Combining admission tests for heuristic partitioning of real-time tasks on ARM big.LITTLE multi-processor architectures*

Agostino Mascitti[a], Tommaso Cucinotta[a], Luca Abeni[a]

[a]*Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56124, Pisa, Italy*

Abstract

This paper tackles the problem of admitting real-time tasks on non-symmetric multiprocessor platforms, where partitioned EDF-FF is used. We extend the already obtained results for symmetric multiprocessor platforms and provide an improved equation based on the number of tasks in the taskset that considers the first few heaviest tasks in the taskset. This equation has been enveloped in three effective and efficient admission tests that are suitable for being used online and in dynamic environments to partition and admit tasks on NUMP platforms and, specifically, on ARM big.LITTLE architectures. This results in an improvement of the state of the art in terms of the considered architectures and looks promising in terms of admitted tasksets as shown by an extensive number of tests performed on tasksets generated randomly with the widely adopted *randfixedsum* algorithm.

Keywords: Real-time scheduling, ARM big.LITTLE, heterogeneous multicore processing, EDF admission test

Email addresses: agostino.mascitti@santannapisa.it (Agostino Mascitti), tommaso.cucinotta@santannapisa.it (Tommaso Cucinotta), luca.abeni@santannapisa.it (Luca Abeni)

*This work has received funding from the European Commission through the EU H2020 research project AMPERE (A Model-driven development framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimization) under the grant agreement no. 871669.

1. Introduction

In recent years, embedded systems have embraced multi-core architectures to face the relentless growth of performance required by users. In particular, in the mobile domain, the performance growth also faced the important problem of energy-efficiency, which led to the adoption of DVFS-capable architectures, first and foremost the ARM big.LITTLE one. This is nowadays a fundamental characteristic of a plethora of mobile and tablet devices on the market, where soft real-time applications are becoming more and more important, for example in the multimedia and the gaming domains.

The big.LITTLE design deviates from classical symmetric multi-processing (SMP), in that it introduces two different types of cores sharing the same instruction-set architecture (ISA), so that tasks can be seamlessly migrated among them, as in SMP, but with different frequency vs power consumption curves: *LITTLE* cores specialize in low-energy computing whilst *big* cores specialize in performance. The two types of cores are normally capable of switching among principally different but partially overlapped frequency steps. However, the differences in the internal micro-architecture and pipeline design for the two core types causes a task running on a LITTLE core to take longer for execution and to consume less power than when running on a big core at the same frequency. Nowadays, the Dynamic Voltage and Frequency Scaling (DVFS) capabilities of big.LITTLE architectures are constrained to being able to set a single frequency for each of the two core type islands, albeit the most recent and advanced developments of the architecture, named DynamIQ[2], will remove this constraint.

The presence of real-time workloads requires the scheduler to make fast decisions on whether to admit or not new tasks in the system, where at a time t there can be any number of tasks ready to be scheduled, and tasks can enter or leave the system at any time. In such an open and dynamic enviroment, the need for fast and effective admission tests is a relevant feature of any operating system (OS) scheduler. In the case of the Linux kernel, for example, real-time workloads are supported either via POSIX RR/FIFO policies, or by the SCHED_DEADLINE scheduler. This is a multi-processor variant of the well-known constant bandwidth server (CBS) employing a reservation-based scheduling strategy that can be conveniently configured as using either

[2]More information is available at: `https://www.arm.com/why-arm/technologies/dynamiq`.

global or partitioned or clustered EDF scheduling underneath. However, no fast and effective admission test for the ARM big.LITTLE architecture is available.

1.1. Contributions

In this paper, a simple equation (see Equation (15) below) is proposed to admit a taskset on a *non-uniform multi-processor* (NUMP) platform such as the ARM big.LITTLE one and under P-EDF scheduling[3]. The proposed equation follows the interesting technique proposed in [1], where we take into account the utilizations of the heaviest few k tasks in the set, bounding the utilization of the remaining tasks with the one of the k^{th} heaviest task.

The proposed equation is embedded within three different admission tests and combined with the SMP tests for big.LITTLE architectures, i.e., the utilization-based test [2] and the heuristic proposed in [1] over task sets randomly generated using the well-known `randfixedsum` algorithm [3]. The evaluation is performed on a reference ARM big.LITTLE platform (ODROID-XU3) and focuses on the capability to admit additional tasks into the system, as well as the additional overheads due to the computational complexity needed for the new tests. We show that the technique can be proposed as an on-line admission test for dynamic real-time systems where real-time tasks can enter and leave at any time. Our approach does not require to know the whole taskset from the beginning and it allows for incrementally handling new tasks entering the system.

2. Related Work

The problem of real-time tasks partitioning can be formulated as an instance of the well-know bin-packing problem, where items must be allocated into bins such that the final bins weights are less than 1. In the context of real-time task scheduling, CPU cores are modelled as bins (with a size equal to the core's speed) and tasks are modelled as items (with a size equal to the task's utilization).

[3]Note that in this paper we refer to non-homogeneity in computational capabilities and speed of computations of the various cores/processors, which is different from the well-known NUMA acronym, that refers to non-homogeneity in memory access timing.

Optimal solutions to the bin-packing problem can be found by using a MILP (Mixed Integer Linear Programming) formulation. In fact, the bin-packing problem is known to be NP-hard in the strong sense, thus making other related interesting problems, like partitioning tasks to achieve the minimum energy consumption, intractable in polynomial time as well. Different bin-packing heuristics (designed to work around the problem complexity) have been investigated in previous works and some interesting conclusions are reported in the state of the art. Simchi-Levi [4] provides worst-case results for the heuristics First-Fit, Best-Fit and their respective versions in which items are ordered in decreasing order. The authors also provide the absolute performance ratios, which gives the heuristic solution's maximum deviation from optimality. Likewise, [5] revisits some of the heuristics of the bin-packing problem and compares the approximation ratio of these algorithms as a function of the total size of the input items. Johnson [6], and similarly [7], proposes simple, polynomial-time, heuristic algorithms for finding approximate solutions, which are analyzed with respect to their worst-case behavior. Finally, [8] shifts the focus from bin-packing towards bounding the scheduling anomalies on multiprocessor systems when scheduling tasksets of DAG tasks. The reader can refer to [9] for a survey on the heuristics for the bin-packing problem and its variants. The survey emphasizes the worst-case performance guarantees that are provably achievable, and it discusses work that has been done on the expected performance and behavior "in practice", mentioning also some of the many applications of these problems. Notice, however, that the approaches mentioned above can only be used for off-line tasks partitioning.

Efficient heuristics for on-line bin-packing (which allow partitioning tasks that start and terminate dynamically) have also been investigated. Gambosi [10] allows a constant number of elements to move from one bin to another, as a consequence of the arrival of a new input element, and possible algorithms are presented. Ivković [11] shows that the ability to move more than a constant number of items is necessary for accomplishing highly competitive, time-efficient fully dynamic approximation algorithms for bin packing. Balogh [12] continues the work in [11] by improving the lower bound on the asymptotic worst-case ratio, while [13] uses a new dynamic rounding technique and novel methods to handle small items in a dynamic setting such that no amortization is needed. Balogh [14] defines and analyzes a semi-on-line algorithm where for each step at most k items can be repacked, for some positive integer k. Semi-on-line algorithms for the bin-packing problem al-

low, in contrast to pure on-line algorithms, the use of repacking, reordering or lookahead before packing the items. Finally, [15] splits bins and items into classes and proposes an algorithm to dynamically partition items into bins based on their classes.

In the context of real-time scheduling, one of the first important results is the computation of a utilization bound for the First-Fit heuristic [2] on homogeneous multi-cores. This result has then been extended to uniform heterogeneous multi-cores, where different cores may have different speeds. Methods for finding an approximate utilization bounds for partitioned scheduling on heterogeneous multiprocessors are presented in [16], while [17] proposes a strategy to partition sporadic tasks onto cores and then uses Rate Monitonic to schedule tasks on each core.

The next logical extension is to consider platforms such as ARM big. LITTLE, where cores are clustered in islands of homogeneous cores (see [18] for a survey of real-time scheduling on such kind of heterogeneous systems). In this case, tasks can be partitioned between the various islands (allowing tasks to migrate within an island), or can be statically assigned to cores [19]. If inter-island migrations are allowed, then optimal scheduling algorithms can also be developed [20], by allocating fractional parts of the tasks to the various islands ([21] is another example of this approach). Other works [22, 23] use ILP formulations (thus considering only off-line assignments) to partition independent constrained-deadline sporadic tasks upon heterogeneous multiprocessor platforms, or extend the study to tasks with shared resources [24]. Finally, some other works [25] tried to adaptively minimize the energy consumption under a dynamically partitioned EDF scheme on big.LITTLE CPUs.

In this paper, we propose an improved equation for partitioned EDF systems starting from the approach proposed in [1] and suitable for being used on DVFS-enabled platforms such as the ARM big-LITTLE and DynamIQ ones. The latter combines the original utilization-based admission test [2] (which uses the utilization of the heaviest task to compute the utilization bound) with a novel test based on the *number of tasks* in the taskset, using not only the highest utilization task in the taskset, but also the subsequent highest few ones, resulting in the capability to save a number of partitionable tasksets that the original test would not admit.

3. Background

A well-know utilization-based admission test for a taskset $\Gamma = \{\tau_i\}_{i=1..m}$ with utilization $U_i \leq 1$ on a symmetric multicore platform (SMP) with n cores with capacity $B_j = 1 \; \forall j \in 1..n$ where tasks are scheduled with EDF and using the First-Fit allocation strategy (EDF-FF) is given in [2]:

$$\sum_{i=1}^{m} U_i \leq \frac{n\beta + 1}{\beta + 1} \tag{1}$$

where $\beta = \left\lfloor \frac{1}{\alpha} \right\rfloor$ and $\alpha = \max_{i=1..m}\{U_i\}$.

