Work-in-Progress: System-wide DVFS for real-time systems with probabilistic parameters

Roberto Medina, Liliana Cucu-Grosjean
Inria, Paris, France
{roberto.medina-bonilla, liliana.cucu}@inria.fr

***Abstract*—Power consumption and the ever-increasing service demand are key issues that real-time embedded systems are facing nowadays. Approaches based on Dynamic Voltage and Frequency Scaling (DVFS) are capable of reducing the energy consumed by processors while still guaranteeing real-time constraints. In this paper we present short-comes on the state-of-the-art techniques that need to be leveraged in order to be more effective when reducing energy consumption: our experimental results clearly show that tasks' execution time is not proportional to processor speed, and that DVFS techniques could also be applied on other components like bus and memory without compromising time constraints. We also discuss the applicability of real-time probabilistic systems and DVFS, and argue that by adopting a frequency-aware model, we can (i) capture more detailed behaviors of tasks w.r.t. processor speeds and (ii) apply DVFS techniques to gain in energy consumption.**

I. Introduction

During the past few years, embedded real-time systems have been confronted to increased processing capabilities (by incorporating more functionalities) and, at the same time, reduce their energy consumption. In fact since many of these systems are battery powered and recharging or changing a battery is not always practical or feasible, energy management has become a prime design objective for the conception of embedded real-time systems.

A common effective technique for reducing energy consumption is Dynamic Voltage and Frequency Scaling (DVFS) [1]. The majority of modern processors offer the possibility to switch frequencies in order to reduce input voltage given to the unit in only a few microseconds. By doing so, tasks' execution time tends to increase, while energy consumption decreases. The main goal of DVFS techniques is to derive proper frequency values capable of guaranteeing timing constraints while minimizing energy consumption. A common assumption made by many DVFS real-time scheduling policies is the scalability of tasks' Worst Case Execution Time (WCET). It is often assumed that WCET is fully scalable w.r.t. processor speed, meaning that the faster the processor goes, the faster tasks will complete their execution. As a matter of fact, actual scheduling algorithms on real-time operating systems adopt this hypothesis in their implementations [2]. Nevertheless, we will show through experimental results that this is not always the case, and that other components like the bus and memory play major roles on tasks' execution time.

At the same time, while DVFS techniques have mostly been applied to processors on real-time systems, in reality, energy consumed by the processor only represents a fragment of the energy consumed by the whole system. For example, when we look into modern computer architectures, energy consumed by the memory can take up to 25% of the total energy consumed [3] and it has been demonstrated that DVFS techniques can also be applied to other components like memory and buses in the context of real-time systems [4].

Finally, to cope with the ever-increasing demand in terms of functionalities and services for modern embedded systems, we are interested in probabilistic real-time systems where the WCET is defined thanks to a probabilistic distribution. A probabilistic real-time systems can guarantee a minimum degree of schedulability while other scheduling policies would deem the system as non-schedulable [5], [6]. Nevertheless, DVFS techniques have never been used on these types of systems. Therefore, in this paper, our main objective is to propose an appropriate model for WCET distributions when frequencies change on various components of the system. Finally, we will show how existing DVFS techniques can be applied to this type of probabilistic real-time systems and discuss future research directions.

II. Background: Task model and DVFS techniques

In this section, we introduce notions and results about DVFS on non-probabilistic real-time systems. To define an appropriate probabilistic model with DVFS, we look into existing works. A typical real-time systems is implemented as a set of concurrent tasks being executed by the operating system. A task τ_i is defined as a tuple $\tau_i = (C_i, D_i, T_i)$, where C_i is its WCET, D_i its deadline and T_i its period (for *periodic* systems) or its minimal inter-arrival time (for *sporadic* systems).

A. Scalable WCET

A common hypothesis taken by many DVFS techniques is the scalability of the WCET w.r.t. the processor speed [1]. The WCET could therefore be defined thanks to the following function: $C_i(s) = C_i/s$, where s is the speed of the processor.

