On the Effectiveness of Energy-Aware Real-Time Scheduling Algorithms on Single-Core Platforms

Mario Bambagini[1,2], Marko Bertogna[2] and Giorgio Buttazzo[1]

[1]Scuola Superiore Sant'Anna, Pisa, Italy
[2]University of Modena and Reggio Emilia, Italy

***Abstract*—Energy-aware scheduling is a challenging problem that has been studied for decades, investigating the trade-off between performance and energy consumption.**

In early CMOS circuits, Dynamic Voltage and Frequency Scaling (DVFS) techniques allowed drastically reducing the power consumption. Recent technological advancements have decreased the portion of dissipation which is affected by speed scaling, making Dynamic Power Management (DPM) algorithms more effective. However, the adoption of simplistic power models often biased the decision on which technique to adopt, decreasing the effectiveness of the selected implementation.

This paper discusses the factors to consider when deciding which technique to implement on a given single-core architecture, highlight the limitations of the current mainstream.

I. Introduction

Energy saving algorithms became crucial in actual embedded platforms for extending the lifetime of battery-operated devices.

The two techniques to reduce energy dissipation are Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM). DVFS strategies aim at reducing the system performance by adjusting the voltage and/or frequency to reduce the overall energy consumption. As scaling frequency down makes execution times longer, the objective of this technique is to exploit the slowest frequency that still guarantees real-time constraints. Conversely, DPM techniques aim at switching the processor in a low-power inactive state for the longest possible time, thus postponing tasks execution as long as possible, still guaranteeing the task real-time constraints.

In CMOS technology, power consumption is due to both dynamic and leakage components, which are ascribable to system activity and static dissipation, respectively. Unless the system is off, the static contribution is always present, independently of the actual performance level. Thus, DVFS approaches that modify the clock frequency are more suitable for reducing the dynamic power, whereas DPM solutions are best suited for decreasing the impact of the static component.

Historically (before 2005), CMOS circuits used to operate at high supply voltages, much higher than their threshold voltages, making the impact of dynamic power consumption overwhelming the static dissipation. This fact caused a proliferation of DVFS approaches, which are more suitable for reducing dynamic power. With the progress of electronic technologies, miniaturization has considerably shrunk transistor size, reducing the supply voltage to decrease the dynamic power consumption. Even though the threshold voltage has also been lowered, the gap between supply and threshold voltages has been significantly reduced, leading to an exponential increase of the leakage consumption [1], [2], [3]. Since static dissipation has become the dominant cause of power consumption, DPM approaches are today preferred over DVFS ones, as they extend the time spent in low-power states.

Paper contribution: The main contribution of this work consists of analyzing which scheduling technique is more effective for actual single-core architectures. The principal DVFS and DPM algorithms are compared on three representative platforms, evaluating their behaviors for different task characteristics. The current belief [4], [5], [6], [7], [8] that considers DVFS algorithms less effective than DPM ones is questioned first, evaluating the system parameters that mostly affect the validity of this assumption on actual hardware. Moreover, although multi-core architectures are steadily growing in importance, single-core systems are still of interest because many results can be used locally on each core in partitioned approaches.

Detailed experiments are presented to show the technique that is more suitable under each considered scenario.

Paper organization: Section II details the state of art concerning the problem under analysis. Section III introduces the system and power models. Then, Section IV shows a motivational example. Section V details the considered algorithms while Section VI provides the power models of several real processors. Section VII analyzes the algorithm performance and Section VIII ends the paper with the final remarks.

II. State of art

One of the first papers about power management exploiting frequency scaling was due to Yao et al. [9]. The authors proposed an offline algorithm that, given a task set, computes the minimum energy schedule under the Earliest Deadline First scheduling (EDF) algorithm [10]. Then, they use an online method to scale the speed according to the actual workload at every scheduling event. The analysis compares the efficiency

This work has been supported by the 7th Framework Programme P-SOCRATES (FP7-ICT-2013.10) project, founded by the European Community under grant agreement n. 611016.

of the algorithm with respect to different power models, but without taking switching overheads into account.

Aydin et al. [11] proposed three algorithms with growing complexity. The first one computes the lowest CPU speed such that the task set is schedulable under the assumption that all tasks execute for their worst case. The second algorithm (DRA) keeps track of the times at which a task is going to be dispatched. At runtime, if a task is dispatched earlier, the CPU is slowed down to prolong the execution until the original finishing time. The third algorithm (AGR) estimates the tasks completion times based on past instances and computes the lowest CPU speed to keep the task set feasible assuming that tasks execute for such estimates. However, since the estimations can be optimistic, the algorithm may speed the CPU up to recover from a task overrun.

Pillai and Shin [12] proposed three algorithms considering both EDF and RM scheduling policies. The first approach runs offline and exploits only the static slack: when the system starts, the running speed is set equal to the lowest available one which guarantees the task set feasibility. Then, the Cycle Conservative algorithm (cc-EDF and cc-RM) is introduced which, at every scheduling event, sets the running speed as the slowest available one that guarantees timing constraints using the last execution time for terminated tasks (whose new job has not arrived yet) and the worst case for the other ones. The last proposed algorithm, called Look-Ahead RT-DVS, runs only under EDF and aims at further reducing the running speed by executing the task with the earliest deadline until its time limit at the slowest possible speed, while pushing pending and incoming jobs as close as possible to their deadlines. Although the actual speed until the first deadline can be really low, for executing the other tasks a high speed may be necessary. However, this side effect is significantly reduced thanks to task early terminations.

