Modelling the Energy Consumption of Soft Real-Time Tasks on Heterogeneous Computing Architectures

Houssam Eddine Zahaf, Richard Olejnik, Giuseppe Lipari, Abou El Hassen Benyamina

Modelling the Energy Consumption of Soft Real-Time Tasks on Heterogeneous Computing Architectures

Houssam Eddine Zahaf,
Richard Olejnik
and Giuseppe Lipari
University of Lille, CNRS,
Centrale Lille
UMR 9189 – CRIStAL
F-59000 Lille, France

Abou El Hassen
Benyamina
University of Oran 1
BP 50 El m'nouer
3100, Oran, Algerie

ABSTRACT

The problem of reducing the energy consumption of embedded processors is of paramount importance for mobile devices powered by batteries. Many of these devices embed heterogeneous processors like the ARM bitLITTLE for adapting to the performance requirements and energy consumption profiles of modern applications. However, before proposing algorithms for optimally managing the energy consumption of a device, it is necessary to build a realistic model of the performance and power profile of an application.

In this paper we address this problem by proposing a simple model for the execution time of soft real-time applications and the energy consumption of the hardware platform. We also propose a simple benchmarking methodology for obtaining the parameters of the model from measures. We identify in the memory access pattern of software tasks a key factor that influences both the performance and the energy consumption. We then show how to apply our methodology on the ODROID development board, and we present some preliminary results of our study. We conclude the paper by discussing current research and future directions.

1. INTRODUCTION

Many modern electronics embedded devices are powered by batteries, hence it is of paramount importance to reduce their power consumption as much it is possible, so to prolong their autonomy.

There are two contrasting objectives: reducing the energy consumption of the device (for lowering the "cost" of recharging but also for reducing heating and prolonging the autonomy of battery powered embedded systems), while at the same time optimising the Quality of Service provided to the user. Many of the applications running on model smartphones and tablets have soft real-time requirements, because the quality of service experienced by the users depends on the response time of the computations. For example, users want to watch audio/video content streamed from Internet at the maximum quality on their tablets while at the same time prolonging the duration of the battery. A delayed visualisation of some video frames, or even worse, the delayed decoding of an audio sample, may greatly compromise the user experience.

In order to propose efficient techniques for optimising the energy consumption of a complex device consisting of heterogeneous computing cores without compromising the response time of the running applications, it is of foremost importance to build a *model* of the system. Therefore, we need to identify the main factors that impact on the energy consumption and on the response time of an application.

The complexity of the problem forces us to make a choice between accuracy and complexity. An exact model of the hardware/software architecture would have the same complexity as the hardware/software architecture itself, and so it would be of no use in practice. We need then to abstract away the many details and focus on the main parameters that can influence energy consumption and execution time.

This paper is a first step toward a model of the energy consumption and of the execution time of a task running on a ARM big/LITTLE architecture. For the hardware architecture, our model considers a few parameters as the type of the processor being used (big or LITTLE), the operating frequency, and the memory architecture (size of the cache). For the execution time of a soft real-time task we consider the number of cache misses, the processor on which it is running and the operating frequency. We show how to compute the model parameters with a set of measurement, and we show that our model captures the behaviour of the application in a sufficiently precise manner.

The proposed model will be used in the future to build scheduling and allocation algorithms that aim at minimising energy consumption without compromising on the Quality of Service.

2. RELATED WORK

In the research literature, many works have focused on measuring execution time of executing tasks, both in the real-time scheduling domain and in High Performance Com-

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

puting (HPC). However, not so many papers focus on benchmarking both execution time and energy consumption in mobile embedded systems.

Erich et al., in [1], proposed an approach for designing benchmarks for scientific computing. They focused basically on task granularity, and on temporal and spatial locality. The main result of their investigation is that one of the most important factors influencing task performances is the memory access. However, they only addressed *big* processors (like Intel Xeon Phi, ...) designed for HPC, and did not consider low-power embedded processors, nor heterogeneous architectures. In particular, here we are interested in the trade-off between power consumption and performance.

