# Energy-Aware Scheduling for Real-Time Systems: A Survey

MARIO BAMBAGINI and MAURO MARINONI, Scuola Superiore Sant'Anna
HAKAN AYDIN, George Mason University
GIORGIO BUTTAZZO, Scuola Superiore Sant'Anna

This article presents a survey of energy-aware scheduling algorithms proposed for real-time systems. The analysis presents the main results starting from the middle 1990s until today, showing how the proposed solutions evolved to address the evolution of the platform's features and needs. The survey first presents a taxonomy to classify the existing approaches for uniprocessor systems, distinguishing them according to the technology exploited for reducing energy consumption, that is, *Dynamic Voltage and Frequency Scaling* (DVFS), *Dynamic Power Management* (DPM), or both. Then, the survey discusses the approaches proposed in the literature to deal with the additional problems related to the evolution of computing platforms toward multicore architectures.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Real-time operating systems**; • **Software and its engineering** → **Scheduling**; **Power management**

Additional Key Words and Phrases: Energy, power, real-time scheduling, dynamic voltage and frequency scaling, dynamic power management, low power, sleep, idle, single core, multicore

**ACM Reference Format:**
Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. 2016. Energy-aware scheduling for real-time systems: A survey. ACM Trans. Embed. Comput. Syst. 15, 1, Article 7 (January 2016), 34 pages.
DOI: http://dx.doi.org/10.1145/2808231

## 1. INTRODUCTION

In the last two decades, energy management has become a prime design and operation dimension for many real-time embedded platforms. In fact, effective energy management is crucial for all battery-powered embedded systems, such as those deployed in autonomous mobile robots, wearable devices, industrial controllers, and wireless sensor networks. In many of these systems, recharging or replacing the batteries is not always practical or feasible; hence, minimizing energy consumption translates to a longer lifetime and clear operational and financial advantages. Even for systems that are directly connected to the power grid, reducing energy consumption provides significant monetary and environmental gains.

In real-time embedded systems, two widely used techniques for reducing energy consumption in the processing unit are *Dynamic Voltage and Frequency Scaling* (DVFS) and *Dynamic Power Management* (DPM). DVFS approaches trade energy with performance by decreasing the voltage and the frequency of the processor to reduce the overall energy consumption. Since reducing the frequency increases the task execution times, a common objective in real-time systems is to derive processor/task speed values that still guarantee the timing constraints while minimizing the total energy

Authors' addresses: M. Bambagini, M. Marinoni, and G. Buttazzo, Scuola Superiore Sant'Anna, Pisa 56127, Italy; emails: m.bambagini@sssup.it; g.buttazzo@sssup.it; m.marinoni@sssup.it; H. Aydin, Department of Computer Science, George Mason University, Fairfax, VA 22030; email: aydin@cs.gmu.edu.

DOI: http://dx.doi.org/10.1145/2808231

consumption. On the other hand, DPM techniques switch the processor to a low-power inactive state as long as possible, while guaranteeing that all real-time tasks will finish within their deadlines.

In CMOS technology, which is still the dominant approach in the VLSI circuit design, the power consumption has both *dynamic* and *static* components, which are due to the system activity and leakage dissipation, respectively. Unless the system is in an off state, the static contribution is always present, regardless of the actual performance level. Thus, DVFS approaches that modify the voltage and clock frequency are more suitable for reducing the dynamic power, whereas DPM solutions are best suited for decreasing the impact of the static power component. These techniques also can be integrated to exploit their complementary features, in order to further increase the energy savings.

Historically, CMOS circuits used to operate at a supply voltage level much higher than the threshold voltage, making the impact of dynamic power consumption dominant with respect to the static power consumption. This resulted in the proliferation of DVFS approaches that are more suitable for reducing the dynamic power consumption. With the progress of the VLSI technologies, miniaturization has considerably shrunk the transistor size, lowering the supply voltage, thereby reducing the dynamic power consumption. Even though the threshold voltage has also been lowered, the gap between supply and threshold voltages has been reduced. This led to a significant increase in the leakage consumption, because the smaller the gap, the higher the subthreshold dissipation [Soudris et al. 2002; Narendra and Chandrakasan 2010]. As a result, the static power consumption has become as important as the dynamic power consumption, and DPM approaches that target reducing the leakage power have recently increased in popularity.

