Document downloaded from:

http://hdl.handle.net/10251/81399

This paper must be cited as:

Sahuquillo Borrás, J.; Hassan Mohamed, H.; Petit Martí, SV.; March Cabrelles, JL.; Duato Marín, JF. (2016). A dynamic execution time estimation model to save energy in heterogeneous multicores running periodic tasks. Future Generation Computer Systems. 56:211-219. doi:10.1016/j.future.2015.06.011.

The final publication is available at

http://dx.doi.org/10.1016/j.future.2015.06.011

Copyright Elsevier

Additional Information

this is the author's version of a work that was accepted for publication in Future Generation Computer Systems. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Future Generation Computer Systems, vol. 56 (2016). DOI 10.1016/j.future.2015.06.011.

A Dynamic Execution Time Estimation Model to Save Energy in Heterogeneous Multicores Running Periodic Tasks

Julio Sahuquillo, Houcine Hassan, Salvador Petit
José Luis March, and José Duato
Department of Computer Engineering (DISCA)
Universitat Politècnica de València
Valencia, Spain
{jsahuqui,husein,spetit,jomarcab,jduato}@disca.upv.es

Abstract

Nowadays, real-time embedded applications have to cope with an increasing demand of functionalities, which require increasing processing capabilities. With this aim real-time systems are being implemented on top of high-performance multicore processors that run multithreaded periodic workloads by allocating threads to individual cores. In addition, to improve both performance and energy savings, the industry is introducing new multicore designs such as ARM's big.LITTLE that include heterogeneous cores in the same package.

A key issue to improve energy savings in multicore embedded real-time systems and reduce the number of deadline misses is to accurately estimate the execution time of the tasks considering the supported processor frequencies. Two main aspects difficult this estimation. First, the running threads compete among them for shared resources. Second, almost all current microprocessors implement Dynamic Voltage and Frequency Scaling (DVFS) regulators to dynamically adjust the voltage/frequency at run-time according to the workload behavior. Existing execution time estimation models rely on off-line analysis or on the assumption that the task execution time scales linearly with the processor frequency, which can bring important deviations since the memory system uses a different power supply.

In contrast, this paper proposes the *Processor-Memory* (*Proc-Mem*) model, which dynamically predicts the distinct task execution times depending on

the implemented processor frequencies. A power-aware EDF(Earliest Deadline First)-based scheduler using the *Proc-Mem* approach has been evaluated and compared against the same scheduler using a typical *Constant Memory Access Time* model, namely *CMAT*. Results on a heterogeneous multicore processor show that the average deviation of *Proc-Mem* is only by 5.55% with respect to the actual measured execution time, while the average deviation of the *CMAT* model is 36.42%. These results turn in important energy savings, by 18% on average and up to 31% in some mixes, in comparison to CMAT for a similar number of deadline misses.

Keywords: Heterogeneous multicore architectures, time predictable multicore architectures, time aware energy efficiency, energy savings, real-time embedded systems

---

1. Introduction

Nowadays, real-time systems are implemented of top of high-performance multicore processors due to the growing functionality demands of the applications. These processors support the execution of multithreaded workloads by allocating each thread to a specific core. Current multicores present resources that are private to cores and resources that are shared among cores. Which resources are designed private and which ones shared vary among commercial machines. Typical resources implemented as private to individual cores are the register file and the first-level caches, and examples of shared resources are the interconnection network and the main memory.

The schedulability problem has been widely studied [1, 2], in distinct types of applications. However, real-time systems require an estimation of the Worst Case Execution Time (WCET) of applications in order to ensure the schedulability. The WCET must be estimated with the highest accuracy in order to provide either a high quality of service in *Soft* Real-Time (SRT) systems (e.g. multimedia or video-streaming) or to prevent possible damages (e.g., automotive or avionics) due to deadline misses in *Hard* Real-Time (HRT) systems running periodic tasks. On the other hand, a high estimation accuracy allows the system to save power, improve schedulability, or both.

Two main characteristics of multicore architectures difficult the estimation of the task execution time. First, the running threads compete among them for shared resources. Second, almost all current multicores implement Dynamic Voltage and Frequency Scaling (DVFS) regulators that permit ad-

justing the voltage and frequency at run-time according to the dynamic workload behavior. This technique allows the system to manage power consumption more efficiently.

Some research works assume that the memory access time (quantified in processor cycles) is constant regardless of the processor frequency. From now on, we refer to this model as *Constant Memory Access Time* (*CMAT*). This model assumes that all of the processor components scale their speed at the same pace, which can bring important deviations in the estimation of the execution time since main memory devices have their own power supply and work independently of the DVFS regulator. Despite this fact, this model is implicitly assumed in important research work like [3, 4, 5]. For instance, in [3], an example is shown where the processor takes 20s (or 1000Mcycles) at 50MHz (i.e. 20ns processor clock) to run a given program, and 25s (i.e. 25ns processor clock) when working at 40MHz. In other words, although not explicitly said, it can be observed that the execution time grows in the same factor as the processor cycle time. To deal with this shortcoming, other researchers have devised alternative models [6, 7] to achieve better estimations. These models, however, are static and rely on either analyzing the workload source code [6] or performing off-line characterization of the architectural parameters [7].

In contrast to previous work, this paper proposes a model that predicts at run-time the execution time of real-time applications running periodic tasks on heterogeneous multicores supporting different *frequency domains* (i. e., local DVFS), without the need of analyzing any source code or hardware platform. Periodic tasks have been used in a wide segment of real-time control applications, ranging from automotive, robotics or avionics [8]. In spite that experimental evaluation section focuses on periodic tasks, the proposed model would also work with aperiodic tasks, since aperiodic servers [9, 10] can process aperiodic workloads without compromising the execution of periodic tasks. We show that the proposed model is highly accurate even when applied to an heterogeneous multicore. The studied system implements cores with different ranges of DVFS levels. This type of designs are being introduced in the embedded market since they can improve performance of both parallel and sequential applications while providing a high energy-efficiency. This is the reason why some recent Samsung smartphone models such as the Galaxy S5 or the Note 4 implement processors based on the ARM big.LITTLE [11] heterogeneous processor architecture.