Kambel et al.[2] investigate benchmarking for mobile phones. They assume that classic benchmarking methodologies are not suitable for the target devices. However, their work focuses on a subset of applications, notably application heavily relying on GUI interaction, and low-intensive computation applications (e-mail clients, chats, etc.), where the emphasis is on networking.

Gal-On and Levy [3] discussed the effectiveness of current benchmarks for modern processors. They focus on parallelism, heterogeneity, memory access impacts and on performances. The report does not present any experiments and discusses in a general way the possible impact of these aspects.

In the area of Real-Time scheduling, benchmarking is focused on *worst-case execution time* rather then on performance and energy consumption. In addition, benchmarks are used to compare scheduling and worst-case execution time algorithms, rather than on build a model of the task behaviour. Here instead we are interested in *average execution time* and its relationship to average memory access patterns, and how this influence the power consumption of the architecture.

3. SYSTEM MODEL AND ASSUMPTIONS

The objective of this paper is to build a model of the execution time and of the energy consumption of soft real-time task executing on a heterogeneous multi-core architecture. In this section we describe the model of a soft real-time task and of the hardware architecture.

A soft real-time task is a software process executing periodically. An example of C code that uses the POSIX RT API for implementing a periodic real-time task is reported in Figure 1.

```
void *task(void * arg) {
  struct timespec_t next;
  // initialization
  clock_gettime(CLOCK_REALTIME, &next);
  while (1) {

    // * periodic code *

    // suspend until next period
    timespec_addto(next, period);
    clock_nanosleep(CLOCK_REALTIME, 0, &next, 0);
  }
}
```

Figure 1: Pseudo-code of a soft-real time periodic task

From a mathematical point of view, a periodic real-time task is a sequence of *jobs* $J_i = (a_i, c_i, d_i)$. Each job corresponds to the execution of the code denoted as *periodic code* in the figure. The arrival time a_i is the activation time of the job (corresponding to the values of variable `next`); the computation time c_i corresponds to the execution time of the jobs, that is the processing time necessary to complete the periodic code; d_i is the job's *deadline* that is the absolute time within which the job should complete its execution in order to satisfy the performance requirements of the user.

Per a periodic task, we have that $a_i = a_{i-1} + T$, where T is the task's period. Very often the job deadline is equal to the next arrival time of the job (each job should complete before the next one is activated). Hence $d_i = a_i + T_i = a_{i+1}$.

The execution time of a job depends on many factors: 1) the number of hardware instructions to be executed, 2) how many clock cycles per each instruction 3) the operating frequency of the processor and of the main memory.

In turn, 1) depends on the input data and on the internal state of the task; 2) depends on the hardware architecture and its internal state. Notably, the pipeline state and the cache state may have a great influence on the execution time of the task. Typically, we can control the processor operating frequency, to a certain extent. In particular, it is possible to set the frequency so to reduce energy consumption without degrading too much the performance of the task. For example, under the assumption that the system executes one single task, and that we have an exact model of the execution time of a job, we could set the processor frequency so that $c_i = d_i - a_i = T$, that is the task finishes always exactly at its deadline.

This technique is called Dynamic Voltage and Frequency Scaling (DVFS), and will be described in the following. Unfortunately, it is impossible to have a precise model of the execution time of a task, so we have to derive approximate models.

In the following, we restrict our attention to tasks with a weak dependence from input and state variables; in other words, we assume that the number of processor instructions to execute in each jobs is approximately constant. The variability due to input data and state variables is very dependent on the application class, and it will be the focus of future works.

In the next section we describe the hardware architecture used for our experiments, and a set of benchmarks for measuring execution time and power consumption of a specific task.

