An adaptive, utilization-based approach to schedule real-time tasks for ARM big.LITTLE architectures

Agostino Mascitti, Tommaso Cucinotta, Mauro Marinoni
Scuola Superiore Sant'Anna
firstname.lastname@santannapisa.it

ABSTRACT

ARM big.LITTLE architectures are spreading more and more in the mobile world thanks to their power-saving capabilities due to the use of two ISA-compatible islands, one focusing on energy efficiency and the other one on computational power. This architecture makes the problem of energy-aware task scheduling particularly challenging, due to the number of variables to take into account and the need for having lightweight mechanisms that can be readily computed in an operating system kernel scheduler.

This paper presents a novel task scheduler for big.LITTLE platforms, combining the well-known Constant Bandwidth Server algorithm with a power-aware per-job migration policy. This achieves real-time adaptation of the CPU islands' frequencies based on the individual cores' overall utilization, as available in the scheduler thanks to the use of the resource reservation paradigm. Preliminary results obtained by simulations based on modifications to the open-source RTSim tool show that the proposed technique is able to achieve interesting performance/energy trade-offs.

CCS CONCEPTS

• **Computer systems organization → Real-time operating systems**; *Embedded systems*;

KEYWORDS

Real-time systems, DVFS, ARM big.LITTLE, heterogeneous processing, multicore platforms, energy-efficiency

ACM Reference Format:
Agostino Mascitti, Tommaso Cucinotta, Mauro Marinoni. 2019. An adaptive, utilization-based approach to schedule real-time tasks for ARM big.LITTLE architectures. In *Proceedings of Embedded Operating Systems Workshop (EWiLi'19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Heterogeneous systems play a fundamental role in mobile computing nowadays, taking the dominating position in the hardware market previously held by homogeneous computing solutions. In the relentless effort to produce more and more powerful CPUs, hardware producers already stopped years ago the rush for higher CPU frequencies in favor of multicore architectures, in order to keep under control the heat generated by processors, so to be able to dissipate it timely. Furthermore, the tremendous increase in mobile computing led processor manufacturers to focus on novel designs, able to deal in an energy-efficient way with quite heterogeneous workloads, from personal computing to high-performance 3D gaming. This led to the rise of heterogeneous platforms, not only for accelerating graphics, incorporating GPUs on the chip, but also having different general-purpose processing cores with different profiles of computing capacity vs power consumption. This is the case of the ARM big.LITTLE [3, 20], which is a very popular architecture for Android smartphones nowadays. In these processors we have the coexistence of high-performance "big" cores and energy-efficient "small" cores. These share the same Instruction Set Architecture (ISA) and have a coherent view of the system RAM, so it is possible to schedule and migrate tasks among them as is normally done in a symmetric multi-core system. Hence, the operating system (OS) can make intelligent use of scheduling and placement algorithms coupled with energy-saving features, to execute tasks timely and in an energy-efficient way. For example [6], with *Dynamic Frequency Voltage Scaling* (DVFS) it is possible to scale down the frequency of the CPU(s) whenever possible depending on the workload requirements, and with *Dynamic Power Management* (DPM) techniques an idle core can be put into a number of possible low-power states corresponding to different trade-offs between energy saving and wake-up latency. In general, these techniques tend to slow down and/or switch off various parts of a processor circuitry, to control heat and energy consumption.

Some efforts have been made to make joint task placement and energy management decisions, in order to lengthen battery duration, as witnessed by the *Intelligent Power Allocation* (IPA) and *Energy Aware Scheduling* (EAS) frameworks. EAS has been merged into mainline Linux from version 5.0 onwards, extending the Linux scheduler to make it aware of the underlying power/performance capabilities of the available CPUs.

In general, the scheduler needs to have a sufficiently detailed power consumption model to make proper decisions. The power consumption estimation must be computed given the core type, its frequency, and the current task workload. However, the problem of minimizing energy consumption while respecting tasks deadlines is known to be NP-complete [25].