The proposed model uses the first hyperperiod of the execution of the

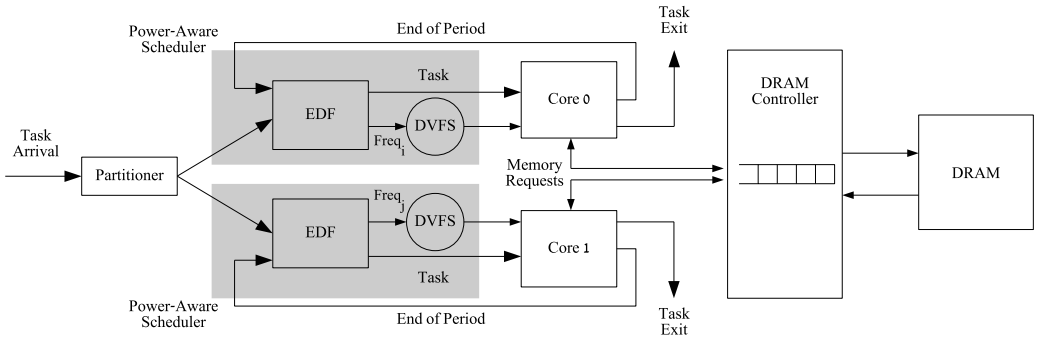


Figure 1: System model.

tasks to investigate the workload characteristics. Then, this information is used by the power-aware EDF-based scheduler to choose the most suitable frequency for the following hyperperiods. In this way, not only important energy savings are achieved but also system schedulability is improved. For instance, if the system utilization decreases and more slack time is available, this extra time could be used to reduce the frequency for energy savings or introduce additional tasks in the system.

This paper proposes the *Processor-Memory* (*Proc-Mem*) model, which predicts the execution time for each individual task and frequency level. To this end, *Proc-Mem* uses performance monitoring counters to measure the time that each core spends performing computation (CPU), waiting for memory (MEM), and overlapping time ($OVERLAP$) between computation and memory access. Since the overlapping time of a given task depends not only of itself but also on the co-running tasks, the input values of the model must be taken at run-time. The proposed model uses the first hyperperiod to gather the required values. Then, the scheduler uses the model estimates to choose the most suitable working frequency in each active period of the following hyperperiods to address both energy and deadline misses (only soft real-time tasks are considered).

A power-aware scheduler using the *Proc-Mem* approach has been evaluated against the same scheduler using a typical *Constant Memory Access Time* model, namely *CMAT*. Results on a heterogeneous multicore processor show that the average deviation of *Proc-Mem* is by 5.55% with respect to the measured execution time, while the average deviation of the *CMAT* model is 36.42%. These results turn in important energy savings, 18% on average and

up to 31% in some mixes, for a similar number of deadline misses. Finally, we would like to remark that the devised frequency selection policy is orthogonal to the implemented scheduling algorithm so it can be implemented for energy savings in any other scheduler.

The rest of the paper is organized as follows. Section 2 presents the baseline system. Sections 3 and 4 propose and validate, respectively, the *Proc-Mem* execution model. Section 5 explains the frequency selection policy. Section 6 describes the related work. Finally, Section 7 presents some concluding remarks.

2. System Architecture

The modeled system, as shown in Figure 1 consists of a heterogeneous superscalar multicore. Heterogeneity comes from the fact that cores can work at a different range of frequency levels, as adopted in recent embedded systems [12].

Table 1 shows the machine parameters. Each core can issue to execution up to two instructions every cycle. Besides, due to important energy constraints in a wide segment of embedded systems, an in-order issue logic has been assumed, as deployed in some embedded processors like the Intel Atom [13]. Nevertheless, to reduce pipeline stalls due to memory latencies, the processor is allowed to dispatch instructions while a memory access is being performed. The working frequency of each core is controlled by a local DVFS regulator [14], that is, cores can run at different speeds. Cores have been assumed to work at the same frequency levels of a Pentium M [15] as depicted in Table 2. The considered DVFS local regulators implement 7 and 4 frequency levels for core 0 and core 1, respectively. The 7L configuration allows the system to work at all the frequencies indicated in the table, whereas the 4L (Low-Power) mode permits running tasks at the four lowest frequencies (1.4, 1.3, 1.2 and 1.1 GHz).

The system executes multiple soft real-time tasks. A soft real-time task is executed during each of its active periods, and it should finish its execution before reaching its deadline. The end of the period and the deadline of a task are assumed to be equal for a more tractable scheduling process. There are also some periods where a task is not active (i.e., inactive periods), so it is not executed. In short, a task arrives to the system, executes during several active periods, leaves the system, remains out of the system for some inactive

Table 1: Machine Parameters.

Microprocessor core	
Issue policy	In order
Branch Prediction	Two-level global history 256 entries BTB, 4096 2-bit saturating counters GHB
Issue width	2 instructions/cycle
# Int ALUs, mult/div	2,1
# FP ALUs, mult/div	2,1
Memory access latency	10 cycles

periods, and then it enters the system again. This sequence of alternative active and inactive periods allows modeling real systems with mode changes.