This article presents a survey of energy-aware scheduling algorithms for uniprocessor and multiprocessor hard real-time systems. Although several surveys have been published on energy management algorithms, most of them focused on DVFS approaches only or did not take real-time constraints into account. The increasing relevance of leakage dissipation led to interesting integrated DVFS-DPM approaches, which were not considered in previous surveys.

For instance, Chen and Kuo [2007] addressed single- and multiprocessor systems by classifying algorithms according to the task periodicity, but they mainly focused on DVFS algorithms. Similarly, Kim [2006] surveyed the intra- and intertask DVFS algorithms by considering only the single-core systems. Saha and Ravindran [2012] reported a performance comparison of a number of single-core DVFS algorithms for an implementation in the GNU/Linux kernel. More recently, Mittal [2014] presented a general survey of energy management techniques for embedded systems, including the microarchitectural techniques.

The present survey, on the other hand, provides an in-depth overview of the existing DVFS- and DPM-based approaches, analyzing integrated DVFS-DPM algorithms and offering a wider spectrum of analysis. Considering the space limitations and the large number of proposed approaches, we decided to focus on methods for hard real-time systems, briefly discussing some solutions for soft real-time systems in Section 9, which has been dedicated to present other related problems.

**Article Organization.** The rest of this article is organized as follows: Section 2 presents the main system models adopted in the literature. Section 3 details the proposed taxonomy for the various algorithms under consideration. Section 4 introduces the DVFS algorithms for uniprocessor real-time systems, whereas Section 5 discusses the relevant DPM algorithms. Section 6 considers the integrated algorithms that combine both DVFS and DPM approaches. Section 7 introduces the algorithms for multiprocessor systems with independent frequencies, whereas Section 8 presents

the integrated solutions for multiprocessor systems based on voltage islands. Section 9 presents an overview of other problems related to energy management in real-time systems. Section 10 concludes the survey with final remarks.

## 2. MODELS

This section presents the most relevant models used in the literature for the design and analysis of energy-aware scheduling algorithms. Specifically, Section 2.1 overviews various power models, and Section 2.2 presents the computational workload models.

### 2.1. Power Model

The power consumption of a single active gate in CMOS technology has been modeled accurately in the literature [Chandrakasan et al. 1995]. Specifically, the power consumption $P_{gate}$ of a gate is a function of the supply voltage $V$ and clock frequency:

$$P_{gate} = \alpha C_L V^2 f + \alpha V I_{short} + V I_{leak}, \tag{1}$$

where $C_L$ is the total capacitance driven by the gate, $\alpha$ is the gate activity factor (i.e., the probability of gate switching), $I_{short}$ is the current between the supply voltage and ground during gate switching, and $I_{leak}$ is the leakage current, which is independent of the actual frequency and system activity. The three components of the sum in Equation (1) correspond to *dynamic, short circuit,* and *static* power components, respectively.

In essence, the dynamic power is the power required to load and unload the output capacitors of the gates. Unlike the dynamic component, the short circuit current $I_{short}$ depends on the temperature, size, and process technology. The leakage current is a quantum phenomenon where mobile charge carriers (electrons or holes) pass by tunnel effect through an insulating region, leading to a current that is independent from switching activity and frequency. That dissipation is due to three causes: gate leakage (from gate to source losses), drain junction leakage (losses in the junctions), and subthreshold current (from drain to source losses).

In Equation (1), the two variables that do not depend on the physical parameters are the supply voltage $V$ and the clock frequency $f$. However, they are not completely independent, because the voltage level limits the highest frequency that can be used: the lower the voltage, the higher the circuit delay. Specifically, the circuit delay is related to the supply voltage $V$ by the following equation:

$$circuit\ delay = \frac{V}{(V - V_T)^2}, \tag{2}$$

where $V_T$ denotes the *threshold voltage*, which is defined as the minimum voltage needed to create a channel from drain to source in a MOSFET transistor.

In the literature, the processor is assumed to be able to dynamically scale the clock frequency $f$ in a given range $[f_{min}, f_{max}]$. Often, the analysis is performed by replacing the clock frequency by the processor *speed* $s$, defined as the normalized frequency $s = f/f_{max}$, so that the maximum processor speed is considered as $s_{max} = 1.0$. In some work, the speed range is assumed to be *continuous* (for processors where the frequency can be varied with a fine granularity), whereas other works consider a *discrete* set of $k$ frequencies $\{f_1, \ldots, f_k\}$, based on the observation that current processors typically offer a small number of discrete frequency levels.