The problem of obtaining an optimal frequency from a discrete frequency range was discussed by Bini et al. [13]. The authors provided a method for computing the optimal speed offline (that could be unavailable in a specific architecture) and introduced a speed modulation technique to achieve the required speed using two discrete values. The analysis selects the pair of frequencies that minimizes energy consumption also considering switching overheads. Despite its innovative contribution, such an offline approach does not take advantage of tasks early terminations to further reduce consumption.

Some authors [14], [15] reported that online DVFS techniques that frequently scale the execution speed may lead to transient faults. Another side effect of DVFS techniques was emphasized by Kim et al. [16], who noticed that such algorithms increase the number of preemptions, leading to a higher system utilization and, therefore, a higher energy consumption. To mitigate such a problem, they proposed two preemption control DVFS techniques.

The raising impact of leakage power in modern architectures, highlighted by Kim et al. [4] and empirically tested by Bambagini et al. [5], is driving the research on power management toward DPM techniques.

Lee et al. [6] proposed two leakage control algorithms for procrastinating tasks execution as long as possible, both under dynamic (LC-EDF) and fixed (LC-DP) priority scheduling. Using a dual priority scheme [17], LC-DP computes the longest delay (*promotion time*) each task can suffer still satisfying its deadline.

Jejurikar et al. [18] proposed an approach (CS-DVS-P) based on the critical speed analysis and task procrastination working for periodic tasks under EDF. First, an offline DPM algorithm computes the maximum time each task can spend in the sleep state within its period. Then, at runtime, sleep management is delegated to an external controller that switches the system off for the corresponding pre-computed time. Jejurikar and Gupta [19] extended the previous method to consider early terminations and fixed priority scheduling [20].

Chen and Kuo [7] proposed two solutions for fixed priority systems. The first method simulates the execution of periodic tasks to compute the idle time available until the next deadline. Then, such a time is used to postpone the task activations and switch the system into the sleep state. The other algorithm introduces the *virtual blocking*, which is the maximum blocking that tasks can suffer, to extend the procrastination interval.

Awan and Petters [8] proposed to accumulate task execution slack to switch the processor off during such intervals under EDF, considering tasks with different criticality and several low-power states and different break-even times. However, tasks are always executed at the maximum speed.

Huang et al. [21], [22] proposed an offline analysis that combines DPM and Real-Time Calculus to estimate tasks arrivals, compute the CPU idle intervals and then, at runtime, modulate between active (at maximum speed) and sleep states. This approach leads to sleep intervals that are generally smaller and more frequent than those obtained by procrastination algorithms.

Bambagini et al. [23] proposed an algorithm for fixed priority tasks which exploits the limited preemptive scheduling model to further reduce energy consumption with respect to the fully-preemptive model. More precisely, at design time, when the non-preemptive regions are computed, the slowest feasible speed and the minimum among all the blocking tolerances (maximum time interval during which a task can be blocked from the execution of lower priority tasks) are also returned. At runtime, when an idle event occur, the inactivity is extended for the minimum blocking tolerance among all the tasks, delaying the execution of the incoming jobs and prolonging the time spent in a low-power state.

With respect to other papers [24], [25], [26], [27] which compared energy-aware scheduling algorithms, our proposal extends the analysis to DPM solutions rather than focusing only on the DVFS approach.

Moreover, many papers [28], [29], [30] have been proposed for modeling the power dissipation of the processors, providing effective algorithms for estimating the actual power consumption at runtime. However, an offline model has been preferred in this work as it provides an excellent trade-off between representativeness and simplicity.

III. System model

We consider a system consisting of a task set Γ composed of n independent periodic tasks, $\{\tau_1, \tau_2, \ldots, \tau_n\}$, executing upon a single processor platform according to the EDF policy [31]. The processor can vary the clock frequency f by selecting one of the available frequencies in a discrete set $\{f_1, \ldots, f_m\}$, ordered by ascending values. In the following, the normalized speed s ($s = f/f_m$) is used as a more convenient parameter ($s = 1$ denotes the maximum speed).

Each task τ_i generates an infinite sequence of jobs $\tau_{i,j}$, with the first job arriving at time zero and subsequent arrivals separated by T_i time-units. Each job of τ_i is characterized by a worst-case execution time (WCET) $C_i(s)$, a best-case execution time (BCET) $B_i(s)$ and a relative deadline D_i equal to the task period T_i. The computation time of each job at speed s is within the range $[B_i(s), C_i(s)]$. The WCET of τ_i is computed as $C_i(s) = \alpha_i C_i^m + (1 - \alpha_i) C_i^m / s$, where C_i^m denotes the worst-case time to execute τ_i at the maximum speed and α_i is the fraction of execution time that does not scale with the speed (i.e., memory, I/O operations, etc.).

