Dynamic Partitioned Scheduling of Real-Time DAG Tasks on ARM big.LITTLE Architectures*

Agostino Mascitti
Tommaso Cucinotta
agostino.mascitti@santannapisa.it
tommaso.cucinotta@santannapisa.it
Scuola Superiore Sant'Anna
Pisa, Italy

ABSTRACT

This paper evaluates the combination of a Directed Acyclic Graph (DAG) task splitting technique already proposed in the literature and the state-of-the-art, energy-aware version of the well-known CBS server (BL-CBS), which dynamically partitions and schedules real-time task sets in an energy-efficient way on multi-core platforms based on the ARM big.LITTLE architecture. The approach is designed to be used with any DAG in a transparent way as an on-line and adaptive scheduler supporting "open" systems. The approach is validated and evaluated through the open-source RTSim simulator, which has been extended integrating an energy model of the ODROID-XU3 board and the code-base needed to perform the DAG task decomposition and scheduling. Simulations on randomly generated DAGs show that the approach leads to promising results.

CCS CONCEPTS

• **Computer systems organization → Real-time operating systems**; • **Hardware → Platform power issues**.

KEYWORDS

Real-time scheduling, ARM big.LITTLE, heterogeneous multicore processing, energy-efficiency, DAG scheduling

ACM Reference Format:
Agostino Mascitti and Tommaso Cucinotta. 2021. Dynamic Partitioned Scheduling of Real-Time DAG Tasks on ARM big.LITTLE Architectures* . In *29th International Conference on Real-Time Networks and Systems (RTNS'2021), April 7–9, 2021, NANTES, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3453417.3453442

1 INTRODUCTION

In recent years, embedded systems faced a relentless growth in the requirements posed by users on their computing capabilities, with a pervasive switch to multi-core platforms. Specifically, in the area of mobile and battery-operated embedded systems, we have witnessed a widespread adoption of novel energy-efficient architectures, first and foremost the ARM big.LITTLE one, nowadays a fundamental component of a plethora of smartphone and tablet devices on the market, where the timeliness of soft real-time applications in domains like multimedia and gaming is becoming more and more important.

The big.LITTLE design deviates from classical symmetric multiprocessing (SMP), in that it introduces two different types of cores sharing the same instruction-set architecture (ISA), but with different frequency vs power consumption curves. In these platforms, the operating system can migrate tasks among different core types at run-time, as needed according to its power-aware scheduling and placement strategies. *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 [4], will remove this constraint.

However, much of the work on energy saving has been carried out until now on independent and sequential real-time tasks, without considering intra-task parallelism (refer to [7] for a survey), while in this paper we focus on that. These complex tasks can be deterministically represented as Directed Acyclic Graphs (DAGs) of tasks at the operating system (OS) level. An efficient scheduling of these *DAG tasks* is imperative, for example, in case of low-latency audio and multimedia processing, where soft real-time scheduling techniques can be used to ensure the expected Quality of Service (QoS) for the application. For example, in audio/video processing, filters and effects to apply can be represented as nodes in a DAG characterized by their WCETs, and precedence constraints whose presence adds complexity to the scheduling problem. Indeed, the deadline miss of a node delays its successors, which can be potentially perceived by the user and degrade the final experience with annoying audio hiccups or undesirable audio/video delays.

[0]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.

RTNS'2021, April 7–9, 2021, NANTES, France

ACM ISBN 978-1-4503-9001-9/21/04…$15.00
https://doi.org/10.1145/3453417.3453442

When dealing with real-time workloads, the Linux default scheduler, the completely fair scheduler (CFS), is not effective in coordinating the energy-management features of the hardware with the timing requirements of the deployed real-time applications, leading to experiencing unnecessary deadline misses. More appropriate solutions are needed like the SCHED_DEADLINE scheduler, which has been recently made available in the mainline Linux kernel, and that is a multi-processor variant of the CBS server [1], exploiting a reservation-based scheduling strategy that can be conveniently configured as using either global, partitioned or clustered EDF scheduling underneath. However, it cannot be used directly for scheduling real-time DAGs in an energy efficiency way on ARM big.LITTLE architectures, which is the subject of this paper investigations.

1.1 Paper Contributions

In this paper, we deal with the problem of correct use and configuration of a reservation-based scheduler like SCHED_DEADLINE, for scheduling real-time DAG tasks (i.e., sets of OS tasks with precedence constraints linking them into DAG topologies) on ARM big.LITTLE architectures in an energy-efficient way. To this purpose, we leverage on our previous work [33] on energy-aware scheduling for independent real-time tasks on big.LITTLE architectures, adding the support for soft real-time DAG tasks. The focus is on an "open" environment, where real-time DAGs can enter and leave the system at any time, due to activation or termination of applications at run-time. The proposed approach is based on combining the mentioned prior work of ours with task decomposition techniques proposed in [24, 25] for transparently grouping the nodes of each DAG into CBS servers, and partitioning them among the available CPU cores. The proposed approach is extensively evaluated by simulation under various conditions, where the OS tasks belonging to each DAG are dynamically assigned to the available cores, trying to minimize the power consumption of the platform. The simulations focus on synthetic, albeit realistic scenarios for mobile personal computing devices (e.g., smartphones, tablets) in which DAG tasks are dynamically activated (or terminated) due to user interactions. Simulations have been performed extending the well-known RTSim simulator platform [38] with a realistic big.LITTLE energy model [33] reflecting tightly the capabilities of a real octa-core big.LITTLE platform, and the possibility to host arbitrary real-time DAGs, integrating the DAG decomposition and acceptance tests mentioned above.

From the presented evaluation, our approach: (1) guarantees that the DAG tasks admitted into the system are scheduled within their deadlines; (2) achieves the lowest expected power consumption for executing the DAG tasks.

1.2 Paper Organization

This paper is organized as follows: after a brief review of the related research in Section 2, the computing platform we focus on and its energy model are described in Section 3, along with some accompanying notation that is adopted throughout the rest of the paper. Then, key background concepts related to the adopted task placement strategy and CBS scheduling are presented in Section 4. The proposed technique is described in Section 5, along with an example of its usage in Section 5.1. Then, the proposed BL-CBS technique is evaluated by simulation in Section 6. Finally, conclusions are drawn in Section 7, sketching out possible directions for future research on the topic.

2 RELATED WORK

The problem of energy-aware scheduling of *sequential* real-time tasks on multicore architectures has been widely studied over the years and a survey is available in [7]. Some of the authors [15, 17] use linear programming-based approaches to find the optimal solution, while others [5, 11, 26, 32, 34, 36, 37, 40, 44] exploit different techniques and the hardware DVFS capability to reduce the energy consumption, like we do with DVFS. Notably, [31] also considers GPUs in the platform model and this brings to a more complete solution for saving energy. A previous work of ours [33] also explores the problem of scheduling sequential real-time tasks on ARM big.LITTLE platforms in an energy-aware way, proposing an innovative dynamic partitioning algorithm that works for "open" systems, where tasks can come and go at any time. This last work is used as a base upon which the approach proposed in this paper builds, extending it to the case of tasks with dependencies (DAGs). However, the problem becomes more challenging in the case of DAG tasks because the single DAGs can execute partially on different cores at the same time and nodes have dependencies among each other, and none of the mentioned works on energy-aware real-time scheduling deals with DAGs.

A variant of this problem is the one of thermal-aware scheduling of *sequential* real-time tasks on multicore systems. While some approaches aim at minimizing the peak temperature [14, 20, 21, 48], and some of them exploit ILP methods and other ones the core speeds, other apporaches employ feedback loops to manage adaptively the platform temperature [16, 22, 23]. Finally, other solutions [47] are based on a pattern-based approach that divides a given time horizon into several time segments, and the processor can either be making computation or in a dormant mode that reduces the temperature.