To characterize the power consumption $P(s)$ of the system as a function of the processor speed, one of the most general formulations has been proposed in Martin and Siewiorek [2001]:

$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0. \tag{3}$$

The $K_3$ coefficient expresses the weight of the power consumption components that vary with both voltage and frequency. The second-order term ($K_2$) captures the nonlinearity of DC-DC regulators in the range of the output voltage. The $K_1$ coefficient is related to the hardware components that can only vary the clock frequency (but not the voltage). Finally, $K_0$ represents the power consumed by the components that are not affected by the processor speed.

Another variant of Equation (3) used in the literature (e.g., Zhu and Aydin [2009]) is

$$P(s) = P_{ind} + P_{dyn}(s), \tag{4}$$

where the power dissipation is explicitly divided into static ($P_{ind}$) and dynamic ($P_{dyn}(s)$) power components. $P_{ind}$ is assumed to be independent of the system speed, and $P_{dyn}$ is assumed to be a polynomial function of the speed $s$. In some work [Pillai and Shin 2001; Aydin et al. 2001], such a polynomial function is assumed to be equal to $P(s) = \beta \cdot s^{\alpha}$, where $(2 \leq \alpha \leq 3)$.

A more specific power model adopted in Bini et al. [2009] considered the set of operating modes supported by the processor. Each mode is described by the frequency $f$, the lowest voltage $V$ that supports that frequency level, and the corresponding power consumption. To some extent, Martin's equation can be considered a generalization of this model, as it provides an interpolation of the various operating points on an ideal processor where the speed/voltage can be adjusted in a continuous manner.

Switching from one speed level to another one involves both a *time* and *energy* overhead. These overheads depend both on the original and final speed levels [Xu et al. 2007; Mochocki et al. 2007]. When scaling the speed, the execution is suspended and the overhead is mostly due to the time required to switch the crystal on and/or adjust the Phase-Locked Loop (PLL). Generally, the wider the difference between the two frequencies, the higher the introduced overhead. In this article, we use the notation $\mu_{s_1 \to s_2}$ to denote the time overhead when transitioning from the speed level $s_1$ to the speed level $s_2$.

An additional feature provided by almost all the current processors is the ability to switch to *low-power* states when the task execution is suspended. Each low-power state $\sigma_x$ is characterized by its power consumption ($P_{\sigma_x}$) and the time and energy overheads involved in entering and exiting that state, denoted as $\delta_{s \to x}$, $\delta_{x \to s}$, $E_{s \to x}$, and $E_{x \to s}$, respectively. For the sake of simplicity, we use the overall time and energy overheads associated with the low-power state $\sigma_x$, namely, $\delta_x$ and $E_x$, as the sum of the initial and final transition overheads. In general, the "deeper" a low-power state, the lower the power consumption is, but also the higher time and energy overheads involved in the transition. An exhaustive analysis of the low-power states in actual architectures has been undertaken by Benini et al. [2000].

Considering the time and energy overheads involved in transitions to low-power states, there is, in general, a minimum time interval that justifies switching to a specific low-power state; this is because, if the system returns to an active state too quickly, the energy overhead of the transition would offset the power savings of the low-power state. Consequently, the parameter $B_x$, referred to as the *break-even time*, corresponds to the length of the shortest idle interval that must be available in the schedule to effectively exploit the sleep state $\sigma_x$. Specifically, $B_x$ is the maximum of the time required to perform a complete transition and the minimum idle time length that can amortize the switching energy [Quan et al. 2004; Zhao and Aydin 2009]:

$$B_x = \max\left(\delta_x, \frac{E_x - \delta_x \cdot P_{\sigma_x}}{P_{ref} - P_{\sigma_x}}\right), \tag{5}$$