To characterize the power consumption of the processors when they are active, we adopt the following relation derived by Martin et al. [32]:

$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0. \quad (1)$$

When processors are not needed to be active, low-power states can be exploited for reducing the power consumption while code execution is suspended. Each low-power state has a given power consumption P_σ and a *Break-Even Time* (BET), i.e., the overhead time required to enter and exit the corresponding state. The BET determines the shortest idle interval that must be available in the schedule to take advantage of the sleep state. Deeper sleep states characterized by a lower power consumption have typically longer BETs. In this paper, two low-power states are considered: *idle* and *sleep*. Their break-even times are negligible and non-negligible, respectively.

IV. Limits of existing approaches

The energy needed to execute a job is the product of the active power and the execution time at the selected speed. Note that a higher speed reduces the execution time, but increases the power consumption. For this reason, the concept of critical speed s^* [7] has been introduced for defining the speed that minimizes the overall active energy consumption. Analytically, s^* is computed as the speed that minimizes the active energy consumption per cycle $\frac{P(s)}{s}$, and can be derived from $\frac{dP(s)/s}{ds} = 0$.

As an example, assume a processor with ten speeds uniformly distributed from 0.1 to 1.0, and with active power consumption $P(s) = 0.9s^3 + 0.1$. The dominant non-linearity in the power function makes it a DVFS-sensitive architecture, where the speed that minimizes the energy consumption is $s^* = 0.4$. When instead the power consumption is characterized by a significant constant component (independent of the speed), as in $P(s) = 0.3s + 0.7$, the critical speed results to be equal to the maximum available ($s^* = 1$), hence speed scaling is not effective to minimize the active energy consumption. Such a behavior is typical in DPM-sensitive architectures, which integrate a significant amount of memory, I/O controllers and other devices whose power consumption does not depend on the processing speed.

One big limitation of the above approach is that it only analyzes the active power consumption, neglecting the power consumed when the processor is not executing. Therefore, the critical speed gives a reliable indication of the best operating frequency *only if the system is assumed to consume no power when the processor is idle*. As the static power consumed during idle intervals gets bigger, the critical speed is less suitable to characterize the best operating frequency of the processor. To have a more precise characterization of the power consumption, it is therefore necessary to account for the time spent in low-power states.

Depending on the ability of the system to exploit deeper sleep states, the best operating speed can be higher or smaller than the critical speed. For example, a DVFS-sensitive architecture with $s^* = 0.4$ could be better operated at a higher speed if the slack created could be spent in a low-power state with $P_\sigma < P(s^*)$. Conversely, it could be beneficial to reduce the speed of a DPM-sensitive platform, even with $s^* = 1$, if the idle intervals are not sufficiently large to allow entering a low-power state.

Intuitively, the highest consumption is obtained when the algorithm is never able to put the processor in sleep state, so that the slack time is entirely spent in the more consuming idle state. Conversely, a lower bound on the overall power consumption can be derived considering that any idle interval fully exploits the deepest sleep state.

Figure 1 shows the average power consumption of a NXP LPC1768 chip ($P(s) = 0.3s + 0.7$) for different task utilizations. The power consumed in idle and in sleep state is taken from real measurements, as detailed in Section VI. The straight lines represent the upper and lower bounds on the DPM power consumption, while the staircase line shows the DVFS consumption obtained from executing the generic task set at the slowest available feasible speed. Even if the device is DPM-sensitive, with a critical speed equal to one, the performance of a simple DVFS approach is rather close to the ideal DPM performance. Unlike the actual general opinion that DPM algorithms work always better than DVFS ones, the example shows that task set characteristics are crucial to decide which technique works best, as short periods and reduced idle intervals (or, equivalently, large BETs) can forbid the use of deeper low-power states, making DVFS approaches more competitive.

What said above is even more relevant when task computation times do not entirely scale with the speed, that is when each task τ_i has $\alpha_i > 0$. Indeed, memory-bound tasks with a large constant part $\alpha_i C_i$ tend to privilege DVFS techniques, as similar execution times can be obtained executing at a lower speed, with a smaller power consumption.

Finally, when the entire consumption of the SoC is con-

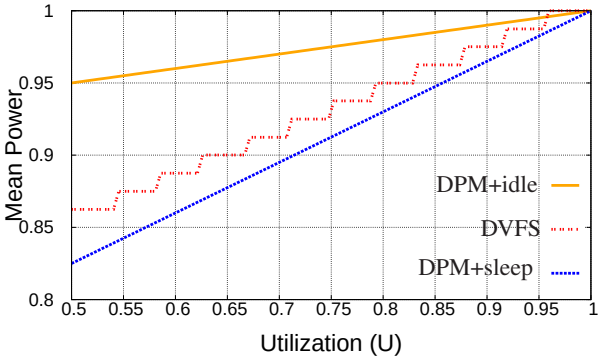


Fig. 1: DPM bounds vs. simplest DVFS algorithm.

sidered, the dissipation in low-power states is not as low as expected, because many components can not be turned off (such as the main memory), thus reducing the impact of DPM-based algorithms.

V. Algorithms

This section introduces the algorithms that have been considered in this paper. Speed scaling algorithms are considered in Section V-A, while DPM algorithms are analyzed in Section V-B. Although other recent algorithms might provide a slightly better performance, the presented algorithms have been selected for their popularity and their reasonable run-time complexity, well representing their scheduling class.