Some well known scheduling algorithms include *Dynamic Reclaiming Algorithm* (DRA), *Agressive Speed Reduction* (AGR) [7] and *Greedy Reclaim of Unused Bandwidth* (GRUB) with its power-aware extension (GRUB-PA) [8]. The above-mentioned algorithms rely on the following principle: because WCET is difficult to estimate and safe upper-bounds are often used, tasks tend to complete their execution earlier than their

WCET. This unused processing time can be then reclaimed in order to reduce the processor speed and still meet deadlines. The transition from one frequency to another is performed during the context switch.

B. Partially scalable WCET

Nevertheless, it is unrealistic to think that during tasks execution no memory access takes place. Therefore, the hypothesis of a fully scalable WCET can be too optimistic for certain systems. In [9] authors propose a static analysis framework used to obtain Worst Case Execution Cycles and demonstrate that, in the presence of a memory hierarchy, the number of cycles does change when the processor speed changes. For instance, when the processor goes faster than memory and tasks perform many memory requests, more cycles will be spent waiting for the memory. In their analysis, authors consider a fixed number of cycles for memory requests, defining the following function: $C_i(s) = C_i/s + C_{fix}$. By doing so, they are capable of estimating tighter WCET bounds. Another advantage of this framework is that it can be used "on top" of any existing DVFS scheduling algorithm, improving energy savings on existing algorithms.

C. Extending DVFS to other components

It is clear that DVFS techniques are capable of limiting the energy consumption of the processor and many scheduling algorithms have been developed to take advantage of this feature [1]. As a matter of fact, frequency scaling has also been considered on other components of hardware architectures like the bus and memory. These components also require a large portion of energy in order to operate [3]. For instance, scheduling algorithms have been proposed for architectures capable of supporting DVFS at the processor and memory level [3], [4]. The WCET will be decomposed into two (or more) scalable portions that vary in function of the hardware setup. For instance, one could define the WCET by the following formula (CPU and memory are DVFS capable): $C_i(s_{cpu}, s_{mem}) = C_i^{cpu}/s_{cpu} + C_i^{mem}/s_{mem} + C_{fix}$.

Considering a mixture of scalable and non-scalable parts on tasks' WCET seems to be an appropriate approach when defining a task model for DVFS algorithms since it captures the behavior of all physical components of the architecure. However, existing approaches have gone at the instruction level to determine which part of the program does not scale and in reality COTS are equipped with hardware features like caches limiting the impact of memory hierarchy. These enhancements push towards the scalability of the WCET w.r.t. the processor speed.

III. Frequency scaling profiling

To evaluate how close the fully scalable model is to reality on modern COTS, we have performed experimental evaluations of the TACLe benchmark [10]. Our first goal is to observe the scalability of execution time w.r.t. processor speed. The second goal of these experiments consists in observing the impact of memory speed on some of these programs, if the impact is small then considering DVFS techniques at a memory level would be beneficial for the overall system.

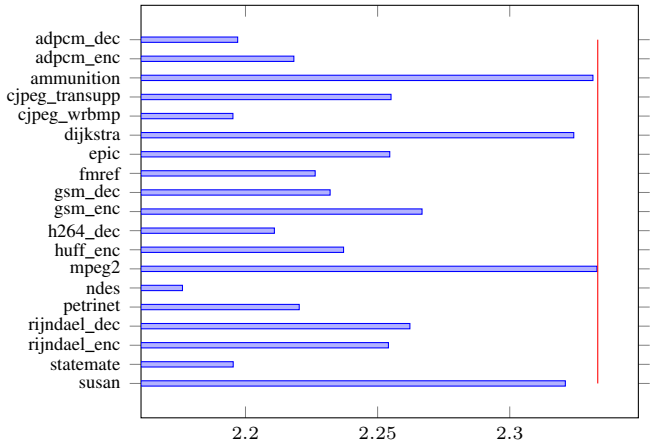


Fig. 1: Ratio of execution time between min. and max. speed for TACLeBench programs

A. Experimental setup

Our experiments are conducted on a Raspberry Pi 3B+. This board contains an ARM Cortex A53 processor with speeds varying from 600 MHz to 1400 MHz. Cores access the memory (default 500 MHz) through a bus (default 400 MHz). We set the Raspberry Pi to use the Linux kernel 4.14 with the `PREEMPT_RT` patch.

