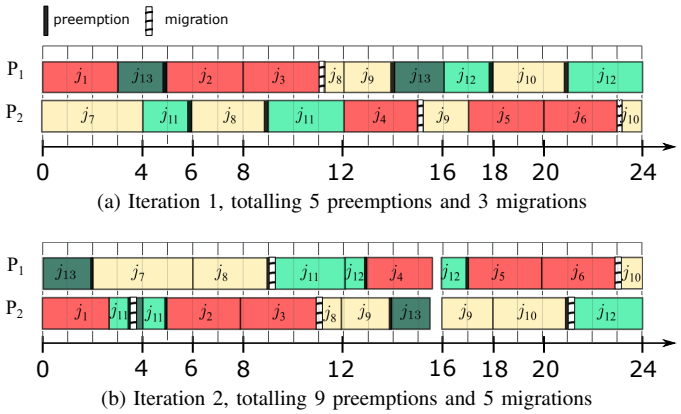
6) For each task, select the job that produced the maximum overhead and take this as the task overhead
7) For each task, check if its new overhead is greater than the previous overhead. If so, update the task with the new, higher, overhead.
8) If any task overhead was changed at the previous step, return to step 1.
9) Otherwise the algorithm has converged.

Line 13 in Alg. 1 specifies the computation of the new operating frequency when the adjusted task set is no longer schedulable due to the increased overhead. AdWCET requires a minimum of two iterations. The first one calculates a first schedule with its corresponding overheads; the second one adjusts the WCET based on this overhead. We now provide an example.

**Example 1** Consider the task set in Tab. II, to schedule on $m = 2$ processors with the available discrete frequencies $f \in F = \{1000, 1020, 1040, \ldots, 2000\}$ (Hz). The cost in cycles per preemtpion is $p_{cost} = 10$ and per migration is $m_{cost} = 20$ (cycles). This task set spawns 13 jobs. Jobs $j_1$ to $j_6$ are generated by $\tau_1$, $j_7$ to $j_{10}$ by $\tau_2$, $j_{11}$ and $j_{12}$ by $\tau_3$, and $j_{13}$ is the only job of $\tau_4$. This example requires three iterations to converge, but the overhead does not increase from the second to the third iteration. Thus the resulting task set and CE are the same. Fig. 3 displays the two CEs computed. On each figure we highlight the preemptions (black bars) and migrations (hatched bars). Both iterations are described bellow.

**Iteration 1** (Fig. 3a)

Step 1 Add overhead to original task set (none needed in 1st. iteration)
Step 2 Check schedulability. The system is schedulable with $f = 1000$



(a) Iteration 1, totalling 5 preemptions and 3 migrations

(b) Iteration 2, totalling 9 preemptions and 5 migrations

Fig. 3: Iterations for AdWCET to adjust the WCET of task set from Table II, using CAlECS as scheduling policy.

Step 3 Compute a cyclic executive: see Fig. 3a
Step 4 Count migrations and preemptions per job:
Migrations per job $[(j_8, 1), (j_9, 1), (j_{10}, 1)]$.
Preemptions per job:
$[(j_8, 1), (j_9, 1), (j_{10}, 1), (j_{11}, 1), (j_{12}, 1)]$
Step 5 Calculate overhead per job: zero for every job but $[(j_8, 30), (j_9, 30), (j_{10}, 30), (j_{11}, 10), (j_{12}, 10)]$
Step 6 Select new overhead per task: [0, 30, 10, 10]
Step 7 Check if the new task overhead $>$ current task overhead: this is true because current overhead is zero, since this is the first iteration.
Step 8 Update current task overhead: [0, 30, 10, 10]