Paper contributions. This paper tackles the problem of energy-aware scheduling mechanisms for soft real-time applications suitable for big.LITTLE multicore architectures. We introduce *BL-CBS*, a novel adaptive utilization-based scheduler that is designed to work at runtime, within the OS kernel scheduler, with the knowledge normally possessed by this component. Our solution couples a well-known reservation-based paradigm [18] in the form of a hard constant bandwidth server (CBS) scheduler, with on-line task placement and migration decisions that are taken whenever tasks activate or suspend within the scheduler queue, considering the energy consumption peculiarities of big.LITTLE architectures. An implementation of the proposed mechanism has been realized by modifying the open-source RTSim real-time simulator [21]. This

EWiLi '19, October 17, 2019, New York, USA.

has been used to gather preliminary results demonstrating the advantages of the proposed technique in terms of energy saving with respect to the common baseline approach for multicore platforms, typical for example of the current implementation of the CBS in Linux. The obtained results seem promising, demonstrating that the proposed approach may constitute a sound base to build practical and viable new mechanisms for real-time and multimedia workloads in the Linux kernel on mobile platforms, such as adopted on common Android devices.

2 BACKGROUND

ARM big.LITTLE is a heterogeneous computing architecture composed by two groups, called islands or clusters, of homogeneous cores: the *big* one is performance-oriented, while the *LITTLE* one focuses on energy efficiency. Both islands cores share the same ISA, while the differences in the micro-architecture results in different energy profiles. For example, a Cortex-A15 has higher performance than a Cortex-A7 since it runs more instructions in parallel and with a bigger and more complex pipeline. Moreover, the islands are connected by the Cache Coherent Interconnect, that is another crucial feature for migration of tasks across islands, since it allows avoiding to go through the main memory, saving time.

A commonly used technique to reduce power consumption is *Dynamic Voltage Frequency Scaling* (DVFS), where the island frequency can be dynamically changed. Also, *Dynamic Power Management* (DPM) is an available feature, but its usefulness is limited in this kind of platform since the whole island must be idle (i.e., all cores without jobs), which is quite a rare situation, as reasoned in [12]. Cores in the same big.LITTLE island are constrained to the same frequency, limiting the hardware complexity but also the power saving capabilities. However, the use of smart scheduling algorithms aware of the islands' power profiles may achieve energy-efficiency effectively [13]. On a related note, the recent DynamIQ architecture[1] no longer has this constraint.

The CBS is a well-known technique [1] to provide temporal isolation among soft real-time tasks. It grants each reservation s an execution budget Q_s to access the CPU over each time period of duration T_s, with Q_s and T_s being per-server parameters that can be assigned according to the requirements of the workload to be served, under the constraint that $\sum_s Q_s/T_s \leq 1$. A hard reservation, multi-processor variant of the CBS, is for example available in the SCHED_DEADLINE scheduler within the Linux kernel since version 3.14 [15]. It relies on global EDF (G-EDF), but it can be configured in partitioned EDF mode via cpusets. One known limitation of the mainline kernel is that, whenever the POSIX real-time or SCHED_DEADLINE policies are used, the maximum CPU frequency is forced in the system [3]. The baseline Android kernel includes a variant in which the minimum frequency satisfying the real-time utilization requirements of SCHED_DEADLINE tasks on each CPU is forced. However, tasks are migrated strictly following the global EDF rules, differing from what is proposed in this paper.

[1]More information at: https://www.arm.com/why-arm/technologies/dynamiq.

3 RELATED WORK

Energy-efficient scheduling for real-time systems has received much attention in the last two decades, albeit the majority of works focused on *homogenous* multi-core platforms [5]. In what follows, we briefly recall related research on energy-aware schedulers for *heterogeneous* multi-core platforms only, most of which are based on task-splitting, optimization methods, and task speed profiles.