3.1 Odroid architecture description

The *ODROID XU3* [4] board (Figure (2)) is compound of a *samsung Exynos 5422*, a *Mali GPU*, a RAM memory and I/O peripherals. The *Samsung Exynos 5422* is an ARM big.LITTLE multicore architecture. It is composed of 8 cores: 4 big cores (*ARM cortex A15*) and 4 LITTLE cores (*ARM Cortex A7*). The *ODROID XU3* board embeds 4 power *sensors*: a sensor for big cores, a sensor for little cores, a sensor for the GPU and the last one for the memory. An external energy *sensor* (Odroid SmartPower) is plugged to the power supply of the board in order to measure the overall power consumption.

Each core of the Exynos 5422 has a private L1 cache of $2 \times 32Kb$. Little cores share $512Kb$ of L2 cache. Big cores

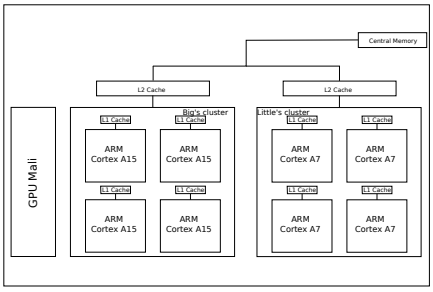

Figure 2: ODROID XU3 board architecture

share $2Mb$ of L2 cache. Both big and little cores share a RAM memory of $2Gb$.

The frequency of little cores can be calibrated homogeneously from 200Mhz to 1400Mhz in discrete steps of 100Mhz (13 modes). Respectively, the frequency of big cores can be calibrated from 200Mhz to 2000Mhz, in discrete steps of 100Mhz (19 modes). In this paper, core operational frequency and state are set offline and do not change dynamically. Therefore, for the moment we are not interested in the core state and frequency changing costs.

3.2 Odroid bench design

In order to build a simple task and architecture models, we implemented a simple bench of periodic matrix multiplication of size $S \times S$. The matrix multiplication example allows us to change the data size and can be easily parallelized. Also, the matrix multiplication implementation allows us to control the spatial and temporal locality which make the cache behaviour *easily* predictable. To measure the power consumption, we used 3 out of the 4 available embedded sensors: the big cores sensor; the little cores sensor; the memory sensor. The external power sensor allows us to have an external view on the power consumption of all components of the ODROID board.

The thread has a higher priority than all other threads in the system. In Figures 3, 5, 6, 7, 8, 10, the task is implemented in one periodic thread locked on a big core or a little core using POSIX thread affinity. The frequencies are set using the `CPUFreq` tool. Each experiment is run 500 times, the presented results are the average values. After each experimentation, L1 and L2 caches are reset.

3.3 Benchmark results

Figure 3 presents the average execution time of the thread locked on a big core (red squares) and on a little core (blue circles) as a function of the frequency. When big and little cores operate at the same frequency, big cores perform approximately 3 times faster than little cores. This can be explained by the more complex architecture of ARM A15 compared to ARM A7 (Figure (4)), a big core executes in average more instructions than a little core per one cycle.

Figure 5 presents the average power consumption of big (red squares) and little (blue circles) cores as functions of the frequency. A little core consumes $3-5$ times less power than a big core operating at the same frequency. This is due may be to the complex architecture of big cores compared to little cores.

Figure 6 presents the average consumed energy of one instance of matrix multiplication thread (the cumulative