This test implicitly assumes that all tasks have the same utilization α and thus it discards many tasksets that would be actually schedulable. This behaviour is particularly showy when considering tasksets where the highest-utilization task is considerably bigger than the other ones.

To overcome this issue, we proposed in [1] an extended test that considers also a few smaller tasks, besides the biggest one α, and works in combination with Equation (1) in the SMP case. The test considers the possible placement options for the heaviest $k-1$ tasks of the taskset, assuming that the remaining tasks have a utilization equal to the one of the k^{th} heaviest task. This results in a test on the maximum number m^{max} of tasks in the taskset that can be admitted on the platform under partitioned EDF scheduling with any placement strategy (e.g., First-Fit and Worst-Fit). It is suitable to be used for incrementally partitioned tasksets and dynamic scenarios, where tasks can enter and leave the system at any time. For example, in the taskset $\Gamma = \{0.799, 0.342, 0.196, 0.192, 0.182, 0.124, 0.064197\}$, Equation (1) would consider only the task with utilization $\{0.799\}$ and would implicitly assume that the remaining tasks have the same (high) utilization 0.799. On other hand, the test in [1], with $k = 4$, would consider the possible placements of the tasks with utilizations $\{0.799, 0.342, 0.196\}$, and implicitly assume that the remaining tasks have utilization 0.192, which makes the latter test more flexible and robust in terms of number of admitted tasksets in the system. Notice that the taskset Γ can be increasingly built over time and the test can be repeated at each task arrival.

Two formulations of the test have been proposed in [1], both growing in computational complexity with the parameter k, and we provide an intuition below for completeness (see Section 3.1). They are different not only in computational complexity but also in their effectiveness (the combinatorial

test is always better or equivalent to the logarithmic test). Both formulations assume that tasks are sorted in decreasing utilization order. The one with combinatorial computational complexity (with the parameter k) is:

$$m^{max} = k - 1 + \min_{\{\Gamma_j\}} \left\{ \sum_{j=1}^{k-1} \frac{1 - \sum_{i \in \Gamma_j^H} U_i}{U_k} \right\} + (n - k + 1) \left\lfloor \frac{1}{U_k} \right\rfloor \qquad (2)$$

where the minimum is carried out over the possible allocations $\{\Gamma_j\}$ of the heaviest $k-1$ tasks over $k-1$ cores, Γ_j^H denotes the subset of the first $k-1$ heaviest tasks hosted on core j and U_k is the utilization of the k^{th} heaviest task. The logarithmic-complexity approximated test is:

$$m^{max} \geq 1 + \left\lfloor \frac{k - 1 - \sum_{i=1}^{k-1} U_i}{U_k} \right\rfloor + (n - k + 1) \left\lfloor \frac{1}{U_k} \right\rfloor \qquad (3)$$

In this case, we must check if $m \leq m^{max}$ and the final admission test for SMP becomes the combination (i.e., the logical OR) of Equation (2) or Equation (3), with Equation (1), which can be performed in logarithmic time on the number of tasks if the taskset needs to get sorted in decreasing order.

Simulations on tasksets randomly generated using `randfixedsum` [3] showed that already with $k = 4$ the technique increases in a good way the number of admitted tasksets with respect to the case in which only Equation (1) is used. This is shown in Figure 1 for example, where on the X axis we report the total utilization of the tasksets that have been generated in the range $1.5 - 3.0$, and on the Y axis the number of admitted tasksets, over the total of 100 tasksets of 6 tasks that were generated for each utilization, to be admitted onto a symmetric quad-core system (maximum utilization 4.0).

The various coloured bins correspond to different admission tests: the *Utilization-based test* refers to Equation (1); the *Combinatorial test* refers to Equation (2) for $k \in \{3, 4\}$, and the *Linear test* refers to the approximated test in Equation (3) for $k \in \{3, 4\}$; the *Util.-based or Comb.* refers to the logical OR between Equation (1) and Equation (2) and the *Util.-based or Linear* refers to the logical OR between Equation (1) and Equation (3). As evident, both the combinatorial and the approximated faster linear tests, in logical OR with Equation (1), manage to admit a substantial number of tasksets in the range of high utilizations, whereas the original test of Equation (1) alone would admit no tasksets at all.

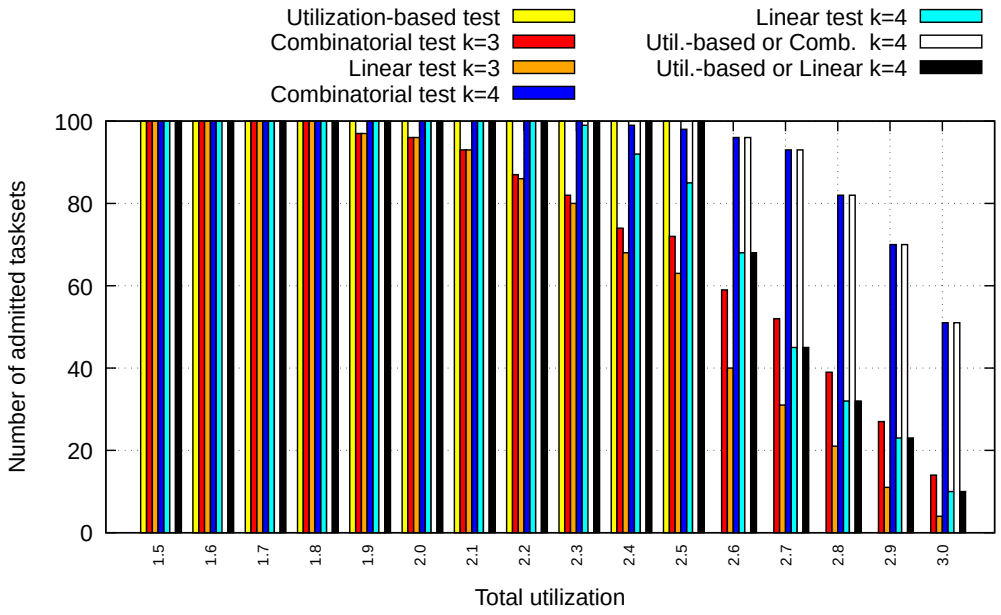


Figure 1: Comparison among different admission tests as in [1] in the case of a symmetric multicore platform with 4 cores and considering tasksets of 6 tasks.

3.1. Intuition about the tests in [1]

We now give an intuition about Equation (2) and Equation (3) and the reader can refer to [1] for a complete description of the state-of-the-art approach.

For simplicity and without loss of generality, we assume that $m \geq n \geq k$. Also, remember that here we take into account a SMP platform, where each core maximum capacity is 1, while in the rest of the paper we consider a NUMP platform, where cores may also have lower maximum capacities (for example, in the ARM big.LITTLE case, the LITTLE cores maximum capacity is $B_L = 0.345328$).

The main idea is that we have to consider all of the possible allocations of the heaviest $k-1$ tasks partitioning them across a subset of $k-1$ processors. Without loss of generality, we can assume these are the first $k-1$ processors among the n ones available on the platform. Whenever each of said processors j hosts a subset Γ_j^H of the first $k-1$ heavy tasks, with overall utilization $U_{\Gamma_j^H} \triangleq \sum_{i \in \Gamma_j^H} U_i \leq 1$ (note Γ_j^H can be an empty set for one or more processors here), its utilization available to host small tasks is $1 - U_{\Gamma_j^H}$, therefore each

processor $j < k$ can host a maximum number of small tasks m_j^S equal to:

$$m_j^S = \left\lfloor \frac{1 - \sum_{i \in \Gamma_j^H} U_i}{U_k} \right\rfloor, \text{ for } j < k, \tag{4}$$

where the $n - k + 1$ remaining processors will exclusively host small tasks, so the maximum number of tasks m_j^S for $j \geq k$ is:

$$m_j^S = \left\lfloor \frac{1}{U_k} \right\rfloor, \text{ for } j \geq k. \tag{5}$$

Therefore, the number of tasks m^{max} that can be admitted is:

$$\begin{aligned} m^{max} &= \sum_{j=1}^{k-1} \left(\left| \Gamma_j^H \right| + m_j^S \right) + \sum_{j=k}^{n} m_j^S \\ &= k - 1 + \sum_{j=1}^{k-1} \left\lfloor \frac{1 - \sum_{i \in \Gamma_j^H} U_i}{U_k} \right\rfloor + (n - k + 1) \left\lfloor \frac{1}{U_k} \right\rfloor \end{aligned} \tag{6}$$

However, (i) we do not know a-priori where the $k - 1$ heaviest tasks will be placed, and thus the sets $\{\Gamma_j^H\}$ above are unknown, and (ii) we are interested in formulating a test that holds regardless of the exact location of the heaviest tasks $\{\Gamma_j^H\}$. So, we want to compute the minimum possible m^{max} as derived accounting for all the possible distributions of the $k - 1$ heaviest tasks over $k-1$ processors respecting the single-processor EDF schedulability test $U_{\Gamma_j} \leq 1 \; \forall j < k$.

Therefore, the test to be applied requires to verify that $m \leq m^{max}$ as derived from eq. (6) for any possible partitioning of the $k - 1$ tasks with highest utilization across $k - 1$ processors that respect the saturation bound of $U_{\Gamma_j^H} \leq 1 \; \forall j < k$ and, since we need to consider the worst-case scenario (in fact, the actual scheduling strategy that will be applied is unknown a-priori), we add the minimum to the formulation, obtaining Equation (2).