**End of iteration 1**

**Iteration 2** (Fig. 3b)

Step 1 Add overhead to original task set:
Adjusted WCET by task: [3000, 4030, 5010, 4010]
Step 2 Check schedulability: $U = 2.00625$, the system is no longer schedulable with $f = 1000$, since $U > m$, then the new $f = \frac{1}{m}\sum_{i=1}^{n} \frac{adjustedWCET_i}{\omega_i} = 1003.12$, but because of the discrete values $f = 1020$
Step 3 Compute a cyclic executive, see Fig. 3b
Step 4 Count migrations and preemptions per job:
Migrations by job:
$[(j_8, 1), (j_{10}, 1), (j_{11}, 1), (j_{12}, 1), (j_{13}, 1)]$.
Preemption by job:
$[(j_8, 1), (j_9, 1), (j_{10}, 1), (j_{11}, 2), (j_{12}, 2), (j_{13}, 2)]$
Step 5 Calculate overhead by job: zero for every job but $[(j_8, 30), (j_9, 10), (j_{10}, 30), (j_{11}, 40), (j_{12}, 40), (j_{13}, 40)]$
Step 6 Select new overhead per task: [0, 30, 40, 40]
Step 7 Check if the new task overhead $>$ current task overhead: $0 = 0, 30 = 30, 40 > 10, 40 > 10$
Step 8 Update current overhead per task as: [0, 30, 40, 40]

**End of iteration 2**

In this second iteration the overhead of the tasks increased from [0, 30, 10, 10] to [0, 30, 40, 40], therefore the adjustment requires a third iteration, which we omit here for the sake of space. On such third iteration the overhead does not increase

**Algorithm 1** AdWCET - WCET adjustment

**Input** $\mathcal{T}$:Task set; $m$: Number of CPUs; $F$: set of CPUs frequencies; $L$: mapping of jobs to tasks, $p_{cost}$: preemption cost, $m_{cost}$: migration cost
**Output** A set of adjusted WCET;
**Aux. functions**
· CE(*task set, m, f*) – *CE computes a CE*;
· countPreemptionsMigrations(*CE*) – *Returns the number of preemptions and migrations per job from a given cyclic executive*
· taskOverhead(*L*, $overhead_j$) – *Returns the task's overhead, as the maximum overhead from its jobs*;
$ad\mathcal{T} = \mathcal{T}$, // each task $ad\tau_i \in ad\mathcal{T}$ is a 3-tuple $(adcc_i, w_i, d_i)$
$converge$ = false,
$overhead_j = [0, ..., 0]$ // holds the overhead per job $j_x \in \mathcal{J}$
$new\_overhead_\tau = [0, ..., 0]$ // holds new overhead per task $\tau_i \in \mathcal{T}$
$current\_overhead_\tau$=[0,...,0] // current overhead per task $\tau_i$
**while not** $converge$ **do**
  $ad\mathcal{T} = \{ad\tau_i | adcc_i = cc_i + current\_overhead_\tau[i]\}$
  // check schedulability of adjusted task set
  $U = \sum_{i=1}^{n} \frac{adcc_i}{f\ \omega_i}$
  **if** $U > m$ **then**
    $f = \min\{f \in F | f \geq \frac{1}{m} \sum_{i=1}^{n} \frac{adcc_i}{\omega_i}\}$
  **end if**
  $CE(ad\mathcal{T}, m, f)$
  $[pre_j, mig_j]$=$countPreemptionsMigrations(CE)$
  **for all** $j \in \mathcal{J}$ **do**
    $overhead_j[j] = (p_{cost} \times pre_j[j]) + (m_{cost} \times mig_j[j])$
  **end for**
  $new\_overhead_\tau = task\_overhead(L, overhead_j)$
  **if any** $new\_overhead_\tau > current\_overhead_\tau$ **then**
    $current\_overhead_\tau =$ max$(current_overhead_\tau, new_overhead_\tau)$
  **else**
    $converge$ =true
  **end if**
**end while**

anymore for any task, and the algorithm converges. The resulting CE appears in Fig. 3b; the adjusted WCET of the tasks are: [3000, 4030, 5040, 4040], with frequency $f = 1020$. The adjusted task set increments system utilization by $0.5\%$, but this increase ensures the accomplishment of temporal constraints.