Nevertheless, the problem of scheduling real-time DAG tasks on multicore platforms has been tackled in other works without considering the energy issue. The problem is tackled from two points of view and while [3, 8–10, 12, 28] consider the analysis and schedulability problem, other works [2, 13, 27, 29, 35, 45, 46, 49, 50] propose partitioning techniques based on various scheduling strategies like federated and semi-federated scheduling and G-EDF. One interesting variant of the problem uses the Gang scheduling [2, 39], in which all parallel instances of a same task start and stop using the processors at the exact same time. Also, specific works [19] dealing with the particular case of *linear* topologies, have appeared in the literature.

As for the problem of energy-aware scheduling of real-time DAG tasks, it has not received much attention until now, and we report the major works available in the literature on the topic.

The first work is from Guo et al. [24], and it is based on (and extends) some previous works of theirs [25, 42], which did not consider ARM big.LITTLE platforms. Our research is based on [24], which is deepened in Section 4, and exploits the algorithm to decompose the DAG tasks into groups of nodes (Task Decomposition) that can be dispatched onto the cores and executed with EDF. These

works are based on the concept of *Speed Profile*, which, given a DAG task, provides the time percentage/likelihood needed by the DAG for each speed available on the platform. This is used to identify an energy-efficient core-to-task mapping.

In contrast, we propose to use a scheduling strategy combining an admission test based on the utilization of the tasks, designed specifically for the ARM big.LITTLE architecture, developed in our prior research [33], with the DAG decomposition technique proposed in [24].

The same authors also propose [41] an energy-aware federated scheduling approach based on ILP to partition DAG tasks on the ARM big.LITTLE architectures and determine the speeds of the nodes. Then, when the nodes run on their cores, frequencies are set based on the speeds of the nodes. This approach requires some offline computations, while ours totally works online for "open" systems and without any prior knolwedge of the DAGs that will need to be admitted onto the system at run-time.

3 NOTATION AND ENERGY MODEL

This section presents the models and notation used throughout the paper to represent the underlying processing platform, the energy model, and the scheduled task sets.

3.1 Platform Model

The system under consideration has an ARM big.LITTLE architecture with two homogeneous islands where each island s has a total of m_s cores. DVFS capabilities are available with all cores in the same island sharing the same CPU frequency. At a higher level of abstraction, a core is characterized by a power model and a speed, and $\bar{x}_s$ is the maximum speed of the island s. The *power model* is given by [6] and it tries to be a good compromise between representativeness and complexity:

$$P_{CPU} = P_{sw} + P_{lk} = \delta + (1+\eta)(1+\gamma)KfV^2, \tag{1}$$

where P_{sw} is the power required to charge the transistors and P_{lk} is the power due to leakage effects. Also, K envelopes the capacitance of the transistor gates and the number of transistors involved in the frequency switching, γ is a temperature-related parameter, η is the proportionality factor between the power consumption due to charging the gates and the power loss due to brief short-circuit conditions while toggling logic states, and δ is used to introduce some degree of freedom. The formula states that the power consumption depends on voltage and frequency and is in quadratic proportion to the voltage. These parameters have been tuned via genetic optimization on real data gathered from a ODROID XU3, as detailed in [6]. Equation (1) does not consider interferences due to other tasks in the system nor the implied cache and bus contention. Including other details would make the CPU model more accurate, but it would also increase the complexity. In line with observations in [6], this approach results in a sufficient approximation for our simulations. We define the *speed of a core* (either big or LITTLE) as a number between 0 and 1 representing its computational capacity, relative to a *big* core at maximum frequency. On the ODROID XU3, each big core has maximum speed $\bar{x}_B = 1$ and each LITTLE core has maximum speed $\bar{x}_L = 0.345328$, which are the maximum per-core nominal utilizations admissible under EDF (see Equation (4)).

3.2 Task Model

In this paper we consider a set of real-time DAG tasks $\Gamma = \{\tau_1 \dots \tau_n\}$ to be scheduled on a number of CPUs. Each $\tau_i \in \Gamma$ is represented as a DAG and it is characterized by a minimum inter-arrival period T_i equal to its relative deadline D_i (*implicit deadline* case). Each DAG task τ_i is made up of n_i nodes, each denoted by N_i^j with $1 \le j \le n_i$ ($\tau_i = \{N_i^j\}_{j=1}^{n_i}$) and characterized by the worst-case execution time (WCET), which will be discussed in depth in the following sections for the case of the ARM big.LITTLE architecture. An edge from N_i^j to N_i^k ($N_i^j \rightarrow N_i^k$) means that the execution of N_i^k can only begin when N_i^j finishes for every instance (precedence constraint). In this case, N_i^j is called *parent* of N_i^k and N_i^k is called *child* of N_i^j. A node N_i^j may have multiple parents or children. The set of parent nodes $\{N_i^j\}$ of a node N_i^k is denoted as $pred(N_i^k)$.

Each DAG task τ_i generates a sequence of activations, where each activation τ_i^j arrives at time r_i^j (causing the j^{th} activation of all nodes N_i^j with $pred(N_i^j) = \emptyset$) and has a finishing time (equal to the maximum among the finishing times of the j^{th} activation of all nodes in the graph) denoted by $f_i^j > r_i^j$. A DAG task τ_i respects all of its deadlines if all of its activations respect the relative deadline constraint $D_i \equiv T_i$: $\forall j, f_i^j \le r_i^j + T_i$. Generally, $r_i^{j+1} \ge r_i^j + T_i$, but for a *periodic* real-time task, we have $r_i^{j+1} = r_i^j + T_i$.

A *critical path* is a directed acyclic path with the maximum total execution among all the other paths in a DAG. L_i is the sum of the execution times of all the nodes that belong to the critical path. It is a min-span of τ_i, i.e. in order to make it schedulable, at least L_i time units are required in every period T_i even when the number of cores is unlimited. Therefore, the condition $T_i \ge L_i$ must hold for τ_i to be schedulable. A schedule is said to be feasible if upon satisfying the precedence constraints, all the sub-tasks (nodes) receive enough execution time on the CPUs, so that every task τ_i completes within its deadline $D_i \equiv T_i$ for each activation.

Each node N_i^j of the DAG task τ_i has a known *nominal WCET* C_i^j being the WCET of the node when running on a big core at maximum frequency. Whenever the task is running on a core of an island s at an Operating Performance Point (OPP) o, corresponding to the frequency $f_{s,o}$, its timing is characterized by the *scaled WCET* $\tilde{C}_{i,s,o}^j$ defined as:

$$\tilde{C}_{i,s,o}^j = \frac{C_i^j}{x_{s,o}} \left[= \frac{C_i^j}{\bar{x}_s} \frac{f_{s,k_s}}{f_{s,o}} \right], \tag{2}$$

where k_s denotes the maximum-frequency OPP available on island s, and $x_{s,o}$ denotes the speed of a core of island s when at OPP o, $\bar{x}_s$ denotes the speed of any core of island s when running at its maximum-frequency OPP and f_{s,k_s} denotes the maximum frequency of island s.