Semi-partitioned scheduling [2] deals with the fragmentation problem faced when trying to partition a real-time task set over multi-processor platforms. It relies on splitting a real-time task into two parts with reduced demand that fit into two processors and execute one after the other, requiring a migration, keeping schedulability. These techniques have been studied for homogeneous multicores [2, 7, 8] as well as for heterogenous architectures [17], but they require complex partitioning and schedulability analysis techniques. Some authors [8] investigated the use of linear-time approximation methods to perform the splittings, to make the algorithm more useful in real-world settings. The present work advocates a simpler utilization-based method where task splitting is not used: a task resides mostly on one core for each job, and it is migrated normally only across subsequent activations. This is easier to compute in an OS kernel (see Section 5.2 for details).

Some authors [14] studied whether it is better to execute at the highest possible frequency and then set cores to an idle state as long as possible or to find the lowest frequency needed to make tasks respect their deadlines and never go idle. Interestingly, they find out that no strategy is the best one for all platforms and all applications. In fact, one application executed on an embedded device can have lower energy consumption when using the lowest frequency, while on another platform it would be more convenient to make it run at the highest frequency and maximize the slack time, where the platform can go into an idle state.

Several authors formulated the problem of energy-efficient real-time scheduling for multi-cores as an optimization problem [9, 22–24]. Polynomial-time algorithms have been proposed [16] to divide real-time streaming applications across DVFS capable islands, where tasks are assigned statically to an island and then globally scheduled inside it via an optimal scheduler. In [10], it is found that the most efficient way to allocate real-time tasks and save energy is neither balancing the load nor choosing the most power-efficient core. They find the optimal load distribution via integer linear programming (ILP) and try to approximate that result via heuristics. Other authors [24] divide the scheduling problem into workload partitioning and next task ordering. The first step determines what parts of tasks should be executed at what frequency within a time interval such that feasibility constraints are satisfied, while the second part establishes how to order the pieces of tasks for each core. Also, an analysis of the task code structure has been proposed [22] coupled with an ILP returning the minimum frequency and location to be used for each code segment. In [22], the authors tried to moderate the use of the *LITTLE-Core-First* principle, according to which one should always fill LITTLE cores while possible before selecting a *big* one. Using the task execution variance (i.e., the ratio between the WCET on LITTLE and on big for a given task) an ILP formulation is used to compute the optimal distribution of the

utilization of the tasks between the islands and their minimum frequencies to respect the deadline. A heuristic is then used to assign tasks and set the frequencies. This approach, however, is related to ARM DynamIQ and it makes use of per-core-DVFS, which is not feasible in big.LITTLE. Optimization methods have been used [19] for choosing the best frequency for each node of stream processing applications represented as Synchronous Data Flows with end-to-end deadlines, making their parallelism explicit in the model.

However, most of the above works focus on optimizing the configuration of a whole partitioned system, whilst in this paper we deal with designing a strategy that can be applied on-line within an OS kernel scheduler considering both task migrations and possible task overruns, which we handle by using CBS servers.

Finally, an interesting approach is the one in [4], where authors highlight that power consumption may significantly vary depending also on the workload type being computed, supporting the argument with real data measured on a ODROID XU3 platform, and making the resulting energy model available via modifications to RTSim. This is the work we actually start from, in this paper, albeit we do not deal with heterogeneous workloads.

4 SYSTEM MODEL

This section describes the task and CPU models used in the paper.

4.1 CPU model

The system under consideration is an ARM big.LITTLE consisting of two homogeneous islands with DVFS capabilities, where all cores in the same island share the same frequency. At a higher level of abstraction, a core is characterized by a power model and a speed. The CPU *power model* is given by [4] and it tries to be a good compromise between representativeness and complexity:

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

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 [4]. 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 [4], this approach results in a sufficient approximation for our simulations. We define the speed of a core as a number between 0 and 1 representing its computational capacity, relative to a big core at maximum frequency.