2.1. Partitioning and Scheduling

The system implements a partitioner module (labeled as partitioner in Figure 1) that is in charge of distributing tasks among cores. In this regard, Worst Fit (WF) algorithm is considered as one of the best choices to balance the workload [16]. This algorithm requires task utilization values which are obtained as the quotient between the WCET of the task to its period as shown in Equation 1. To obtain the utilization, the WCET is typically estimated for each task in a stand-alone execution.

$$U = \frac{WCET}{Period} \tag{1}$$

The WF algorithm balances the workload by assigning each incoming task to the least loaded core. If more than one task arrives to the system at the same time, WF arranges the incoming tasks in a decreasing utilization order and assigns them to the cores starting with the task with the highest utilization.

Table 2: Frequency (F) vs power (P).

F[GHz]	1.7	1.6	1.5	1.4	1.3	1.2	1.1
P[Watts]	24.5	24.5	24.5	22	22	12	12

Since in the modeled system each core works with a different set of frequency levels, the WF policy must be properly adjusted. For this purpose, the following extension has been adopted. In case that the partitioner detects that core 1 is fully loaded (100% utilization) when working at its maximum working frequency (1.4 GHz), then the new incoming tasks are allocated to core 0, even if this action introduces imbalance into de system, since core 0 can work at higher frequencies than core 1.

To simplify the implementation, the system assumes that once a task is allocated to a given core, it is inserted into the task queue of that core, where incoming tasks are ordered according to the EDF scheduling algorithm, which prioritizes the execution of tasks with the closest deadlines.

The devised schedulers are also in charge of calculating the required target speed of each core according to its utilization. In this sense, the power-aware EDF scheduler implemented in each core chooses the minimum frequency that fulfills the temporal constraints of its task set in order to minimize power consumption.

2.2. Memory System

Regarding the memory system, all cores send their memory requests to a common memory controller that handles the accesses to a shared scratchpad memory. The memory controller handles its internal request queues according to the FCFS-RR (First-Come First-Served, Round-Robin) policy [17].

The scratchpad memory is composed of eight banks. Bank conflicts are taken into account so that if a bank is being accessed, and a younger request demands the same bank, this request waits at the memory controller until the previous access finishes. Otherwise, the new request can proceed.

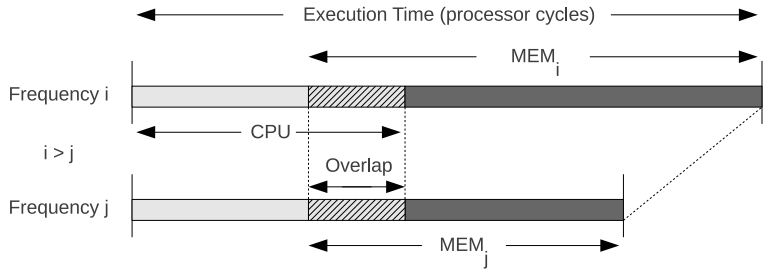


Figure 2: Execution time model.

3. Processor-Memory Model

The execution time of a task can be considered as composed by two main components, (CPU) time and the memory (MEM) time. The former can vary according to data, structural and control hazards presented by the application at the core side, while the latter grows with the number of memory accesses. Both times can overlap since the core can dispatch instructions while the memory is being accessed. This time, from now on referred to as $OVERLAP$, is defined as the time the processor is executing non-dependent instructions while a memory request is being served.

Figure 2 depicts a simplified overview of the execution time components. The model assumes that performance monitoring counters (PMC) [18] typically implemented in most current processors, are available in the multicore system. These registers allow the processor to gather multiple variables. This paper assumes that the variables required by the scheduler can be gathered in the target processor.

Since the MEM value is gathered in processor cycles, the smaller the processor cycle time (i.e., the higher the processor frequency) the higher the MEM value gathered in the corresponding performance counter. Therefore, the MEM value for a target frequency j can be estimated from the MEM value collected for the current frequency i as given by Equation 2. This effect can be appreciated in Figure 2. It also can be appreciated that if the elapsed time is quantified in processor cycles, the CPU value remains constant.

$$MEM_j = MEM_i \times \frac{Freq_j}{Freq_i} \tag{2}$$

On the other hand, in-order processors write the results to the corresponding destination register in program order at the writeback (WB) stage. This means that after a long latency event (e.g. a memory access), the execution (EX) stages of subsequent instructions are delayed to perform the WB stage in program order. The memory latency in current systems, regardless of the working frequency, is typically much longer than that of arithmetic operators; thus, the number of instructions that can be executed while a previous memory request is being processed remains constant to comply with the WB order. In other words, the $OVERLAP$ time, measured in processor cycles, can be assumed to be constant.

Figure 3 illustrates this behavior. It depicts, a possible instructions-time diagram corresponding to the execution of five instructions (one memory ref-

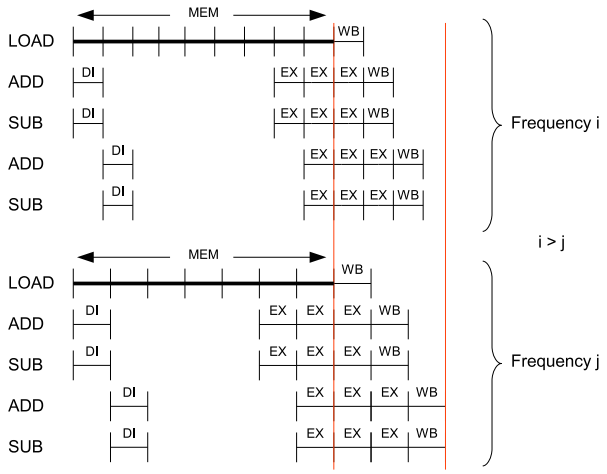


Figure 3: Execution overlap between processor and memory for two different frequencies in a superscalar architecture.

