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

Gabriele Ara
Scuola Superiore Sant’Anna
Pisa, Italy
gabriele.arasantannapisa.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:

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 multipile “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 [16]. 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
drivers can also change the voltage of the processor when changing frequency, with a non-negligible execution time 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 temporally 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 DynamicIQ 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 DynamicIQ 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 an 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
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.

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.

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 state2. [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).

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

3Most 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 model that iterates all possible system configurations and starts a variable number 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.

#### 3.2.2 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 “re-playing” 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 single core CPU island, PARTSim uses the following model to calculate the power consumption of an island with $N_{CPU}$ cores:

$$
\text{Power Island} = \sum_{i=1}^{N_{CPU}} P_i(f) \cdot W_i(f)
$$

Where $P_i(f)$ is the power consumption of the $i$-th core, $W_i(f)$ is the workload running on the $i$-th core, and $f$ is the frequency.

$^4$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.
where $P_{Wi}$ 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 $\alpha_i$ 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 between 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 CPU 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:

\[
\begin{bmatrix}
    T_1(t) \\
    T_2(t)
\end{bmatrix}
= \begin{bmatrix}
    -\frac{\alpha_1}{C} & -\frac{\alpha_1}{C} \\
    \beta_{1,2} & -\frac{\alpha_1}{C}
\end{bmatrix}
\begin{bmatrix}
    T_1(t) \\
    T_2(t)
\end{bmatrix}
+ \frac{1}{C} \begin{bmatrix}
    P_{w,1} + \alpha T_e \\
    P_{w,2} + \alpha T_e
\end{bmatrix}
\]

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: $T(t) = (I - e^{A(t-t)}) b + e^{A(t-t)} T(t_0)$.

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.

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

---

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

---

6See also https://docs.scipy.org/doc/scipy/reference/optimize.html
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, featuring four ARM Cortex-A7 (LITTLE) and four Cortex-A15 (big); (ii) a Xilinx ZCU102 board, equipped with a Broadcom BCM2711B0, which includes four Cortex-A72 CPUs; for our tests, we used the official ODROID Linux kernel version 4.14.180. (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 EXPERIMENTAL RESULTS

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. 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.

4.1 Collecting Data With PARTProf

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.

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 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);

4 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.
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 waiting in the OS’s ready queue. Notice that the methodology applied for power consumption estimation varies from platform to platform.

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.

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.
As we increase the cache miss rate, the behavior of these 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 multicore 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.

### 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 multicore 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.
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.

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 one running task to one task per CPU on each frequency island; eq. (1) is then used again with the \( P_{idle} \) and \( P_W \) 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 \( P_W \) 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