Simulating Execution Time and Power Consumption of Real-Time Tasks on Embedded Platforms

Gabriele Ara
Scuola Superiore Sant'Anna
Pisa, Italy
gabriele.ara@santannapisa.it

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

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

ABSTRACT

In this paper, we present PARTSim, an open-source power/thermal-aware simulator for embedded real-time systems. This tool is a fork of the well-known RTSim simulator, which can simulate the timing behavior of a set of real-time tasks with various characteristics when running on a multi-processor platform in presence of a number of real-time scheduling policies. PARTSim extends the functionality of RTSim by introducing support for power-aware embedded platforms exhibiting frequency scaling and specific architectural patterns like the ARM big.LITTLE and DynamIQ ones. Experimental results that compare simulated data against execution profiles collected on real platforms show a simulation error under 10 % for both execution time and power consumption at 90th percentile when simulating the effects of DVFS.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Real-time system architecture*; • **Software and its engineering** → *Scheduling*; **Power management**; • **Hardware** → **Temperature simulation and estimation**; **Power estimation and optimization**;

KEYWORDS

Real-Time Systems, Scheduling, Simulation, DVFS, Heterogeneous Embedded Systems, ARM big.LITTLE

ACM Reference Format:
Gabriele Ara, Tommaso Cucinotta, and Agostino Mascitti. 2022. Simulating Execution Time and Power Consumption of Real-Time Tasks on Embedded Platforms . In *Proceedings of ACM SAC Conference (SAC'22).* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3477314.3507030

1 INTRODUCTION

Engineering real-time and distributed systems has always been a cumbersome task due to the need for software mechanisms to ensure *predictable execution*, despite the presence of unpredictable elements in the underlying hardware, primarily because of the need for higher and higher performance to support applications with higher and higher complexity, to fulfill increasingly demanding requirements by users of real-time systems [23].

However, the last decade has also seen increasing attention dedicated to *power efficiency*, with hardware being enriched with more and more power management features, resulting in platforms with high peak computational capabilities and low power consumption under regular workloads within normal-usage scenarios. One of the mechanisms that support this model is Dynamic Voltage and Frequency Scaling (DVFS) [14], often coupled with multiple "deep-idle" states of the processor characterized by different trade-offs between power consumption while idle vs. wake-up latency when reacting to external events [10]. Additionally, the so-called "turbo" technologies [15] add the capability to opportunistically spike up the frequency whenever allowed by the processor's thermal and current absorption conditions as continuously monitored in hardware. In this context, heterogeneous architectures [13] leverage different processor types to dedicate to different tasks to reach particularly energy-efficient operating points for different workloads. Indeed, heterogeneous single-Instruction Set Architecture (ISA) multi-core architectures have been introduced, like ARM big.LITTLE, which now dominate the market smartphones and tablet markets, introducing the distinction between "big" and "LITTLE" core islands [12]. The two core types exhibit different trade-offs between computing speed and power efficiency while maintaining cross-compatibility by supporting the same ISA. The DynamIQ architecture further extends this concept with the possibility of even more core islands.

A common practice for soft real-time system development is the one to disable many hardware capabilities that lead to non-predictable execution of real-time tasks; however, energy efficiency requirements are first-class citizens in the design of modern heterogeneous computing platforms, leading to the impracticality to disable energy management features like DVFS altogether [3, 23].

Research on real-time systems very often relies on non-functional simulators (see section 2), which can simulate the timing and scheduling of tasks on a platform, under various conditions, including worst-case scenarios, to perform experimental comparisons among the timing properties of various solutions. However, these platforms rarely include energy-awareness features (i.e., DVFS). Even the ones that do include these features typically rely on simplified theoretical models that may struggle to reflect a real platform's behavior accurately. For example, when modeling DVFS, power consumption is often assumed to vary according to the square or cube of the processor frequency, but nowadays, operating system

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.

SAC'22, April 25 –April 29, 2022, Brno, Czech Republic

ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/10.1145/3477314.3507030

drivers can also change the voltage of the processor when changing frequency, with a non-negligible impact on these simplified models. Additionally, (worst-case) execution times of tasks are assumed to vary with the frequency according to simple algebraic equations, often without distinguishing between CPU-intensive and data-intensive workloads.

Another aspect that is crucial for accurate simulations of the behavior of modern embedded platforms, is the one of correct modeling of the *thermal properties* of such platforms under a variety of workload and environmental conditions. Many modern CPUs are affected by excessive heating problems when operating at peak performance conditions for a prolonged amount of time. This means that these platforms often rely on kernel-level protection mechanisms that kick in when excessive heating might lead to physical damages of the hardware. These may vary from simple ventilation activation or speed increase, to temporarily limiting the maximum frequency of affected CPU(s). These phenomena cannot be ignored nowadays, with a growing number of embedded platforms deployed in very different environmental conditions, where it is expected that these protection mechanisms will be triggered quite often (i.e., small form factor or fan-less devices, as typical in smartphones and tablets). Therefore, it is essential to support the variation of the CPU(s) temperature over time from a soft real-time perspective, to enable the emulation of these protection mechanisms, so to enhance realism in the simulated platform. For example, this modeling feature is key to effectively experiment with thermal-aware scheduling logics capable of preventing or controlling the occurrence of overheating conditions, e.g., as in [22, 24].

Therefore, to improve realism in the validation of real-time systems research, it has become increasingly important to have open real-time simulation frameworks with proper support for various power management and thermal management features typical of embedded platforms, which is the subject of the present paper.

1.1 Contributions

This paper presents PARTSim, a novel open-source tool for simulating embedded real-time systems running on multi-processor power-aware platforms, realized as an extension of the well-known RTSim open-source simulator [20]. The new tool enables users to conveniently simulate the timing behavior of real-time software running on multi-core embedded platforms with power-aware features like DVFS, aiming at an accurate simulation of power consumption and temperature of the various cores throughout the simulations. While extensions to RTSim supporting the simulation of power consumption and task speed-up due to DVFS already exist [2], this paper describes a more generic power/timing simulation approach, introducing a modular mechanism allowing for model-based, as well as table-based emulation of the impact of DVFS on task speed-up, power consumption and thermal behavior of the platform. For this purpose, we developed another open-source tool, PARTProf, that collects various metrics relative to the execution of real-time tasks on embedded platforms running Linux and generates a suitable model that PARTSim will use to simulate each platform accurately. Simulator internals related to big.LITTLE have been generalized to support arbitrary sets of CPU cores per CPU island, generic DynamIQ architectures, and the automatic generation of a complete system model based on configuration files. Finally, we show the soundness of this methodology by comparing simulated data and execution profiles obtained from the corresponding real platform.

2 RELATED WORK

The literature offers many real-time task simulators, including real-time scheduling analysis tools and a graphical user interface (GUI) to visualize the tasks' temporal behavior, easing a visual inspection of the effect of various task placement and scheduling policies.