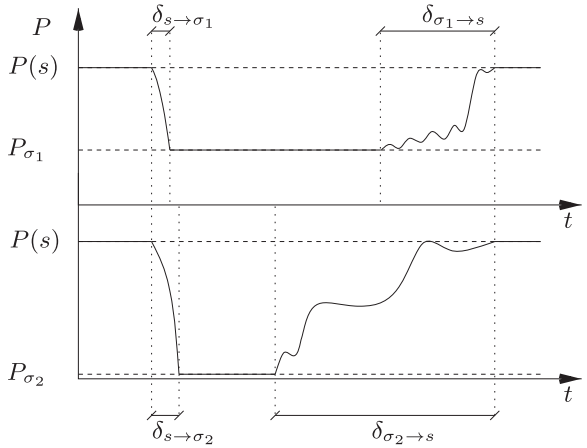



Fig. 1. An example with two low-power states.

where $P_{ref}$ is the power consumption of the processor in a default state when tasks do not execute. For instance, $P_{ref}$ can be a particular inactive state that requires a negligible transition overhead, or, in case the processor is kept active during idle intervals, it may be the power consumption at the minimum speed level.

Different low-power states are characterized by different parameters. Figure 1 illustrates two different state transitions. The first case illustrates a low-power state $\sigma_1$ with a medium power consumption and a relatively short break-even time. On the other hand, the second low-power state $\sigma_2$ guarantees the lowest power consumption but introduces a significant temporal overhead from active to sleep and back to active. Finding the most suitable low-power state depends on the length of the available idle interval, which, in turn, is determined by the timing constraints.

Different solutions have been proposed to provide DVFS capabilities to multicore processors. In particular, they can be distinguished based on the capability of setting the clock frequencies independently among cores. Historically, the first platform model considered core frequencies to be independent and has been used to analyze architectures where each core is located in a dedicated chip. However, as noted by Herbert and Marculescu [2007], the potential energy gains of such an architectural solution are not significant enough to justify the higher design complexity of the hardware. Therefore, in modern multicore and many-core architectures, a good tradeoff between flexibility and complexity is obtained by grouping CPUs in voltage islands sharing the same voltage and frequency.

### 2.2. Workload Model

In hard real-time systems, the computational workload is typically characterized by a set $\Gamma$ of $n$ *periodic* or *sporadic* tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task $\tau_i$ is cyclically activated on different input data and therefore generates a potentially infinite sequence of instances $\tau_{i,1}, \tau_{i,2}, \ldots$, referred to as *jobs*. The jobs of a *periodic* task $\tau_i$ are regularly separated by a *period* $T_i$, so the release time of a generic job $\tau_{i,k}$ can be computed as

$$r_{i,k} = \Phi_i + (k-1)T_i,$$

where $\Phi_i$ denotes the activation time of the first job, also referred to as the *task offset*. On the other hand, in the case of *sporadic* task $\tau_i$, the period $T_i$ indicates the *minimum interarrival time* of its jobs: $r_{i,k+1} \geq r_{i,k} + T_i \ \forall k$. A real-time task $\tau_i$ is also characterized by a *relative deadline* $D_i$, which specifies the maximum time interval

(relative to its release time) within which the job should complete. Depending on the specific assumptions, relative deadlines can be less than, equal to, or greater than periods. In the most common case, the relative deadlines are equal to periods, which is commonly called *implicit-deadline task sets*. Once a job $\tau_{i,k}$ is activated, the time at which it should finish its execution is called the *absolute deadline* and is given by $d_{i,k} = r_{i,k} + D_i$.

Each task $\tau_i$ is also characterized by a *worst-case execution time (WCET)* $C_i(s)$, which is a function of the processor speed. In a large body of works, WCET is considered to be fully scalable with the speed, that is, $C_i(s) = C_i/s$. However, a number of research works [Seth et al. 2003; Aydin et al. 2006] noted that this is only an upper bound, because several I/O and memory operations are performed on devices and memory units that do not share the clock frequency with the CPU. For instance, if a task moves data to/from a hard disk drive, the operation depends mostly on the bus clock frequency, the hard disk reading/writing speed, and the interference caused by other tasks accessing the bus. To take the speed-independent operations into account, the task's WCET can be split into a fixed portion $C_i^{fix}$ not affected by speed changes and a variable portion $C_i^{var}$, which is fully scalable with the speed. Hence,

$$C_i(s) = C_i^{fix} + C_i^{var}/s.$$

To better characterize the complexity of modern parallel applications, more detailed task models have been proposed.