4.2 Task model

A periodic task τ is characterized by an absolute Worst-Case Execution Time $WCET$ and a period P equal to its deadline (i.e., implicit deadline). Without loss of generality, we assume that its absolute WCET is determined under the hypothesis of one workload type only (e.g., *bzip2* among the ones in [4]), to simplify the presentation. It would be possible to use many instructions with different workload types since the used model is already fitted for that. The $WCET$ associated to each task is assumed to correspond to when the task is running on a big core at maximum frequency. The actual WCET when executing on a big or little core at a given frequency is $WCET_{scaled}$ and it is a function of the core speed, computed according to the execution time model in [4].

Each periodic task is then enveloped in its own CBS server. This allows for throttling the task if it exceeds its WCET, and it allows the use of the virtual time and the active utilization during the acceptance test. The corresponding active utilization, stored on the core where the ending task is located, is equal to the task scaled utilization until the virtual time expires or the core goes idle. A similar idea is in [26], where tasks are statically assigned to CPUs, enveloped in servers and they globally reclaim unused bandwidth.

5 PROPOSED APPROACH

In this section, we start providing an easy-to-follow example to explain the main idea behind BL-CBS. Then, we detail our proposed task placement strategy.

5.1 Main idea

The main idea behind BL-CBS is the one to realize an adaptive partitioned scheduling of real-time EDF/CBS-based reservations: consider the two main instants in which the scheduler is invoked by the OS, i.e., when a task wakes up (or is created) and when it goes to sleep (or is terminated), and, among the available placement options that respect schedulability, *pick the one that brings the minimum foreseen energy consumption for the whole system.* For a task arrival, the placement options concern preempting one of the currently running tasks on any of the CPU(s), or placing the task on an idle CPU. For a task departure, we restrict in this paper to the option of possibly moving one of the tasks ready to run from other CPUs, to the one that just got additional spare capacity. However, note that, in presence of CBS servers, in order to guarantee schedulability of real-time tasks, the latter decision can be taken immediately only when the CPU goes idle, otherwise we have to post-pone said decision to the virtual time of the departing task.

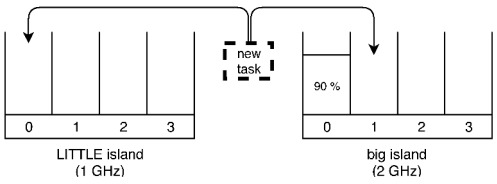


Figure 1: Placement decision on task wake-up or task creation. Percentages represent cores utilizations.

Consider for example the scenario depicted in Figure 1: a heavy task that cannot fit on a little core is running on one of the big cores, with the rest of the system idle. When another lighter task that can fit on both cores wakes up or just arrives in the system,

the most direct decision seems to be the one to place it on a little core, setting the lowest possible frequency on the island so as to make it fit. However, the little island is completely idle at this time, whilst the big island cannot be shut down because of the heavy task presence. Moreover, placing the new task on another big core results in it executing much faster. The overall energy consumption comparison between the two decisions needs to consider both the difference in power consumption and the difference in processing time, plus the fact that when an island is completely idle it might bring additional savings. As highlighted in [4], the execution-time and power consumption profiles of tasks when deployed on different islands might be workload dependent, so a comprehensive approach would imply the use of a monitoring strategy that adaptively catches this level of details. However, this paper focuses on a single workload type for the sake of simplicity, deferring the more complex approach to future work.

Consider now the scenario in Figure 2, where both little and big cores are quite heavily occupied, when the only task keeping the first little core busy goes away. As the core would remain idle, it can pull immediately another task. G-EDF, as e.g., implemented in SCHED_DEADLINE, would simply pull the ready-to-run task with the earliest deadline from other cores that fits, and assume this is the task waiting for execution on the 3rd big core. However, as evident from the picture, pulling the task waiting for execution on the 4th big core (that fits as well on little core 0) is more advantageous, because it reduces the maximum utilization among cores of the big island, allowing for scaling down its frequency to a lower value. This is exactly what our algorithm does in such a case. The full algorithm is detailed in the next section.