This paper focuses on approaches where the initial task set of DAG tasks Γ is turned into a set of CBS servers Γ'. Each server $\sigma_{i,k}$ of the DAG task τ_i groups a set of nodes $\sigma_{i,k} = \{N_i^j\} \subseteq \tau_i$. This, when running on island s at OPP o, has a scaled budget equal to the sum of the scaled WCETs of its associated DAG nodes $\tilde{Q}_{i,k} = \sum_{N_i^j \in \sigma_{i,k}} \tilde{C}_{i,s,o}^j$. At any point in time, each core h of an island s hosts a subset of CBS servers $\Gamma_{s,h} \subseteq \Gamma'$ that, in the case of

a partitioned EDF-based scheduler, need to meet the well-known condition (for schedulability of the CBS reservations – see Equation (5)):

$$\sum_{\sigma_{i,k} \in \Gamma_{s,h}} \frac{\tilde{Q}_{i,k}}{P_{i,k}} \leq 1 \; \forall h \in \{1..m_s\} . \tag{3}$$

With reference to a core h of island s at OPP o where the reservation is deployed, we can introduce the *nominal budget* $Q_{i,k} = \tilde{Q}_{i,k}/x_{s,o}$ and the *nominal utilization* $B_{i,k} \equiv Q_{i,k}/P_{i,k}$, obtaining:

$$\sum_{\sigma_{i,k} \in \Gamma_{s,h}} B_{i,k} \leq \bar{x}_s \; \forall h \in \{1..m_s\} . \tag{4}$$

However, we assume to have no prior knowledge as to what real-time tasks will appear into the scheduler queue over time; thus we focus on an adaptive approach that can be used for "open" systems (details in Section 5).

4 BACKGROUND

In this paper, we consider a system where many soft real-time tasks may be concurrently active on the same CPU and they interfere with each other, jeopardizing their deadlines. Therefore, we make use of *resource reservations* to enforce *temporal isolation* among tasks. In this paper, we use the Constant Bandwidth Server (CBS) [1] as a resource-reservation mechanism. So, node groups of each DAG task are scheduled by means of a CBS reservation $\sigma_{i,k}$ with associated scheduling parameters (Q, P), with the meaning that Q time units (a.k.a., *budget*) are reserved for scheduling the corresponding task nodes on the CPU in every time interval of duration P (a.k.a., *reservation period*).

In CBS, reservations are realized by means of the Earliest Deadline First (EDF) scheduler, which schedules reservations based on their *scheduling deadlines*, assigned by the CBS algorithm. When a reservation is activated, the server checks whether the current deadline is sufficient to schedule it, otherwise it assigns a new deadline postponing the current one of a time equal to P. While a reservation is scheduled, the budget of the associated CBS is accordingly decreased. If tasks in the reservation try to execute for more than Q time units, the reservation scheduling deadline is postponed by P. Therefore, each reservation is prevented from executing for more than Q time units with the same scheduling deadline, and it is guaranteed a computation bandwidth of Q/P regardless of the behaviour of other reservations. This ensures temporal isolation, preventing a misbehaving task to cause deadline misses on jobs of other tasks with farther away deadline. To guarantee schedulability of, and temporal isolation among, the reservations $\{(Q_i, P_i)\}$ on the same core, the following *schedulability condition* must hold:

$$\sum_i \frac{Q_i}{P_i} \leq U_{max} \tag{5}$$

with $U_{max} = 1$ in case of EDF. A CBS server may be associated with many tasks, which belong to the same DAG task and that are related by a dependency relationship in this paper.

An energy-aware extension of the CBS server is GRUB-PA (*Greed Reclamation of Unused Bandwidth - Power Aware*), integrating the ability to reclaim unused processor capacity (*bandwidth*) that is not used because some of the servers may have no jobs awaiting execution and exploiting the hardware DVFS capabilities to reduce the cores frequency. GRUB-PA has been implemented in the mainline Linux running SCHED_DEADLINE CBS reservations, starting from version 3.14 released in September 2017. Its energy-related behaviour on multiprocessor platforms could be summarized as follows. When a new task instance $J_{i,j}$ arrives, the first free core is selected if available; otherwise, the core with the latest-deadline task is chosen and the new task is dispatched onto it. Then, in the case of ARM big.LITTLE, the highest-utilization core of each island is used to determine the frequency (or the highest frequency is picked if the busiest core has utilization greater than 1.0). When a task in a server ends, leaving its core idle, the task with closest deadline is pulled onto it and the islands frequencies are adjusted. The reader can find more details about the GRUB-PA in [43].

In this paper, we introduce the support to DAG tasks, whose nodes are split into groups, enveloped into CBS servers and dispatched onto cores. Task Decomposition is a technique to group nodes and, in this paper, we use a slightly modified and optimised version of the original one [24, 25], where for example we introduce the *Relaxing Finishing Times* phase to further reduce the utilizations of the CBS servers, which is important to reduce the energy consumption. Also, we adapt and refine the original technique, thought to deal with the *Speed Profiles* (see above), for being used in conjunction with partitioned EDF and with the partitioning strategy proposed in [33]. The technique is made up of different phases that aim at assigning each node an arrival and finishing time with respect to the DAG arrival time and deadline. Since the technique is a central part of this paper, it is extensively explained in Section 5 and Section 5.1 with images.

4.1 Real-time task partitioning

The Task Decomposition technique is coupled with our technique [33] for partitioning OS tasks dynamically, so we report in this section this strategy for the sake of completeness. The placement algorithm is called to decide a target core and its frequency. It loops over all cores and their frequencies, starting with the same island where the task is already located, keeping its current frequency if possible. The first check is that jobs do not miss their deadlines and that they are schedulable with EDF. Then, it computes the difference between the power consumption due to the current core assignment, with the one of an alternate assignment to a different core, considering the possible impact on frequency scaling. Power consumption is calculated based on the island and utilization of the new task. When the loop is over, the core with minimum additional power consumption is chosen. Note that the power consumption of a core for a given frequency is actually the difference with respect to its consumption on idle. Our modified RTSim is actually able to exploit the full optimized CPU power model as available from [6].

One more check and policy is applied to refine the choice: if the core that would give minimum additional consumption is already busy, the algorithm tries to balance the cores load by opting for the next free one with the same extra consumption, if it exists. This way, we avoid unneeded migrations.

When a job in a core ends its $WCET_{scaled}$, that core can get either idle or there might be some ready tasks, which would be scheduled. If there is no ready task, then the core would go idle. In

this case, to maximize its utilization and because it may be impossible to scale down its frequency to the minimum for saving energy (cores in an island share the same frequency in ARM big.LITTLE), migrations are performed. A ready task from the big island is picked and migrated. However, it may not be schedulable in the LITTLE island with EDF. Therefore, migration is confirmed if:

$$WCET_{f''} \leq (1 - U_{\rho_{final}, f''})(D_{abs} - t), \quad (6)$$

where we want to move a task with absolute deadline D_{abs} from its core with frequency f' to core ρ_{final} with frequency f'' at time t. If no task can be moved and all queues in the core island are empty, its frequency is set to the minimum. The CBS server active utilization is stored on its ending CPU when it finishes and it is kept while in releasing state. It is considered when computing the core total utilization and it is removed when the virtual time [30] expires or the core goes idle.

Note that the Placement Algorithm depends on the instantaneous state of the system and that the scheduling decisions are dynamic (i.e., performed at each task wake up and termination event). It works with any taskset and, for each task, it only needs its WCET and period. Moreover, it makes decisions based on 3 factors: (i) task deadlines are respected; (ii) reduce power consumption when possible; (iii) accept as many periodic tasks as possible. However, since the focus has been on the functionality, the placement logic has not been optimized for efficient execution. This is left as future work, necessary for a viable solution to be embedded in a real OS kernel scheduler.

Besides, we clarify that using this scheduling algorithm, we determine the core allocation that minimizes power consumption using a greedy heruristics. This works by comparing the difference in power consumption due to placing the new task on the variuos cores (and at the minimum associated frequencies to keep the reservations schedulable). This way we can determine that a specific core provides the heuristically best choice that gives the minimum (additional) power consumption.

5 PROPOSED APPROACH

The proposed approach is the combination of the tecniques proposed in [24], to split the DAG tasks in input into groups of nodes, and the partitioning technique as proposed in [33], to dispatch the parallel tasks (nodes) onto their cores.