Among open-source real-time simulators, we recall MAST [8, 9], which supports scheduling strategies like Earliest Deadline First (EDF), Rate Monotonic (RM), and their hierarchical composition, and various servers (e.g., Sporadic and Polling servers), allows to perform worst-case schedulability analysis and supports many shared resources protocols, like Priority Inheritance Protocol (PIP), Priority Ceiling Protocol (PCP), and Stack Resource Policy (SRP). The MAST tool suite's heart lies in the MAST model, similar to the one defined in the MARTE UML profile for embedded real-time systems [19]. Unfortunately, MAST does not support energy awareness. Cheddar [25] is a simulator that implements well-known scheduling algorithms like EDF and RM and shared resources protocols (PCP and PIP). It supports multi-processor systems, but it does not consider energy-related issues and does not directly support ARM big.LITTLE architectures, while the simulator proposed in this paper actively supports it with a focus on the recent DynamIQ extensions. Similarly, SIMSO [6] supports multi-processor architectures and various schedulers, but it is not energy aware.

Moving on to simulators that consider the energy-related features of the underlying hardware, Simulation Tool for Energy Efficient Real Time Scheduling and Analysis (STREAM) [7] is not open source and shifts the focus to energy-efficient scheduling while supporting multi-processor architectures. It implements several scheduling algorithms, synthetic task set generation, modules for performance analysis, and the generation of execution traces. Likewise, SPARTS [18] is another simulator designed to simulate power-aware scheduling strategies; its extensible design considers various task properties, scheduling algorithms, and hardware models for a wide variety of applications. YARTISS [4, 5] evaluates scheduling algorithms considering overheads and effects due to the target hardware platform; in this simulator, energy consumption is an optional parameter associated with each task. Finally, the EWARDS framework [1] explores at design time the performance and energy capabilities of modern Massively Parallel Multi-Processors System-on-Chip (MP2SoC) architectures. It combines power management techniques with clustering-based scheduling and extends MARTE with power aspects typical of MP2SoC systems, with the ultimate goal of saving energy at runtime. This framework demonstrates the validity of Model-Driven Engineering (MDE) techniques to the design and implementation of energy-aware scheduling techniques. In general, no real support to the ARM big.LITTLE architecture is available in these simulators.

In addition to open-source and research-focused simulators, commercial tools are also available on the market, like the well-known RapiTime[1] and MATLAB. These tools implement a comprehensive set of features, including performing on-target software verification,

[1]RapiTime home page: `www.rapitasystems.com/products/rapitime`

structural coverage analysis, simplifying the software verification process, and producing evidence for specific code certifications. MATLAB has been used together with partitioning algorithms designed on purpose in [11] and with Integer Linear Programming (ILP) methods (without real-time constraints) in [26] to partition the task set for many-core hardware with cluster structures, like ARM big.LITTLE. However, these tools are closed-source, and the degree of customization that users can make is limited compared with the open-source solution that we present in this paper.

A final notable example among open-source real-time simulators is RTSim [20], a simulation library that focuses on the timing behavior of real-time applications, implementing multiple scheduling algorithms, resource sharing protocols (Priority Inheritance), resource reservations (like the CBS server), supporting multi-core system architectures (with partitioned and global strategies). It has partial support for power-aware systems, thanks to an extension that introduces an ARM big.LITTLE energy model fitted on data collected on an existing platform [2]. This work also highlighted how the CPU type in use (big vs. LITTLE), its operating frequency, and the type of workload running on it (e.g., CPU vs. memory-bound) are all contributing factors that determine both how the power consumption and the execution time scale when changing the system configuration. This extension has been recently used to evaluate the performance of energy-aware task placement strategies for real-time systems running on ARM big.LITTLE architectures [16, 17]. However, the RTSim energy-aware support for just one hardware architecture limits its usefulness. In this work, we extend RTSim to provide a more flexible approach to power-aware real-time systems simulation, introducing the capability to support various hardware architectures, focusing on embedded devices.

3 PROPOSED APPROACH

In this section, we illustrate the approach we followed to address the problem of realistically simulating real-time tasks on embedded platforms with DVFS capabilities. In principle, we can divide our approach into two main phases: (**i**) data collection and elaboration and (**ii**) system simulation.

In the first phase, a custom profiling suite is deployed on the target platform, on which it performs a set of automated profiling runs, under various DVFS settings, collecting several metrics over time: execution time of the tasks, temperature of the cores and power consumption of the platform. Then, another software component, typically running on a general-purpose machine, can post-process the collected data, calculating statistics and model parameters that are converted into a suitable format to be used when simulating the corresponding platform. This data is used in the second phase, along with a system description provided by the user, to properly simulate the timing behavior of real-time tasks on the designated platform, along with its estimated power consumption and thermal behavior throughout the simulation.

The software is open-source, and it is freely available, under a GPLv3 license, at: `https://gitlab.retis.santannapisa.it/parts`. Researchers or practitioners can conveniently extend them, should they need to provide support for additional embedded platforms or experiment with novel energy-aware schedulers.

Algorithm 1: Algorithm used to profile an embedded platform.

```
foreach I ∈ Islands do
    for n ← 1 to #CPUs(I) do
        foreach F ∈ Freqs(I) do
            set frequency of island I to F ;
            foreach W ∈ Workloads do
                for r ← 1 to N_rep do
                    for i ← 1 to n do
                        start an instance of W pinned
                            on the i-th core of island I ;
                    start the data collection application ;
```

3.1 Data Collection

This section presents PARTProf, the software we realized to profile and analyze the execution of real-time tasks on embedded platforms. This software comprises two interacting components: the *host* and the *embedded* component. The first runs on a workstation machine, while the latter is automatically deployed and runs on the embedded platform under analysis. Both components can easily be deployed on Linux hosts and embedded devices running Linux[2].

PARTProf automatically performs several tests on the target platform; each test consists of running a specific type of workload on the target machine and collecting key information to simulate the execution of similar real-time tasks on the given platform. PARTProf provides a set of tasks that represent typical workload types of both CPU and data-intensive applications: (**i**) *hash*: SHA-256 checksum algorithm. (**ii**) *encrypt/decrypt*: triple DES encryption and decryption algorithm. (**iii**) *gzip*: compression algorithm, run with various compression levels, from the fastest (1) to the slowest (9). (**iv**) *cache stress app*: application purposely developed to generate a configurable rate of cache misses to simulate generic workload types, from always-hit (cache saver) to never-hit (cache killer): it accesses elements in an array bigger than the size of the Last Level Cache (LLC), generating one access with a displacement bigger than the cache line size each time a miss should occur. (**v**) *idle*: no task is executed; the system switches to the clock-gating idle state[3] [21].

As it will be shown in section 4, these applications exhibit diverse behaviors both with respect to DVFS and to multi-core scalability. Even among "real" application workloads (so excluding the *cache stress applications*), we registered a difference of more than 20 times between maximum (*gzip*) and minimum (*encrypt/decrypt*) observed cache misses, showing that the selected applications set does represent both CPU and memory-bound applications alike.