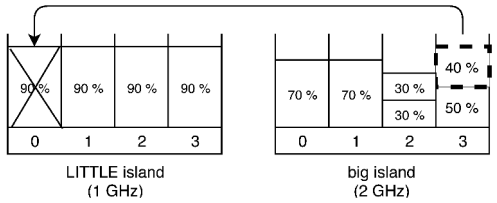


Figure 2: Placement decision on task departure.

5.2 Task placement and DVFS

Our proposed Placement Algorithm is sketched out in Algorithm 1. It works with any taskset and it does not need previous information about tasks but their WCET and period. Specifically, when a job arrives the Algorithm 1 is called to decide a core and its frequency. It loops over all cores and their frequencies, starting with the island frequency. The first check is that jobs do not miss their deadlines and that they are schedulable with EDF. Then, it computes the difference between current power consumption and the one if the job is dispatched to that core, considering the 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

Algorithm 1 Conceptual placement algorithm.

```
1:  procedure PLACEMENT ALGORITHM(job, cores ρ)
2:      Δ = ∅ // set of triplets (core, freq, power)
3:      for each core i ∈ ρ do
4:          for each f_i' ≥ current one f_i do
5:              if isAdmissibleEDF(job,f_i') then
6:                  Let U_{tot,f_i'} be the util. on core i at freq. f_i'
7:                  Let U_{job,f_i'} be the job util. on core i at freq. f_i'
8:                  Let λ be the island of core i
9:                  if U_{tot,f_i'} + U_{job,f_i'} > 1 then continue
10:                 end if
11:                 oldIslandUtil = Σ_{c∈λ}(U_{c,f_i} + U_{active,c})
12:                 newIslandUtil = Σ_{c∈λ}(U_{c,f_i'} + U_{active,c})
13:                 oldPower_i = oldIslandUtil × powerCons_{f_i}
14:                 newPower_i = (newIslandUtil+U_{job,f_i'})×powerCons_{f_i'}
15:                 Add (i, f_i', newPower_i − oldPower_i) to Δ
16:             else
17:                 Job not admissible
18:             end if
19:         end for
20:         Return (core, frequency) with min power from Δ.
21:     end for
22: end procedure
```

RTSim is actually able to exploit the full optimized CPU power model as available from [4].

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:

$$WCET_{f''} \leq (1 - U_{\rho_{final},f''})(D_{abs} - t), \tag{2}$$

where we want to move a task 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 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. 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, the focus in this paper has been on the functionality of BL-CBS, but we have not optimized Algorithm 1 for efficient execution. This is left as future work, necessary for a viable solution to be embedded in a real OS kernel scheduler.

6 IMPLEMENTATION DETAILS

The mechanism just described has been implemented within RTSim [21], a portable, open-source event-based simulator written in C++ that allows to simulate the execution timing of real-time tasks running on multi-processor platforms using various schedulers.

We started from the variant of RTSim used in [4], containing a first attempt at modeling the power consumption of big.LITTLE architectures. We extended RTSim modifying the classes `CPU`, `CBServer`, `EnergyMRTKernel` and `MultiCoreScheds`. In particular:

(1) `CPU` now supports the concept of island so that all CPUs in the same island are constrained to use the same frequency;
(2) `MultiCoreSched` implements local, per-core scheduling queues, as in the Linux kernel scheduler design;
(3) `EnergyMRTKernel` implements the Algorithm 1 and various utility methods to compute and store active utilization, task utilization scaled with the core speed and the island utilization. Moreover, it makes use of `MultiCoreSched`, through which all CPUs have their own queue of running and ready jobs. Jobs can be added and removed from a queue and migrations can be traced as well.
(4) `CBServer` has been improved to compute correctly the remaining capacity (since it must be scaled with the current core speed) and with some necessary callbacks to `EnergyMRTKernel`. For example, when the CBS server goes from releasing to idle, the kernel must be informed so to remove the active utilization of that server.