erence and four arithmetics) that overlap their execution in a 2-instruction issue width superscalar processor working at two different frequencies i (upper side) and j (lower side). This example assumes 3-cycle latency for arithmetic operations. The number of overlapped cycles is exactly the same in both frequencies due to the part of CPU time overlapping with the memory access is fixed (all stages DI, two stages EX of the two first arithmetic instructions and one EX stage of the two latter arithmetic instructions). This fact is taken into account in Equation 3, which estimates the total execution time in cycles for a given working frequency j. Note that both CPU and $OVERLAP$ values do not depend on the working frequency.

$$Texe_j = CPU - OVERLAP + MEM_j \tag{3}$$

Substituting MEM_j from Equation 2 in Equation 3, we can derive Equation 4, which estimates the overall execution time for any working frequency j as a function of the inputs of the model gathered during the program execution at a different working frequency i. Note that the computational cost of the equation is negligible. Besides, the overhead caused by the libraries to access the hardware counters required as input for the equation is also very low.

$$Texe_j = CPU - OVERLAP + MEM_i \times \frac{Freq_j}{Freq_i} \quad (4)$$

Finally, it is important to note that in multicore processors, where different cores compete among them for shared resources, new interferences appear. These interferences mainly rise when memory requests from different cores reach the memory controller, where they are scheduled for main memory access. Consequently, the individual execution time (measured either in cycles or in temporal units) of each task increases with respect to stand-alone execution in a single-core system. In spite of this fact, as shown in the next section, the proposed model remains mostly valid in this scenario since the extra time waiting at the memory controller is already included in the *MEM* component.

4. Model Validation

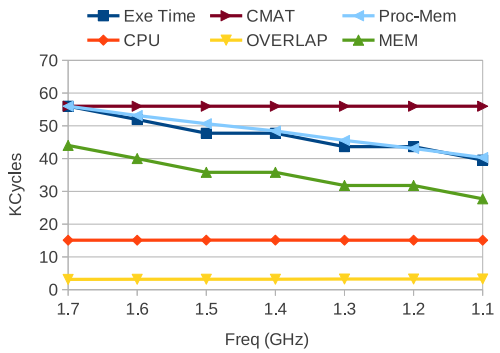
Simulators are a commonly used tool to validate and evaluate novel approaches in many research fields [19, 20]. In this particular case, to evaluate the goodness of the model the Multi2Sim [21] simulation framework has been used, which is a detailed cycle-by-cycle execution driven simulator of multicore and multithreaded processor architectures. Multi2Sim is being used by companies like AMD or NVIDIA. It has been extensively extended to model the real-time system and power related characteristics, including the power-aware scheduler, the DVFS regulators, and the memory controller (see [22] for more details).

Experiments have been conducted with 20 different benchmarks from WCET Malardalen Project [23], although for illustrative purposes, only the results of two representative benchmarks (*fir* and *sqrt*) are presented. These benchmarks are executed in 2-way superscalar cores that include multicycle operators whose latencies range from 1 cycle to 3 cycles.

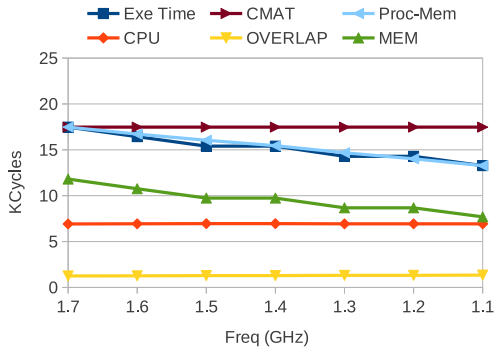
Figure 4 compares the execution time estimates provided by the *Proc-Mem* and the *CMAT* models for fir (upper side) and sqrt (lower side) benchmarks. The latter assumes a constant number of processor cycles for each memory access (i.e. the memory speed depends on the DVFS frequency). The stand-alone execution time (labeled as *Exe Time*) of both benchmarks in a single-core superscalar architecture is also represented. Model inputs were taken when running the processor at the highest frequency, that is, at 1.7 GHz. The remaining points of the plot (from 1.6 GHz down to 1.1

GHz) were estimated with the studied models. For comparison purposes, the three main components (*CPU*, *OVERLAP* and *MEM*) of the execution time are represented. As observed, *CPU* and *OVERLAP* values keep constant for both benchmarks, while *MEM* increases with the frequency, which illustrates that *Proc-Mem* is suitably designed to model the target system.

Proc-Mem estimates closely follow the execution time regardless the target frequency; in contrast, the error introduced by the *CMAT* model grows



(a) Fir



(b) Sqrt

Figure 4: Estimates of the *Proc-Mem* model in stand-alone execution in the single-core superscalar architecture.

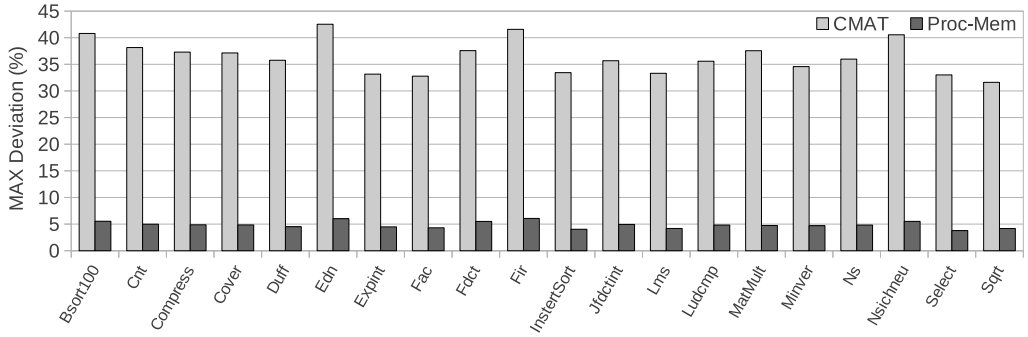


Figure 5: Maximum deviation in processor cycles in a single-core superscalar architecture.