In our experiments we isolate the programs we consider for measurements, therefore some setup at the operating system level is performed:

• TACLeBench programs are executed in a single core isolated at a kernel level by setting the `isolcpus` option at boot. • Each program has its affinity and priority changed to be executed on the isolated core at the maximum priority. • Real-time throttling is turned off. • CPU frequencies are setup thanks to the `userspace` governor. Chosen frequencies are 600 and 1400 MHz. • Memory frequencies can be changed thanks to the Raspberry Pi firmware at boot. Chosen frequencies are 250 MHz, 375 MHz and 500 MHz. • Execution time, CPU cycles and instructions are measured thanks to `perf`. • Each program is executed 500 consecutive times.

B. Experimental results: CPU frequencies

Our first experimental results are presented in Fig. 1. We have plotted the ratio of the average execution time for each TACLeBench program, between the min. and max. speed of the processor. This can be seen as the *speedup* achieved by the program when the processor switches its speed from 600 MHz to 1400 MHz. The red vertical line represents the theoretical speedup that should be achieved if tasks are fully scalable w.r.t. processor speed. As it is shown in the figure, few applications are close the theoretical speedup achieved by increasing the frequency of the processor (*e.g.* `ammunition`, `mpeg2`). Like it was expected, this is due to the fact that programs request other resources than just the CPU and hardware optimization like caches and branch predictors are

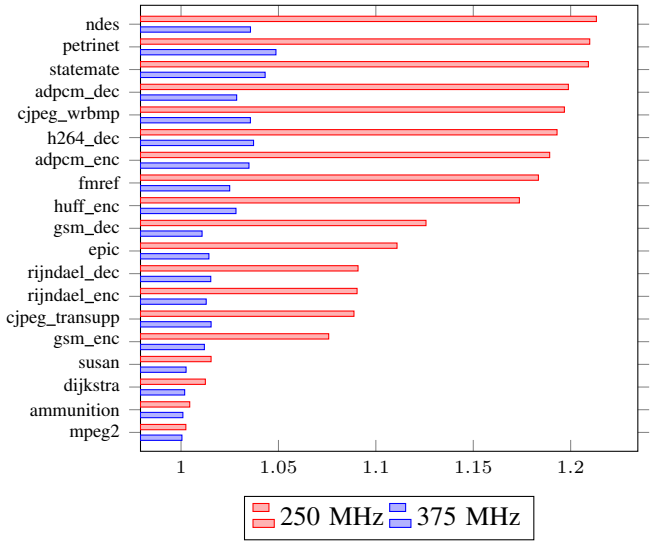


Fig. 2: Performance degradation when memory speed is set at different frequencies

not able to fully limit the impact of memory accesses. For instance, programs like `adpcm_dec`, `ndes` and `statemate` are performing more memory requests which provokes some delays when the processor actually goes faster. The maximal slowdown that was observed was close to 7%, nevertheless it is important to notice that in our experiments, programs are executed in isolation and the system has very minimal services running in parallel. This gap should be more noticeable when other programs interfere or pollute shared caches for example.

C. Experimental results: Memory frequencies

To further reduce energy consumption on modern embedded architectures, DVFS capabilities have been included in other components like memory and buses. As a matter of fact, energy consumed by the memory also represents a large portion of the total energy utilized by the system [3]. Previous works [4] have demonstrated that similarly to scaling down the processor, the memory can be slowed down to save in energy and still meet real-time constraints.

We have performed some experiments thanks to the TACLeBench suite in order to observe the impact of memory frequency on this benchmark. The Raspberry Pi firmware allows us to set a given frequency for the SDRAM included on the board. Our goal is to demonstrate that for CPU intensive programs, memory can be slowed without having an important impact on the program's performance. By doing so, further energy savings can be achieved. Fig. 2 illustrates our experimental results. We present the performance degradation in terms of execution time, when the system's memory is slowed down to 250 MHz and 375 MHz respectively (default memory speed is 500 MHz). The processor speed is fixed at its maximum: 1400MHz. Having a ratio close to 1 means that the memory speed had almost no effect on the program execution time time. As we can see, programs like `dijkstra` and `mpeg2` perform almost identically even when memory speed is brought down. On the contrary, programs like `ndes` and `petrinet` should be executed without changing the memory speed, otherwise their WCET would change significantly.