Real-time DAG tasks are transparently decomposed into a set of CBS servers as illustrated with an example in Section 5.1 (Task Decomposition phase), and then they are partitioned among the available cores as described in Section 4.1, where they are scheduled on the assigned core using the CBS scheduling policy, based on EDF. Notice that the partitioning technique is applied at job level when the tasks wake up to maximize the saved energy. The schedulability of the admitted tasks is guaranteed on each core using the EDF admission test as in [24], and the schedulability of the admitted DAG tasks is assured by the fact that, at each DAG task wake up (i.e., at each period), all of its servers are dispatched onto their cores, so that the DAG task is scheduled "all or nothing". In fact, for example in the case of audio processing, where a number of filters are applied to the original audio (the reader can think about it as a DAG task), the output is correct only if all the filters are applied in the required order and within their deadlines, which motivates our DAG admission choice. Notice that, even when a DAG instance is not schedulable, it still activates at its next period and we try to schedule it.

5.1 Motivational example

To clarify the approach, we present an example of Task Decomposition, which is the first step applied when a DAG task enters the system. The original technique is taken from [24], and it is adapted and refined for being used in conjunction with partitioned EDF and with the partitioning strategy described in Section 4.1 and to further decrease the utilization of the resulting CBS servers.

Consider a DAG task τ_i representing a possible task with deadline $D_i = T_i = 50$ that we want to schedule in Table 1, which shows the nodes and their nominal WCET (i.e., referred to the big island at maximum speed). For semplicity, the node Nj corresponds to the parallel task N_i^j of τ_i.

Table 1: Nodes of DAG task for the motivational example.

Node	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11
WCET	10	5	1	6	3	6	5	3	4	10	2

The Task Decomposition algorithm decomposes the DAG task τ_i into multiple Sequential Tasks and computes arrival times and absolute finishing times for all nodes in a way that the Sequential Task utilization is reduced whenever possible to fit inside the LITTLE cores and to decrease the island frequency. Then, each Sequential Task with its nodes corresponds to a CBS server, and they are dispatched onto cores based on their utilization. The technique is made up of different phases, which optimize the result of the Base Step.

Task decomposition - Base Step. The first step of the Task Decomposition algorithm is the same as in [24] and finds a number of Sequential Tasks containing nodes. Figure 1 (Top) is a graphical representation of its output, where different colors represent different Sequential Tasks. For example, $\{N1, N2\}$ and $\{N5, N6, N8\}$ are two Sequential Tasks.

In addition to assigning nodes to Sequential Tasks, the basic step also finds for each node N_i^j its its arrival time $a_i^j = \sum_{N_i^k \in pred(N_i^j)} C_i^k$ and its absolute finishing time $f_i^j = a_i^j + C_i^j$ for the first DAG instance released at time t=0. Using these values, the output of the basic step can be represented as in Figure 1 (Bottom).

Slack allocation. The basic step leaves some slack time $\sigma = D_i - L_i$, which can be used to relax the nodes deadlines and thus their utilizations. σ is allocated uniformly among the nodes by multipling the deadline of each node by the factor $\delta \equiv \frac{T_i}{L_i} = \frac{50}{39} = 1.28$. The result is shown in Figure 2, where for example:

$$f_i^{N11} = \lceil 39 \times 1.28 \rceil = 50$$
$$f_i^{N10} = \lceil 37 \times 1.28 \rceil = 48$$
$$f_i^{N9} = \lceil 31 \times 1.28 \rceil = 40$$

Segment extension. The absolute final time computed at the previous step is too strict for the nodes without successors and thus it can be further extended. The main refinement of this phase is the reduction of the utilization of some Sequential Tasks. The

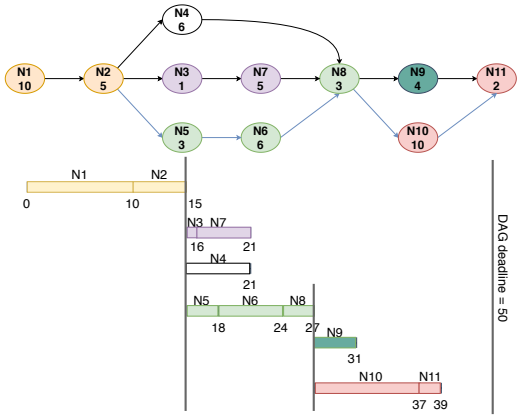


Figure 1: Result of the "Base Step" of the Task Decomposition. Top: The input DAG is divided into Sequential Tasks, represented by different colours. Bottom: Arrival and finishing time assigned to each of the produced Sequential Task, represented by group of nodes in different rows.

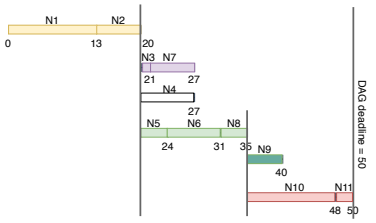


Figure 2: Result of the "Slack Allocation" phase. All arrival and finishing times are updated.

result is shown in Figure 3, where the nodes that can be extended are highlighted ($N7, N4, N9$), like N7 that has no reason to finish within $f_i^{N7} = 27$ and can be extended to $f_i^{N7} = a_i^{N8} = 35$. Notice that this way the utilization of the Sequential Task is reduced and this is a key point for the placement algorithm when exploiting the LITTLE cores.

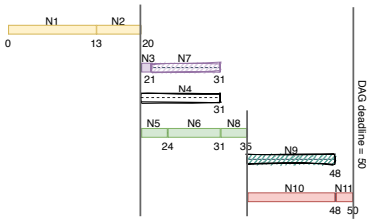


Figure 3: Result of the "Segment Extension" phase.

Relaxing finishing times. The finishing times of the nodes can be further reduced in this step by distributing the time window of each Sequential Task to its nodes proportionally to the node WCET. The result is shown in Figure 4 and the modified deadlines are highlighted ($N3, N7, N6, N8$). The nodes of each Sequential Task $\mathcal{S}$ with nodes $\{N_{\mathcal{S}}^1 \dots N_{\mathcal{S}}^n\}$ are multiplied by a factor $\mu \equiv \frac{\Delta}{\sum_{N_{\mathcal{S}}^j} C_{\mathcal{S}}^j}$, where $\Delta \equiv f_{\mathcal{S}}^{N_{\mathcal{S}}^n} - a_{\mathcal{S}}^{N_{\mathcal{S}}^1}$. For instance, in the Sequential Task with nodes $\{N3, N7\}$, $\mu \equiv \frac{11}{6} = 1.83$:

$$f_i^{N3} = \lceil a_i^{N3} + C_i^{N3} \times \mu \rceil = \lceil 20 + 1 \times 1.83 \rceil = 22$$

$$f_i^{N7} = \lceil 22 + 5 \times 1.83 \rceil = 32$$

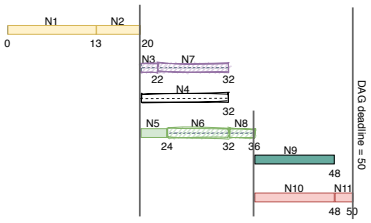


Figure 4: Result of the "Relaxing finishing times" phase.

The produced Sequential Tasks and their nodes are then turned into CBS servers, which are dynamically partitioned based on their utilization using our placement strategy to achieve the minimum expected energy consumption of the platform and guarantee the deadlines of the nodes and their DAG tasks. Notice that our placement algorithm is based on the task utilization, while in contrast the one of [24] is based on the *Speed Profile* of the nodes.

6 SIMULATION RESULTS

In what follows, we introduce the schedulers taken into account over all the experiments in Section 6.1, we introduce the paramters of the experiments for their reproducibility in Section 6.2 and in Section 6.3 we present our results about the energy efficiency of our approach compared to other partitioning strategies.