### A. Convergence of AdWCET algorithm

AdWCET algorithm will always converge due to the fact that the overhead of the tasks is always increased or maintained, even if the current iteration yields a smaller overhead. Furthermore, the number of iterations is bounded because the same scheduling policy holds throughout the adjustment. Hence, the maximum overhead is bounded by the maximum number of preemptions ($\rho$) and migrations ($\mu$) of the preferred scheduling policy, i.e. $n(\rho+\mu)$ iterations on the worst case, where $n$ is the cardinality of $\mathcal{T}$. Actually, this bound is unlikely reached. The following section shows AdWCET performance using CAlECS [2] as the scheduling policy.

## V. Experimental results

This section shows empirically that AdWCET adjust the CE in a few iterations. Also, we will show the maximum percentage of WCET increment endured in each experiment and the overall system utilization increase. We carry out the experiments using Tertimuss [23], an open-source framework to model a RT multiprocessor system, simulate different RT schedulers, and process the results.

### A. Experimental setup

Task sets are generated using the Dirichlet Rescale (DRS) algorithm [24]. The total utilization of each task set is equal to the number of processors in the experiment, with available frequencies $f \in F = \{1000, 1020, 1040, \ldots, 2000\}$ (Hz). Task periods are randomly selected between the divisors of 60, to obtain a major cycle of at most 60 s. The task sets are executed on systems with 2 and 4 cores, with task-to-core ratios of 4, 8, 12, 16, 20, 24,28, 32, 36, 40, 44 and 48. This amounts to 100 experiments per combination, totalling 2400 experiments in all.

### B. Methodology for task generation

The task sets for the experiments are generated such that they are valid and feasible under a $m$-processor system, i.e $U \leq m$ and $u_i \leq 1\ \forall \tau_i \in \mathcal{T}$.

We first calculate task utilization leveraging the DRS algorithm, which produces $n$-dimensional vectors uniformly distributed over the valid region, such that the components sum to a specified value (*total task set utilization*). Each component accepts lower and upper bounds. The algorithm has the signature $\mathbf{u}$ = DRS($n$, $U$, $\mathbf{u^{max}}$, $\mathbf{u^{min}}$), where $\mathbf{u} = (U_1, U_2, \ldots, U_n)$ is the output vector of task utilization values, $n$ is the cardinality of the task set, $U$ is the specified total utilization and $\mathbf{u^{max}} = (U_1^{max}, U_2^{max}, \ldots, U_n^{max})$ and $\mathbf{u^{min}} = (U_1^{min}, U_2^{min}, \ldots, U_n^{min})$ are optional vectors to constrain the utilization values for the task set, by default $\mathbf{u^{max}} = (1, 1, \ldots)$ and $\mathbf{u^{min}} = (0, 0, \ldots)$.

To mimic a realistic setup, the minimum WCET and the preemption/migration costs are of different orders of magnitude. This follows the commonly accepted notion that a context switch is negligible with respect to the execution time of a task.

To this purpose, we define $\text{wcet}_{min}$ as the minimum WCET that any generated task can receive. The DRS algorithm allows imposing a lower bound for the computed utilization of every task in the generated task set, that we calculate as:

$$u_{min} = \frac{\text{wcet}_{min}}{fH} \tag{3}$$

where $f$ is the frequency and $H$ the hyperperiod. This expression is deduced from the fact that the utilization $u_i$ of any task $\tau_i$ is inversely proportional to its period $\omega_i$ (Eq. (1)). Hence, the minimum utilization is achieved with the largest possible period, which is the hyperperiod $H$.