A. DVFS algorithms

DVFS algorithms can be further divided according to the kind of slack (unused computational time) they take advantage of:

- SVS [12] (Static Voltage Scheduling): only the static slack $(1 - \sum_{\tau_i} \frac{C_i}{T_i})$ is exploited;
- DRA-OTE [11] (Dynamic Reclaiming Algorithm - One Task Extension): mostly the dynamic slack due to task early terminations is used to scale the speed down;
- LA-DVS [12] (Look-Ahead RT-DVS): both static and dynamic slacks are considered.

The first algorithm, SVS, sets the running speed equal to the slowest one that guarantees the task set feasibility. Note that the speed is set at system start time, and it is never changed during execution. Although the algorithm is simple (the online complexity is null), it does not exploit task early terminations.

The second algorithm extends the previous idea by introducing a queue to keep track of task executions. Once a new job is about to be scheduled, the dynamic slack due to task early terminations is exploited to further scale the speed down. In other words, unexploited computation time is passed from the owner to the executing job.

The last algorithm aims at further reducing the running speed by executing the task with the earliest deadline at the slowest possible speed, while pushing pending and incoming jobs as close as possible to their deadlines. Although the speed until the first deadline is reduced, a high speed may be necessary to execute the other tasks.

Whenever the ready queue is empty, the processor is possibly put in low-power state until next task arrival.

B. DPM algorithms

The following DPM algorithms are considered:

- CS-DVS-P [18] (Critical Speed DVS with Procrastination): the maximum time that a task can be delayed is computed offline;
- LC-EDF [6] (Leakage-Control EDF): job delays are entirely computed at run-time.

CS-DVS-P computes at design time the maximum amount of time (Z_i) each job of τ_i can spend in the sleep state within its period without leading to any deadline miss:

$$\frac{Z_i}{Ti} + \sum_{k=1}^{i} \frac{C_k}{T_k} = 1.$$

Tasks are sorted by non-decreasing relative deadlines. At run-time, when there is no pending job, the processor is put in the deepest low-power state until next job arrival. When a job arrives and the processor is still in sleep mode, an external controller keeps the processor in such a state for the minimum between the remaining estimated sleep time and Z_i of the new job.

Conversely, LC-EDF computes at each job arrival the maximum delay the job can suffer without missing its deadline, extending the time spent in sleep state. More precisely, when an idle event occurs, LC-EDF computes the maximum extension Δ_k that the first arriving task τ_k can exploit to fully utilize the processor:

$$\sum_{i \in \{1,\ldots,n\}/\{k\}} \frac{C_i}{T_i} + \frac{C_k + \Delta_k}{T_k} = 1.$$

Then, the sleep time is extended for Δ_k units. If another task τ_j with absolute deadline earlier than τ_k's arrives when the processor is in sleep, the procedure is executed again, considering for τ_k the elapsed delay interval δ_k since the previous invocation of the algorithm:

$$\sum_{i \in \{1,\ldots,n\}/\{k,j\}} \frac{C_i}{T_i} + \frac{C_k + \delta_k}{T_k} + \frac{C_j + \Delta_j}{T_j} = 1.$$

VI. Measurements

This section aims at providing a detailed analysis of the power characteristics of the following processors:

1) Microchip dsPic33FJ256MC710, with frequencies within $[4, 40]$MHz with step of 1MHz;
2) NXP LPC1768 (ARM Cortex M3), characterized by frequencies in $[36, 96]$MHz with step of 4MHz;
3) Intel Pentium4, providing frequencies from 375MHz up to 3.0GHz with step of 375MHz.

These architectures have been selected to cover a wide spectrum of real platforms, from small digital signal processors to

Processor	K_3	K_2	K_1	K_0
LPC1768	0.0	0.0	0.3	0.7
dsPic33	0.0	0.0	0.55	0.45
Pentium4	0.0	0.09	0.44	0.47

TABLE I: Parameters of the power models.

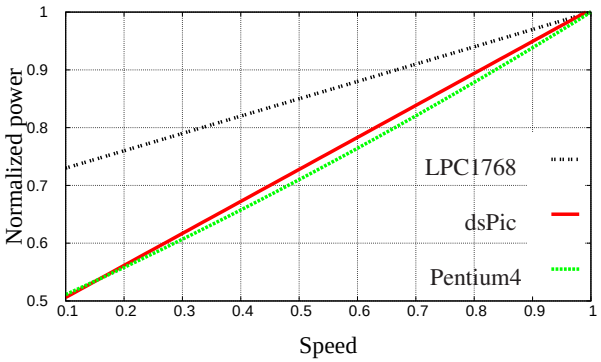


Fig. 2: Power consumption models.

average embedded controllers, including high-level computing processors.

Except for the last processor, the other measurements refer to the entire platform (i.e., core, cache, memory and peripherals) as everything is embedded on a single chip. Moreover, only the frequency is scaled, while the voltage is kept constant.

A. Performance evaluation

The following measurements have been carried out running the Coremark benchmark [33], which implements CPU bound code ($C_i(s) = C_i/s$).

The benchmark scores at the maximum speed for the analyzed platforms are 57 (without hard FPU), 218.7 and 14413.0, respectively.

These results have been achieved on systems with light workload, meaning that the processor was entirely assigned to the benchmark in execution.