6.1 Compared Schedulers

We validated the BL-CBS algorithm described in Section 5 and measured its performance and energy consumption through the class *EnergyMRTKernelDAG* in RTSim [1] with respect to: (i) G-EDF, as available through the class *MRTKernel_Linux5_3_11_DAG* in RTSim, simulating the behaviour of the mainline Linux kernel a few years back, running SCHED_DEADLINE CBS reservations without GRUB nor power awareness features; (ii) and *MRTKernel_Linux5_3_11_GRUB_PA_DAG* in RTSim, implementing the energy-related behaviour of GRUB-PA, i.e. decreasing the frequency to the minimum required to sustain the utilization of every CPU, and reflecting (part of) the behaviour of the implementation of GRUB-PA in the mainline Linux running SCHED_DEADLINE CBS reservations. In fact, a complete implementation of GRUB-PA would add the bandwidth reclaiming mechanism, which would not benefit our simulated scenarios since task overruns are not considered.

[1] The source code of RTSim with BL-CBS is available at: https://gitlab.retis.santannapisa.it/a.mascitti/dag_sched_energy_c2021

6.2 Comparison Results

The hardware energy consumption model is the one of the ODROID-XU3 board, which uses the Samsung Exynos 5422 SoC. This is an ARM big.LITTLE architecture with four Cortex-A15 and four Cortex-A7 cores. The model has been taken from [6], where it has been implemented in RTSim [38].

We performed a number of experiments with DAG nominal utilization $U_i \equiv \sum_{N_i^j \in \tau_i} U_i^j$ (for the DAG task τ_i) in the range [0.25;0.7] with spacing 0.05, for the reservations. For each experiment, we generated a number of random DAGs with the tool Ggen[2] by Cordeiro et al. [18] for each value in the set {1,2,3} with {24,12,8} nodes for each DAG respectively, corresponding to a total of 24 nodes per experiment, and each node has a probability to get edges of 0.25. Each experiment has been repeated with BL-CBS, GRUB-PA and G-EDF using the same task set of DAGs. To represent a more realistic and dynamic environment (i) for each server containing n_{CBS} nodes, the (nominal) WCET of each of its nodes is sized down so as to be uniformly distributed between $\frac{0.6}{n_{CBS}}$ and $\frac{0.9}{n_{CBS}}$ times the budget of their CBS server; (ii) the Ggen tool is used to generate random budgets and periods in the range [1;100] ms with a relatively rough granularity of 0.5 ms, in order to keep under reasonable limits the resulting hyperperiods; and (iii) to represent an even more dynamic environment, the execution time of the node *instances* is set so to be uniformly distributed between 0.1 ms and the node nominal WCET. Moreoever, each node N_i^j is parametrized as in Section 3 and assigned to a CBS server $\sigma_{i,k}$ with parameters:

$$\begin{cases} Q_{i,k} = \sum_{N_i^j \in \sigma_{i,k}} C_i^j \\ P_{i,k} = \sum_{N_i^j \in \sigma_{i,k}} (f_i^j - a_i^j) \end{cases} \quad (7)$$

Since all the servers of a DAG are dispatched onto cores at the beginning of the DAG period for the reasons explained in Section 5, cores have been assigned an overprovisioning factor of 0.8, so that the big cores maximum speed is 1.8 and the LITTLE cores maximum speed is about 1.15.

6.3 Energy Saving Results

In this section, the obtained energy saving achieved with our approach (BL-CBS) when compared with GRUB-PA and G-EDF is discussed. DAGs are inserted in succession in the system and nodes assigned to their own CBS servers through the Task Decomposition as in Section 5.1, which is performed only once per DAG task. Then, at each DAG period, the servers and their nodes are dispatched onto cores that give the minimum increase of energy consumption to the system.

Figure 6 depicts the average energy consumption for (i) BL-CBS, (ii) GRUB-PA and (iii) G-EDF for different DAG nominal utilizations (different curves) and different number of DAG tasks (different graphs). The vertical bars associated to each point represent the standard deviation. BL-CBS has lower energy consumption than both GRUB-PA and G-EDF for all the utilizations, and this is because BL-CBS always chooses the core with the minimum power increase and tries to not raise the island frequencies when possible, while GRUB-PA chooses the core with latest deadline and,

[2]The source code of the Ggen tool is availbale at: https://github.com/perarnau/ggen

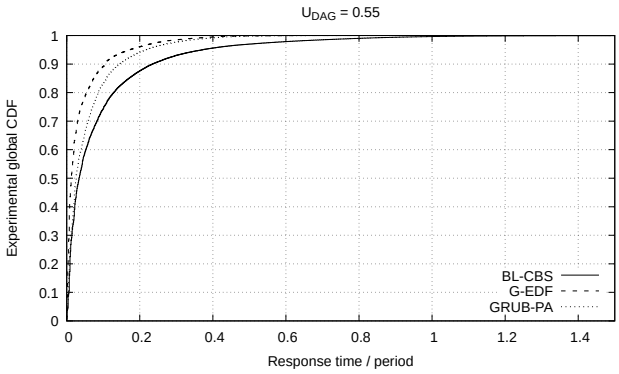
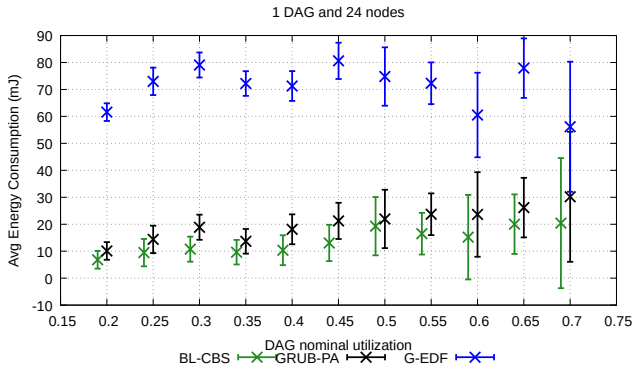


Figure 5: Experimental global cumulative distribution function (CDF) for 10 experiments with DAG nominal utilization 0.55. Jobs take more to complete with BL-CBS than with G-EDF and GRUB-PA, but all of them terminate before their deadlines.

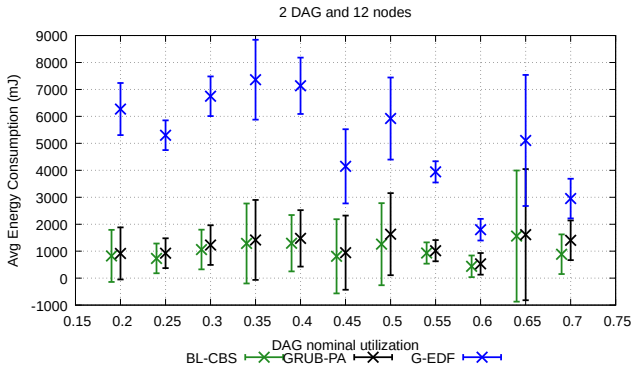
like G-EDF, does not take into account the consequent frequency increase. Moreoever, GRUB-PA consumes less than G-EDF because the latter keeps the highest frequencies throughout the simulations. When the DAG nominal utilization grows the energy consumption is pretty stable for both BL-CBS and GRUB-PA and with a growing number of DAGs the total energy consumption of the system increases dramatically, since higher frequencies are used.

Figure 7 shows the same data of Figure 6 in a hyperperiod-independent way and shows the ratio of the total energy consumption (on the Y axis) between (i) BL-CBS and GRUB-PA and (ii) BL-CBS and G-EDF (different curves) for each DAG nominal utilization over 10 experiments. Also, the vertical bars represent the minimum and maximum ratios found among the experiments. While the average ratio between BL-CBS and GRUB-PA is generally stable for all DAG nominal utilizations and in all the considered cases (1,2 and 3 DAG tasks), the ratio between BL-CBS and G-EDF decreases with growing DAG utilization. In general, the consumption of G-EDF becomes more and more similar to the one of BL-CBS with growing DAG nominal utilization, while BL-CBS saves more or less the same with respect to GRUB-PA with both growing DAG utilization and number of DAG tasks.