Users can easily customize the set of workloads tested on each platform by editing a couple of configuration files. This allows for gathering data that more tightly represents workloads of interest. The input/output files required by some of the aforementioned data-intensive workloads are randomly generated at the beginning of each experiment and stored in a *ramfs* partition, limiting interactions with disks and SD card devices to the bare minimum (storage access is not simulated).

[2] At the moment, we successfully deployed the *embedded* component of PARTProf on two Linux distributions, Ubuntu and PetaLinux.

[3] Most multi-core embedded platforms can access deep idle states only when all CPUs on the same frequency island are idle; simulating the power consumption under various idle states is among the planned future extensions.

The profiling of a platform is performed according to Algorithm 1, which iterates all possible system configurations and starts a variable number n of tasks (at most one per CPU and only on one frequency island at a time). Profiling multiple tasks on different cores simultaneously is particularly relevant and an essential improvement over similar related work [2] because it allows for a more accurate representation of the system's behavior when multiple tasks are running concurrently. Each test can be repeated automatically for a configurable number of times (N_{rep}). The software collects each task execution time for each test run and runs a data collection application concurrently to the profiled tasks[4].

The data collection application periodically samples a set of key metrics useful for the simulation of real-time tasks; the set of supported metrics varies from platform to platform because different devices may expose different kinds of sensors for the same metric. In general, metrics collected during each run include power consumption, the temperature of the CPU, and actual CPU frequency, which may differ from the one selected by PARTProf due to thermal issues. Sections 4 and 5 provide more details about the practical use of PARTProf and discuss potential issues that should be addressed to improve the performance of this tool.

The design of the data collection application is modular and easily extensible to expand its support to more platforms in the future. Platforms that do not ship with internal sensors for power consumption or other useful metrics (e.g., CPU temperature) can also be profiled using external power meters attached to the embedded platform itself. Indeed, we profiled a Raspberry Pi 4 Model B using this approach in section 4.1.

After completing the first phase, all data samples and logs are collected to be post-processed by the *host* application component. The *host* application calculates statistics on the collected data and generates a set of CSV format tables, to be used as input to PARTSim or manually inspected.

3.2 Simulation

PARTSim is the simulator we developed to support DVFS-capable platforms as an extension to RTSim. Partial support for DVFS in RTSim has already been introduced in [2], which implements power consumption and execution time models. However, these are limited to ARM big.LITTLE platforms with exactly two islands. For example, the recent ARM DynamIQ architecture could not be supported.

The original RTSim [20] is an extensible, event-based, open-source library written in C++ that allows for simulating the timing behavior of real-time systems. Typically, RTSim is used to simulate worst-case scheduling scenarios for real-time task sets using a given scheduling policy to check whether the resulting schedule is feasible or not (i.e., whether any deadline miss occurs). Each RTSim simulation can be traced, and users can inspect the resulting schedule or visualize the events that occurred during the simulation.

3.2.1 Power Consumption and Execution Time Models in RTSim. Each simulated CPU in RTSim must be associated with its own *CPU Model*, which implements the estimation of both power and execution time for any task running in the system, adjusting automatically task duration and power consumption depending on the DVFS configuration and selected CPU. The simulator keeps track of activation/termination of tasks and the per-CPU power consumption, providing an estimate of the final overall energy consumption throughout a simulation. For this purpose, RTSim already implements two models: a simple linear model and a task model based on [2], specific for ARM big.LITTLE platforms.

The original RTSim implementation provides a simple generic model that simulates the variation of the power consumption and execution time of real-time tasks with the frequency. This first model does not distinguish among different workload types that could run on the system; it merely estimates the power consumption P and the execution time C of a real-time task depending on the current Operating Performance Point (OPP), uniquely identified by a pair of voltage V and frequency f values, as: $P(V, f) = V^2 \cdot f; \quad C(f) = \frac{C_{max} \cdot f_{max}}{f}$, where C_{max} represents the expected execution time of the task at the maximum frequency f_{max}.

RTSim was later extended in [2], which introduced a more accurate model for ARM big.LITTLE platforms, taking into account the type of workload running on each CPU at any time. The power consumption of a CPU island is computed as: $P = \sum_{i=1}^{N_{CPU}} P_{W_i}(f)$, where W_i represents the workload currently in execution on the i-th core, or *idle*[5], and $P_{W_i}(f_i)$ is its associated *single-CPU* power consumption under a frequency f for the island. The $P_{W_i}(f)$ function is approximated with a per-workload analytical model, using four real parameters fitted on the measured data on an ODROID-XU3 platform. For a detailed description of the model, please refer to [2]. This model has been used for example to compare different power-aware real-time schedulers on ARM big.LITTLE platforms [16].

3.2.2 PARTSim Data-Driven Simulation Model. PARTSim extends some of the concepts already present in RTSim for big.LITTLE platforms to generic or heterogeneous multi-core platforms and introduces new mechanisms that ease the set-up of a multi-core platform to simulate real-time power-aware scheduling policies.

PARTSim introduces an additional model to the ones described above that does not make any assumption on the relationship between execution time and power consumption of the platforms with respect to the CPU frequency. This *data-driven model* estimates variations of power consumption and execution times directly "replaying" the real data as made available by the PARTProf output. The tables produced by PARTProf for each platform provide these estimations for each *core type*, *frequency* and *workload type*; the combination of these three values precisely identifies the correct data to use to estimate variations of execution times and power consumption due to each type of task when running on each CPU type. Also, PARTSim is capable of interpolating the data provided by PARTProf, in case some points are skipped (useful to avoid measuring all the frequencies, skipping some of them to reduce the profiling time for a platform).

For a multi-core CPU island, PARTSim uses the following model to calculate the power consumption of an island with N_{CPU} cores:

[4]Preferably, the data collection application runs on a separate frequency island; on platforms that only have one frequency island, preference goes to cores not involved in the current test run, if any.

[5]Internally to RTSim, the *idle* condition of a CPU is treated similarly to when the CPU is running a specific task, i.e., it is associated with an island-and-frequency specific power consumption.

$$P = P_{idle} + \sum_{i=1}^{N_{CPU}} (P_{W_i} - P_{idle}). \quad (1)$$

where P_{W_i} is the power consumption measured by PARTProf for the whole island when running a benchmark of the same type as the real-time task being simulated on core i of the island, and P_{idle} is the measured power consumption with all cores idle. This reflects the fact that most embedded platforms lack per-core power monitors and provide power information for each island, or do not have any embedded power monitor. Examples of these platforms are shown in section 4. Note that, for systems with multiple core islands like big.LITTLE platforms, values in Eq.(1) need to be summed up together to estimate the power consumption of the entire platform.

Furthermore, differently from RTSim, PARTSim uses a disjoint declaration of power consumption and execution time models as functions of the frequency: for any island, different estimation models for power and execution time can be selected, providing users the capability to mix different models.

3.2.3 Multicore CPU Thermal Modeling. In PARTSim, we are working on a new high-level model that might predict the thermal evolution of a multicore CPU system over time, starting from a limited subset of parameters that can be estimated using PARTProf.