Nevertheless, when constraining the minimum value of every task we must ensure that the utilization constraint allows for a feasible task set. Therefore, the utilization lower bound should not be greater than the total of processors, i.e:

$$\sum_{i=1}^{n} \frac{\text{wcet}_{min}}{fH} \leq m \tag{4}$$

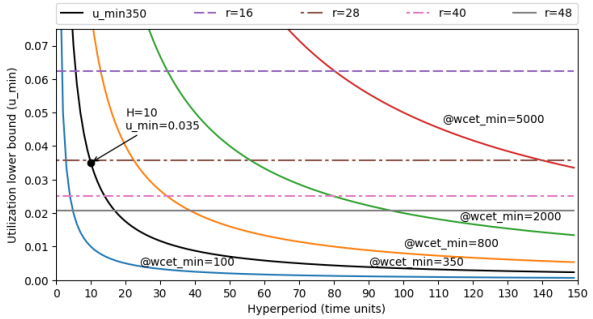



Fig. 4: Utilization lower bound $u_{min}$ for different hyperperiods. The horizontal lines represent the constraint $1/r$, where r is the task-to-core ratio. $f = 1000$, fixed for every curve

Hence, the task-to-core ratio ($r = n/m$) cannot be arbitrarily chosen. There exists a trade off between hyperperiod, frequency and task-to-core ratio. From Eq. 4 it follows that:

$$\frac{\text{wcet}_{min}}{fH} \leq \frac{m}{n} \tag{5}$$

Therefore, $u_{min}$ should always be less or equal than $1/r$ in order to generate a valid task set:

$$u_{min} \leq \frac{1}{r} \tag{6}$$

To illustrate the importance of Eq. (6) on the utilization lower bound, we plot in Fig. 4 the relation of $u_{min}$ to different values of $H$. The frequency value is fixed at $f = 1000$, and every $u_{min}$ curve is computed for each $\text{wcet}_{min} \in [100, 350, 800, 2000, 5000]$. The flat colored lines represent the constraint from Eq. (6), for four task-to-core ratios $r$. Therefore, to generate valid task sets, $u_{min}$ must be below the $1/r$ desired line.

Notice from Fig. 4 that it is easier to satisfy Eq. (6) for smaller values of $\text{wcet}_{min}$. Also, as $\text{wcet}_{min}$ increases the number of feasible task-to-core ratios decrease.

For example, with $\text{wcet}_{min} = 350$ and $H = 10$, we get $u_{min} = \text{wcet}_{min}/(f \times 10) = 0.035$, then a task set with task-to-core ratio $r = 40$ will be unfeasible, but for $H \geq 14$ this task-to-core ratio becomes feasible. On the other hand, valid task sets for $H = 10$ can be generated with any $r \leq 28$. In our setup we chose $\text{wcet}_{min} = 350$, and $H = 60$, therefore every task set on the experiments is valid.

### C. Results

AdWCET iterates to adjust the WCET of a task set according to the overhead introduced by migrations and preemptions. The number of iterations is highly dependent on the scheduling algorithm, but this number is bounded.

We generate task sets as explained in Subsec. V-A and focus on WCET and frequency increases after adjusting the CE, and on the number of iterations to reach convergence. The following figures share all the same scheme. The configurations, i.e. number of processors over the number of tasks, appear on the x-axis, whereas the y-axis holds the studied feature.

The results on WCET and frequency increments are both normalized and plotted as percentage increase in Fig. 5. In the case of WCET, we selected the maximum percentage of increase among the WCETs on each task set, i.e. it shows the WCET increment from the task that had the maximum overhead, and does not represent the behaviour of the whole task set. In the case of frequency, the percentage increase was computed as the increment between the frequency on the first iteration and the frequency at the convergence iteration. The latter indicates the required increment in system capacity. The results in Fig. 5 show that the maximum WCET increase is bounded by 30% and is more variable than the frequency increase, which lies below 6% for most cases. The increment in frequency is more stable than the maximum increment in WCET as the number of tasks per core increase. This is consistent with the fact that the latter represents the maximum overhead of a single task while the former measures the overhead on the task set.

Fig. 6 shows the number of iterations it takes AdWCET to converge to a solution for each task set in every configuration. Fig. 6a shows the iteration analysis for different task set sizes on 2-processors. The number of iterations is always below 15 except for an spurious case. Fig. 6b shows that it takes more iterations to reach convergence on 4-processors than on 2-processors. This is the logical result because the number of possible schedules increases with the number of tasks and processors.