To derive the logarithmic formula Equation (3), we exploit the fact that

$$\forall x, y \in \mathbb{R}, \; \lfloor x \rfloor + \lfloor y \rfloor \geq \lfloor x + y \rfloor - 1. \tag{7}$$

m^{max} as from eq. (6) above can be bounded as Equation (3), since the m^{max} formulation in eq. (6) contains the summation of the result of $k - 1$ ceil operations (one for each of the $k-1$ first cores possibly hosting heavy tasks),

which can be bounded applying the approximation in eq. (7) exactly $k-2$ times, resulting in the first term of $(k-1)$ in eq. (6) being reduced to just 1. Note that this bound is correct for any possible partitioning $\{\Gamma_j^H\}$ of the $k-1$ heaviest tasks across the first $k-1$ cores, and regardless of which subsets of said tasks actually fit onto a single core. Indeed, the final bound for m^{max} turns out to be identical in all these cases, and equal to the one in Equation (3).

4. Proposed approach

Consider a NUMP platform where cores have capacities $B_j \leq 1 \; \forall j \in \{1 \dots n\}$ and let $\Gamma \equiv \{\tau_1 \dots \tau_m\}$ be a generic taskset where tasks have *nominal* utilizations $U_i \leq 1 \; \forall i \leq m$ (i.e., each task utilization U_i is referred to the core with maximum capacity). We are interested in a test that is suitable to be used online and in a dynamic environment and that works for a generic NUMP architecture, as for example a multicore platform with DVFS capabilities. A simplistic and fast admission test is the trivial application of Equation (1):

$$\sum_{i=1}^{m} U_i \leq U(m, n, \alpha) = \frac{\beta n + 1}{\beta + 1}$$

with $\beta = \left\lfloor \frac{min_{j=1 \dots n} B_j}{\alpha} \right\rfloor$ (*alpha* is the highest utilization) where we consider the platform as if it had symmetric cores and each core had capacity the minimum one among the processors. However, this would result in a very pessimist test since: 1) it considers the minimum among all the core utilization bounds B_j, discarding a potentially large part of the cores capacity; 2) it considers all tasks as if they had the maximum utilization α.

A possible way to tackle the pessimism is to use the same idea of [1], which is to consider not only the heaviest task of the taskset but also the k heaviest ones. Without loss of generality, from now on assume the tasks are sorted in decreasing utilization order, i.e., $U_1 \geq U_2 \geq \dots \geq U_m$ (and $\alpha \equiv U_1$). Also, we assume for simplicity that $m \geq n \geq k$.

In the simplest (non-trivial) case $k = 2$ we must consider, for each possible placement of the heaviest task with utilization U_1 onto any j of the n cores, that the remaining "small" $m-1$ tasks, all bounded with utilization $\leq U_2 \leq U_1$, would have to fit in the leftover utilization $B_j - U_1$ on the CPU j where

U_1 was deployed, plus the full capacity available on the remaining $n-1$ cores. Thus the maximum number of small tasks m^S that fit on the platform is:

$$m^S = \min_{\substack{j=1\dots n,\\ B_j \geq U_1}} \left\{ \left\lfloor \frac{B_j - U_1}{U_2} \right\rfloor + \sum_{\substack{h=1\dots n,\\ h \neq j}} \left\lfloor \frac{B_h}{U_2} \right\rfloor \right\} \tag{8}$$

Generalizing for $k \geq 3$, we need to consider all the possible ways to distribute the $k-1$ heavy tasks among the n cores so that each core is not overcommitted. We obtain the following general formula:

$$m^S = min_{\{\Gamma_j\}} \sum_{j=1\dots n} \left\lfloor \frac{B_j - \sum_{i \in \Gamma_j} U_i}{U_k} \right\rfloor \tag{9}$$

where the minumum is carried out over all the possible assignments $\{\Gamma_j\}$ of the "heaviest" $k-1$ tasks onto the n cores s.t.

$$\begin{aligned} \cup_{j=1\dots n} \Gamma_j &= \{\tau_1 \dots \tau_{k-1}\}, \\ \Gamma_j \cap \Gamma_h &= \emptyset \ \forall j \neq h, \\ \sum_{i \in \Gamma_j} U_i &\leq B_j \ \forall j = 1 \dots n \end{aligned} \tag{10}$$

Therefore, the maximum number m^{max} of tasks that can be admitted is:

$$m^{max} = k - 1 + min_{\{\Gamma_j\}} \sum_{j=1\dots n} \left\lfloor \frac{B_j - \sum_{i \in \Gamma_j} U_i}{U_k} \right\rfloor \tag{11}$$

With respect to Equation (2) presented in [1], Equation (11) must consider every single core in the computation and cannot rely on the fact that all cores have the same maximum capacity $B_i = 1 \ \forall i = 1 \dots n$, thus the formula is less straightforward since we cannot assume to have $n-k+1$ free cores that simplify the final equation. Figure 2 shows an example of partitions that we must check with Equation (11) for the taskset $\Gamma = \{0.9237, 0.5331, 0.3762\}$ to be placed on cores with capacities $\{1.0, 0.6, 0.8, 0.7\}$, in the cases $k = 3$ and $k = 4$.

However, Equation (11) must check all the possible assignments of the heaviest $k-1$ tasks onto the n cores, and this makes its computational complexity explode. We observe that all the partitions of the $k-1$ heaviest tasks on n cores will have at most $k-1$ cores with some of those tasks and

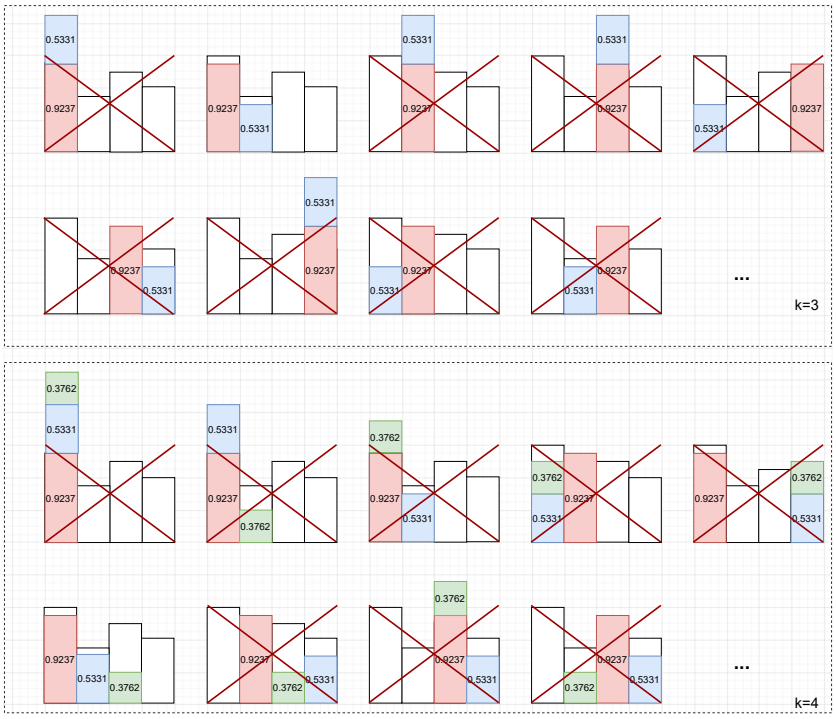


Figure 2: Some of the possible partition configurations that Equation (11) must check for a generic NUMP, for k = 3 (top) and k = 4 (bottom) when applied to $\Gamma = \{0.9237, 0.5331, 0.3762\}$ and core capacities $= \{1.0, 0.6, 0.8, 0.7\}$ for the cores from left to right respectively. Configurations that are impossible due to violation of EDF schedulability are marked with a red cross. All the cores must undergo the check.

at least $n - k + 1$ cores without any of them, so they only host lightweight tasks $\{\tau_k \dots \tau_m\}$. On the $k-1$ cores we still have to try all the possible ways to partition the heavy tasks $\{\tau_1 \dots \tau_{k-1}\}$. This is shown in Figure 2, where (in the case of $k = 3$ for simplicity) we try to partition the two heaviest tasks on one or two cores in all the possible ways, and in each valid partitioning the remaining cores are left free and they can host the remaining "smaller" tasks with utilization $U_k = U_3$. In other words, all the possible partitions of $k - 1$ tasks onto n cores can be thought of as all the possible subsets of $k - 1$ cores, on which we try all the possible assignments of $k - 1$ tasks, thus Equation (11) can be written as:

$$m^{max} = k - 1 + \min_{\substack{\mathcal{C} \subseteq \{1 \dots n\}, \\ |\mathcal{C}| = k-1}} \left\{ \min_{\{\Gamma_j\} \in \Gamma(\mathcal{C})} \left\{ \sum_{j=1}^{k-1} \left\lfloor \frac{B_j - \sum_{i \in \Gamma_j} U_i}{U_k} \right\rfloor + \sum_{j \notin \mathcal{C}} \left\lfloor \frac{B_j}{U_k} \right\rfloor \right\} \right\} \tag{12}$$

where $\mathcal{C}$ denotes a subset of cores, $|\mathcal{C}|$ denotes the cardinality of $\mathcal{C}$ and $\Gamma(\mathcal{C})$ is the set of the possible ways to assign the first $k-1$ tasks to the $k-1$ cores in $\mathcal{C}$:

$$\begin{aligned} \cup_j \Gamma_j &= \{\tau_1 \dots \tau_{k-1}\}, \\ \Gamma_j \cap \Gamma_h &= \emptyset \ \forall j \neq h, \\ \Gamma_j &= \emptyset \ \forall j \notin \mathcal{C} \\ \sum_{i \in \Gamma_j} U_i &\leq B_j \ \forall j = 1 \dots n \end{aligned} \tag{13}$$

The test in Equation (12) can be further simplified by exploiting the fact that:

$$\forall x, y \in \mathbb{R}, \ \lfloor x \rfloor + \lfloor y \rfloor \geq \lfloor x + y \rfloor - 1 \tag{14}$$

so the exact formulation of m^{max} in Equation (12) can be bounded as:

$$m^{max} \geq 1 + \min_{\substack{\mathcal{C} \subseteq \{1 \dots n\}, \\ |\mathcal{C}| = k}} \left\{ \left\lfloor \frac{\sum_{j \in \mathcal{C}} B_j - \sum_{i=1}^{k-1} U_i}{U_k} \right\rfloor + \sum_{j \notin \mathcal{C}} \left\lfloor \frac{B_j}{U_k} \right\rfloor \right\} \tag{15}$$

In fact, after applying Equation (14), the terms $\sum_{j=1}^{k-1} \sum_{i \in \Gamma_j} U_i$ in Equation (12) turn into $\sum_{i=1}^{k-1} U_i$ and, since we have a sum of $k-1$ terms, we need to apply Equation (14) $k-2$ times, obtaining the final formula in Equation (15).