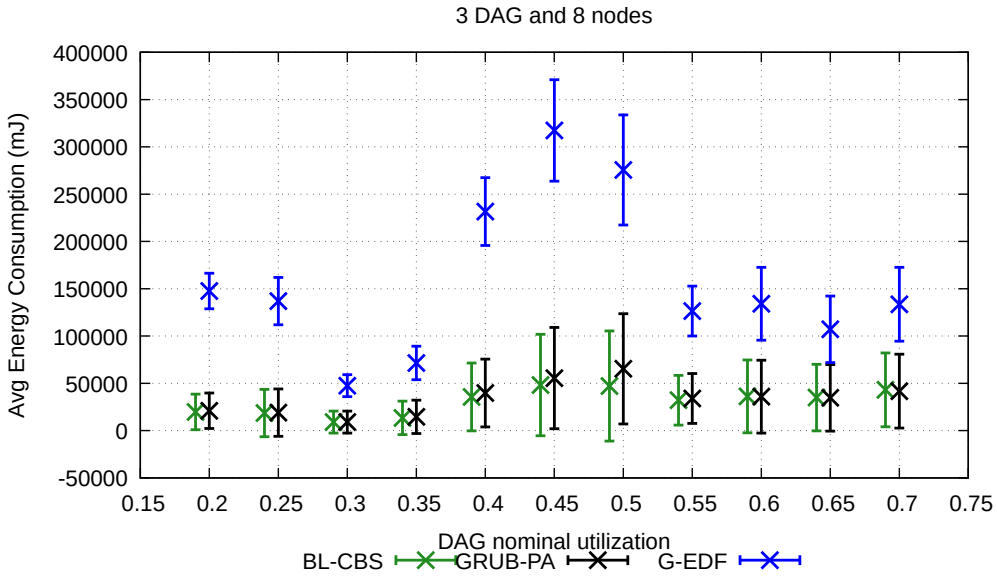
Figure 8 depicts the average frequency of the big and the LITTLE island for BL-CBS and GRUB-PA (different curves) over 10 experiments (on the Y axis) for each DAG nominal utilization (on the X axis) and for the big island (left) and the LITTLE island (right). The vertical bar associated to each point represent the average minimum and maximum values found over the experiments. In the case of 1 DAG in Figure 8a and Figure 8b, BL-CBS keeps the lowest frequencies for both the big and LITTLE island, since the utilization of the CBS servers is low (even if the DAG utilization is higher), and thus the tasks can be spread among the available cores to reduce the frequencies, while GRUB-PA uses higher frequencies and G-EDF keeps the highest ones. For a growing number of DAGs, the frequencies used by BL-CBS raise and they grow with increasing DAG nominal utilization. With both BL-CBS and GRUB-PA and for each number of DAG tasks, the LITTLE island uses higher average



(a) 1 DAG and 24 nodes



(b) 2 DAG and 12 nodes



(c) 3 DAG and 8 nodes

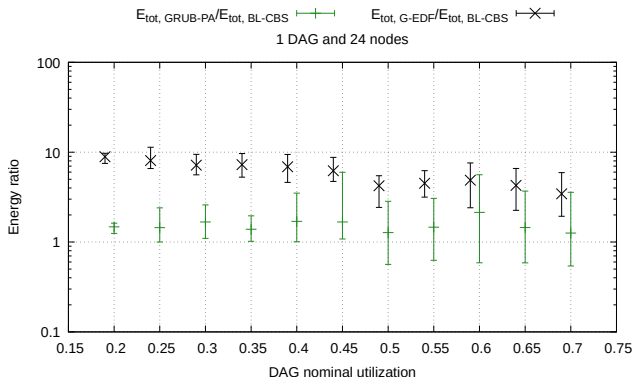
Figure 6: Average energy consumption for both BL-CBS, GRUB-PA and G-EDF for different DAG nominal utilizations, considering 10 experiments for each utilization. Y axes are in logarithmic scale.

frequencies than the big islands for each nominal utilization. Moreover, the frequencies used by GRUB-PA are higher than the ones used by BL-CBS in all the considered cases for both the big and the LITTLE islands.

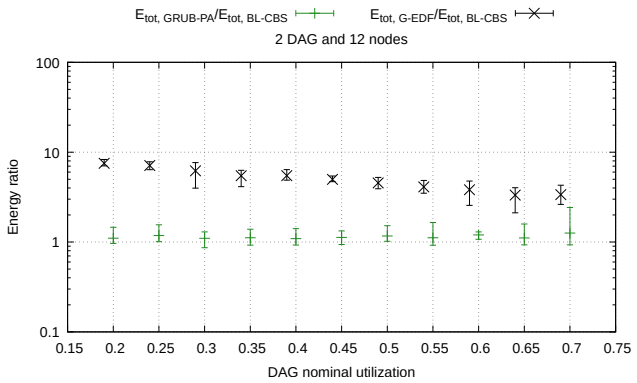
Finally, the frequency change just discussed has implications on the jobs response time, which is inversely proportional to the frequency. Figure 5 shows the obtained global experimental CDF of the response time relative to the period for all instances of the nodes) considering 10 experiments and in the cases of BL-CBS, GRUB-PA and G-EDF. Since BL-CBS keeps the lowest frequencies on both islands with respect to both GRUB-PA and G-EDF, the response time of the jobs is higher with BL-CBS than with GRUB-PA and G-EDF. Moreover, the jobs take more to finish with GRUB-PA than with G-EDF because G-EDF just keeps the highest frequencies, and in all the three cases the jobs end within their deadlines.

7 CONCLUSIONS AND FUTURE WORK

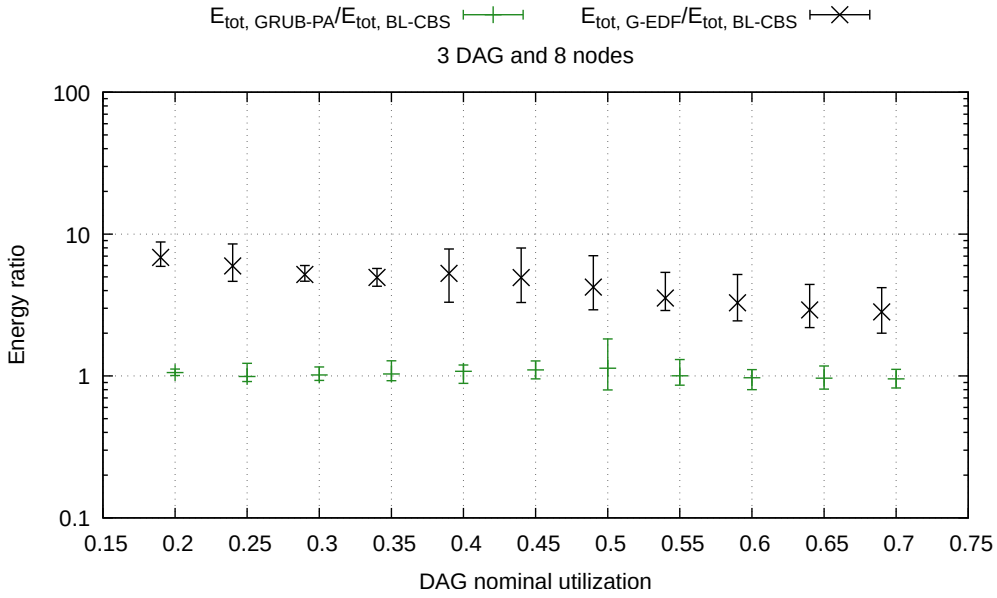
In this paper, we have presented and simulated a version of big-LITTLE Constant Bandwidth Server (BL-CBS) supporting DAG scheduling in a transparent way and suitable for being used on-line with "open" systems and able to reduce the energy consumption on platforms based on the ARM big.LITTLE architecture. An extensive number of experiments on randomly generated DAG tasks shows that this approach leads to interesting energy saving, which is in average 10% over all the performed experiments with respect to the state of the art GRUB-PA, already implemented in the current code-base of SCHED_DEADLINE in the mainline Linux kernel.