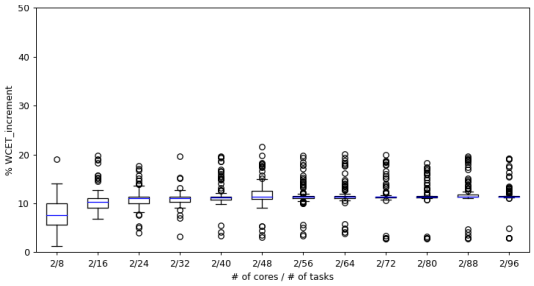
Fig. 7 shows the analysis of preemptions and migrations per job in the experiments on 2-processors. We only take into account the data from the first and last iteration. The total number of preemptions and migrations never exceeds 1, because of the chosen scheduler (CAlECS). As long as the number of tasks and processors keeps the same, the number of preemptions and migrations will be consistent in both iterations for any other scheduler.

The number of iterations rises with the ratio of tasks per processor, whereas preemptions and migrations decrease (Fig. 7, Fig. 6). This is because the theoretical bound for the iteration is $n(\rho + \mu)$, and therefore the number of iterations will always be proportional to the task set cardinality, and to the number of preemptions and migrations. Fortunately, on CAlECS both migrations and preemptions decrease as the number of tasks increase, thus keeping the number of AdWCET iterations low.

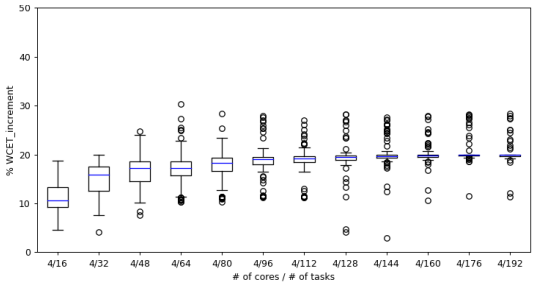
## VI. Conclusions

In this work we presented a safe methodology to adjust the WCET of a task set with HRT constraints, yielding a cyclic executive (CE) schedule. It is fully deterministic and produce minimal run-time overhead.

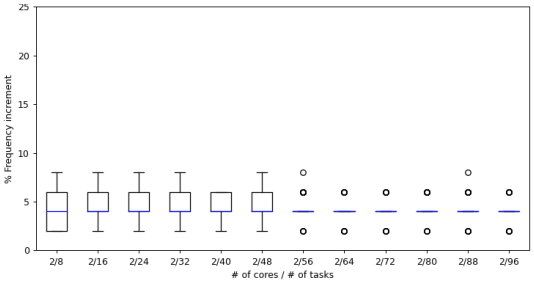
A very common rule when estimating WCET through measurement based analysis (MBA) is to increase the WCET