About the computational complexity of Equation (15), for $k = 2$ we need to find all the subsets of cores of cardinality $k - 1 = 1$, and thus the complexity is quadratic in the number of cores (n computations, each requiring the calculation of n terms). In the general case of $k > 1$ we need to find all the possible ordered subsets of $k-1$ cores given n processors, and compute n terms each time. Therefore, the computational complexity is:

$$O\left(n(k-1)!\binom{n}{k-1}\right) = O\left(n\frac{n!}{(n-k+1)!}\right) \tag{16}$$

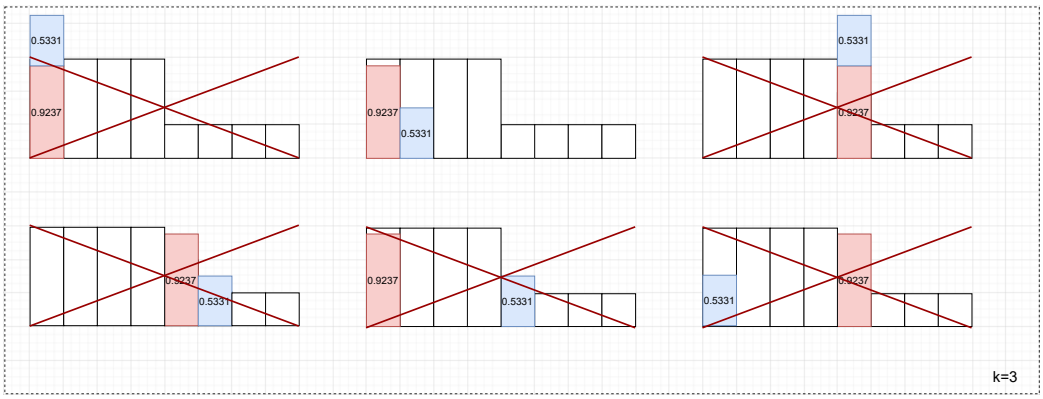


Figure 3: The possible partition configurations that Equation (15) must check in the case of the ARM big.LITTLE architecture with $n_L = n_B = 4$ and $k = 3$ when applied to $\Gamma = \{0.9237, 0.5331, 0.3762\}$. Configurations that are impossible due to violation of EDF schedulability are marked with a red cross. For this example, the big island is on the left with maximum core capacity $B_B = 1.0$ and the LITTLE island is on the right with maximum core capacity $B_L = 0.345328$. Notice that the case $k = 3$ is exhaustive and that only the first $k - 1$ cores must undergo the check.

The computation complexity of the final test based on the number of subsets of $k-1$ cores in Equation (15) may seem pretty high, but it is actually lowered in the specific case of ARM big.LITTLE architectures, where there are two islands: the n_B cores of the big island have the same capacity B_B at fixed frequency, while the n_L cores of the LITTLE island share the same capacity B_L at fixed frequency, and $n = n_B + n_L$. In fact, in this special case of a non-symmetric architecture, we must only take into account a smaller number of combinations of cores.

For example, in the case $k = 3$, Equation (15) must consider:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

combinations of cores in the general NUMP case, which for $n = 4$ means 6 combinations, while in the ARM big.LITTLE case it reduces to just 3 combinations since the two heaviest tasks can only be deployed (i) both on the big island; (ii) both on the LITTLE island; or (iii) one per island. The same promising results applies for $n = 8$ cores. Therefore, Equation (15) is greatly simplified in an interesting number of practical cases.

This concept is illustrated in Figure 3 for an ARM big.LITTLE platform with $n_B = n_L = 4$ (and thus $n = 8$ cores) and $B_B = 1$ and $B_L = 0.345328$ for the case $k = 3$ and $\Gamma = \{0.9237, 0.5331, 0.3762\}$ and it shows that many combinations that we must consider in Figure 2 are actually not needed for Equation (15). For example, it is useless to consider the combination of tasks in which all of them are deployed on the last big core, since it is equivalent to deploying them on the first big core.

As a side note, the ARM big.LITTLE architecture can never reduce to the general case of a NUMP architecture presented in Section 4 since all cores of the same island share the same frequency and thus the same capacity.

4.2. Splitting tasksets for big.LITTLE platforms

It is noteworthy to mention that, in the case of ARM big.LITTLE architectures, an alternative admission test might be designed in cooperation with a prefixed partitioning of the tasks in the taskset among big and LITTLE islands. Indeed, splitting the taskset Γ to be placed into a Γ_B to be admitted on n_B big cores each with capacity B_B, and a Γ_L to be admitted on n_L LITTLE cores each with capacity B_L, we can apply twice the tests in Equation (2) or Equation (3) to the two islands, which are SMP when considered individually. However, we need to adopt a simple partitioning scheme that can be implemented efficiently on a real system, with tasks potentially arriving and leaving dynamically.

The taskset splitting strategy that we propose to use in this paper is based on assigning to each island a percentage of the total taskset utilization close to the percentage of the island overall capacity in proportion to the overall system capacity, i.e., assuming $U_1 \geq \cdots \geq U_m$:

$$\Gamma_B = \Gamma_H \cup \left\{ \tau_1, \ldots, \tau_h \mid \frac{\sum_{i=1}^{h-1} U_i}{\sum_{i=1}^{m} U_i} < \frac{n_B * B_B}{n_B * B_B + n_L * B_L} \wedge \frac{\sum_{i=1}^{h} U_i}{\sum_{i=1}^{m} U_i} \geq \frac{n_B * B_B}{n_B * B_B + n_L * B_L} \right\}, \tag{17}$$

where $\Gamma_H = \{\tau_i \in \Gamma | U_i > B_L\}$ is the set of heavyweight tasks that can only be placed on the big island, and $\Gamma_L = \Gamma \setminus \Gamma_B$. An alternative taskset splitting technique relies simply on placing on the big island a *number* of tasks in the same percentage as the big island capacity in proportion to the overall system capacity, i.e.:

$$\Gamma_B = \Gamma_H \cup \left\{ \tau_1, \ldots, \tau_h \mid \frac{h-1}{m} < \frac{n_B * B_B}{n_B * B_B + n_L * B_L} \wedge \frac{h}{m} \geq \frac{n_B * B_B}{n_B * B_B + n_L * B_L} \right\} \tag{18}$$

$\Gamma_L = \Gamma \setminus \Gamma_B$.

It is relevant to choose the right partitioning technique for Γ as it affects the effectiveness of the SMP tests in Equation (1) and Equation (3), as it will be shown in the next section. Moreover, each test needs additional details to be realized in the scheduler implementation, on new tasks arriving in the scheduler or existing tasks leaving. For example, a test relying on the taskset splitting in Equation (18) may require to move just one task among the islands, on a task arrival or leave. Which task to move can be found efficiently by keeping tasks into a min-heap data structure for the big island, and a max-heap one for the LITTLE island. On the other hand, the splitting in Equation (17) might need additional migrations among islands, as a big task arriving may result in the migration of a multitude of smaller tasks to the LITTLE island, for example.

Moreover, one could observe that, in Equation (17), we include τ_h to Γ_B, which, added to the previous tasks, causes the percentage of utilization hosted on big cores to exceed the percentage of computational capacity of the island. However, we prefer to try keeping such a task on the big island (rather than the little one), to increase the possibility to schedule the taskset (at least in cases where the overall utilization of all the tasks is sufficiently below the overall computational capacity of the system). In fact, as this is a heuristic, we cannot pretend it is optimum and, actually, it should guarantee at least that all heavyweight tasks go to Γ_B.

Finally, note that the NUMP test proposed in Equation (15) is meant to be used without any prior taskset splitting technique. In this paper, we propose to use three different admission tests, some of them applying directly Equation (15), others recurring to the taskset splitting technique in Equation (17). In what follows, the proposed tests are described in detail, while their effectiveness is evaluated comparatively in Section 5.

4.3. Proposed admission tests

The NUMP tests in Equation (15) can be combined with the SMP tests in Equation (1) and Equation (3) to generate a number of admission tests for any type of taskset.

We define three main admission tests in this paper:

Admission Test 1: the test passes if the approximated NUMP test of Equation (15) passes, OR, after splitting the taskset with eq. (17), both tests on big and LITTLE islands made separately using the utilization-based test in Equation (1) pass;

Admission Test 2: the test passes if the approximated NUMP test of Equation (15) passes, OR, after splitting the taskset with eq. (17) on the two islands, either the simple utilization-based test in Equation (1) OR the approximated test in Equation (3) pass on both islands;

Admission Test 3: the test passes if, after splitting the taskset in heavyweight and lightweight tasks, the heavyweight ones fit on the big cores using a first-fit strategy, then the lightweight ones pass the approximated NUMP test in Equation (15) using the LITTLE island at its full capacity plus the residual utilization left by heavyweight tasks on the big cores.