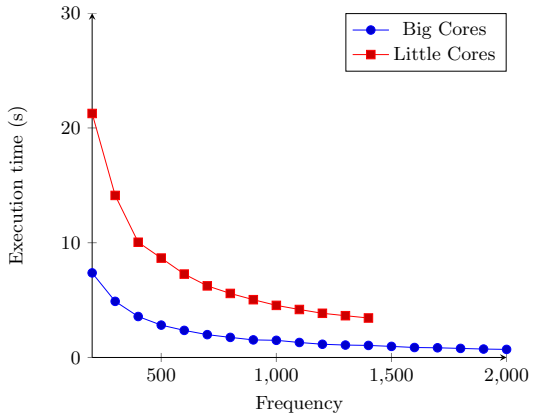

Figure 3: Execution Time with different frequencies on big and little cores

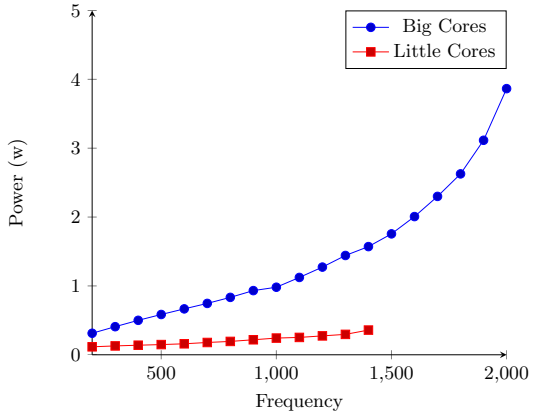

Figure 5: Power consumed by frequency: Big vs little cores

power by one instance of the thread) for big (red squares) and little (blue circles) cores as functions of the frequency respectivly. Even if it finishes the executions in a shorter time, a big core still consumes more energy than a little one that operates on the same frequency. Even more, a big core set on the lowest frequency (200Mhz) still consumes more than a little core set at the maximum frequency (1400Mhz).

Figures 8 and 7 present power consumption of the whole Odroid Board as function of frequency. The power consumption of all componenets except the big and little cores is constant.

Figure (9) presents the power consumption ratio of big (red squares) and little (blue circle) cores as functions of frequency. When the frequency is low, the board consumes (40-60) times more than little cores, and (6-9) more than little core. However, this ratio decreases substantially when frequency is high (only 8 time for little cores and 1.7 for big cores).

Figure (10) presents the memory power consumption as function of the frequency. The memory consumption is higher When the thread is locked on a little core than when the same thread is locked on a big core. This is probably due to the fact that Little cores have only $\frac{1}{4}$ of big core $L2$ cache, Then, more cache-misses occur causing more energy consumption. This assumption can not be verified because

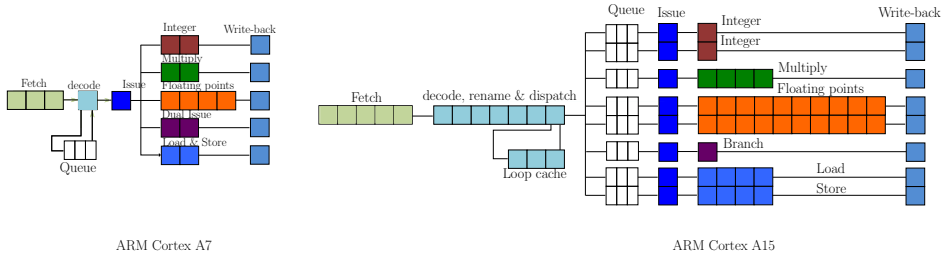

Figure 4: Pipeline architecture of A7 (left) and A15 (right) cores.