7 RESULTS

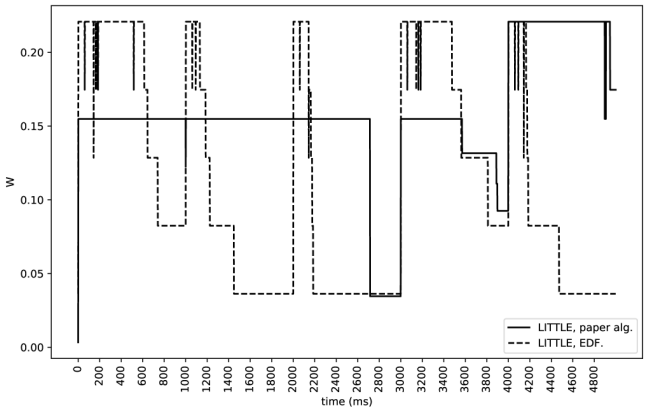
Table 1: Comparison of BL-CBS vs G-EDF

	BL-CBS	G-EDF
Avg f_{big} (MHz)	1228	2000
Avg f_{LITTLE} (MHz)	1238	1400
Energy big (mJ)	3385	4842
Energy LITTLE (mJ)	788	571
Energy tot. (mJ)	4173	5413
Avg rel. resp. time	0.37	0.14

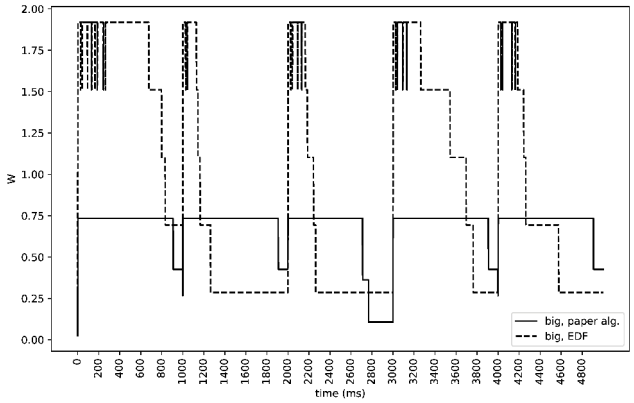
In this section we present simulation results comparing the effectiveness of the novel algorithm in terms of energy consumption with respect to the basic G-EDF algorithm as available in RTSim through the original `MRTKernel` class, simulating the mainline Linux behavior running SCHED_DEADLINE CBS reservations.

As said before, the hardware energy consumption model is the one of ODROID XU3, which uses a Samsung Exynos 5422 SoC. This is an ARM big.LITTLE architecture with four Cortex-A7 and four Cortex-A15 cores. The model has been taken from Balsini et al. [4], where it has been embedded in RTSim. Experiments are performed with 3 tasks per core with total utilization of 50%. Their periods are in range 1000-5000 ms. Task sets have been randomly generated using the open-source `taskgen.py`[2] by Emberson [11]. Experiments are repeated 10 times with different tasksets, each being scheduled by either BL-CBS or G-EDF.

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



(a) Consumption for the LITTLE island



(b) Consumption for the big island

Figure 3: Comparison of BL-CBS with G-EDF for each island

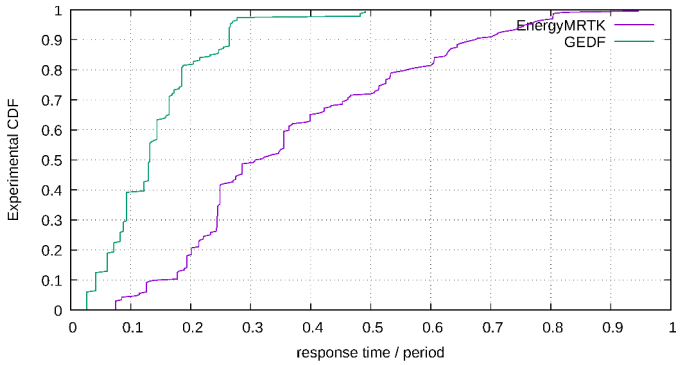


Figure 4: Response-time CDF for the first generated taskset, when scheduled by G-EDF vs our new algorithm.