As for Admission Test 1, it performs the SMP admission test with Equation (1) on both Γ_L and Γ_B obtained by splitting Γ with eq. (17), and Equation (15) on the whole taskset Γ. However, Equation (1) may easily discard many tasksets since it only considers the heaviest task of Γ_B and Γ_L, whose utilization depends on the split of the taskset discussed above. Therefore, Admission Test 2 follows an interesting way to tackle this weakness and still uses an SMP test by performing the logical OR between Equation (1) and Equation (3), which are SMP tests performed independently on the two islands.

Finally, a big part of the core capacities is discarded by the Admission Test 1 and 2. To make use as much as possible of the core capacity left by the heavy tasks, Admission Test 3 splits the taskset into heavy $\Gamma_H = \{\tau_i | U_i > B_L\}$ and lightweight tasks $\Gamma_L = \Gamma \setminus \Gamma_H$ and performs an EDF-FF allocation of the heavy tasks on the big island. If this is possible, the remaining core capacities are computed and the test in Equation (15) is performed on the lightweight tasks on the whole platform (with decreased capacities for the big island).

These three strategies are comparatively evaluated later in Section 5.

4.4. Motivational example

In this section, we provide a simple example justifying the need for the new tests proposed in this paper. Consider the taskset of 7 tasks in Table 1, already sorted for convenience, and having total nominal utilization 1.9 and an ARM big.LITTLE platform with $n_B = n_L = 2$ cores per island (for a total of $n = 4$ cores) and $B_B = 1.0$ and $B_L = 0.345328$, as the ODROID-XU3 platform.

Table 1: Taskset for the motivational example

U_1	U_2	U_3	U_4	U_5	U_6	U_7	$\sum_i U_i$
0.799	0.342	0.196	0.192	0.182	0.124	0.064197	1.9

First, the taskset Γ is partitioned into $\Gamma_B = \{0.799, 0.342, 0.196, 0.192\}$ and $\Gamma_L = \{0.182, 0.124, 0.064197\}$ with Equation (17). Equation (1) would admit Γ_L for the LITTLE island:

$$0.370197 \leq 1.5 = \frac{2\left\lfloor \frac{0.345328}{0.182} \right\rfloor + 1}{\left\lfloor \frac{0.345328}{0.182} \right\rfloor + 1}$$

the same equation applied to the big island would not:

$$1.529 > 1.5 = \frac{2\left\lfloor \frac{1}{0.799} \right\rfloor + 1}{\left\lfloor \frac{1}{0.799} \right\rfloor + 1}$$

Therefore, the SMP test of Equation (1) (i.e., the logical AND between the two islands) would fail.

Also the SMP test of Equation (3) would fail for $k = 2$ (and thus also Admission Test 2 fails). The test applied to the big island would fail:

$$4 > 3 = 1 + \left\lfloor \frac{2 - 1 - 0.799}{0.342} \right\rfloor + (2 - 2 + 1) \left\lfloor \frac{1}{0.342} \right\rfloor$$

while it would admit the taskset on the LITTLE island:

$$3 \leq 9 = 1 + \left\lfloor \frac{2 - 1 - 0.182}{0.124} \right\rfloor + (2 - 2 + 1) \left\lfloor \frac{0.345328}{0.124} \right\rfloor$$

For $k = 3$ the test of Equation (3) admits the tasksets Γ_B and Γ_L. For the big island:

$$4 \leq 5 = 1 + \left\lfloor \frac{3 - 1 - (0.799 + 0.342)}{0.196} \right\rfloor + (2 - 3 + 1) \left\lfloor \frac{1}{0.196} \right\rfloor$$

and for the LITTLE island:

$$3 \leq 27 = 1 + \left\lfloor \frac{3 - 1 - (0.182 + 0.124)}{0.064197} \right\rfloor + (2 - 3 + 1) \left\lfloor \frac{0.345328}{0.064197} \right\rfloor$$

Notice that Equation (1) and Equation (3) are used in combination within Admission Test 2 above, which fails for $k = 2$, while for $k = 3$ the test admits the taskset.

On the other hand, Equation (15) in case $k = 2$ must consider all the possible partitionings of the "heaviest" task U_1 on one core, and for each valid partitioning the remaining cores are left free and they can host "smaller" tasks with utilization $U_k = U_2$. In other words, Equation (15) considers (i) the case in which U_1 is on the first big core, (ii) the case in which U_1 is on the second big core, (iii) the case in which U_1 is on the first LITTLE core and finally (iv) the case in which U_1 is on the second LITTLE core. However, since we are in the case of an ARM big.LITTLE platform, equivalent cases (for example, U_1 on the first big core is equivalent to U_1 on the second big core) are not considered in the final formula and the only valid cases are (i) the case in which U_1 is placed on a big core and (ii) the case in which U_1 is placed on a LITTLE core. Removing impossible cases, the only valid case is the case in which $U_1 = 0.799$ is placed on a big core (since $U_1 > B_L$, which means that U_1 does not fit in a LITTLE core) and the remaining cores can be used for deploying the other tasks with (increased) utilizations $U_i = 0.342 \ \forall i \geq 2$. This test would not admit the taskset since it does not meet the schedulability check, being the number of tasks in the taskset greater than the maximum number of admissible tasks m^{max} (there is only one element in the minimization general formula):

$$7 > 5 = 1 + \min \left\{ \left\lfloor \frac{1 - 0.799}{0.342} \right\rfloor + \left\lfloor \frac{1}{0.342} \right\rfloor + 2 \left\lfloor \frac{0.345328}{0.342} \right\rfloor \right\} \tag{19}$$

For the case $k = 3$, Equation (15) must consider all the possible partitionings of the first two "heaviest" tasks U_1 and U_2 on two cores, and for each valid partitioning the remaining cores are left free and they can host "smaller" tasks with utilization $U_k = U_3$. In other words, Equation (15) must consider (i) the case in which U_1 and U_2 are both on the first big core; (ii) U_1 and U_2 are both on the second big core; (iii) U_1 is on the first big core and U_2 on the second big core; (iv) U_1 is on the second big core and U_2 on the first big core; (v) U_1 is on the first big core and U_2 on the first LITTLE core; (vi) U_1 and U_2 both on the first LITTLE core; (vii) U_1 on the first LITTLE core and U_2 on the second LITTLE core; (viii) and so on. However, since we are in the case of an ARM big.LITTLE platform, equivalent cases, like U_1 on the second big core and U_2 on the second LITTLE core (equivalent to

Table 2: Comparison of the results of the admission checks for the presented motivational example.

	k=2	k=3	(without k)
Equation (1) big island	-	-	no
Equation (1) LITTLE island	-	-	yes
Equation (3) big island	no	yes	-
Equation (3) LITTLE island	yes	yes	-
Equation (15)	no	yes	-
Admission Test 1	no	yes	-
Admission Test 2	no	yes	-
Admission Test 3	yes	yes	-

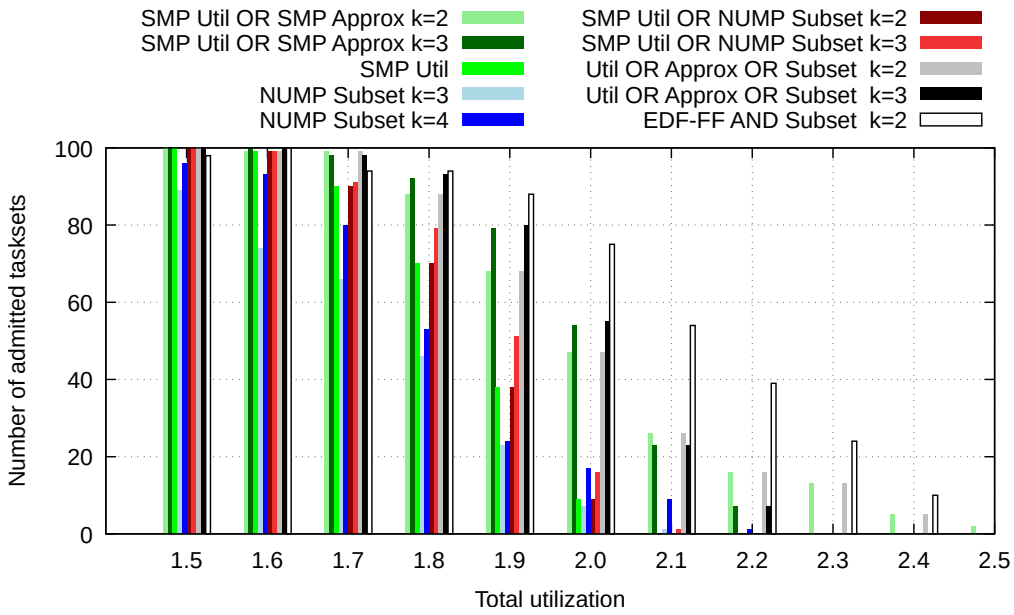
U_1 on the first big core and U_2 on the first LITTLE core), and impossible cases, like U_1 and U_2 on a LITTLE core, are not considered in the equation. Therefore, the following equation only shows (i) the case in which U_1 and U_2 are both on the big islands on two different cores; and (ii) U_1 is on the big island and U_2 on the LITTLE island. The admission check would admit the taskset:

$$\begin{aligned} 7 \leq 7 = & 1 + \min \left\{ \left\lfloor \frac{(1+1) - (0.799 + 0.342)}{0.196} \right\rfloor + 2 \left\lfloor \frac{0.345328}{0.196} \right\rfloor , \right. \\ & \left. \left\lfloor \frac{(1 + 0.345328) - (0.799 + 0.342)}{0.196} \right\rfloor + \left\lfloor \frac{1}{0.196} \right\rfloor + \left\lfloor \frac{0.345328}{0.196} \right\rfloor \right\} . \end{aligned} \tag{20}$$