B. Active state

The power functions of the platforms, considering normalized speed and normalized power, are reported in Figure 2, whereas the numerical values of the parameters of Equation 1 are summarized in Table I. Note that, on these platforms, the critical speed is always equal to the maximum available ($s^* = 1.0$).

Concerning the components of the power consumption, the static dissipation, mostly due to leakage currents, is represented by the coefficient K_0 as it is speed independent, while the rest of consumption is ascribed to the dynamic dissipation. Note that the static component includes the consumption due to the cache, which can not be turned off.

Considering the absolute power consumption, the dsPic and the LPC1768 processors have similar consumptions (0.56W and 0.695W, respectively), but the latter is around four times more performing. On the other hand, the Pentium4 consumes 88.8W when active.

C. Low-power states

Concerning the low-power states, the dsPic processor provides two states: Idle and Sleep. The first one switches few components off, leading to a consumption equal to 66% of the power at the maximum speed, with a state transition of 8 clock cycles. The Sleep state consumes 31% of the maximum power, with a break-even time around 15ms (the clock crystal and PLL are put off).

The LPC1768 offers Sleep, Deep Sleep, Power Down, and Deep Power Down states, which consume, with respect to the consumption at the maximum speed, 90%, 70%, 70% and 65%, respectively. The overhead is negligible for the lighter state, whereas it takes about 10 milliseconds for the deepest.

On the Pentium4, the ACPI module exploits only a single state, called Idle, with a relative consumption of 34% and a break-even time of a few hundreds milliseconds.

VII. Experimental results

This section presents a set of simulation experiments carried out for evaluating the considered algorithms on the different platforms under different scenarios.

The synthetic task sets are composed of 10 tasks randomly generated using the UUniFast algorithm [34]. For each utilization step of 0.05, 30 different task sets were generated and tested.

The speed scaling overhead (in the order of μs) was considered negligible with respect to the task execution times (in the ms). The frequencies of buses and memories were assumed to be constant and independent of the processor frequency.

The following simulations are divided into three categories: analysis of the algorithm performance in the worst case, online improvement due to task early terminations and impact of speed-independent code.

A. Worst-case analysis

This section presents the average power consumption obtained by the considered algorithms assuming always the worst-case execution. Since task early terminations are not considered, the algorithm DRA-OTE is not taken into account as it would exhibit the same performance of SVS. Computation times are assumed to scale linearly with the speed ($\forall \tau_i \in \Gamma : \ \alpha_i = 0$).

The results are reported in Figure 3 considering two scenarios in which the shortest task period is larger than or comparable to the sleep break-even time, respectively. When the shortest period is much larger than the break-even time, DPM-based algorithms tend to work better, especially at lower utilizations when a considerable amount of slack is available. When instead the shortest period becomes comparable to the break-even time the performance of DPM-algorithms drops significantly.

The first case is shown in Figure 3a, 3c and 3e, where periods are generated in the range $[25, 250]$ms for the dsPic and LPC1768, and in the range $[300, 3000]$ms for the Pentium4. Among DPM algorithms, CS-DVS-P has always a lower consumption than LC-EDF. Among DVFS-based approaches,

SVS and LA-DVS obtain the same performance on the dsPic because the power function is linear. For example, consider a job of 10ms. When it is executed at $s = 0.5$, hence for 20ms, the energy consumption is 14.554 mJ. If the same job is executed for 15ms at $s = 0.\bar{3}$ and for 5ms at $s = 1.0$, leading to the same overall execution time (20ms), then the overall energy consumption is again $P(0.\bar{3}) \cdot 15 + P(1.0) \cdot 5 = 14.554$ mJ. Hence, an aggressive DVFS algorithm may be ineffective when the power consumption is linear. However, LA-DVS consumes less than SVS on the other two platforms because the limited number of speeds does not allow SVS to exploit the entire static slack. The best DPM algorithm (CS-DVS-P) has always a better performance than DVFS ones, although the difference is rather small for the LPC1768 due to the limited savings allowed in sleep mode with this architecture. Instead, LC-EDF has always the largest power consumption.

In the second case, the minimum task period is decreased, becoming comparable to the low-power state transition overhead (Figure 3b, 3d and 3f). More precisely, periods were generated in the range $[8, 80]$ms for the dsPic and LPC1768, and in the range $[100, 1000]$ms for the Pentium4. As expected, while the performance of DVFS strategies remains the same, that of DPM algorithms drops significantly, making DVFS strategies more competitive.

B. Average execution analysis

This section considers the average power consumption obtained by the algorithms taking into account that jobs may terminate earlier than their worst case. More precisely, the actual execution time of each job of τ_i is generated in the range $[B_i = C_i/10, C_i]$.

The results for the three platforms are shown in Figure 4, assuming periods in $[25, 250]$ms for the dsPic and LPC1768 platforms, and in $[300, 3000]$ms for the Pentium4 (shortest period larger than sleep BET).

The trend among DPM algorithms is not altered, with CS-DVS-P still guaranteeing better performance than LC-EDF. Among DVFS techniques, SVS does not improve as it cannot take advantage of the online slack freed by task early terminations. However, DRA-OTE has always worse performance than SVS, because it scales the speed down in order to make jobs last as long as the worst case. Since the critical speed is equal to the maximum one, DRA-OTE increases the energy consumption without introducing any benefit. LA-DVS is the best DVFS algorithm as jobs can usually end before scaling up to high speeds, leading to a lower consumption.