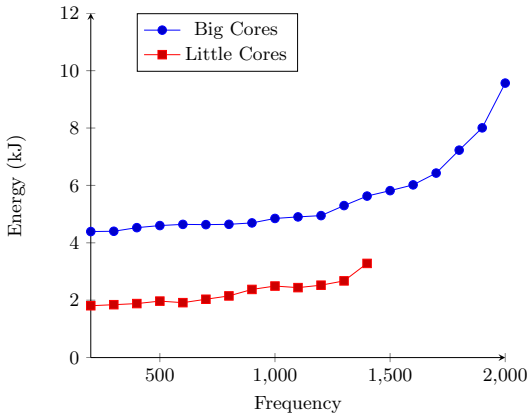

Figure 6: Big and little cores energy consumption

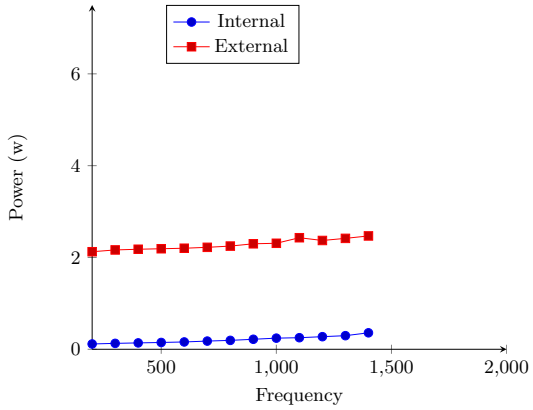

Figure 7: Little Core power consumption: Internal vs External measures

the current linux version of ARM exynos 5422 is still does not support the hardware calls to read the cache-misse registers.

4. ARCHITECTURE MODELING

To flexibly model heterogeneous architectures as big.LITTLE, we introduce the concept of core group. A multicore architecture $\mathcal{A}$ of m cores is compound of G core group. All cores (P_j) of the same group $\mathcal{G}_g$ have the same micro-architecture and frequency characteristics (minimal frequency (f^g_{min}), maximal frequency (f^g_{max}), operating frequency (f^g_{op})). Each group has a set of discrete frequencies modes and its own energy calculation coefficients (discussed in Section 4.1).

Cores are indexed alternatively, core j belongs to the group $g = (j\%G)$. For a processor with 4 cores and 2 groups, cores $0, 2$ belong to group 0 and cores $1, 3$ belong to group 1. We choose this representation to allow to compute the core group just by having its index j. Moreover, by using the concept of core group we can address homogeneous architecture (e.g. SMP) by setting the number of groups $G = 1$, and heterogeneous architectures like ARM big.LITTLE architecture by setting the number of groups to 2 or more.

$$\begin{aligned}\mathcal{A} &= \{\mathcal{G}_g, g \in \{1 \dots G\}\}\\ \mathcal{G}_g &= (\{\mathsf{P}_j, (j \mod G) = g, j < m\}, \mathsf{f}_{min}, \mathsf{f}_{max}, \mathsf{f}_{op}),\\ &\quad g \in \{1 \dots G\}\end{aligned}$$

4.1 Energy model

Increasing a core's frequency involves switching its transistors more rapidly, and transistors that are switched more rapidly dissipate more power. The power dissipated due to switching is called *dynamic power*. In order to reduce dynamic power, we calibrate the operating frequency for big and little cores.

Even transistors that aren't switching will still leak current during idle periods. This leakage current constantly dissipates power. The amount of power dissipated per unit of area due to leakage current is called static power. In order to reduce the static power consumption, we have the ability to set the cores in a deep state power using the ARM *Wait For Interrupt* and *Wait For Event* hardware instructions.

The overall power consumption is the sum of the dynamic and static power (total power consumption).Jing Mie et al. [5] defined the dynamic power dissipated by a CMOS circuit as the product of a constant coefficient ξ that depends on the technology, by the square of the voltage, and by the frequency.

$$E = \xi \times V^2 \times f \tag{1}$$

They also defined the frequency as the ratio of the difference between the actual voltage V and V_{Th} raised to the power of a, where V_{th} is the threshold voltage by the product of a constant K and the logic depth L_d. a and K are constants that depend on the technology.

$$f = \frac{(V - V_{Th})^a}{K \times L_d} \tag{2}$$

By combining the two equations, the dynamic power E can be expressed as the product of a constant *Const* by the

