Abstract—Nowadays, real-time embedded systems are facing concerns like power consumption and increased functionalities demand. Approaches based on Dynamic Voltage and Frequency Scaling (DVFS) reduce the energy consumed by processors while guaranteeing real-time constraints. In this paper, we present short-comes on existing models reducing energy consumption. Our experimental results clearly show that the execution time of the tasks is not exclusively proportional to the processor speed. Thus, we believe that DVFS techniques could also be applied to other components like busses and memory. We discuss the applicability of a probabilistic model combined with DVFS techniques, and argue that by adopting a probabilistic frequency-aware model, we can (i) capture more detailed behaviors of tasks w.r.t. hardware frequencies and (ii) apply DVFS techniques to gain in energy consumption.

I. INTRODUCTION

During the past years, real-time embedded systems have been confronted to increased processing capabilities (by incorporating modern processors) while expecting to reduce their energy consumption. In fact, many of these systems are battery powered and recharging or changing a battery is not always feasible. Thus, the energy management has become a prime design objective for the conception of real-time embedded 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 few microseconds. The execution time of tasks tends to increase, while energy consumption decreases. DVFS techniques 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 real-time 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 \(\tau_i\) 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), Aggressive 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 accesses take place. Therefore, the hypothesis of a fully scalable WCET can be too optimistic. 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, further improving their energy savings.

C. Extending DVFS to other components

It is clear that DVFS scheduling algorithms are capable of limiting the energy consumption of the processor [1]. Nevertheless, frequency scaling has also been considered on other components of hardware architectures like the bus and memory. Since these components also require a large portion of energy in order to operate [3], scheduling algorithms have been proposed for architectures capable of supporting DVFS at the processor and memory level [3], [4]. The WCET, in these cases, is 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): \( C_i(s_{cpu}, s_{mem}) = C_{cpu} + C_{mem} + C_{fix} \).

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 minimum and maximum 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 (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 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...
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 feasible, we are interested in proposing a new model of tasks. In this paper we choose a task model with WCETs described by probability distributions. We define the probabilistic WCET of a task as the least upper bound on all possible distributions of the execution times of that task. We denote by $C_t$ the pWCET of a task $\tau_t$ as follows:

$$F_{C_t}(c) \geq F_{C_j}(c), \forall c$$

where $F_{C_j}(c) = P(C_j = c)$ is the distribution of the $j$-th set of possible execution times $c$ of the task $\tau_j$.

By using a probabilistic description, we may guarantee a minimum degree of schedulability, while increasing functionalities of modern embedded systems.

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 task’s behavior. Fig. 3 illustrates this: measured execution times for the cpjeg_wrbmp program of the TACLeBench suite under two different CPU frequencies (1400 and 600MHz) are illustrated, in addition to their density functions.

Therefore, our task model defines different pWCET for each task in function of frequencies considered for the different hardware components. 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 mostly executed at runtime. A drawback from considering different pWCETs for each combination of hardware is the storing space. We leave 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. 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.

Thanks to the TACLeBench suite we are able to observe the impact of memory frequency. The Raspberry Pi firmware allows us to set a 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 slowdown had almost no effect on the program execution 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.
following task set being scheduled according a given fixed priority policy on a single-core processor:

\[
\tau_1 : \left( \begin{array}{ccc} 3 & 4 & 5 \\ 0.2 & 0.75 & 0.05 \end{array} \right)^{\text{min}}, \left( \begin{array}{ccc} 2 & 4 & 5 \\ 0.95 & 0.05 & 0.05 \end{array} \right)^{\text{max}}
\]

\[
\tau_2 : \left( \begin{array}{ccc} 2 & 4 & 10 \\ 0.7 & 0.3 & 10 \end{array} \right)^{\text{min}}, \left( \begin{array}{ccc} 1 & 3 & 10 \\ 0.99 & 0.01 & 0.01 \end{array} \right)^{\text{max}}
\]

The \text{min} values are obtained from minimum CPU frequency and all possible memory frequencies. The same convention goes for the \text{max} value of the pWCET. In this example, even if the CPU is running at its maximum speed, the task system would be deemed as non-schedulable for any deterministic scheduling policy. Nevertheless, a probabilistic response time analysis can be used. For instance if we apply the response time analysis from [5] we obtain a probabilistic deadline success ratio of 99.9975%. Moreover, for a minimum CPU frequency, the schedulability ratio is 52.65% which is, 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 starting at minimum speed, then switch to the maximum speed for the last execution of \(\tau_1\) during the hyperperiod, giving a schedulability rate of 96.9%. However, if the first job of \(\tau_1\) finishes at 3 time units and \(\tau_2\) finishes within 2 time units, then the second job of \(\tau_2\) can complete its execution even when the processor is at its minimum speed. This allows the system to complete its execution with its minimal 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 a fully scalable WCET. We have also demonstrated how DVFS techniques can be applied to probabilistic real-time systems without compromising timing guarantees. For future works we plan to fully evaluate the extension of probabilistic DVFS scheduling algorithms in terms of energy reduction and complexity. We plan to compare our model to existing approaches in order to measure the improvement in terms of energy reduction. We also want to propose a method to derive pWCET distributions for tasks with and without frequency scaling hardware as this is a major necessity for probabilistic task models.

REFERENCES