A frame-based system is composed by a task set $\Gamma$ where all tasks $\tau_i$ are repeated every frame of length D. Hence, they share the same deadline $D_i = D$.

A more general model considers applications composed by tasks with dependencies described as a directed acyclic graph (DAG), where vertexes represent tasks and edges denote precedence relations among tasks.

In terms of CPU scheduling, tasks may be assigned a *fixed-priority* level $P_i$, representing the relative importance or urgency of the task with respect to the others. In systems with dynamic priorities, the priority levels of jobs of a given task may vary over time: for instance, with the *Earliest Deadline First* (EDF) policy [Liu and Layland 1973], the priorities are determined according to the absolute deadlines of the current active jobs of the periodic tasks and hence naturally vary over time.

In most algorithms, tasks are assumed to be fully *preemptive*, meaning they can be suspended at arbitrary points in favor of higher-priority tasks. Preemption simplifies the schedulability analysis but introduces a runtime overhead $\xi$ (preemption cost) during task execution due to context switch cost, the pipeline invalidation delay, and the cache-related preemption delay. The preemption cost is often assumed to be constant and speed independent. On the other hand, *nonpreemptive scheduling*, while characterized by negligible runtime overhead, introduces significant blocking delays on high-priority tasks that heavily penalize schedulability.

Scheduling approaches for multicore systems can mainly be divided into two classes: *partitioned* approaches, which statically assign tasks to dedicated cores and schedule them with uniprocessor scheduling algorithms, and *global* approaches, where tasks are handled through a single ready queue and can migrate between cores during their execution. Typical algorithms belonging to such a class are the multicore extensions of Rate Monotonic and EDF, called Global Rate Monotonic (GRM) and Global Earliest Deadline First (GEDF), respectively.

## 3. TAXONOMY OF ENERGY-AWARE SCHEDULING ALGORITHMS

This section presents the taxonomy used to organize the energy-aware CPU scheduling algorithms discussed in this survey. First, it presents the approach used to classify the

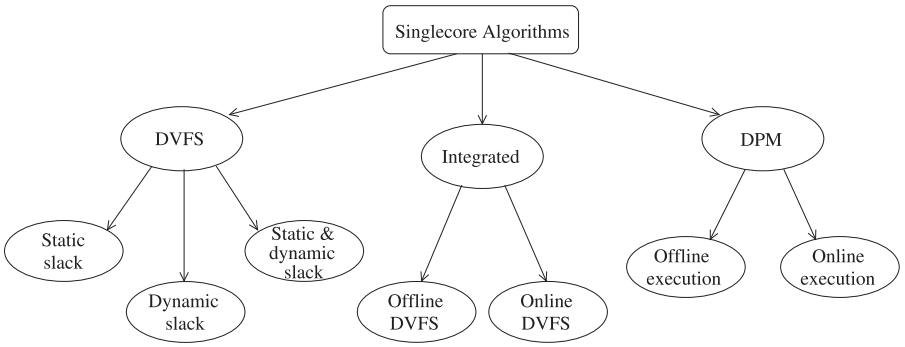



Fig. 2. Taxonomy for single-core algorithms.

algorithms for platforms powered by a single-core CPU, and then it introduces the parameters considered to cope with the extra degrees of freedom that are present in multicore systems. Figure 2 illustrates the taxonomy for algorithms running on single-core CPUs. They are first classified along the DVFS and DPM dimensions, based on the primary power management technique that they use. The DVFS algorithms are then divided according to the type of *slack* (the unused CPU time) that they reclaim for scaling speed to save energy: *static, dynamic*, or *both*. Specifically, the algorithms that exploit only the *static slack* consider the residual processor utilization in the worst-case execution, whereas those that reclaim the *dynamic slack* take advantage of the difference between the worst-case and the actual execution time of the jobs. In other words, the DVFS algorithms that exploit the dynamic slack take advantage of the runtime variability of the workload, since in practice many real-time jobs complete earlier than their worst-case work-case finishing time.

Such a classification does not immediately apply to DPM algorithms, since, due to their work-conservative nature, the dynamic slack is automatically accounted in almost all the cases. Thus, they are classified as *offline* and *online* approaches. Finally, the algorithms that use both DVFS and DPM techniques are designated as *integrated* algorithms. These algorithms are further divided according to when the task speed assignment decisions are made, that is, either *offline* or *online*.