This model is a very simplified view of a thermal system composed of n CPUs where each CPU i has associated: a time-varying temperature $T_i(t)$; a thermal capacitance C_i; a coefficient α_i regulating heat transfers towards the environment at temperature T_e, proportional to the difference $T_e - T_i(t)$ (as per Newton's cooling law); coefficients $\beta_{i,j} \equiv \beta_{j,i}$ regulating heat transfers towards any other CPU $j \neq i$, proportional to the difference $T_j(t) - T_i(t)$; a point-like heat pump that generates heat $P_{w,i}$ when executing a specific workload w (or even when idle, as a CPU is not entirely turned off when idle). A simplifying assumption is the one of CPUs all considered uniform ($C_i = C$), and with similar heat transfer capabilities with the environment ($\alpha_i \equiv \alpha$), which has a constant temperature T_e. This would result in, e.g., a dual-core system modelled as:

$$\begin{bmatrix} \dot{T}_1(t) \\ \dot{T}_2(t) \end{bmatrix} = \begin{bmatrix} -\frac{\alpha+\beta}{C} & \frac{\beta}{C} \\ \frac{\beta}{C} & -\frac{\alpha+\beta}{C} \end{bmatrix} \begin{bmatrix} T_1(t) \\ T_2(t) \end{bmatrix} + \frac{1}{C} \begin{bmatrix} P_{w1} + \alpha T_e \\ P_{w2} + \alpha T_e \end{bmatrix} \quad (2)$$

For n cores this can easily be generalized introducing the vector of temperatures $T(t)$, the symmetric heat-transfer matrix A and the heat-pump vector b, as: $\dot{T}(t) = AT(t) + b$, which has the general solution: $T(t) = \left(\int_{t_0}^{t} e^{A(t-\tau)} d\tau\right) b + e^{A(t-t_0)} T(t_0)$.

Figure 1 exemplifies the thermal model in use by highlighting all heat transfers considered between each component. This model can easily be solved numerically to simulate the evolution over time of the thermodynamic system given its initial conditions (i.e., the initial temperature $T_i(t_0)$ for each CPU in the system) and the external stimuli (i.e., the power-pump associated to each CPU P_i due to its instantaneously running workload in the current DVFS settings, as well as the environment temperature T_e). The parameters of this model remain constant as long as the CPUs keep running the same workloads. However, as the simulated OS scheduler runs different tasks or puts cores in idle mode, or changes the DVFS settings of the cores, then the model can be updated, using the current value of the temperature vector $T(t)$ as the new initial temperature vector $T(t_0)$ in the above model with updated parameters. This can be repeated, simulating the thermal behavior of the platform, adapting the parameters each time a configuration change happens. This enables also simulation of active thermal protection measures activated by the OS, like activating (or speeding up) fans, by changing the coefficients modeling heat transfer among parts of the system, switching among a number of models with parameters fitted under the different configuration states of the active cooling subsystem.

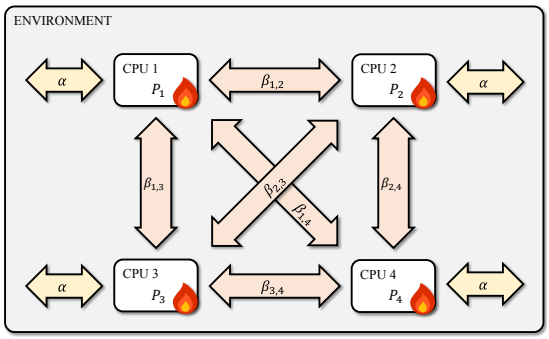


Figure 1: Graphical representation of all heat transfers across CPUs on a 4-core CPU island and the environment in our thermal model.

Since the external stimuli can be estimated using PARTProf, the thermal properties of the system (i.e., the heat-transfer coefficients and thermal capacitances) must be estimated numerically from experimental data. PARTProf already collects all necessary data from the embedded platforms, including the evolution of the system temperature over time for each workload and DVFS condition. This data can be used to fit the model's unknown parameters, using common tools such as the SciPy Optimize library[6], so that the simulated platform closely resembles the corresponding real one. For that purpose, the steps described in algorithm 1 are still valid, with the additional step of profiling each workload on each of the CPUs for each island: that way, the heat transfer coefficients between the different CPUs can be estimated.

The preliminary experiment in Figure 2 shows that the ideal model described above, once its parameters have been fitted on experimental data, can be used to estimate the thermal evolution of an entire CPU island quite effectively. PARTSim already implements the described thermal model, while integration in post-processing tools inside PARTProf is still underway. We plan to use it to simulate and test thermal-aware real-time scheduling strategies while realistically emulating thermal protection mechanisms, for further enhancing the accuracy of the power/timing simulations.

3.2.4 Implementation Details and Other Improvements to RTSim. Simulating a multi-core platform in RTSim requires the user to instantiate and interconnect several data structures, each representing a different aspect of the hardware/software components being simulated. Among the most common components we find the OS kernel

[6]See also `https://docs.scipy.org/doc/scipy/reference/optimize.html`

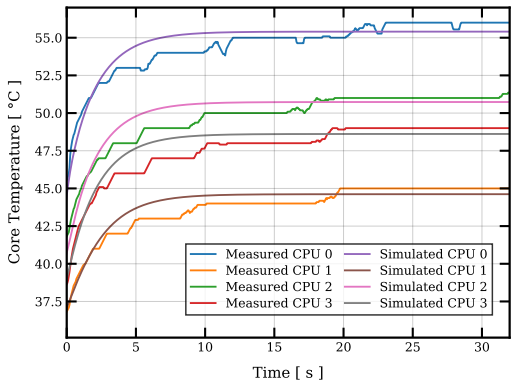


Figure 2: Comparison between an evolution of the temperature of 4 Cortex-A15 (big) CPU cores and the simulated evolution by our work in progress model for the same scenario.

(`MRTKernel`), the real-time scheduler in use (`Scheduler`), the processing hierarchy of the system (composed by a set of `Island` and `CPU` instances). Furthermore, to simulate a real-time taskset a number of `Task` instances must be declared, depending on the task model in use (e.g., periodic, aperiodic). Finally, each `CPU` must be associated with a corresponding `CPUModel`, which defines how task durations and power consumption should scale depending on the DVFS configuration. With so many components to instantiate, RTSim requires many set-up operations.