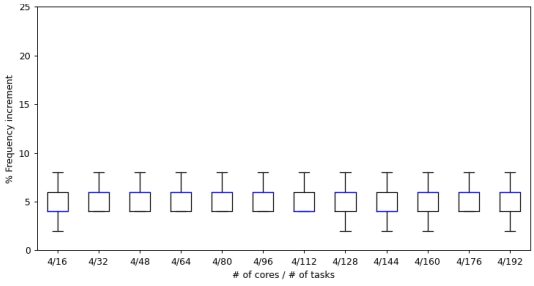

(a) Maximum WCET increment analysis with $m = 2$




(b) Maximum WCET increment analysis with $m = 4$

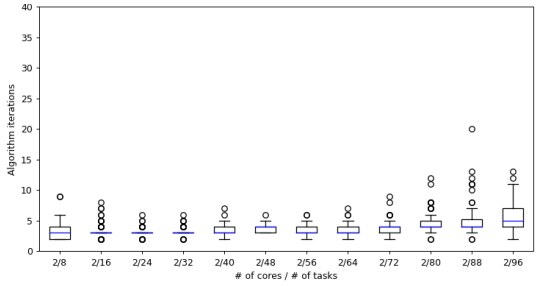



(c) Frequency increment analysis with $m = 2$

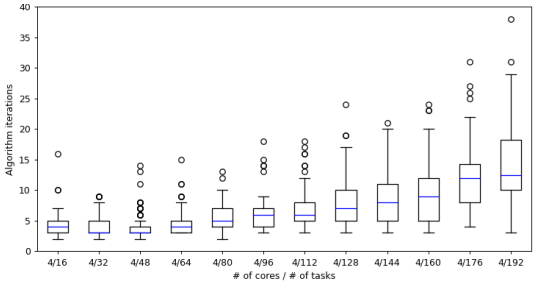



(d) Frequency increment analysis with $m = 4$

Fig. 5: Percentage of WCET maximum increase (a and b) and percentage of frequency increase (c and d)

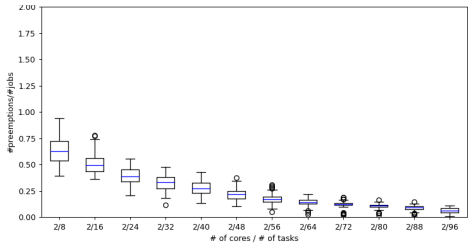



(a) Iteration analysis with $m = 2$

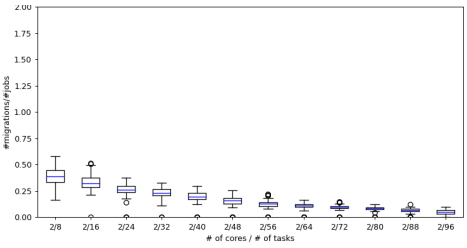



(b) Iteration analysis with $m = 4$

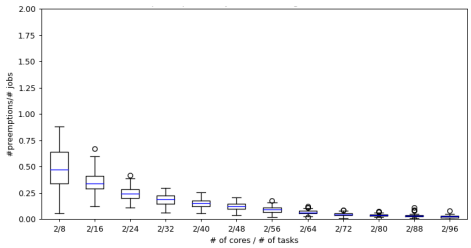
Fig. 6: Iteration analysis of AdWCET under CAlECS



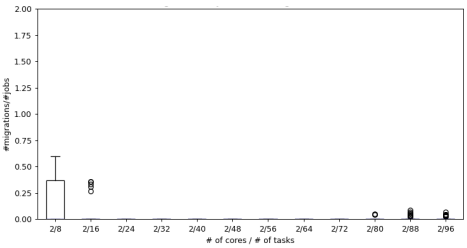

(a) Preemption analysis on first iteration




(b) Migration analysis on first iteration




(c) Preemption analysis on convergence iteration




(d) Migration analysis on convergence iteration

Fig. 7: Analysis of preemption and migrations per job on first and convergence iteration

by a given factor, usually 20% [16], such that the WCET accounts for any other overheads. Nevertheless, according to our results this thumb rule is not quite accurate and can result in both overprovisioning and deadline misses. On the one hand, if every task's WCET is increased by a 20% factor, the system capacity is also incremented, but from results in Figs. 5.c,d the system capacity only requires around 10% increment, thus incurring in overprovisioning. On the other hand, Figs. 5.a,b showed that some tasks required a higher increment than a 20%, therefore some tasks may miss their deadlines because of incorrect WCET values. In this context, AdWCET is a good methodology to compute safer and tighter bounds for WCET estimation. If used for a particular task set, it will provide accurate scheduling and WCET estimation.

Furthermore, we tested empirically the performance of AdWCET. It yielded an adjusted task set for every experiment, with less than 8% increase of system capacity on most cases. Also, the number of iterations required by AdWCET to converge was way below the theoretical bound. For example, AdWCET required 10 iterations to converge on the average case on a 4-processor system with 192 tasks, against a bound of $192(\rho + \mu)$. There is strong evidence that AdWCET is a viable methodology to obtain tighter WCET bounds when accounting for overheads due to scheduling decisions.