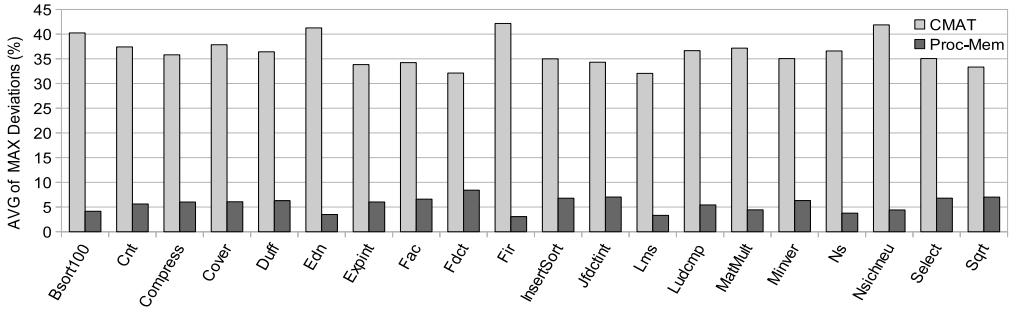


Figure 6: Average of maximum deviations in a multicore superscalar architecture.

as the frequency moves away from 1.7 Ghz. *Proc-Mem* deviation is on average around 2.5% and 1.5% depending on the benchmark, and never exceeds 5.5%. In contrast, the *CMAT* approach incurs in a noticeable higher error, which is already around 7% for both benchmarks in the first 100MHz away from 1.7 GHz. Moreover, this deviation worsens as the frequency decreases, exceeding 30% in both benchmarks at 1.1 GHz.

The maximum deviation for *Proc-Mem* and *CMAT* accross a relatively wide set of 20 evaluated benchmarks is shown in Figure 5. *Proc-Mem* error is always around 5%, while that incurred by *CMAT* is always over 30%.

Once it has been proven that the proposal provides good estimates for single-core execution, *Proc-Mem* results are explored in a two-core architecture. To this end, we have performed an exhaustive study with multiple

experiments by concurrently executing all the possible pairs of benchmarks for the different working frequencies. In these experiments we assume that both cores work at the same frequency. Figure 6 shows the average of the maximum deviations incurred by *CMAT* and *Proc-Mem* for each benchmark across all the experiments. The deviation of *Proc-Mem* for all benchmarks is, on average, 5.55% with respect to the measured execution time, while the deviation of the *CMAT* model is, on average, 36.42%.

For illustrative purposes, Figure 7 presents the results of the previously studied benchmarks (*fir* and *sqrt*) when executing concurrently for the different frequencies. In this case, the maximum deviation of the *CMAT* model is 39.8% for *fir* and 32.7% for *sqrt*, whereas in the *Proc-Mem* model the maximum deviation is 6.9% and 7.3% for *fir* and *sqrt* respectively. Moreover, the plots confirm the robustness of the proposed model even in the presence of interferences due concurrent execution across all the experiments.

5. Frequency Selection Policy based on the Proc-Mem Model

5.1. Scheduler Working Behavior

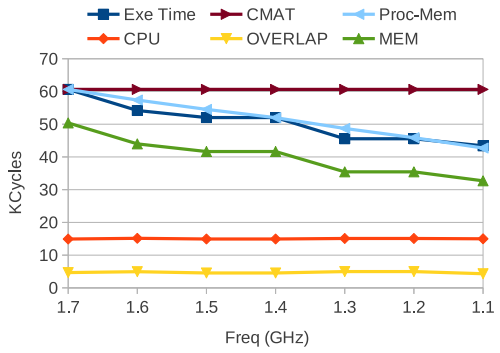
The *Proc-Mem* model provides estimates of the execution time for the different instances of the tasks for each target frequency. These estimates can be used for several purposes by the scheduler such as to predict the task utilization or to choose the target frequency. This section illustrates how the estimates provided by the model can help the power-aware scheduler to choose the most suitable DVFS levels to both save energy or address deadline misses. This component of the scheduler will be referred to as frequency selection policy. Next, the proposed policy is presented.

The actions performed by the scheduler at run-time using the devised frequency selection policy are depicted in Figure 8 for the studied two-core processor. Actions mainly differ depending on the considered hyperperiod; the first one, namely H0, is used as a sampling period to obtain the workload characteristics to be used by the model in the following hyperperiods.

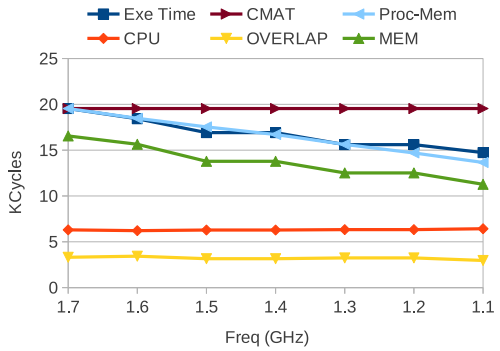
In hyperperiod H0, performance monitoring counters (PMC) are used to gather the main components of the execution time (*CPU*, *OVERLAP*, and *MEM*). This is done for each active period of the different tasks. Each core of the bi-core system is assumed to work at a single frequency during the whole hyperperiod. This initial frequency can be any of the available in the DVFS regulators that ensures meeting all task deadlines, although this

example assumes that each core works at its maximum frequency (i.e. core 0 works at 1.7 GHz and core 1 at 1.4 GHz.)

When hyperperiod H0 expires, the inputs required by the *Proc-Mem* model (i.e., *CPU*, *OVERLAP*, and *MEM*) have already been gathered for each active period. Then, the proposed frequency selection policy is applied to select the target core frequencies to be used in subsequent active periods of the following hyperperiods. The devised policy uses the model to estimate the execution time at each frequency for each task and period. Taking into



(a) Fir



(b) Sqrt

Figure 7: Estimates of the *Proc-Mem* model in a multicore processor.

account these estimates, the policy chooses the lowest frequency for each period that fulfills its deadline while maximizing energy savings. In addition, the obtained estimates are also used by the power-aware scheduler to correct possible deviations in the WCET that affect tasks utilizations, which are required to perform partitioning and scheduling actions (see Section 2.1).