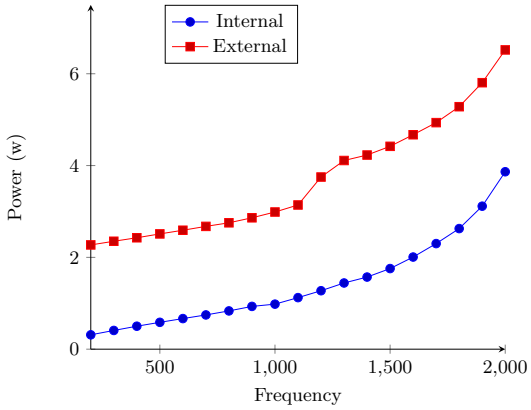

Figure 8: Big Core power consumption: Internal vs External measures

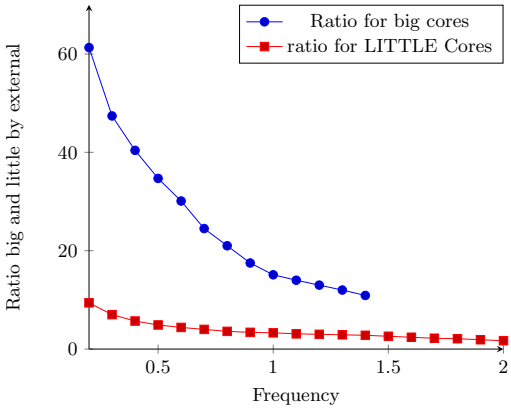

Figure 9: Power ratio between all the board components and cores power consumption

frequency f power λ (3).

$$E(f) = Const \times f^{\lambda} \tag{3}$$

In order to evaluate the total power consumption in the rest of this work, we used a polynomial regression (we used the online regression tool available at http://www.xuru.org/rt/PR.asp) of the power values measured for big and little cores (described in Figure (5) as a function of variable frequency. Figure (11) presents the results of the polynomial regression of degree 3 for big cores (asterisks) and little cores (squares).

Figure (11) shows that the match between the values and the regression curve for both big and little cores is rather satisfactory. The regression polynomial in Equation (4) is for little cores and the one of Equation (5) is for big cores.

$$Power(f)^{l} = 0.07477f^3 + 0.045706f + 0.008425 \tag{4}$$

$$Power(f)^{b} = 1.0561341f^3 + 1.376928f^2 + 2.452f + 1.0216 \tag{5}$$

5. TASK MODEL

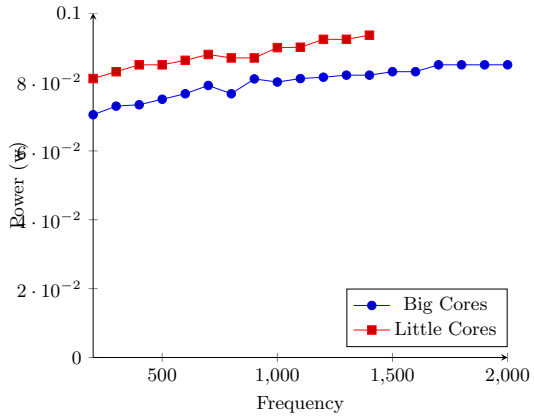

Figure 10: Memory power consumption: Big VS Little cores

We assume that the execution time C of a thread depends on the micro-architecture of the core group on which the thread is allocated and on its operating frequency. Thus, to model the execution time of a task we obviously need to compute model coefficients for every core group, in our case for big and LITTLE cores.

Only a part of the execution time depends on the core operating frequency. For example, when a thread suspends itself waiting for operating on external devices, the suspension time depends on the response time of the external device which may not be linked to the core operating frequency. As an relevant example, consider the time to access main memory in case of a cache miss: the response time depends on the bus contention (due to other cores/devices) and on the response time of the memory circuitry.

The component of the execution time that directly depends on the frequency is denoted by $ct(f)$, whereas the part that does not depends on the frequency is denoted by it. For simplicity, in this paper we consider this second part as a constant which only depends on the thread average characteristics (e.g. number of memory requests and average number of cache misses). Thus, the execution time of a thread on a core of frequency f is modelled by a semi-linear

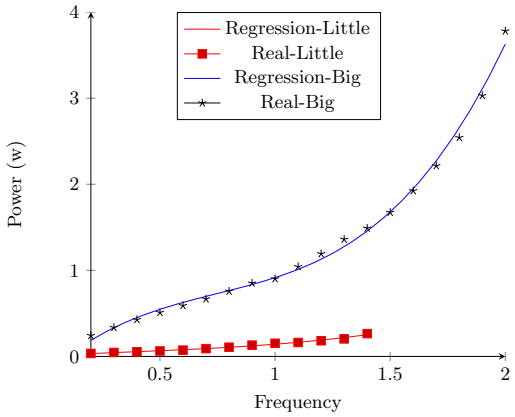

Figure 11: Regression and real values for big and little power consumption: cores consumption