Future work includes taking into account the possible memory conflicts while migrating a job on specific memory hierarchies.

## ACKNOWLEDGMENT

This work was supported by grant PID2019-105660RB-C21 founded by MCIN/ AEI /10.13039/501100011033 and by the Aragón Government (T5820R research group). Also, a special thanks to CONACYT (Mexico), for providing a scholarship for the first author.

## REFERENCES

[1] B. B. Brandenburg and J. H. Anderson, "On the implementation of global real-time schedulers," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*. IEEE, 2009, pp. 214–224.

[2] L. E. Rubio-Anguiano, A. C. Trabanco, J. L. B. Velasco, and A. Ramírez-Treviño, "Maximizing utilization and minimizing migration in thermal-aware energy-efficient real-time multiprocessor scheduling," *IEEE Access*, vol. 9, pp. 83 309–83 328, 2021.

[3] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2011.

[4] E. Massa, G. Lima, and P. Regnier, "Revealing the secrets of run and qps: New trends for optimal real-time multiprocessor scheduling," in *2014 Brazilian Symposium on Computing Systems Engineering*, 2014, pp. 150–155.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[6] D.-I. Oh and T. P. Bakker, "Utilization bounds for n-processor rate monotone scheduling with static processor assignments," *Real-Time Systems*, vol. 15, no. 2, pp. 183–192, 1998.

[7] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE, 2007, pp. 149–160.

[8] J. L. Goossens and S. Funk, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Systems*, pp. 2–3, 2003.

[9] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, ser. ECRTS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 209–218.

[10] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[11] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: a unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Systems*, vol. 47, no. 5, pp. 389–429, 2011.

[12] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservation," in *IEEE Real-Time Systems Symposium (RTSS 2016)*, 2016, pp. 99–110.

[13] C. Deutschbein, T. Fleming, A. Burns, and S. Baruah, "Multi-core cyclic executives for safety-critical systems," *Science of Computer Programming*, vol. 172, pp. 102–116, 2019.

[14] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "A comprehensive survey of industry practice in real-time systems," *Real-Time Systems*, pp. 1–41, 2021.

[15] N. Diniz and J. Rufino, "Arinc 653 in space," in *Proceedings of the DASIA 2005-DAta Systems in Aerospace*, vol. 602, Euro Space - CANADIAN SPACE AGENCY – CNES – ESA EUMETSAT – EUROCONTROL. ESA Publications Division, May 2005.

[16] J. Abella, C. Hernández, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-Askasua, J. Perez, E. Mezzetti, and T. Vardanega, "Wcet analysis methods: Pitfalls and challenges on their trustworthiness," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–10.

[17] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston *et al.*, "Proartis: Probabilistically analyzable real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, pp. 1–26, 2013.

[18] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Multiprocessor scheduling by reduction to uniprocessor: an original optimal approach," *Real-Time Systems*, vol. 49, no. 4, pp. 436–474, Jul 2013.

[19] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "Is semi-partitioned scheduling practical?" in *2011 23rd Euromicro Conference on Real-Time Systems*. IEEE, 2011, pp. 125–135.

[20] S. Baruah, M. Bertogna, and G. Butazzo, *Multiprocessor Scheduling for Real-Time Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2015.

[21] M. L. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proceedings of IFIP Congress (IFIP'74) )*, 1974, pp. 807–813.

[22] L. Rubio-Anguiano, G. Desirena-López, A. Ramírez-Treviño, and J. Briz, "Energy-efficient thermal-aware multiprocessor scheduling for real-time tasks using TCPN," *Discrete Event Dynamic Systems*, pp. 1–28, 2019.

[23] G. Desirena, L. Rubio, A. Ramirez, and J. Briz, "Thermal-Aware HRT Scheduling simulation framework," 2019a, https://webdiis.unizar.es/gaz/repositories/tertimuss.

[24] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 76–88.