Similarly to the previous analysis, decreasing task periods makes DPM algorithms much less effective than DVFS ones. The above trend is the same when computation times have different variances between worst and best case execution times.

C. Speed-independent code

This section analyzes the case in which computation times do not fully scale with the speed. As explained in Section III, such a behavior is modeled by the α parameter. More precisely, $\alpha = 0$ means that the computation fully scales with the speed ($C(s) = C/s$), whereas $\alpha = 1$ leads to a constant computation time, completely speed independent.

For example, a job with $\alpha = 0$ that lasts for 10ms at the maximum speed, will take 20ms when executed at $s = 0.5$. On the other hand, if $\alpha = 0.5$, only half of the execution time is scaled with the speed, so that the job would last for 20ms when executed at $s = 0.\bar{3}$. Both cases have a similar completion time, but the second one has a lower speed, resulting in a smaller power consumption.

Figure 5 reports the average power consumption on the LPC1768 processor when $\forall \tau_i \in \Gamma : \alpha_i = 0.5$. Execution times and periods are generated as in the previous experiment, within $[B_i = C_i/10, C_i]$ and $[25, 250]$ms, respectively.

The introduction of speed-independent code improves the performance of DVFS algorithms. More precisely, with respect to the equivalent analysis in Figure 4b ($\alpha = 0.0$), the performance of LA-DVS gets very close to that of CS-DVS-P, while SVS becomes more convenient than LC-EDF. The only DVFS algorithm that does not take a significant advantage from speed-independent sections of code is DRA-OTE.

The performance on the other two platforms shows the same trend. In general, the effectiveness of DVFS algorithms increases with the fraction of speed-independent code, as foreseen in Section IV.

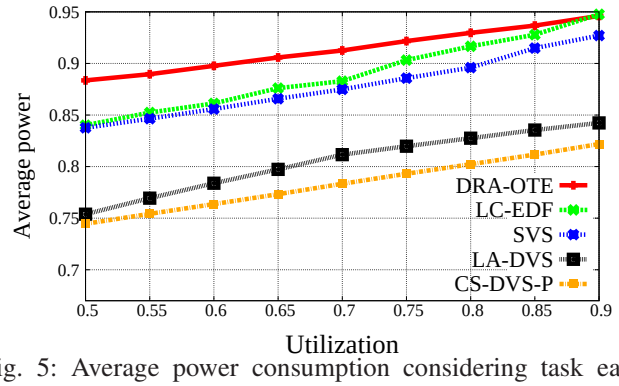
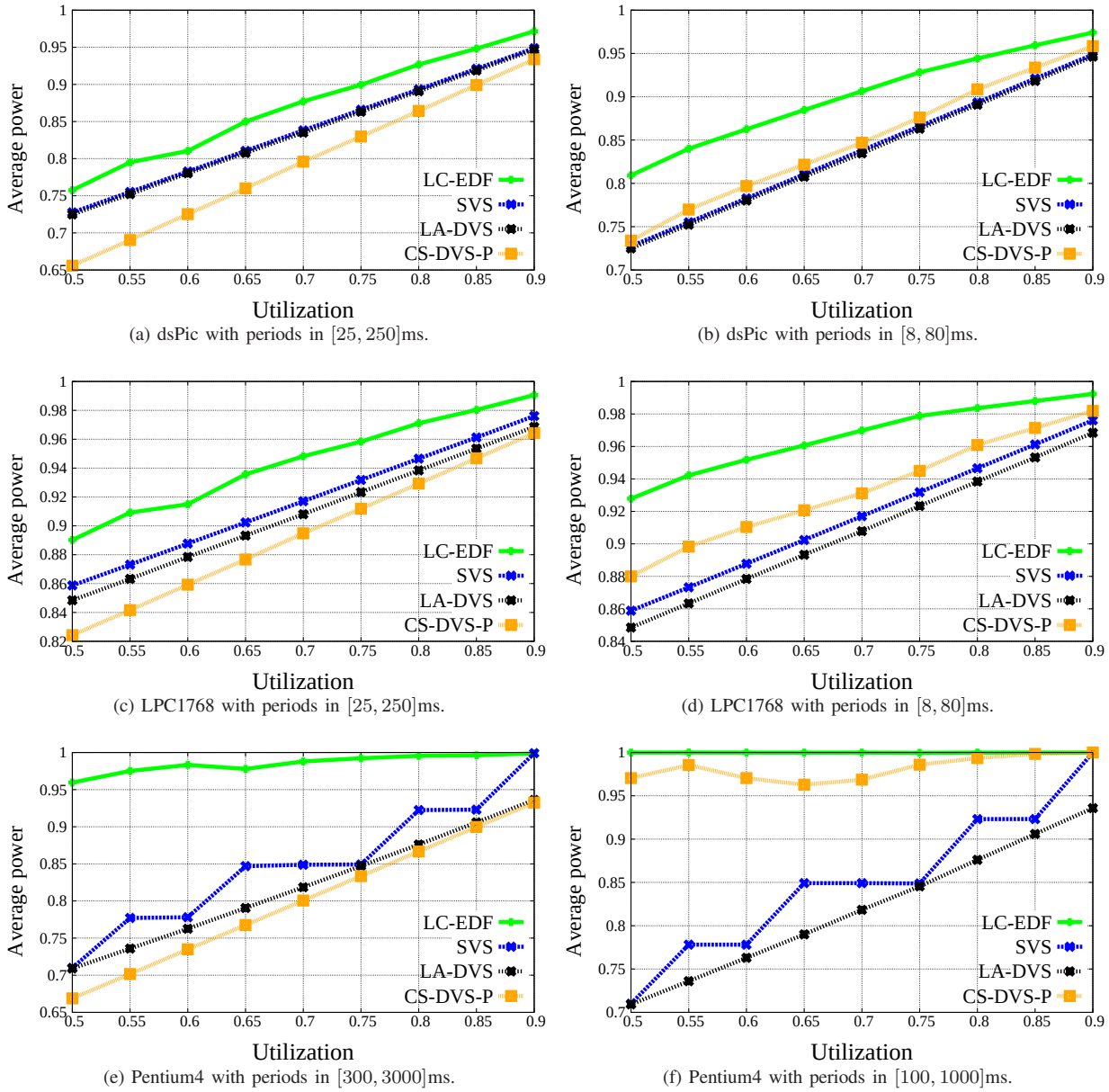