Our experimental results confirm the following: **(i)** WCET cannot be considered as fully scalable w.r.t. CPU speed; and **(ii)** memory plays a major role on tasks' WCET. A system-wide DVFS algorithm with a task model having various terms that vary in function of hardware speeds (Section II-C) would be ideal. Nevertheless, existing approaches [4], [9] have decomposed tasks on number of cycles required for each hardware component, which in practice cannot always be performed on an embedded real-time system.

IV. Proposed Task Model and DVFS algorithm

A. Task model with probabilistic parameters

Since the decomposition of tasks into cycles is not always practical, we are interested in proposing a task model based on probabilistic real-time systems, where the WCET of a task is defined thanks to a probability function, also called probabilistic WCET (pWCET). At the same time, since a probabilistic real-time systems can guarantee a minimum degree of schedulability, we can increase functionalities and services on modern embedded systems.

The pWCET, as defined in [6], is the least upper bound, on the execution time distribution of the program. This upper bound can be defined as follows:

$$\mathcal{C}_i = \begin{pmatrix} C^{min} & C^1 & \dots & C^{max} \\ f_{\mathcal{C}_i}(C^{min}) & f_{\mathcal{C}_i}(C^1) & \dots & f_{\mathcal{C}_i}(C^{max}) \end{pmatrix}$$

where $f_{\mathcal{C}_i}(c) = P(\mathcal{C}_i = c)$ is the probability that task τ_i execution time will be bounded by c time units and $\sum_{j=0} f_{\mathcal{C}_i}(c_j) = 1$.

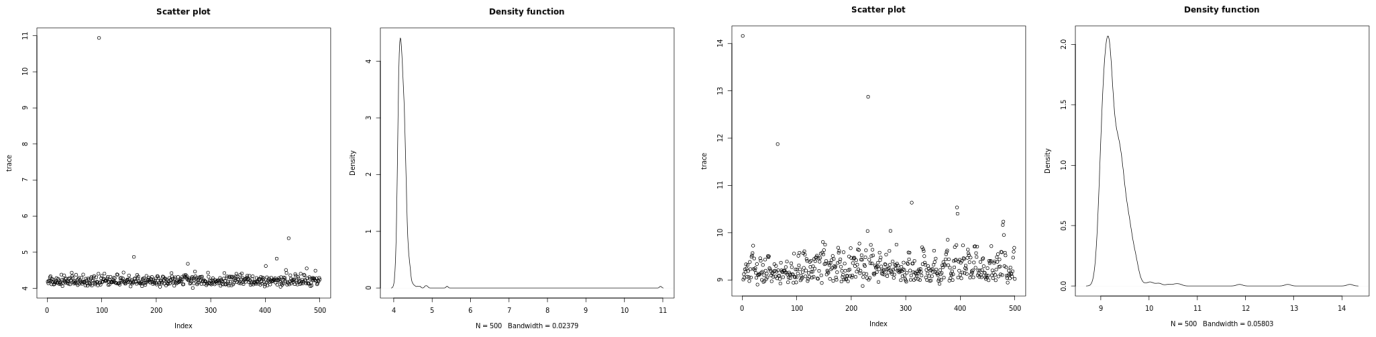
B. Frequency-aware task model

When defining a frequency-aware probabilistic task model, we need to know how the pWCET changes under different frequencies. For instance, the probability functions can change significantly and more values would be necessary to describe the behavior of the task. Fig. 3 shows this behavior thanks to traces of execution times for the `cpjeg_wrbmp` program of the TACLeBench suite under two different CPU frequencies (1400 and 600MHz).

Therefore, our proposed task model defines different pWCET for each task in function of frequencies considered at a hardware level. In our case, the pWCET is defined in function of the CPU and memory frequency. It is important to note that not all hardware combinations need to be considered for all tasks. For example, for tasks where DVFS at a memory level would not be beneficial, we would not require to store the probability functions of the different memory frequencies.