(a) 1 DAG and 24 nodes



(b) 2 DAG and 12 nodes



(c) 3 DAG and 8 nodes

Figure 7: Energy consumption ratio between (i) G-EDF and BL-CBS; (ii) GRUB-PA and BL-CBS for different DAG utilizations, considering 10 experiments for each utilization. Y axes are in logarithmic scale.

Concerning possible lines of future work on the topic, we plan to improve the performance of the placement algorithm and to evaluate its energy saving. Moreover, we plan to consider different workload types, to place the servers on the cores also considering the nodes relationships and the memory consumption, and to investigate on how to modify the proposed mechanism in order to properly consider possible deep-idle states of the CPU. Also, it would be interesting to incorporare bandwidth reclaiming and feedback mechanisms within the CBS servers so to better tolerate the jobs with execution times that exceed the WCET. Besides, it might be interesting to include some of the hard-real-time approaches in the empirical study, so to get an indication of what is the price that a designer must pay for guaranteeing deadlines.

Finally, we plan to realize this version of BL-CBS supporting DAGs within the current SCHED_DEADLINE codebase in the Linux kernel, to perform further experimentation and validation using real application workloads on Linux/Android.

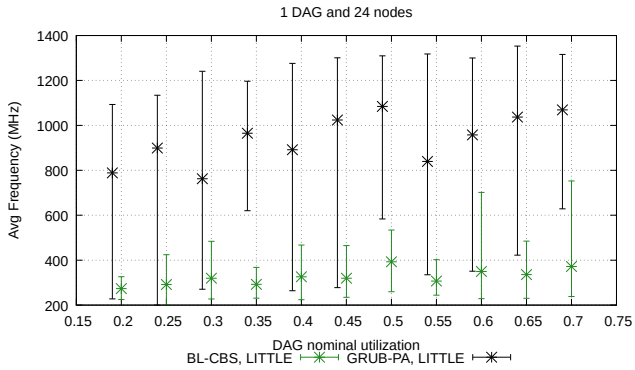
REFERENCES

[1] Luca Abeni and Giorgio Buttazzo. 1998. Integrating multimedia applications in hard real-time systems. In *Proc. 19th IEEE Real-Time Systems Symp.*

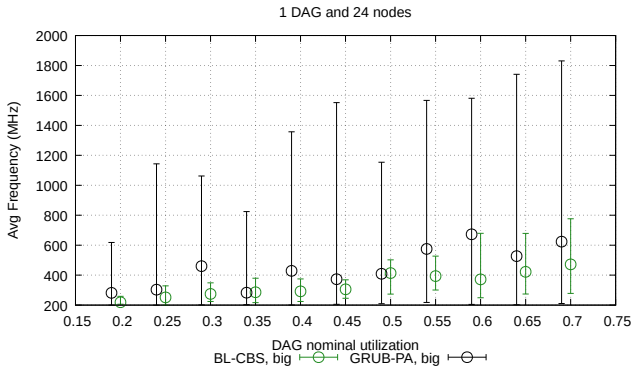
[2] Ashik ahmed Bhuiyan, Kecheng Yang, Samsil Arefin, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. 2019. Mixed-Criticality Multicore Scheduling of Real-Time Gang Task Systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 469–480.

[3] Björn Andersson and Dionisio de Niz. 2012. Analyzing global-edf for multiprocessor scheduling of parallel tasks. In *International Conference On Principles Of Distributed Systems*. Springer, 16–30.

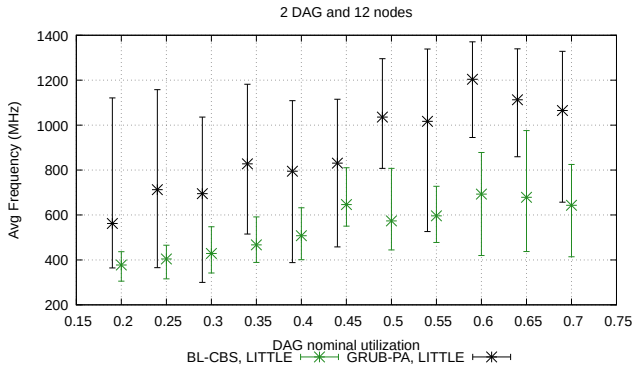
[4] ARM. 2019. ARM Technologies: DynamIQ. https://www.arm.com/why-arm/technologies/dynamiq. Accessed: November 4, 2019.



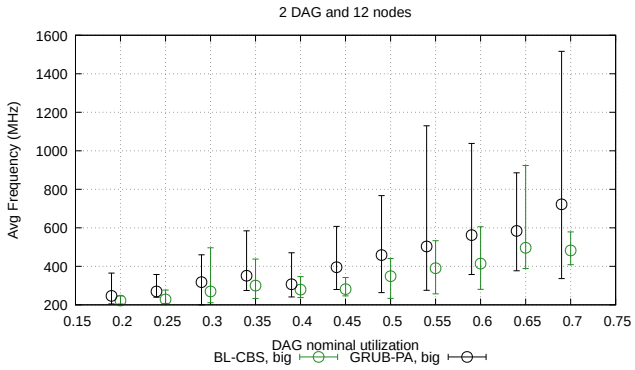
(a) LITTLE island, 1 DAG and 24 nodes



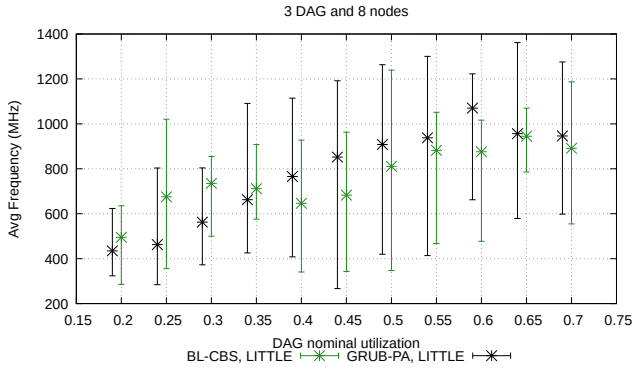
(b) big island, 1 DAG and 24 nodes



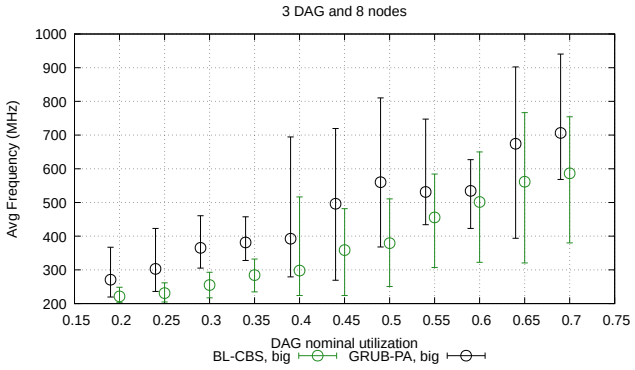
(c) LITTLE island, 2 DAG and 12 nodes



(d) big island, 2 DAG and 12 nodes



(e) LITTLE island, 3 DAG and 8 nodes



(f) big island, 3 DAG and 8 nodes

Figure 8: Average frequency for different DAG nominal utilizations for BL-CBS and GRUB-PA, for the LITTLE island (left) and the big island (right), considering 5 experiments for each DAG utilization.