PARTSim generalizes the implementation of the `Island` and `CPU` components, enabling the instantiation of any generic and possibly heterogeneous multi-core platform. Furthermore, it introduces a novel platform generation paradigm that reduces simulation initialization to a single operation. We implemented a novel class called `System`, which automatically initializes all simulation components, starting from a single YAML configuration file, which the `System` class parses upon initialization. The format of these configuration files is straightforward, yet they provide significant flexibility when declaring a platform building blocks[7]. The configuration file enables users to specify all system parameters, including the list of CPU islands available, number of CPU cores (which is not limited anymore to a fixed number for all CPU islands), the scheduler to use, the OPPs available for each island, and how to model the power consumption and variation of the execution time with the CPU frequency. With this new way of declaring system resources, initializing a simulation in PARTSim is reduced to the following:

```
System sys("system_description.yaml");
```

4 EXPERIMENTAL RESULTS

To test our approach, we collected data from three embedded devices with a heterogeneous set of features and architectures: (**i**) an ODROID-XU3 board, equipped with a Samsung Exynos 5422 SoC, an ARM platform featuring eight CPUs in a big.LITTLE configuration, with four Cortex-A7 (LITTLE) and four Cortex-A15 (big); for our tests, we used the official ODROID Linux kernel version 4.14.180. (**ii**) a Xilinx Zynq UltraScale+ ZCU102 board, equipped with an XCZU9EG-2FFVB1156 MPSoC, featuring four ARM Cortex-A53 CPUs; for our tests, we used the official PetaLinux 2020.2 distribution, with Linux kernel version 5.4.0. (**iii**) a Raspberry Pi 4 Model B board, equipped with a Broadcom BCM2711B0, which includes four Cortex-A72 CPUs; for our tests, we used the official Raspberry Pi OS, a Debian-based distribution, with Linux kernel version 5.4.83.

4.1 Collecting Data With PARTProf

To gather data for PARTSim on each test platform, we used PARTProf as described in section 3.1. Given the high number of combinations of testing conditions (e.g., number of concurrent tasks, CPU frequency, and counting), this section illustrates only the most relevant results.

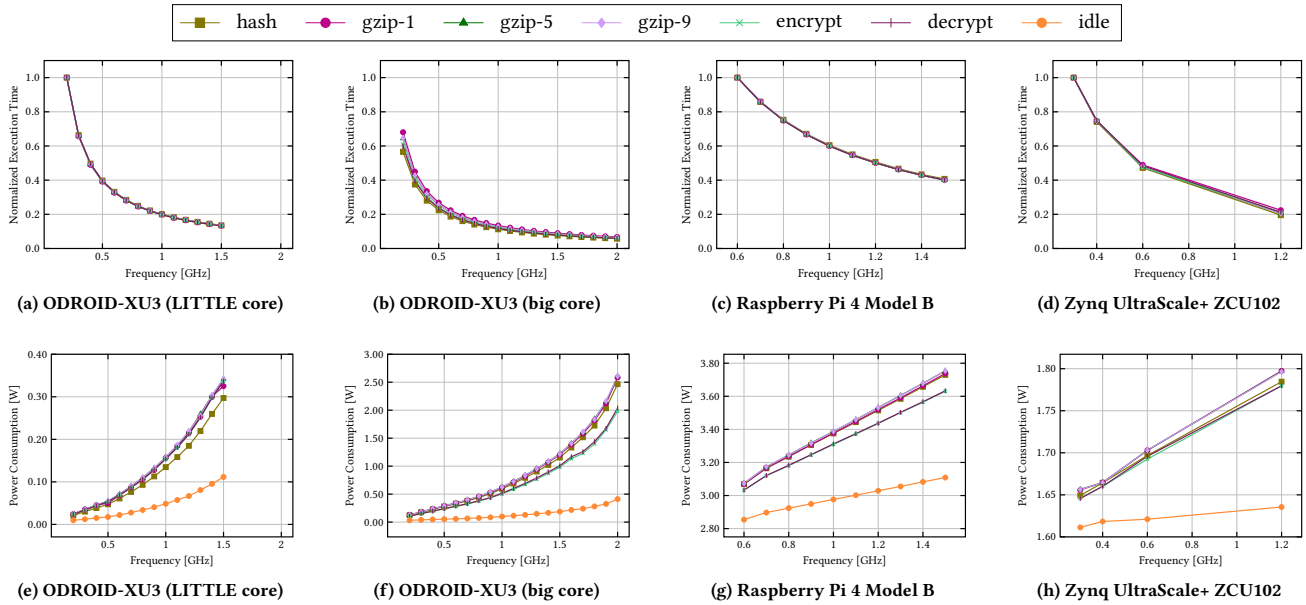
Notice that the three platforms differ for the mechanism used to gather the power consumption of the CPU during each test run: (**i**) the ODROID has a TI INA231 power meter embedded and directly connected to the power lines for each of its CPU islands, so we read directly the power consumption as reported by these meters; (**ii**) the Xilinx ZCU102 has a TI INA226 power meter connected to the power lines of the Processing System (PS) and Programmable Logic (PL) part of the board, so we settled with measuring the power consumption of the PS part, which includes the onboard CPUs; (**iii**) the Raspberry has no embedded power meter, so we used an external ODROID Smart Power Meter v3.0 connected to the entire board power supply.

Figure 3 shows how execution time and power consumption of several tasks vary when changing the CPU frequency on each platform and core type, using only one CPU core at a time. In this figure, we show only a subset of the tasks described in section 3.1, excluding tasks specifically developed to stress the LLC of each embedded platform (*cache stress app*). As we can see, when isolated, these workloads exhibit similar timing and power behaviors to when running on the same platform/core type. The difference between the nearly quadratic increase in power consumption with frequency shown for the ODROID and the linear behavior shown by the other two boards is most likely due to different implementations of DVFS for different hardware platforms.

When we increase the count of concurrent tasks, some tasks tend to deviate from the expected behavior. Figure 4 shows how power consumption and execution time of these tasks is affected by the number of parallel tasks of the same workload type running while keeping the CPU frequency fixed[8]. In an ideal scenario, the execution time should be unaffected by the number of parallel tasks (each working on separate data), while power consumption should increase linearly as the number of active cores increases. In reality, the situation is more complicated than that. While it is true that each concurrent application works on its private data, they all access main memory to read input and write back output data. Each CPU has its private L1 cache on each tested platform, but all CPUs share the LLC and main memory access. For this reason, more memory-intensive applications (and in general applications that generate a higher number of L1 and LLC cache misses) experience blocking

[7] For the sake of brevity, we do not include an example of a system configuration file and we remind to PARTSim documentation for further details.