Fig. 5: Average power consumption considering task early terminations and $\forall \tau_i : \alpha_i = 0.5$.

VIII. Conclusions

This paper presented a comparison of several well-known DVFS and DPM algorithms, showing their behaviors on different scenarios and considering three processors widely used in their respective domains.

Results confirmed that the actual assumption stating that DPM algorithms work generally better than DVFS ones on actual hardware is true. However, experiments also highlighted that such a consideration can easily be invalidated when other aspects are involved in the analysis. For instance, short periods can drastically reduce the effectiveness of DPM algorithms, and tasks that intensively interact with peripherals can make DVFS algorithms more effective. In addition, it was empirically shown that task early terminations help both DPM and DVFS algorithms in further reducing the energy consumption.



(a) dsPic with periods in $[25, 250]$ms.

(b) dsPic with periods in $[8, 80]$ms.

(c) LPC1768 with periods in $[25, 250]$ms.

(d) LPC1768 with periods in $[8, 80]$ms.

(e) Pentium4 with periods in $[300, 3000]$ms.

(f) Pentium4 with periods in $[100, 1000]$ms.

Fig. 3: Average power consumption assuming worst-case execution.

As future work, we aim at extending such an analysis to multi-core platforms, as they represent the actual cutting-edge research topic.

REFERENCES

[1] D. Soudris, C. Piguet, and C. Goutis, *Designing CMOS Circuits for Low Power*, ser. European Low-Power Initiative for Electronic System Design. Springer, 2002.

[2] R. Kaushik, M. Saibal, and M.-M. Hamid, "Leakage current in deep-submicron cmos circuits," *Journal of Circuits, Systems, and Computers*, vol. 11, no. 6, pp. 575–600, 2002.

[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power cmos digital design," *IEEE Journal of Solid State Circuits*, 1995.

[4] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, December 2003.

[5] M. Bambagini, F. Prosperi, M. Marinoni, and G. Buttazzo, "Energy management for tiny real-time kernels," in *Energy Aware Computing (ICEAC), 2011 International Conference on*. IEEE, 2011, pp. 1–6.

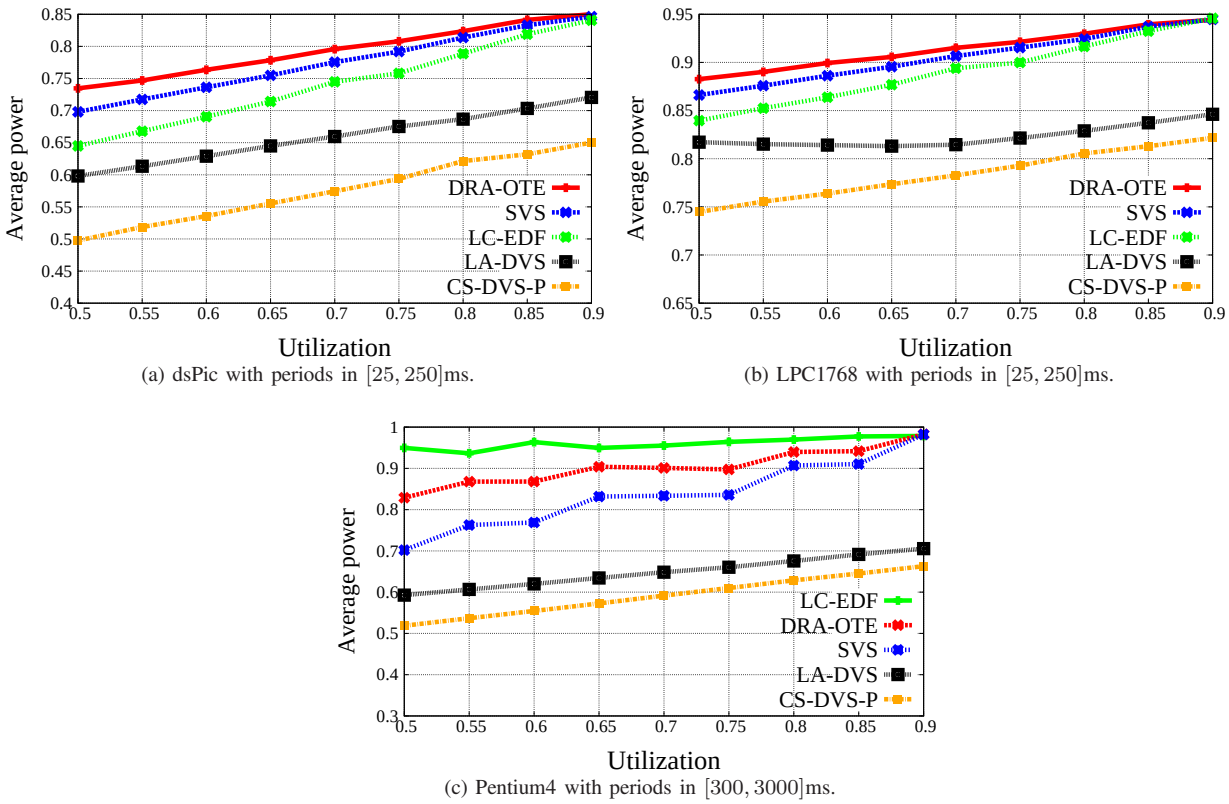
[6] Y.-H. Lee, K. Reddy, and C. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, 2003.