Figure 3a shows a comparison of power consumption between the two algorithms in the LITTLE island and 3b shows the same for the big one. For each time and island, the average consumption of all experiments is considered. The results show how the sum of energy consumption of both islands *in average for 10 experiments*

is 5519 mJ with the G-EDF and 4291 mJ with BL-CBS with a simulation time of 15000 steps, resulting in 22% of energy saving.

It is possible to further reduce the energy consumption by lowering frequencies even more often, for example after a migration. In fact, for these experiments they are reduced only if an island gets idle and when virtual time expires and then increased when jobs arrive. Alternatively, increasing the number of migrations might allow to save energy. However, this is left as future work.

The graphs in Figure 3 show that the most remarkable difference with this approach is in the big island, where more energy is saved. This is due to the fact that we choose the lowest frequency needed to avoid deadline misses and that we try to use as much as possible the LITTLE island, since it is generally power-oriented, and thus the frequency of the big cluster can be reduced more often. Moreover, the energy consumption in the LITTLE island is less uniform over time, whereas the one in the big island is more stable.

Table 1 shows the overall energy consumption *for one experiment only*, representing the integrals of the curves in Figure 3. Our algorithm has 22.9% energy saving wrt G-EDF. G-EDF is set to use the maximum frequency on each core; instead, BL-CBS can change it via DVFS and the average frequencies are very similar on both islands. Also, the average relative response time (relative to the task period) increases in BL-CBS because frequencies are not the maximum possible and therefore jobs experience longer execution times. Therefore their execution time (i.e. $WCET_{scaled}$) tends to the period, as confirmed by the cumulative distribution function (CDF) of their relative response times (response-time / period) visible in Figure 4, generated for all instances of all tasks.

8 CONCLUSIONS

We proposed BL-CBS, a novel approach to migrate CBS-reserved tasks in an energy-aware way on big.LITTLE platforms. The method is based on simple utilization calculations, that are rescaled according to the computational capacity of cores as due to their island type and current frequency. The algorithm has been implemented in the open-source real-time tasks simulator RTSim and preliminary experimental results showed a 22% energy consumption reduction for randomly generated task sets, compared to G-EDF.

As for future works, we plan to study under what theoretical conditions task sets can be guaranteed to be scheduled with our new strategy. Then, we plan to implement the proposed placement strategy in Linux, modifying SCHED_DEADLINE. The technique might need adaptation to deal with non-negligible frequency transition times, as assumed in the current simulations. Also, the algorithm can spread to ARM DynamIQ. Finally, this work can be extended considering more complex task models, like tasks with constrained deadlines, tasks sharing mutexes, or under precedence constraints (DAG) (e.g., following up on [12], one of the few works in that direction on ARM big.LITTLE).

REFERENCES

[1] Abeni, L., and Buttazzo, G. Integrating multimedia applications in hard real-time systems. In *Proc. 19th IEEE Real-Time Systems Symposium* (1998), IEEE.

[2] Andersson, B., and Tovar, E. Multiprocessor scheduling with few preemptions. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)* (Aug 2006), pp. 322–334.

[3] Balsini, A., Cucinotta, T., Abeni, L., Fernandes, J., Burk, P., Bellasi, P., and Rasmussen, M. Energy-efficient low-latency audio on android. *Journal of Systems and Software 152* (2019), 182 – 195.

[4] Balsini, A., Pannocchi, L., and Cucinotta, T. 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 (EWILI 2018)* (Torino, Italy, 2016).

[5] Bambagini, M., Marinoni, M., Aydin, H., and Buttazzo, G. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS) 15*, 1 (2016), 7.

[6] Bhatti, K., Belleudy, C., and Auguin, M. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing* (Dec 2010), pp. 184–191.