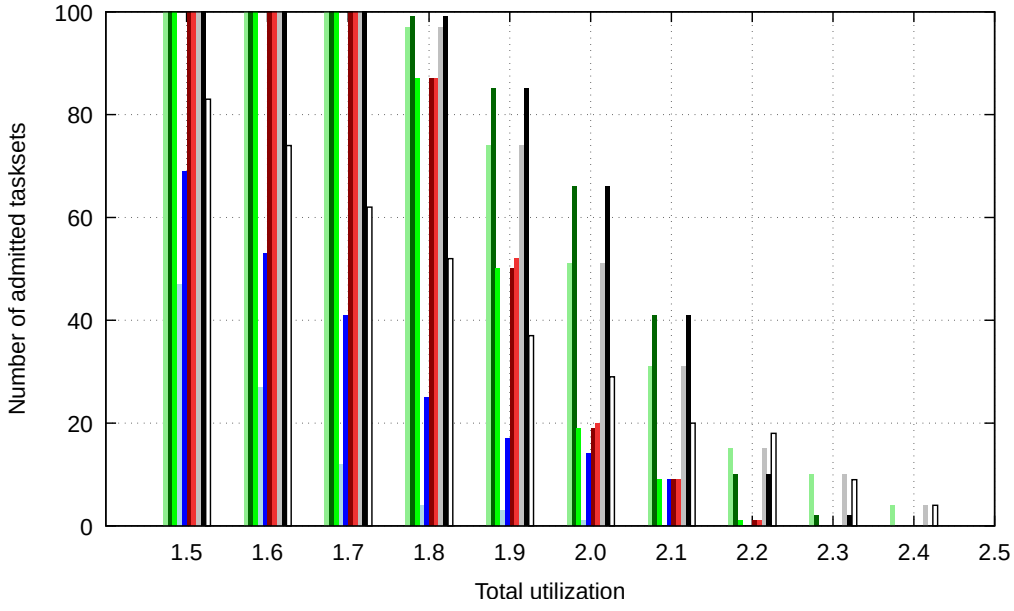
This example compares the basic blocks of the three admission tests, which result in the shown calculations. Table 2 summarizes the results of the tests.

5. Evaluation

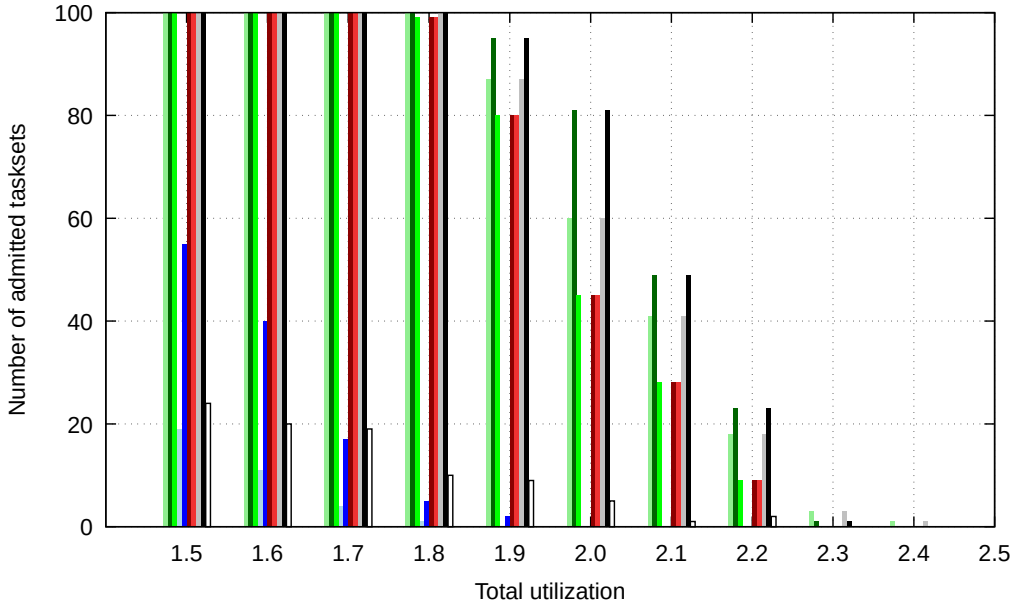
This section evaluates the combinatorial admission tests for the NUMP case given by Equation (15) for the values of $k = \{2, 3, 4, 5\}$, comparing the tests for the SMP case in Equation (1) and Equation (3). As it will become clear, the joint use of the SMP and the NUMP tests results in a viable, efficient and useful admission test for P-EDF that admits more tasksets than the SMP tests alone would do when applied separately to the two islands.