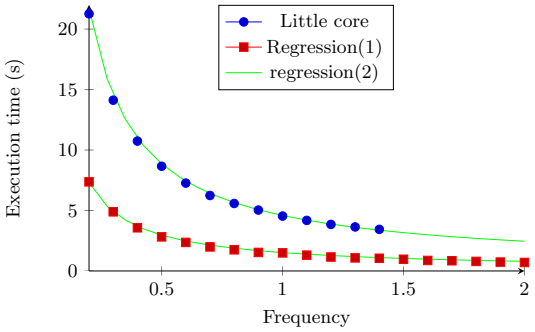

Figure 12: Execution time: regression and real values

function (Equation 6)

$$C^g(f) = \frac{ct \cdot f_{max} \times f_{max}}{\mathsf{f}} + it \quad (6)$$

To estimate the two components of the execution time, we measure the thread execution time under different conditions (different inputs, differents matrix sizes, frequencies, groups). Figure 12 presents a non-linear regression of the execution time values (Figure 3) as function of frequency for matrix multiplication of size $300x300$. Equation (7) presents the semi-linear function of the regression for little cores (resp. Equation 8 for big cores).

$$C^l = \frac{4.27}{\mathsf{f}} + 0.32 \quad (7)$$

$$C^b = \frac{1.45}{\mathsf{f}} + 0.07 \quad (8)$$

We observe that, in this specific case the *it* time for little cores is 4 time bigger than the *it* time of little cores.

We assume that the *it* component of the execution time depends on the number of cache misses experienced by the task, which in turns depends not only on the size of the task data set, but also on the interference by the other threads in the system. However, we cannot currently measure the number of cache misses experienced by a task on the bigLITTLE processor: due to a software error in the Linux device driver for the ODROID platform, we cannot currently collect this precious data. We are currently actively working at solving the problem, and we hope to report experiments directly linking the number of cache misses to the execution time of the thread.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented some preliminary results of a study for modelling the execution time and the power consumption of soft real-time tasks running on a heterogeneous processor architecture, namely the bigLITTLE architecture.

We have shown how to compute the parameters of the power consumption model of the architecture, and how to model the execution time of a thread as a function of the core frequency and the memory access. We found that the number of cache misses has a big impact on the execution time and hence on the power consumption of the memory.

In the future we plan to carefully investigate the relationship between data set size, number of cache misses, execution time and power consumption. In fact, accessing main memory slows down the task and increases the energy consumed by the main memory itself; moreover DVFS techniques do not influence the component of the execution time due to main memory access.

It is clear that this kind of study is preliminary to the proposition of power management heuristics. In addition, we are currently studying the use of machine learning techniques to identify the parameters of a task at run-time.

7. REFERENCES

[1] Erich Strohmaier and Honghang Shan. Architecture independent performance characterization and benchmarking for scientific applications. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pages 467–474. IEEE, 2004.

[2] Brian Kimball Dunn, Dennis F Galletta, Daneka Hypolite, Anuj Puri, and Sandeep Raghuwanshi. Development of smart phone usability benchmarking tasks. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 1046–1052. IEEE, 2013.

[3] Shay Gal-On and Markus Levy. Measuring multicore performance. *Computer*, 41(11):99–102, 2008.

[4] Odroid xu3 board description. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127. Accessed: 2015-09-30.

[5] Jing Mei, Kenli Li, Jingtong Hu, Shu Yin, and Edwin H.-M. Sha. Energy-aware preemptive scheduling algorithm for sporadic tasks on DVS platform. *Microprocessors and Microsystems*, 37(1):99–112, February 2013.