Notice that a frequency change will affect all subsequent tasks' finish time, so the algorithm is not designed to guarantee that deadlines will not be missed for other tasks. That requirement implies additional analyses that are out of the scope of this work.

5.2. Experimental Results

To evaluate the proposed scheduler, it has been compared to a variant using the *CMAT* model in terms of energy and deadline misses. To this end, we have designed eight mixes consisting of benchmarks that are executed multiple times in different active periods across the hyperperiod. Table 3 presents the mix composition and the number of instances of each individual benchmark in one hyperperiod. Benchmarks were randomly selected to build the mixes. We designed mixes with different number of tasks (2, 4, and 5 tasks) and different number of active periods per task (varying from 1 up to 90 active periods) to explore the behavior of the proposal in a wide range of scenarios.

Figure 9 shows the normalized energy consumption during hyperperiod H1 of both proposals with respect to a system working always at the maximum speed for eigth different mixes of benchmarks. To calculate the energy consumption, we measured the number of cycles that each core spends at

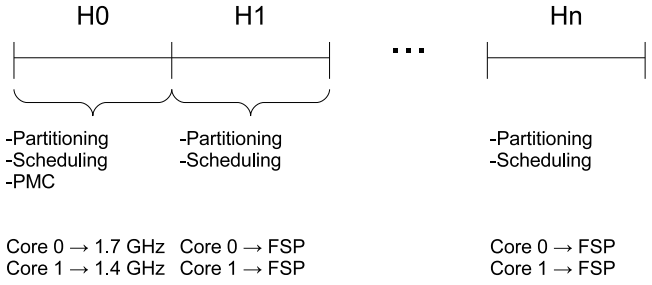


Figure 8: Power-aware scheduler actions of the system across the hyperperiods. Legend: FSP - frequency selection policy.

Table 3: Mix composition: benchmarks and instances of each benchmark.

Mix	Benchmark (instances)				
Mix 1	Bsort100 (1)	Cnt (16)			
Mix 2	Bsort100 (1)	Cnt (90)			
Mix 3	Bsort100 (1)	Cnt (6)	Compress (3)	Cover (3)	
Mix 4	Bsort100 (1)	Cnt (10)	Compress (10)	Cover (10)	
Mix 5	Bsort100 (1)	Cnt (6)	Compress (10)	Cover (16)	
Mix 6	Duff (5)	Edn (1)	Expint (13)	Fac (13)	Fdct (3)
Mix 7	Duff (12)	Edn (1)	Expint (15)	Fac (24)	Fdct (15)
Mix 8	Duff (12)	Edn (1)	Expint (16)	Fac (32)	Fdct (15)

each working frequency and multiplied these values by the energy consumed per cycle at the corresponding frequency. Compared to the *CMAT* model, *Proc-Mem* significantly reduces power consumption across all the mixes. The reason is that *Proc-Mem* provides estimates of the execution time that are much shorter than those provided by the *CMAT* model, which allows the scheduler to select lower frequencies. Energy reduction is as high as 31% in some cases, and around 18% on average. Notice that in mixes 1, 3 and 4, Proc-Mem consumes half the energy of the system working at the maximum speed. The reason is that when executing these mixes with the *Proc-Mem* based scheduler, the system runs all the time at the minimum speed available in both local DVFS regulators, whereas when using *CMAT* higher frequencies are used due to longer execution time estimates. We found that *CMAT* estimate values grow with the weight of the memory access time over the execution time.

As mentioned above, this work focuses on soft real-time systems, where deadline misses are allowed. In these systems, the implemented schedulers must tradeoff energy to deadline misses; that is, energy saving must be achieved but guaranteeing a minimum quality of service (quantified in deadline misses). Table 4 presents the number of misses in the experiments of the *Proc-Mem* and *CMAT* models for each mix. The number of active periods of the tasks of a mix during a hyperperiod is also included. As observed, in four mixes there is not any deadline miss neither in *Proc-Mem* nor in *CMAT*. Nevertheless, in three of them (i.e., Mix 1, Mix 3 and Mix 4) *Proc-Mem* model energy savings are higher (i.e. by 22%) than when using *CMAT*.

Some deadline misses appear in the remaining mixes, where two cases

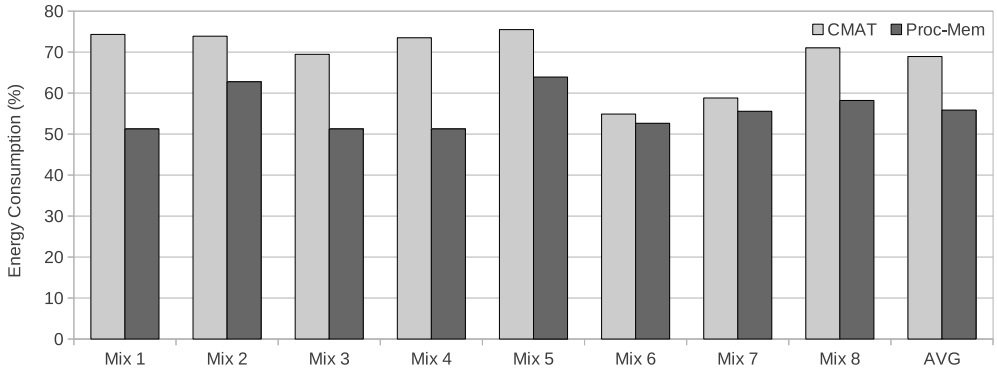


Figure 9: Normalized energy consumption of Proc-Mem and CMAT with respect to a system working at the maximum speed.

can be distinguished. On the one hand, the same number of misses rises in mixes 5, 7 and 8. However, in this case, *Proc-Mem* is more energy efficient (by 10%) than *CMAT*. On the other hand, in mix 2, *Proc-Mem* misses two deadlines while no deadline is missed with *CMAT*. Comparing the latter result to performance, *Proc-Mem* misses by 2% of the active periods' deadlines, while savings in energy consumption are by 15%. Therefore, we can conclude that estimating tight execution times can incur in a limited amount of additional deadline misses. However, in this case, important energy savings can be brought.