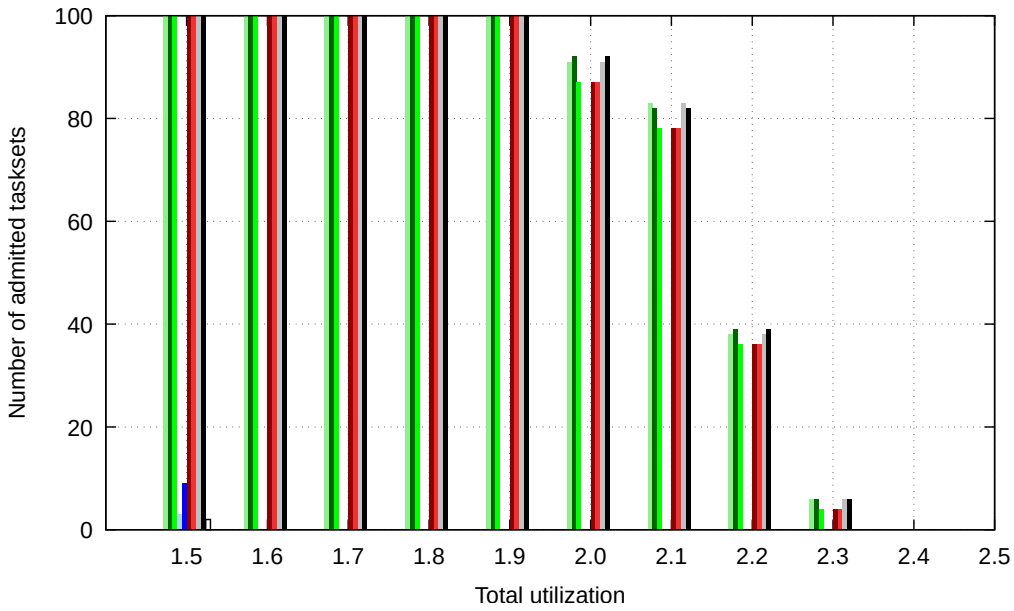
(a) $|\Gamma| = 6$



(b) $|\Gamma| = 8$

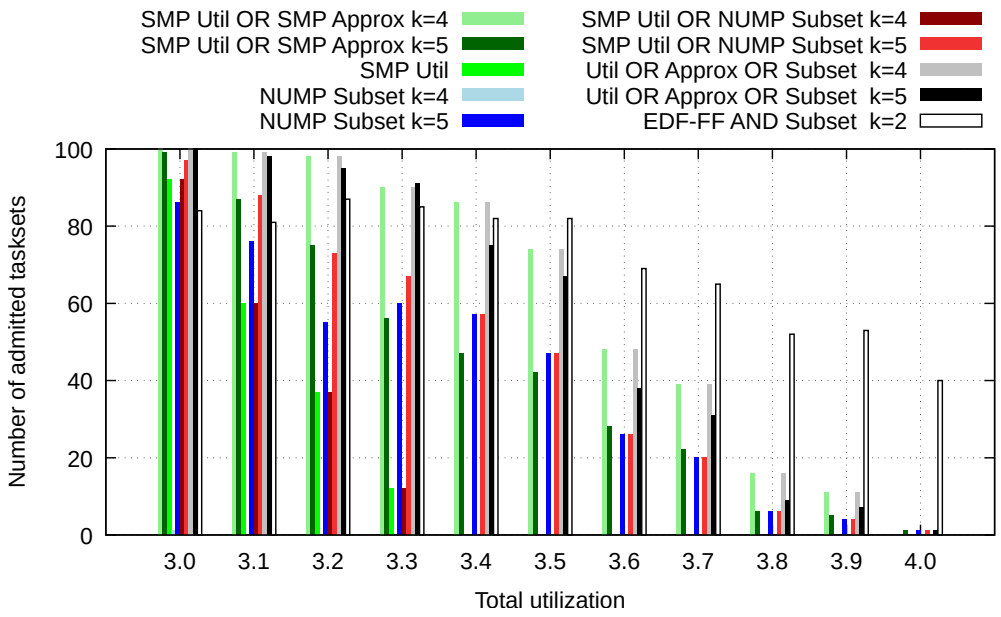


(c) $|\Gamma| = 11$

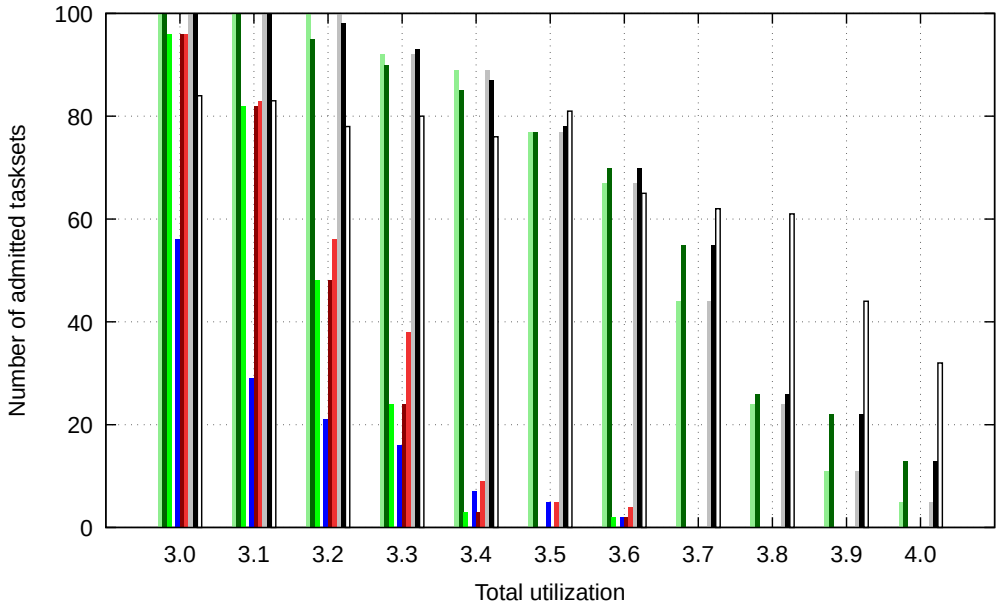


(d) $|\Gamma| = 18$

Figure 4: Comparison of equations behavior for $n = 4$ (i.e., $n_B = n_L = 2$) and varying the number of tasks in each taskset



(a) $|\Gamma| = 9$



(b) $|\Gamma| = 12$

Figure 5: Comparison of equations behavior for $n = 8$ (i.e., $n_B = n_L = 4$) and varying the number of tasks in each taskset

Tasksets have been generated with the `taskgen.py` program[4] by Emberson et al. [3] and the number of CPUs is fixed to $n = 4$ ($n_L = n_B = 2$) and $n = 8$ ($n_L = n_B = 4$). The admission tests have been repeated on a variety of tasksets with different cardinality and overall utilization. For each overall utilization among the values in $\{1.5, 1.6, ..., 2.4, 2.5\}$ for $n = 4$ cores, and in $\{3.0, 3.1, \ldots, 3.9, 4.0\}$ for $n = 8$ cores, 100 random tasksets have been generated and the admitted tasksets have been counted for each configuration. Each taskset contains a number of tasks varying in the range $\{6, 7, \ldots, 17, 18\}$ for $n = 4$ cores and in $\{9, 12, 15, 18, 21, 24\}$ for $n = 8$ and results are shown in Figure 4 and Figure 5 for the two cases of $n = 4$ and $n = 8$. Overall, 20800 tasksets have been tested for $n = 4$ and 6600 for $n = 8$.

5.1. Experimental evaluation of the admission tests

This section evaluates the mentioned admission tests for P- EDF with the settings described in Section 5. We compare the 3 admission tests introduced as Admission Test 1, 2 and 3 above, which are suitable for being used online and in a dynamic environment for small values of k. To take a reference NUMP platform, we concentrate our attention on an ARM big.LITTLE platform with $B_B = 1, B_L = 0.345328$, as the ODROID-XU3 one. Notice that the tests presented below work for any value of B_B and B_L, and that the parameters k given to Equation (3) for the tests on the big and LITTLE islands have been kept equal. Also, given a taskset $\Gamma = \{\tau_1 \ldots \tau_m\}$, Admission Test 1 and 2 must be given a split of the tasks to be dispatched on the big island, as discussed in Section 4.2, and we choose the splitting strategy in Equation (17). Notice that the way we split Γ is another choice that affects the effectiveness of the SMP tests. On the other hand, the NUMP test proposed in this paper in Equation (15) can be used transparently given a value of k.

We now discuss the effectiveness of the tests presented in Section 4.3. In Figure 4 and Figure 5, "*SMP Util*" refers to the logical AND between Equation (1) applied independently on the big and the LITTLE island on Γ_B and Γ_L respectively, "*SMP Util OR SMP Approx*" refers to the logical AND between *Equation* (1) *OR Equation* (3) applied to the big island and applied to the LITTLE island for various values of k, "*NUMP Subset*" refers to Equation (15) for various values of k, and "*SMP Util OR NUMP Subset*",

[4]The tool is available at: `http://retis.sssup.it/waters2010/tools.php`

"*Util OR Approx OR Subset*" and "*EDF-FF AND Subset*" refer to Admission Test 1, 2 and 3, respectively, for different values of k. In Equation (15) and Equation (3) we vary the value of k in the range $k = \{2, 3\}$ for the case $n = 4$ and $k = \{4, 5\}$ for the case $n = 8$, where at each iteration the value of k is the same for both equations. The value of the parameter k given to Equation (3) can be different for the big and the LITTLE island, which is yet another disadvantage of using the SMP tests on the NUMP architecture, since choosing appropriate values for each island may lead to better results, but tuning the value of k on the two islands may not be trivial. Notice that the tests are grouped by color and that "*SMP Util*", "*SMP Util OR SMP Approx*" are state-of-the-art approaches and are depicted with various types of green. The other bins are depicted with other colors to make the groups clearer to the reader (green, blue, red, black and white respectively).

Analyzing the graphs in Figure 4, the joined use of Equation (15) (i.e., the *NUMP check* in the light-blue and blue bins) and Equation (1) (i.e., the SMP check based on the taskset utlization) admits more tasksets than each of the two equations applied alone. For example, for $Total\ Utilization = 1.9$ in Figure 4a, the red bin (corresponding to *SMP Util OR NUMP Subset k=3*) is higher than both the green bin (corresponding to *SMP Util*) and the light-blue bin (corresponding to *NUMP Subset k=3*), which means that joining the two basic checks makes the final admission test in red more robust and effective. When the dimension of the tasksets grows, the effectiveness of the NUMP check decreases fast, and it is interesting to notice that the combination of the SMP tests (Equation (1) and Equation (3)) applied to the islands individually actually results in admitting many tasksets, as generally evident from the light-green and dark-green bins for all the taskset dimensions and total nominal utilizations.

Analogously, the combination of the NUMP and the SMP tests is valuable in that the number of admitted tasksets is greater than or equal to the one obtained by the checks taken individually. This is evident for example in Figure 4a for $Total\ Utilization = 2.0$ for *Util OR Approx OR Subset k=3* (black bin, corresponding to Admission Test 2), where some of the tasksets that are discarded by *SMP Util OR SMP Approx k=3* (dark-green bin) are then admitted by *NUMP Subset k=3* (light-blue bin) and viceversa (in fact, the black bin is higher than both the light-blue and the dark-green bins). Since Admission Test 2 (black bin) considers the logical OR between these two tests, it results in a more robust and effective admission test that increases the number of admitted tasksets into the system. The same holds

for *SMP Util OR NUMP Subset k=3* (red bins) between *SMP Util* (green bin) and *NUMP Subset k=3* (light-blue bin), for example for *Total Utilization = 2.0* in Figure 4a, as exposed above (notice that the red bin denotes an aggregation of different checks with respect to the one discussed in the current paragraph and depicted with black bins).

Nevertheless, in some cases joining two or more checks does not improve the number of admitted tasksets. Considering for example Figure 4a for *Total utilization = 2.2*, we notice that *SMP Util* (green bin) does not admit any taskset, while *SMP Util OR SMP Approx* admits some of them for both $k = 2$ (light-green bin) and $k = 3$ (dark-green bin), which is due to the fact that Equation (1) (green bin) does not admit any further tasksets than Equation (3) (light-green and dark-green bins) alone.

As expected, an increase in the value of k also increases the effectiveness of the proposed tests, at the cost of increased computational complexity. For example, compare the light-blue and blue bins (corresponding to Equation (15)) for *Total Utilization = 2.0* in Figure 4a and also compare the light-green and green bins for *Total Utilization = 1.9* (in Figure 4a). However, also notice that for some tasksets the result of applying Equation (3) (corresponding to the light-green and dark-green bins) with $k = 2$ may produce better results than with $k = 3$, since the formula results degrade, especially for the big island and under the taskset splitting strategy that we have chosen. This is depicted, for example, for *Total Utilization = 2.2* in Figure 4a, where the light-green bin is higher than the dark-green bin.

It is remarkable that *EDF-FF AND Subset k=2* proves to be effective in a number of cases, like Figure 4a and Figure 4b, depicting the behavior of the tests for $n = 4$ cores and $|\Gamma| = \{6, 8\}$ respectively, where it accepts a greater number of tasksets when the total utilization grows also with respect to the other tests with $k = 3$ (white bins vs dark-red, grey, red and black bins), for example with *Total Utilization = 2.4* of Figure 4a. In fact, the Admission Test 3 uses the whole core capacities by first performing an EDF-FF partitioning of the heavy tasks on the big island (under the assumption that these will not be many) and then applies Equation (15) on the lightweight tasks on both islands, as described in Section 5. However, in Figure 4c and Figure 4d, depicting the behavior of the tests for $n = 4$ cores and $|\Gamma| = \{11, 18\}$, *EDF-FF AND Subset k=2* bars are generally lower than all the other tests for $k = \{2, 3\}$ with the splitting strategy we use (white bins vs dark-red, grey, red and black bins), while it may be higher with other splitting strategies. Therefore, *EDF-FF AND Subset k=2* gives better results than the other tests

with higher total utilizations and with smaller tasksets.