C. DVFS probabilistic real-time scheduling

In general, DVFS techniques should have low overhead, *i.e.* the complexity of deriving a frequency should not take much time or space since these algorithms are executed at runtime. A drawback from considering different pWCET for each combination of hardware is the space it would take to



(a) 1400MHz (b) 600MHz

Fig. 3: Execution time traces and density functions for the `cpjeg_wrbmp` benchmark

store all the information. However, we describe the principle of how probabilistic DVFS could work.

Let us consider the following task set being scheduled in a single-core processor:

$$\tau_1 : \left(\begin{pmatrix} 3 & 4 & 5 \\ 0.2 & 0.75 & 0.05 \end{pmatrix}^{min}, \begin{pmatrix} 2 & 4 \\ 0.95 & 0.05 \end{pmatrix}^{max}, 5, 5 \right)$$

$$\tau_2 : \left(\begin{pmatrix} 2 & 4 \\ 0.7 & 0.3 \end{pmatrix}^{min}, \begin{pmatrix} 1 & 3 \\ 0.99 & 0.01 \end{pmatrix}^{max}, 10, 10 \right)$$

For space limitation reason we will consider that DVFS is only applied to the processor and only two speeds are available. However, we could consider frequency scaling at the memory level as well and change frequencies during the context switch.

In this example, even if the system is running at its max. speed, it would be deemed as non-schedulable for any deterministic scheduling policy. Nevertheless, probabilistic response time analysis [5] can be used to derive a degree of schedulability which in this case is 99.9975%. Considering this schedulability rate is acceptable, the problem relies on the fact that at a min. frequency, the schedulability degree becomes 52.65%, *significantly lower than the tolerable threshold*.

The probabilistic DVFS schedulability analysis can be decomposed into two phases: **(i)** derive suitable speeds guaranteeing a schedulability degree offline and; **(ii)** an online phase that further reduces speed whenever it is possible. For instance, in the example, a suitable scheme would be beginning at min. speed, then switch to the max. speed for the last execution of τ_1 during the hyperperiod, giving a schedulability rate of 96.9%. However, if the first job of τ_1 finishes at 3 time units and τ_2 finishes within 2 time units, then the second job of τ_2 can complete its execution even when the processor is at its min. speed, allowing the system to complete its execution with its minimal energy consumption. As opposed to deterministic scheduling algorithms, DVFS on probabilistic systems can accept different speed configurations to guarantee the required schedulability degree, which could lead to less energy consumption.

V. Conclusion and Future Works

This paper introduces a probabilistic task model for real-time systems capable of using DVFS techniques. The proposed model has been based on experiments and differs from most existing works that consider fully scalable WCET. We have also demonstrated how DVFS techniques can be applied to probabilistic real-time systems without compromising timing guarantees for these systems. For future works we plan to fully evaluate the extension of probabilistic DVFS scheduling algorithms in terms of energy reduction and complexity. We also want to propose a method to derive pWCET distributions for tasks with and without frequency scaling hardware.

References

[1] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, p. 7, 2016.

[2] C. Scordino, L. Abeni, and J. Lelli, "Real-time and energy efficiency in linux: theory and practice," *ACM SIGAPP Applied Computing Review*, pp. 18–30, 2019.

[3] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*, 2011, pp. 31–40.

[4] H. Yun, P.-L. Wu, A. Arya, C. Kim, T. Abdelzaher, and L. Sha, "System-wide energy optimization for multiple dvs components and real-time tasks," *Real-Time Systems*, vol. 47, no. 5, p. 489, 2011.

[5] D. Maxim and L. Cucu-Grosjean, "Response time analysis for fixed-priority tasks with multiple probabilistic parameters," in *2013 IEEE 34th Real-Time Systems Symposium*, 2013, pp. 224–235.

[6] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 6, no. 1, pp. 03–1, 2019.

[7] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, 2004.

[8] C. Scordino and G. Lipari, "A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks," *IEEE Transactions on Computers*, vol. 55, no. 12, pp. 1509–1522, 2006.

[9] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "Fast: Frequency-aware static timing analysis," *ACM Transactions on Embedded Computing Systems (TECS)*, pp. 200–224, 2006.

[10] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, 2016, pp. 2:1–2:10.