Table 4: Deadline misses in the CMAT and Proc-Mem models and active periods of the mixes.

Mix	Deadline Misses		# Active Periods
	CMAT	Proc-Mem	
Mix 1	0	0	17
Mix 2	0	2	91
Mix 3	0	0	13
Mix 4	0	0	31
Mix 5	1	1	33
Mix 6	0	0	35
Mix 7	5	5	67
Mix 8	1	1	76

6. Related Work

A number of static execution time analysis methods have been designed in the last two decades, mainly for monoprocessors [24]. These research works use execution time estimates in order to choose the optimal DVFS level to enhance power savings and reduce deadline misses. Seth et al. [6] study the effects of DVFS on static timing analysis taking into account power consumption. They calculate the WCET in any frequency range using a parametric model that depends on the number of cache misses. However, their approach is static and needs the source files of the applications. Another model is proposed by Snowdon et al. [7]. They perform an on-line evaluation of application characteristics using performance counters, nevertheless, it is combined with an off-line characterization of the hardware platform. Miftakhutdinov et al [25] propose a DVFS performance predictor for memory systems with a streaming prefetcher. They also take into account that memory latencies (as measured in seconds) are not affected by chip frequency scaling. However, their proposal is static, without real-time constraints and does not consider multiple cores. Unlike these works, our proposal estimates dynamically the execution time and focuses on heterogenous multicore processors.

Execution time estimation for multicore platforms has been the subject of rather few studies. Most of them focused on cache-aware methods. Hardy et al. [26] present a WCET estimation method for multicore platforms with shared instruction caches. The proposed method provides estimates through the control of the contents of the shared instructions cache, more precisely by caching only the blocks statically known to be reused and bypassing from shared caches the other blocks. Predictability becomes harder when considering the effect of caches embedded in the NoC, like the ones described in [27]. More advanced CMP solutions like the presented in [28, 29] or how feedback driven restructuring techniques can be utilized to improve power savings jointly with the approach proposed in this paper, and are planned to be addressed as for future work.

There are some works that study the effect of the main memory in the estimation of the execution time for multithreaded and multicore architectures but do not address energy savings. Shah et al. [30] make use of bank interleaving and applying Priority based Budget Scheduling (PBS) to share SDRAM among multiple masters. This technique permits to bound the WCET of an application accessing a shared SDRAM using the worst case access pattern. They implemented the memory system in an FPGA. Their proposal produces

safe and low WCET bounds. Ungerer et al. [31] build a predictable multicore architecture for mixed critical applications. Predictability is achieved by giving the highest priority to the hard real-time task while on the shared bus access latency is bounded by a round robin (RR) arbiter. The memory is accessed through an analyzable memory controller (AMC), which implements bank interleaving. Through theoretical analysis, latency parameters are extracted to calculate the WCET. The AMC applies the maximum of Read/Write and Write/Read switching latencies as a constant worst case latency on every access. Such assumption, while making the analysis simple, cannot produce precise bounds. Moreover, the RR policy with one request per master cannot satisfy the need of different bandwidth requirements. If more than one request per master is assigned, the WCET is severely degraded [30].

7. Conclusions

Accurately estimating task execution time in a real-time multicore embedded system supporting DVFS is a critical issue for enhancing the schedulability of the system and improving energy savings. Since this kind of systems support multiple core speeds, different execution times should be estimated, one for each speed. This paper has proposed the *Proc-Mem* model that estimates the execution times, for each instance of a real-time task, at the available frequencies in the multicore. These estimates are used by a power-aware scheduler to choose the most suitable working frequency to address both energy and deadline misses.

To provide accurate execution times, *Proc-Mem* uses performance monitoring counters to measure the time that each core spends performing computation (CPU), waiting for memory (MEM), and overlapping time ($OVERLAP$) between computation and memory access. Based on this information, *Proc-Mem* estimates the execution times for each task and frequency level.

We have devised a frequency selection policy that uses the *Proc-Mem* model to reduce energy consumption while incurring in scarce deadline misses. Compared to the *Constant Memory Access Time* model used in recent works, the use of *Proc-Mem* allows the Power-Aware EDF scheduler to significantly improve energy savings without significantly increasing the number of deadline misses. Experiments show that the accuracy of the proposed model allows the system to reach energy savings by 18% on average, and up to 31% in some workloads. As for future work we plan to explore the use of

the *Proc-Mem* model considering the influence of a Network on Chip in the schedulability and power savings.

Acknowledgement

This work was supported by the Spanish Ministerio de Economía y Competitividad (MINECO) and by FEDER funds under Grant TIN2012-38341-C04-01, and by the Intel Early Career Faculty Honor Program Award.

References

[1] J. Daz, S. Reyes, A. Nio, C. Muoz-Caro, Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers, Future Generation Computer Systems 25 (6) (2009) 617 – 626.

[2] A. Burkimsher, I. Bate, L. S. Indrusiak, A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times, Future Generation Computer Systems 29 (8) (2013) 2009 – 2025.

[3] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on, IEEE, 1998, pp. 197–202.