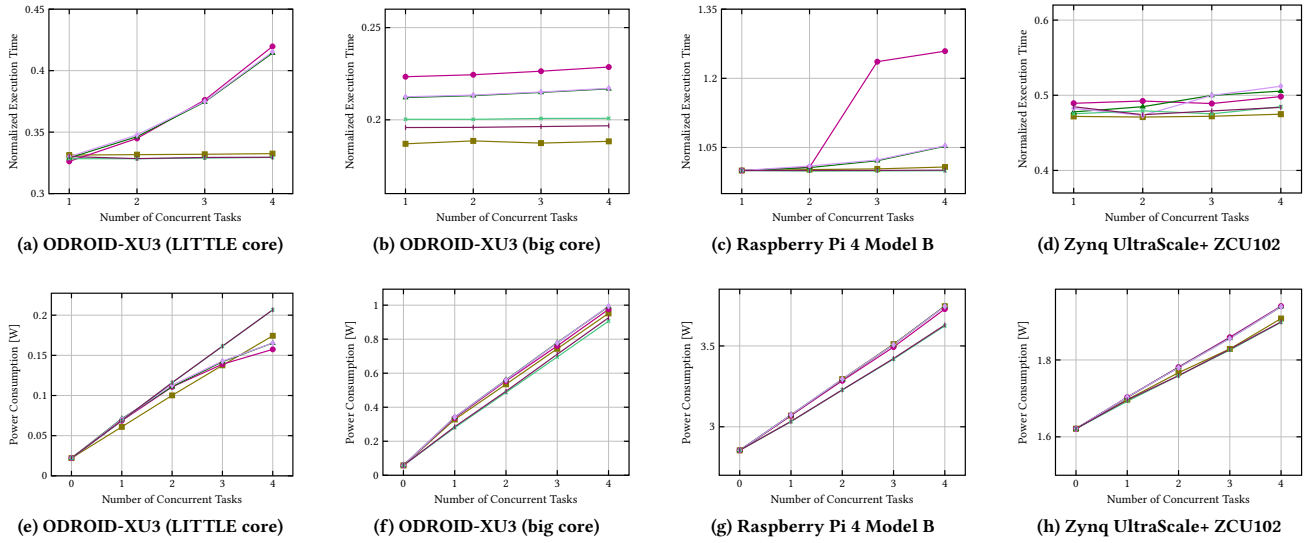
[8] Similar results were obtained for other frequencies.



(a) ODROID-XU3 (LITTLE core) (b) ODROID-XU3 (big core) (c) Raspberry Pi 4 Model B (d) Zynq UltraScale+ ZCU102

(e) ODROID-XU3 (LITTLE core) (f) ODROID-XU3 (big core) (g) Raspberry Pi 4 Model B (h) Zynq UltraScale+ ZCU102

Figure 3: Variation of tasks execution time — (a), (b), (c), (d) — and power consumption — (e), (f), (g), (h) — when varying operating CPU frequency on various embedded platforms and core types. All execution times are normalized with respect to the longest execution time for each workload type, usually registered for the smallest frequency of the least powerful core on each platform. Notice that the methodology applied for power consumption estimation varies from platform to platform.

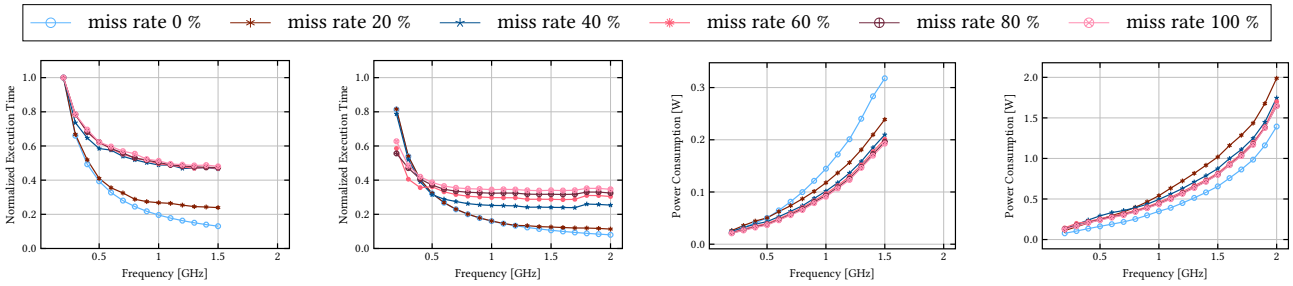


(a) ODROID-XU3 (LITTLE core) (b) ODROID-XU3 (big core) (c) Raspberry Pi 4 Model B (d) Zynq UltraScale+ ZCU102

(e) ODROID-XU3 (LITTLE core) (f) ODROID-XU3 (big core) (g) Raspberry Pi 4 Model B (h) Zynq UltraScale+ ZCU102

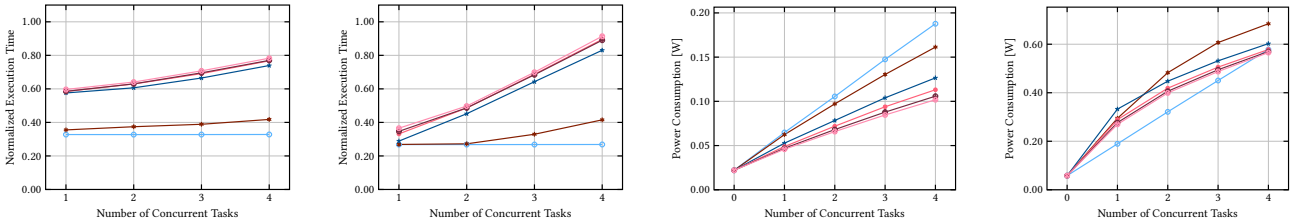
Figure 4: Variation of tasks execution time — (a), (b), (c), (d) — and power consumption — (e), (f), (g), (h) — when varying the number of tasks of the same workload type running on various embedded platforms and core types. Each concurrent task is pinned to a separate core, with frequency fixed at 600 MHz for all platforms. All execution times are normalized with respect to the longest execution time for each workload type, registered for the smallest frequency of the least powerful core on each platform. The value of "0" running tasks indicates the consumption of the target platform/island when no task is running (idle). Notice that the methodology applied for power consumption estimation varies from platform to platform.

time due to contention when accessing data. The final result is that, when increasing the number of concurrent tasks, the execution time is not constant while the average power consumption exhibits a less-than-linear behavior. When the running task is blocked, the CPU reverts to its idle power consumption rate if no other task is



(a) ODROID-XU3 (LITTLE core) (b) ODROID-XU3 (big core) (c) ODROID-XU3 (LITTLE core) (d) ODROID-XU3 (big core)

Figure 5: Variation of tasks execution time — (a), (b) — and power consumption — (c), (d) — when varying operating CPU frequency on various embedded platforms and core types. All execution times are normalized with respect to the longest execution time for each workload type, usually registered for the smallest frequency of the least powerful core on each platform. Notice that the methodology applied for power consumption estimation varies from platform to platform.



(a) ODROID-XU3 (LITTLE core) (b) ODROID-XU3 (big core) (c) ODROID-XU3 (LITTLE core) (d) ODROID-XU3 (big core)

Figure 6: Variation of tasks execution time — (a), (b) — and power consumption — (c), (d) — when varying the number of parallel tasks execution of the same workload type. Frequency for all platforms is fixed to 600 MHz. Each concurrent task is pinned to a separate core on the target platform/island. All execution times are normalized with respect to the longest execution time for each workload type, usually registered for the smallest frequency of the least powerful core on each platform. Notice that the methodology applied for power consumption estimation varies from platform to platform.

ready. This behavior is particularly evident on LITTLE cores of the ODROID and the Raspberry platforms.

To show the tight relationship between memory access patterns and variations of both power consumption and execution time, figs. 5 and 6 show experimental data collected using the custom cache stressing application that we developed for PARTProf, varying the cache miss rate from 0 % (cache saver) to 100 % (cache killer). As we increase the cache miss rate, the behavior of these applications differs even more from the ones shown before, both in the frequency domain and depending on the number of concurrent tasks. For high cache miss rates, improvements in task execution times are less significant increasing the CPU frequency (due to the increased impact of memory accesses on the instruction execution times), and the adverse effects introduced by blocking times discussed above become significantly accentuated.