[7] J.-J. Chen and T.-W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor," *SIGPLAN Notices*, 2006.

[8] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2011.

[9] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proc. of the 36th Annual Symp. on Foundations of Computer Science*, 1995.

[10] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, January 1973.



(a) dsPic with periods in $[25, 250]$ms.

(b) LPC1768 with periods in $[25, 250]$ms.

(c) Pentium4 with periods in $[300, 3000]$ms.

Fig. 4: Average power consumption considering task early terminations.

[11] H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. Melhem, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. of the 22nd IEEE Real-Time Systems Symp.*, ser. RTSS '01, 2001.

[12] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, pp. 89–102, October 2001.

[13] E. Bini, G. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *Proc. of the 17th Euromicro Conference on Real-Time Systems*, 2005, pp. 3–10.

[14] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Proceedings of the conference on Design, Automation and Test in Europe*, ser. DATE '03, 2003.

[15] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proceedings of the International conference on Computer-aided design*, 2004.

[16] W. Kim, J. Kim, and S. L. Min, "Preemption-aware dynamic voltage scaling in hard real-time systems," in *Proceedings of the 2004 international symposium on Low power electronics and design*, 2004.

[17] R. Davis and A. Welling, "Dual priority scheduling," in *Proceedings Real Time Systems Symposium*, ser. RTSS '05, 1995.

[18] R. Jejurikar, C. Pereira, and R. K. Gupta, "Leakage aware dynamic voltage scaling for real time embedded systems," in *In Proc. of the Design Automation Conference*, 2004.

[19] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proceedings of the 42nd annual Design Automation Conference*, ser. DAC '05, 2005.

[20] ——, "Procrastination scheduling in fixed priority real-time systems," in *In Proceedings of the Language Compilers and Tools for Embedded Systems*, 2004.

[21] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Periodic power management schemes for real-time event streams." in *the 48th IEEE Conf. on Decision and Control (CDC)*, 2009.

[22] ——, "Adaptive dynamic power management for hard real-time systems," in *the 30th IEEE Real-Time Systems Symp. (RTSS)*, 2009.

[23] M. Bambagini, M. Bertogna, M. Marinoni, and G. Buttazzo, "An energy-aware algorithm exploiting limited preemptive scheduling under fixed priorities," in *Industrial Embedded Systems (SIES), 2018 8th IEEE International Symposium on*. IEEE, 2013, pp. 3–12.

[24] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S.-L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, 2002.

[25] S. Saha and B. Ravindran, "An experimental evaluation of real-time dvfs scheduling algorithms," in *Proceedings of the 5th International Systems and Storage Conference*, 2012.

[26] D. Grunwald, C. B. Morrey, III, P. Levis, M. Neufeld, and K. I. Farkas, "Policies for dynamic clock scheduling," in *Proceedings of the Symposium on Operating System Design & Implementation*, 2000.

[27] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *ArXiv e-prints*, Jan. 2014.

[28] W. Bircher, M. Valluri, J. Law, and L. John, "Runtime identification of microprocessor energy saving opportunities," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005.

[29] J. Peddersen and S. Parameswaran, "Clipper: Counter-based low impact processor power estimation at run-time," in *Proceeding of Design Automation Conference*, 2007.

[30] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate on-line prediction of processor and memory energy usage under voltage scaling," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, 2007.

[31] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[32] T. Martin and D. Siewiorek, "Non-ideal battery and main memory effects on cpu speed-setting for low power," *IEEE Transactions on VLSI Systems*, vol. 9, no. 1, pp. 29–34, 2001.

[33] "Coremark web site," http://www.eembc.org/coremark/.

[34] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1-2, May 2005.