[4] W.-C. Kwon, T. Kim, Optimal voltage allocation techniques for dynamically variable voltage processors, ACM Transactions on Embedded Computing Systems (TECS) 4 (1) (2005) 211–230.

[5] M. E. T. Gerards, J. Kuper, Optimal DPM and DVFS for Frame-based Real-Time Systems, ACM Transactions on Architecture and Code Optimization 9 (4) (2013) 41:1–41:23.

[6] K. Seth, A. Anantaraman, F. Mueller, E. Rotenberg, FAST: Frequency-Aware Static Timing Analysis, in: Proceedings of the 24th International Real-Time Systems Symposium, IEEE Computer Society, Cancun, Mexico, 2003, pp. 40–51.

[7] D. C. Snowdon, G. V. D. Linden, S. M. Petters, Accurate Run-Time Prediction of Performance Degradation under Frequency Scaling, in: Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Pisa, Italy, 2007.

[8] G. Buttazzo, E. Bini, D. Buttle, Rate-adaptive tasks: Model, analysis, and design issues, in: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, Dresden, Germany, 2014, pp. 1–6.

[9] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 3rd Edition, Springer Publishing Company, Incorporated, 2011.

[10] P. Marwedel, Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, Springer Publishing Company, Incorporated, 2011.

[11] A. Peter Greenhalgh, Big. little processing with arm cortex-a15 & cortex-a7 (2011).

[12] ARM big.LITTLE Processing, [Online]. Available: `http://www.arm.com/products/processors/technologies/biglittleprocessing.php` (ARM Holdings).

[13] T. R. Halfhill, Intel's Tiny Atom: New Low-power Microarchitecture Rejuvenates the Embedded x86, Micro Report 22 (4) (2008) 1.

[14] J. Donald, M. Martonosi, Techniques for Multicore Thermal Management: Classification and New Exploration, in: Proceedings of the 33rd International Symposium on Computer Architecture, IEEE Computer Society, Boston, MA, USA, 2006, pp. 78–88.

[15] INTEL Corp., Santa Clara, CA, USA, Intel Pentium M Processor Datasheet, [Online]. Available: `http://download.intel.com/support/processors/mobile/pm/sb/25261203.pdf` (2004).

[16] T. A. AlEnawy, H. Aydin, Energy-Aware Task Allocation for Rate Monotonic Scheduling, in: Proceedings of the 11th Real Time and Embedded Technology and Applications Symposium, IEEE Computer Society, San Francisco, CA, USA, 2005, pp. 213–223.

[17] E. Ipek, O. Mutlu, J. Martinez, R. Caruana, Self-Optimizing Memory Controllers: A Reinforcement Learning Approach, in: 35th International Symposium on Computer Architecture, ISCA, 2008, pp. 39–50.

[18] S. Eranian, What Can Performance Counters Do for Memory Subsystem Analysis?, in: Proceedings of ACM SIGPLAN workshop on memory systems performance and correctness (ASPLOS'08), ACM, 2008, pp. 26–30.

[19] M. Bux, U. Leser, Dynamiccloudsim: Simulating heterogeneity in computational clouds, in: Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, SWEET '13, ACM, New York, NY, USA, 2013, pp. 1:1–1:12.

[20] S. Camarasu-Pop, T. Glatard, R. F. da Silva, P. Gueth, D. Sarrut, H. Benoit-Cattin, Monte carlo simulation on heterogeneous distributed systems: A computing framework with parallel merging and checkpointing strategies, Future Generation Computer Systems 29 (3) (2013) 728 – 738, special Section: Recent Developments in High Performance Computing and Security.

[21] R. Ubal, J. Sahuquillo, S. Petit, P. López, Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors, in: Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing, IEEE Computer Society, Gramado, RS, Brazil, 2007, pp. 62–68.

[22] J. March, J. Sahuquillo, H. Hassan, S. Petit, J. Duato, Extending a Multicore Multithread Simulator to Model Power-Aware Hard Real-Time Systems, in: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, Springer-Verlag, Berlin, Busan, Korea, 2010, pp. 444–453.

[23] Mälardalen Real-Time Research Center, Västerås, Sweden, WCET Analysis Project. WCET Benchmark Programs, [Online]. Available: `http://www.mrtc.mdh.se/projects/wcet` (2006).

[24] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, The Worst-Case

Execution Time Problem - Overview of Methods and Survey of Tools, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) 36:1–36:53.

[25] R. Miftakhutdinov, E. Ebrahimi, Y. N. Patt, Predicting Performance Impact of DVFS for Realistic Memory Systems, in: Proceedings of the 45th International Symposium on Microarchitecture, 2012, pp. 155–165.

[26] D. Hardy, T. Piquet, I. Puaut, Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches, in: 30th IEEE Real-Time Systems Symposium, 2009, 2009, pp. 68–77.

[27] C. Kim, D. Burger, S. W. Keckler, An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, SIGARCH Comput. Archit. News 30 (5) (2002) 211–222.

[28] P. Foglia, M. Solinas, Exploiting replication to improve performances of nuca-based cmp systems, ACM Trans. Embed. Comput. Syst. 13 (3s) (2014) 117:1–117:23.

[29] S. Bartolini, P. Foglia, M. Solinas, C. A. Prete, Feedback-Driven Restructuring of Multi-threaded Applications for NUCA Cache Performance in CMPs, in: Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing, IEEE Computer Society, 2010, pp. 87–94.

[30] H. Shah, A. Raabe, A. Knoll, Bounding WCET of applications using SDRAM with Priority Based Budget Scheduling in MPSoCs, in: Design, Automation Test in Europe Conference Exhibition, 2012, pp. 665–670.

[31] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiones, M. Gerdes, M. Paolieri, J. Wolf, H. Cass, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, J. Mische, Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability, Micro, IEEE 30 (5) (2010) 66–75.