4.2 PARTSim Accuracy on Multi-Cores

In this section, we discuss the PARTSim approach to deal with accuracy of the simulation of the behavior of real-time applications when multiple heterogeneous applications are running concurrently on a multi-core architecture. The problem is that the theoretical model in eq. (1) does not match exactly with the data gathered by PARTProf, as shown in the above pictures. In our discussion, we ignore the most extreme cases shown in figs. 5 and 6; from the data gathered during our experimentation, most "real" applications exhibit relatively low levels of cache misses and memory contention at most frequencies. This does not mean that all our testing applications are CPU-bound (see section 3). Rather, the most extreme situations shown in those figures are too artificial to be considered close to the behavior of real-world applications. We will also leave out cases in which the target platform activates thermal throttling techniques; the discussion of those cases is postponed for when PARTSim will be fully integrated with the thermal model described in section 3.2.3.

As indicated in section 3.2, PARTSim uses the data collected and elaborated by PARTProf to estimate both execution times and power consumption, by looking up actual values collected on the target platform. The three parameters used to perform this lookup are the core type (if on an heterogeneous platform, like big.LITTLE), the task frequency and the workload type. However, it is not reasonable to produce lookup tables for all kinds of configurations and task combinations on each platform; for this reason, we devised different approaches to "adjust" collected values to simulate certain factors influencing tasks execution times and power consumption on multi-core systems. We will discuss these approaches as we evaluate their performance. Each of the approaches described in this section can be selected to produce PARTSim simulation tables automatically.

Table 1: Error on execution times for homogeneous task sets ("real" workloads only).

Approach	Absolute Percentage Error [%]					
	Overestimation		Underestimation		Both	
	90th	Max	90th	Max	90th	Max
Single	0.03	10.27	8.02	47.40	6.66	47.40
Average	3.66	41.49	2.77	30.99	9.88	41.49
Maximum	16.05	90.10	0.00	0.00	16.05	90.10

To evaluate the various errors introduced by our approximations, we compared the values obtained in simulation against the original experimental data collected on all platforms described in section 4.1, including all possible configurations of "real" homogeneous task sets. Tables 1 and 2 report the simulation error in PARTSim when estimating task execution times and power consumption, respectively; both tables indicate with *90th* the simulation error at the 90th percentile and with *Max* the maximum error obtained in all configurations.

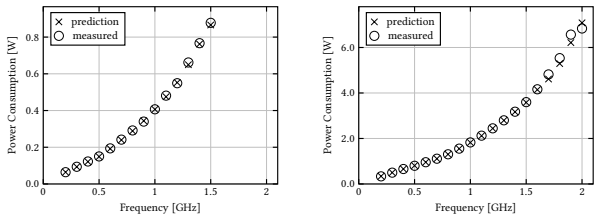
4.2.1 Simulating Tasks Execution Time. The timing behavior of "real" (for fixed frequency/core type) tasks does not deviate much from the ideal constant value. For this purpose, using the execution time of a single task running in isolation can be pretty accurate: for multiple parallel tasks, the error remains mostly limited (if we neglect corner cases and effects of thermal throttling).

Table 1 shows a simulation error (i.e., comparing simulated results against profiled execution times) lower than 10 % in most cases when using either the value collected when running a single task of the given workload running in isolation (referred to as *Single* in the table) or the average of all the values collected in the same core type when increasing the number of parallel tasks (indicated as *Average*). Finally, the *Maximum* approach uses only the maximum value measured for any number of concurrent tasks to simulate tasks execution time. Table 1 divides errors in two categories: *Overestimation* errors indicate that the estimated execution time is greater than the one obtained from experimental data, while *Underestimation* errors indicate the opposite; finally, the *Both* columns considers both kinds of errors. The rationale behind this split is that it is typical in real-time systems to consider more harmful to underestimate the execution time of a task rather than overestimate it, because the latter could result in wrong analysis that would later lead real tasks to miss deadlines.

From the results we can conclude that: the *Single* approach rarely overestimates execution times, because typically tasks take less time to run when they are alone in the system and longer when there are other tasks in parallel on the same CPU island; the *Average* approach, treats both kinds of error equally, at the cost of an increased simulation error when only a single task is running on the system with respect to the previous approach; the *Max* approach, by definition, never underestimates task executions when compared against the same data used to "calibrate" it, at the cost of a greatly increased overestimation error. In general, the first two approaches rarely achieve more than 10 % of errors, with the first one performing better in the general case and the second one achieving smaller errors in test cases deviating the most from the ideal behavior.

Table 2: Error on power consumption for homogeneous task sets ("real" workloads only).

Approach	Absolute Percentage Error [%]	
	90th	Max
Single	16.56	70.53
True regression	13.05	33.32
Fixed regression	10.65	26.82



(a) ODROID-XU3 (LITTLE core) **(b) ODROID-XU3 (big core)**

Figure 7: Comparison between simulated and actual power consumption for a heterogeneous set of tasks on the ODROID-XU3 platform. The task set was composed of one *hash*, one *gzip-9*, one *encrypt* and one *decrypt* task.

4.2.2 Simulating Tasks Platforms Power Consumption. In the power dimension, the situation is slightly more complicated. If we fix the frequency of a specific platform/core type, the most intuitive solution to simulate the execution of multiple concurrent instances of a specific workload type is to use data from the execution of a single task of that workload and to apply a linear model to generalize the behavior for multiple cores. With this approach, we assume that the power consumption for an individual workload increases linearly with the number of concurrent tasks, as in the ideal case described in the previous section, using eq. (1) to obtain the power consumption of each CPU island. While this approach (named *Single* in Table 2) is exact when a single task is running in the system, errors become significant when simulating multiple concurrent tasks on a multi-core platform.

Let us now consider the less-than-ideal behavior particularly visible in fig. 4e. We can approximate it using another linear model: we can fit the almost-linear behavior of measured data using a linear regression model on data collected with PARTProf, from no running task to one task per CPU on each frequency island; eq. (1) is then used again with the P_{idle} and P_{W_i} obtained from the linear fitting for each task. This approach (*True regression* in Table 2) introduces some errors when simulating a single task, but it can also reduce errors for multiple task instances by a similar amount. As often the case with linear fitting, the resulting linear model is not guaranteed anymore to match the measured data for all points, and for the case of no running task (idle), a different displacement for each workload type may occur. A solution to this problem is to fix the P_{idle} of each CPU based on the accurate measured value and to calculate the contribution of each task P_{W_i} as the value that minimizes the error for various numbers of concurrent tasks (*Fixed regression* in Table 2).

Table 2 shows simulation errors when evaluating the power consumption of each platform in each configuration. In this case, the third approach (*Fixed regression*), which minimizes the simulation error while keeping the displacement of the linear model fixed to P_{idle}, outperforms the other two achieving an error below 10 % for 90 % of the configurations, with the *True regression* as a close second. Notice that the original power estimation implemented for RTSim in [2] systematically results in at least 25 % error when compared against its original reference platform (the ODROID-XU3) in multi-core scenarios. This is because the model in [2] was based on single-core data only, thus it is accurate only for single-core power estimations.