[5] Hakan Aydin and Qi Yang. 2003. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings International Parallel and Distributed Processing Symposium.* IEEE, 9–pp.

[6] Alessio Balsini, Luigi Pannocchi, and Tommaso Cucinotta. 2016. Modeling and simulation of power consumption and execution times for real-time tasks on embedded heterogeneous architectures. In *Proc. International Workshop on Embedded Operating Systems.* Torino, Italy.

[7] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. 2016. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 1 (2016), 1–34.

[8] Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. 2015. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *2015 27th Euromicro Conference on Real-Time Systems.* IEEE, 222–231.

[9] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. 2010. Improved multiprocessor global schedulability analysis. *Real-Time*

Systems 46, 1 (2010), 3–24.

[10] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*. IEEE, 63–72.

[11] Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. 2009. Minimizing CPU energy in real-time systems with discrete speed management. *ACM Transactions on Embedded Computing Systems (TECS)* 8, 4 (2009), 1–23.

[12] Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. 2013. Feasibility analysis in the sporadic dag task model. In *2013 25th Euromicro conference on real-time systems*. IEEE, 225–233.

[13] Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. 2018. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 421–433.

[14] Thidapat Chantem, X Sharon Hu, and Robert P Dick. 2010. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 10 (2010), 1884–1897.

[15] Gang Chen, Kai Huang, and Alois Knoll. 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 3s (2014), 1–21.

[16] Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Proactive speed scheduling for real-time tasks under thermal constraints. In *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 141–150.

[17] Alexei Colin, Arvind Kandhalu, and Ragunathan Rajkumar. 2014. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 1–10.

[18] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. 2010. Random Graph Generation for Scheduling Simulations. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques* (Torremolinos, Malaga, Spain) *(SIMUTools '10)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, Article 60, 10 pages.

[19] T. Cucinotta and L. Palopoli. 2010. QoS Control for Pipelines of Tasks Using Multiple Resources. *IEEE Trans. Comput.* 59, 3 (March 2010), 416–430. https://doi.org/10.1109/TC.2009.116

[20] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Thermal-aware global real-time scheduling on multicore systems. In *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 131–140.

[21] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2011. Thermal-aware global real-time scheduling and analysis on multicore systems. *Journal of Systems Architecture* 57, 5 (2011), 547–560.

[22] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D Koutsoukos, and Hongan Wang. 2010. Feedback thermal control for real-time systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 111–120.

[23] Yong Fu, Nicholas Kottenstette, Chenyang Lu, and Xenofon D Koutsoukos. 2012. Feedback thermal control of real-time systems on multicore processors. In *Proceedings of the tenth ACM international conference on Embedded software*. 113–122.

[24] Zhishan Guo, Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, and Nan Guan. 2019. Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 156–168.

[25] Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. 2017. Energy-efficient multi-core scheduling for real-time DAG tasks. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[26] Ravindra Jejurikar. 2005. Energy aware non-preemptive scheduling for hard real-time systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. IEEE, 21–30.

[27] Xu Jiang, Nan Guan, Xiang Long, and Wang Yi. 2017. Semi-federated scheduling of parallel real-time tasks on multiprocessors. In *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 80–91.

[28] Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2013. Outstanding paper award: Analysis of global edf for parallel tasks. In *2013 25th Euromicro Conference on Real-Time Systems*. IEEE, 3–13.

[29] Jing Li, Jian Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. 2014. Analysis of federated and global scheduling for parallel real-time tasks. In *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 85–96.

[30] Giuseppe Lipari and Sanjoy Baruah. 2001. A hierarchical extension to the constant bandwidth server framework. In *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*. IEEE, 26–35.

[31] Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. 2012. Power-efficient time-sensitive mapping in heterogeneous systems. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. 23–32.

[32] Di Liu, Jelena Spasic, Gang Chen, and Todor Stefanov. 2015. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *2015 13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)*. IEEE, 1–10.

[33] Agostino Mascitti, Tommaso Cucinotta, and Mauro Marinoni. 2020. An adaptive, utilization-based approach to schedule real-time tasks for ARM big. LITTLE architectures. *ACM SIGBED Review* 17, 1 (2020), 18–23.

[34] Sujay Narayana, Pengcheng Huang, Georgia Giannopoulou, Lothar Thiele, and R Venkatesha Prasad. 2016. Exploring energy saving for mixed-criticality systems on multi-cores. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 1–12.

[35] Geoffrey Nelissen, Vandy Berten, Joël Goossens, and Dragomir Milojevic. 2011. Optimizing the number of processors to schedule multi-threaded tasks. *RTSS'11-WiP Session, December 2011* (2011), 5–8.

[36] Santiago Pagani and Jian-Jia Chen. 2013. Energy efficient task partitioning based on the single frequency approximation scheme. In *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 308–318.

[37] Santiago Pagani and Jian-Jia Chen. 2014. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 5s (2014), 1–25.

[38] Luigi Palopoli, Guiseppe Lipari, Luca Abeni, Marco Di Natale, Paolo Ancilotti, and Fabio Conticelli. 2001. A Tool for Simulation and Fast Prototyping of Embedded Control Systems. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems* (Snow Bird, Utah, USA) *(LCTES'01)*. Association for Computing Machinery, New York, NY, USA, 73–81. https://doi.org/10.1145/384197.384209

[39] Antonio Paolillo, Joël Goossens, Pradeep M Hettiarachchi, and Nathan Fisher. 2014. Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 1–10.

[40] Xuan Qi and Da-Kai Zhu. 2011. Energy efficient block-partitioned multicore processors for parallel applications. *Journal of Computer Science and Technology* 26, 3 (2011), 418.

[41] Abusayeed Saifullah, Sezana Fahmida, Venkata P Modekurthy, Nathan Fisher, and Zhishan Guo. 2020. CPU Energy-Aware Parallel Real-Time Scheduling. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[42] Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D Gill. 2014. Parallel real-time scheduling of DAGs. *IEEE Transactions on Parallel and Distributed Systems* 25, 12 (2014), 3242–3252.

[43] Claudio Scordino and Giuseppe Lipari. 2004. Using Resource Reservation Techniques for Power-Aware Scheduling. In *Proceedings of the 4th ACM International Conference on Embedded Software* (Pisa, Italy) *(EMSOFT '04)*. ACM, New York, NY, USA, 16–25.

[44] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. 2008. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE transactions on parallel and distributed systems* 19, 11 (2008), 1540–1552.

[45] Corey Tessler, Venkata P Modekurthy, Nathan Fisher, and Abusayeed Saifullah. 2020. Bringing Inter-Thread Cache Benefits to Federated Scheduling. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 281–295.

[46] Leping Wang and Ying Lu. 2008. Efficient power management of heterogeneous soft real-time clusters. In *2008 Real-Time Systems Symposium*. IEEE, 323–332.

[47] Chuan-Yue Yang, Jian-Jia Chen, Lothar Thiele, and Tei-Wei Kuo. 2010. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 9–14.

[48] Buyoung Yun, Kang G Shin, and Shige Wang. 2013. Predicting thermal behavior for temperature management in time-critical multicore systems. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 185–194.

[49] Dakai Zhu, Nevine AbouGhazaleh, Daniel Mossé, and Rami Melhem. 2002. Power aware scheduling for and/or graphs in multiprocessor real-time systems. In *Proceedings International Conference on Parallel Processing*. IEEE, 593–601.

[50] Dakai Zhu, Daniel Mosse, and Rami Melhem. 2004. Power-aware scheduling for AND/OR graphs in real-time systems. *IEEE Transactions on Parallel and Distributed Systems* 15, 9 (2004), 849–864.