[7] Burns, A., Davis, R. I., Wang, P., and Zhang, F. Partitioned EDF scheduling for multiprocessors using a C= D task splitting scheme. *Real-Time Systems 48*, 1 (2012), 3–33.

[8] Casini, D., Biondi, A., and Buttazzo, G. Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)* (2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[9] Chwa, H. S., Seo, J., Yoo, H., Lee, J., and Shin, I. Energy and feasibility optimal global scheduling framework on big. LITTLE platforms. In *Proc. IEEE RTSOPS* (2015), pp. 1–11.

[10] Colin, A., Kandhalu, A., and Rajkumar, R. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *Proc. IEEE 20th International Conf. on Embedded and Real-Time Computing Systems and Applications* (2014).

[11] Emberson, P., Stafford, R., and Davis, R. I. Techniques For The Synthesis Of Multiprocessor Tasksets. In *Proc. 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)* (Brussels, Belgium, 2010).

[12] Guo, Z., Bhuiyan, A., Liu, D., Khan, A., Saifullah, A., and Guan, N. Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2019), IEEE, pp. 156–168.

[13] Herbert, S., and Marculescu, D. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proc. 2007 international symposium on Low power electronics and design (ISLPED'07)* (2007), IEEE, pp. 38–43.

[14] Imes, C., and Hoffmann, H. Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-ISA heterogeneous multi-cores. *ACM SIGBED Review 11*, 4 (2015), 49–54.

[15] Lelli, J., Faggioli, D., Cucinotta, T., and Lipari, G. An experimental comparison of different real-time schedulers on multicore systems. *Journal of Systems and Software 85*, 10 (2012), 2405 – 2416. Automated Software Evolution.

[16] Liu, D., Spasic, J., Chen, G., and Stefanov, T. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous MPSoCs. In *13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)* (2015).

[17] Liu, D., Spasic, J., Wang, P., and Stefanov, T. Energy-efficient scheduling of real-time tasks on heterogeneous multicores using task splitting. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)* (2016), IEEE, pp. 149–158.

[18] Mercer, Savage, and Tokuda. Processor capacity reserves: operating system support for multimedia applications. In *1994 Proceedings of IEEE International Conference on Multimedia Computing and Systems* (May 1994), pp. 90–99.

[19] Nogues, E., Pelcat, M., Menard, D., and Mercat, A. Energy efficient scheduling of real time signal processing applications through combined DVFS and DPM. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)* (2016), IEEE, pp. 622–626.

[20] Padoin, E. L., Pilla, L. L., Castro, M., Boito, F. Z., Alexandre Navaux, P. O., and MÃŀhaut, J. Performance/energy trade-off in scientific computing: the case of arm big.little and intel sandy bridge. *IET Comp. Digital Techniques 9*, 1 (2015).

[21] Palopoli, L., Lipari, G., Lamastra, G., Abeni, L., Bolognini, G., and Ancilotti, P. An object-oriented tool for simulating distributed real-time control systems. *Software: Practice and Experience 32*, 9 (2002), 907–932.

[22] Qin, Y., Zeng, G., Kurachi, R., Li, Y., Matsubara, Y., and Takada, H. Energy-Efficient Intra-Task DVFS Scheduling Using Linear Programming Formulation. *IEEE Access 7* (2019), 30536–30547.

[23] Qin, Y., Zeng, G., Kurachi, R., Matsubara, Y., and Takada, H. Execution-variance-aware task allocation for energy minimization on the big. LITTLE architecture. *Sustainable Computing: Informatics and Systems 22* (2019), 155–166.

[24] Thammawichai, M., and Kerrigan, E. C. Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. *Real-Time Systems 54*, 1 (2018).

[25] Ullman, J. D. NP-complete scheduling problems. *Journal of Computer and System sciences 10*, 3 (1975), 384–393.

[26] Zahaf, H.-E., Lipari, G., and Abeni, L. Migrate when necessary: toward partitioned reclaiming for soft real-time tasks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems* (2017), ACM, pp. 138–147.