4.2.3 Simulating Heterogeneous Task Sets. Finally, we performed additional tests by deploying four different workload types on the ODROID-XU3 board and comparing simulation results with actual data from the board. Figure 7 compares the expected and simulated values for *power consumption* using the third approach described in the previous section (*Fixed regression*). In these tests, the error between predicted and measured values is 3.65 % at 90th percentile, with a maximum overall error of 5.36 %, showing up at the highest frequencies, which could be mostly attributed to thermal throttling.

5 CONCLUSIONS AND FUTURE WORK

This paper addressed some of the main challenges for the simulation of embedded real-time systems running on multi-processor power-aware platforms. We presented PARTProf and PARTSim, two tightly related tools that enable users to conveniently profile and simulate the behavior of real-time software on embedded platforms with power-aware features like DVFS. In particular, we described novel mechanisms for platform simulation that are more general than existing works in this field. We collected experimental data from various ARM embedded platforms with these tools and verified the accuracy of our simulator against the collected data, showing that in realistic use-case scenarios PARTSim can accurately predict task execution times and power consumption.

Regarding possible future research on the topic, we can observe that the PARTProf and PARTSim tools might be improved by adding a number of useful features: non-instantaneous DVFS frequency switching; measurement and simulation of the impact on energy consumption of deep-idle states, commonly present on a variety of modern CPUs; support for a wider and richer set of workload types to be profiled; simulation of actuators controlling heat dissipation as commonly found on many platforms, e.g., fans, which can often be controlled in speed by the OS. These details would allow for obtaining even more realistic energy-aware simulations.

REFERENCES

[1] Manel Ammar, Mouna Baklouti, Maxime Pelcat, Karol Desnos, and Mohamed Abid. 2018. Comparing Three Clustering-Based Scheduling Methods for Energy-Aware Rapid Design of MP2SoCs. *J. Signal Process. Syst.* 90, 4 (April 2018), 34.

[2] Alessio Balsini, Luigi Pannocchi, and Tommaso Cucinotta. 2019. Modeling and simulation of power consumption and execution times for real-time tasks on embedded heterogeneous architectures. *ACM SIGBED Review* 16, 3 (2019), 51–56.

[3] 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.

[4] Younès Chandarli, Frédéric Fauberteau, Damien Masson, Serge Midonnet, and Manar Qamhieh. 2012. Yartiss: A tool to visualize, test, compare and evaluate real-time scheduling algorithms.

[5] Younès Chandarli, Manar Qamhieh, Frédéric Fauberteau, and Damien Masson. 2014. Yartiss: A generic, modular and energy-aware scheduling simulator for real-time multiprocessor systems. (2014).

[6] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. 2014. SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. In *Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*.

[7] Mayuri Digalwar, Pravin Gahukar, Sudeept Mohan, and Biju K Raveendran. 2015. Stream: a simulation tool for energy efficient real time scheduling and analysis. In *Proceedings of 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real Time Systems (WATERS 2015)*.

[8] M González Harbour, JJ Gutiérrez García, JC Palencia Gutiérrez, and JM Drake Moyano. 2001. Mast: Modeling and analysis suite for real time applications. In *Proceedings 13th Euromicro Conference on Real-Time Systems*. IEEE, 125–134.

[9] Michael González Harbour, J Javier Gutiérrez, José M Drake, Patricia López Martínez, and J Carlos Palencia. 2013. Modeling distributed real-time systems with MAST 2. *Journal of Systems Architecture* 59, 6 (2013), 331–340.

[10] Sebastian Herbert and Diana Marculescu. 2007. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED'07)*. IEEE, 38–43.

[11] K. Honda, S. Kojima, H. Fujimoto, M. Edahiro, and T. Azumi. 2020. Mapping Method of MATLAB/Simulink Model for Embedded Many-Core Platform. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 182–186.

[12] Brian Jeff. 2013. big.LITTLE technology moves towards fully heterogeneous global task scheduling. *ARM white paper* (2013).

[13] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. 2003. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36)*. IEEE Computer Society, 81.

[14] Etienne Le Sueur and Gernot Heiser. 2010. Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower'10)*. USENIX Association, USA, 1–8.

[15] S. M. V. N. Marques, T. S. Medeiros, F. D. Rossi, M. C. Luizelli, A. G. Girardi, A. C. S. Beck, and A. F. Lorenzon. 2019. The Impact of Turbo Frequency on the Energy, Performance, and Aging of Parallel Applications. In *IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. 149–154.

[16] 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.

[17] Agostino Mascitti, Tommaso Cucinotta, Mauro Marinoni, and Luca Abeni. 2020. Dynamic partitioned scheduling of real-time tasks on ARM big. LITTLE architectures. *Journal of Systems and Software* (2020), 110886.

[18] Borislav Nikolic, Muhammad Ali Awan, and Stefan M Petters. 2011. SPARTS: Simulator for power aware and real-time systems. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 999–1004.

[19] Object Management Group (OMG). April 01, 2019. MARTE Specification. `https://www.omg.org/spec/MARTE`.

[20] Luigi Palopoli, Giuseppe Lipari, Gerardo Lamastra, Luca Abeni, Gabriele Bolognini, and Paolo Ancilotti. 2002. An object-oriented tool for simulating distributed real-time control systems. *Software: Practice and Experience* 32, 9 (2002), 907–932.

[21] Preeti Ranjan Panda, BVN Silpa, Aviral Shrivastava, and Krishnaiah Gummidipudi. 2010. *Power-efficient System Design*. Springer Science & Business Media.

[22] Javier Perez Rodriguez and Patrick Meumeu Yomsi. 2019. Thermal-Aware Schedulability Analysis for Fixed-Priority Non-preemptive Real-Time Systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. 154–166.

[23] Eduardo Quiñones, Sara Royuela, Claudio Scordino, Paolo Gai, Luís Miguel Pinho, Luís Nogueira, Jan Rollo, Tommaso Cucinotta, Alessandro Biondi, Arne Hamann, et al. 2020. The AMPERE Project: A Model-driven development framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimization. In *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 201–206.

[24] Julius Roeder, Benjamin Rouxel, Sebastian Altmeyer, and Clemens Grelck. 2021. Energy-Aware Scheduling of Multi-Version Tasks on Heterogeneous Real-Time Systems. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21)*. Association for Computing Machinery, New York, NY, USA, 501–510.

[25] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. 2004. Cheddar: a flexible real time scheduling framework. In *Proceedings of the 2004 annual ACM SIGAda international conference on Ada*.

[26] ZHAOQIAN ZHONG. 2019. Model-Based Parallelizer for Embedded Control Systems on Single-ISA Heterogeneous Multicore. *INTERNATIONAL JOURNAL OF COMPUTERS AND TECHNOLOGY* 19 (February 2019), 7470–7484.