Analyzing the graphs in Figure 5 for $n = 8$ cores (i.e., $n_L = n_B = 4$, as in the ODROID-XU3 platform), the behavior of the tests is similar to the one for $n = 4$ cores in Figure 4, where *EDF-FF AND Subset k=2* gives better results than the other tests. In fact, in Figure 5a and Figure 5b and expecially for total utilization greater than 3.5, the white bars are higher than the dark-red, red, grey and black bars. Also, the bars for *SMP Util* (green bin) decay much faster than with $n = 4$ (Figure 4), while the light-green and dark-green bins are taller, meaning that Equation (3) is quite relevant in this context. With respect to Figure 4, an interesting difference is that *EDF-FF AND Subset* (white bin) is generally slightly better when increasing the number of tasks in the tasksets (Figure 4a and Figure 4b vs Figure 5a and Figure 5b). Finally, the relevance of combining the NUMP tests and SMP tests is visible, for example, in Figure 5a for *Total Utilizations 3.5 and 3.7* (black bin vs dark-green bin) and in Figure 5b for *Total Utilizations 3.2 and 3.3* (black bin vs dark-green bin).

Generally speaking, combining the NUMP tests with the SMP ones produces better results since they compensate each other. Also, the choice of the values of k for the big and the LITTLE islands in Equation (3) and how Γ is partitioned are relevant choices in the SMP tests, while the NUMP test in Equation (15) can be used on the entire taskset Γ and for any NUMP platform.

Notice that the conclusions drawn from the comparison performed in this section refers to tasksets randomly generated with the *taskgen.py* program [3]. However, things might be different for alternative distributions of the task utilizations. A more realistic comparison should consider benchmarks or tasksets used in certain industrial domains. Our choice of using the tasksets generator by Emberson et al. has been dictated by its popularity and wide acceptance in the real-time research community.

Finally, we measure the average execution time of each of the presented admission tests (Admission Test 1, 2 and 3) over all tasksets and total utilizations for each number of cores in $n = \{4, 8\}$ and $k = \{2, 3, 4, 5\}$, when running on an Intel i7-8700 at 4.6 GHz and 16 GB RAM and the code is written in C++. The total number of experiments for $n = 4$ is 20800 for each value of k considered and for $n = 8$ it is 6600 for each value of k (the same tasksets of Section 5 have been considered). Results are in Table 3, which shows for each admission test the average execution times in microseconds for each the number of cores in the system n and for each value of k

considered. The admission tests can be generally performed in less than a millisecond for all values of k for $n = 4$ and for $k = \{2, 3, 4\}$ for $n = 8$. Also, the admission tests last significantly more time with $n = 8$ cores and the average execution time for Admission Test 3 is greater than both Admission Test 1 and Admission Test 2 and the one of Admission Test 2 is greater than Admission Test 1.

Table 3: Average execution time for each Admission Test (values in us).

	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$n = 4$				
Admission Test 1	1.01192	12.0065	132.772	-
Admission Test 2	1.01192	12.0065	132.772	-
Admission Test 3	1.96452	15.6746	486.16	-
$n = 8$				
Admission Test 1	1.57515	13.74	580.677	56786.8
Admission Test 2	1.57939	13.7405	580.679	56786.8
Admission Test 3	1.56697	17.7364	1840.9	357999

6. Conclusions and Future Work

This paper tackles the problem of admitting real-time tasks onto a non-uniform multi-processor platform, such as ARM big.LITTLE and DynamIQ, under partitioned EDF-FF scheduling. To do that, we extended [1] to NUMP architectures and proposed an improved equation that must check all the possible partitions of the first $k-1$ tasks on all the cores and a heuristic that checks only the first $k-1$ cores and $k-1$ tasks of the taskset. Then, we focused on the ARM big.LITTLE architecture and embedded the proposed equation within three admission tests which also exploit the SMP tests in [1] applied to the big and the LITTLE island independently. Simulations show that the proposed admission tests can be performed efficiently for small values of k and that the combined tests admit a good number of tasksets.

Concerning the future works on the topic, we plan to specialize the technique presented in this paper to the case of DynamIQ, where clusters contain a mix of big and LITTLE cores and cores may have different micro-architectures and frequencies. Moreover, it can be interesting to research a precise worst-case utilization bound for EDF-FF [2] on the ARM big.LITTLE architecture, and to integrate it with the equation based on the number of

tasks presented in this paper. Also it could be interesting to evaluate the admission tests for a setting with dynamic behaviour, where tasks are entering and leaving the system dynamically and the tasksets are not fixed. Finally, extending the formulas presented in this paper and in [2] to EDF-WF (Worst Fit) would be a relevant contribution to the state of the art of the admission tests, since adopting EDF-WF would more easily allow for the achievement of higher energy savings (than EDF-FF) on ARM big.LITTLE platforms.

References

[1] A. Mascitti, T. Cucinotta, L. Abeni, Heuristic partitioning of real-time tasks on multiprocessors, in: 2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC), IEEE, 2020, pp. 36–42.

[2] J. M. López, M. García, J. L. Diaz, D. F. Garcia, Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems, in: Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS 2000), IEEE, Stockholm, Sweden, 2000, pp. 25–33.

[3] P. Emberson, R. Stafford, R. I. Davis, Techniques for the synthesis of multiprocessor tasksets, in: Proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010), Brussels, Belgium, 2010, pp. 6–11.

[4] D. Simchi-Levi, New worst-case results for the bin-packing problem, Naval Research Logistics (NRL) 41 (1994) 579–585.

[5] J. Boyar, G. Dósa, L. Epstein, On the absolute approximation ratio for first fit and related results, Discrete Applied Mathematics 160 (2012) 1914 – 1923.

[6] D. S. Johnson, Approximation algorithms for combinatorial problems, Journal of Computer and System Sciences 9 (1974) 256 – 278.

[7] D. Johnson, A. Demers, J. Ullman, M. Garey, R. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, SIAM Journal on Computing 3 (1974) 299–325.

[8] R. L. Graham, Bounds on multiprocessing anomalies and related packing algorithms, in: Proceedings of the May 16-18, 1972, Spring Joint Computer Conference, AFIPS '72 (Spring), ACM, New York, NY, USA, 1972, pp. 205–217.

[9] M. R. Garey, D. S. Johnson, Approximation Algorithms for Bin Packing Problems: A Survey, Springer Vienna, Vienna, 1981, pp. 147–172.

[10] G. Gambosi, A. Postiglione, M. Talamo, Algorithms for the relaxed online bin-packing model, SIAM journal on computing 30 (2000) 1532–1551.

[11] Z. Ivković, E. L. Lloyd, A fundamental restriction on fully dynamic maintenance of bin packing, Information Processing Letters 59 (1996) 229–232.

[12] J. Balogh, J. Békési, G. Galambos, G. Reinelt, Lower bound for the online bin packing problem with restricted repacking, SIAM J. Comput. 38 (2008) 398–410. doi:10.1137/050647049.

[13] S. Berndt, K. Jansen, K.-M. Klein, Fully dynamic bin packing revisited, Mathematical Programming (2018) 1–47.

[14] J. Balogh, J. Békési, G. Galambos, G. Reinelt, On-line bin packing with restricted repacking, Journal of Combinatorial Optimization 27 (2014) 115–131.

[15] J. Balogh, J. Békési, G. Dósa, L. Epstein, A. Levin, A new and improved algorithm for online bin packing, arXiv preprint arXiv:1707.01728 (2017).

[16] S. Funk, S. Baruah, Task assignment on uniform heterogeneous multiprocessors, in: 17th Euromicro Conference on Real-Time Systems (ECRTS'05), IEEE, 2005, pp. 219–226.

[17] B. Andersson, E. Tovar, Competitive analysis of partitioned scheduling on uniform multiprocessors, in: 2007 IEEE International Parallel and Distributed Processing Symposium, IEEE, 2007, pp. 1–8.

[18] J. Singh, N. Auluck, Real time scheduling on heterogeneous multiprocessor systems — a survey, in: 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), 2016, pp. 73–78.

[19] G. Raravi, B. Andersson, V. Nélis, K. Bletsas, Task assignment algorithms for two-type heterogeneous multiprocessors, Real-Time Systems 50 (2014) 87–141.

[20] H. S. Chwa, J. Seo, J. Lee, I. Shin, Optimal real-time scheduling on two-type heterogeneous multicore platforms, in: 2015 IEEE Real-Time Systems Symposium, 2015, pp. 119–129.

[21] A. Bertout, J. Goossens, E. Grolleau, X. Poczekajlo, Workload assignment for global real-time scheduling on unrelated multicore platforms, in: Proceedings of the 28th International Conference on Real-Time Networks and Systems, RTNS 2020, Association for Computing Machinery, New York, NY, USA, 2020, p. 139–148. URL: https://doi.org/10.1145/3394810.3394823. doi:10.1145/3394810.3394823.

[22] S. K. Baruah, V. Bonifaci, R. Bruni, A. Marchetti-Spaccamela, Ilp-based approaches to partitioning recurrent workloads upon heterogeneous multiprocessors, in: 2016 28th Euromicro Conference on Real-Time Systems (ECRTS), IEEE, 2016, pp. 215–225.

[23] S. K. Baruah, V. Bonifaci, R. Bruni, A. Marchetti-Spaccamela, Ilp models for the allocation of recurrent workloads upon heterogeneous multiprocessors, Journal of Scheduling 22 (2019) 195–209.

[24] A. Wieder, B. B. Brandenburg, Efficient partitioning of sporadic real-time tasks with shared resources and spin locks, in: 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES), IEEE, Porto, Portugal, 2013, pp. 49–58.

[25] A. Mascitti, T. Cucinotta, M. Marinoni, An adaptive, utilization-based approach to schedule real-time tasks for arm big.little architectures, in: EWiLi, 2019.