Besides the main features considered in the proposed taxonomy and task characteristics (such as periodicity and priority assignment), several algorithms are characterized by other specific assumptions and details that will be discussed in due course. For example, the following aspects also must be considered for the DVFS algorithms:

—*Speed set*: continuous versus discrete
—*Computation time*: fully versus partially scalable with the processor speed
—*Time/energy overhead due to speed changes*: accounted versus neglected

For DPM algorithms, additional features include whether they consider the state transition overhead and whether they explicitly consider task early terminations.

When multicore technology became sufficiently reliable for the market, the CMOS technology already presented a nonnegligible leakage current. Therefore, most of the energy-aware scheduling algorithms for multicore systems integrate DVFS and DPM. Even in the cases where this is not explicitly done, some issues regarding feasible integrations are discussed, typically using slightly modified algorithms for the single-core CPU.

The taxonomy used to classify the multicore algorithms is illustrated in Figure 3. The main classification is made according to the flexibility in the DVFS support provided

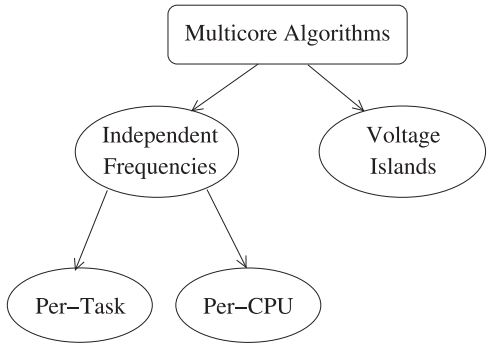



Fig. 3. Taxonomy for multicore algorithms.

by the platform. If the hardware allows setting a different frequency for each core, the DVFS algorithms are classified as *Independent Frequencies*, whereas if a single frequency is shared among a subset of cores, the algorithms are classified as *Voltage Islands*. The DVFS multiprocessor algorithms for independent frequencies can be further distinguished between approaches that assign frequencies to cores independently of the running tasks (Per-CPU algorithms) and those that compute a frequency for each task and use it for the core executing that task (Per-Task algorithms).

Other aspects considered for the classification are as follows:

—*Task scheduling*: partitioned versus global
—Order used for the DVFS, DPM, and scheduling phases

Finally, the algorithms are evaluated according to their computational complexity.

## 4. UNIPROCESSOR DVFS ALGORITHMS

DVFS-based algorithms rely on the system's capability of adjusting the processor supply voltage and frequency (hence, the speed) to reduce power consumption while still meeting the real-time constraints. Historically, such a speed scaling technique was the first approach proposed to deal with energy management, as in CMOS circuits dynamic power consumption was more significant than static power consumption. Most of the early DVFS algorithms assumed a power function equal to $P(s) = s^{\alpha}$ $(2 \leq \alpha \leq 3)$, implicitly ignoring the leakage power. Using such a power function, the lower the speed, the lower the consumed energy; hence, this model favors the algorithms that use the lowest speed that can still meet the deadlines, leaving no idle intervals.

When the leakage power dissipation is not negligible (i.e., $K_0 \neq 0$ and $P_{ind} \neq 0$ in Equation (3) and Equation (4), respectively), scaling the system speed down also increases the computation times and leakage energy consumption, which in turn may *increase* the total energy figures. To address this issue, the concept of *critical speed* (also known as the *energy-efficient speed*), denoted by $s^*$, was introduced to denote the lowest available speed that minimizes the total energy consumption, which consists of dynamic and static power figures [Aydin et al. 2006; Chen and Kuo 2006]. Specifically, if we assume $P(s)$ as in Equation (3), it becomes strictly convex, and $s^*$ is defined as the lowest speed that minimizes the energy consumption per cycle, which is equivalent to the speed value that makes the derivative of $P(s)/s$ null.

For instance, let us consider the power function $P(s) = 0.2 + 0.8s^3$. The derivative of $P(s)/s$ is $\frac{\delta P(s)/s}{\delta s} = 1.6s - 0.2/s^2$, which is null for $s = s^* = 0.5$, implying that scaling the speed below 0.5 is not energy efficient. This can be easily shown by considering a task with WCET = 10 time units while assuming that it can be executed at any speed

Table I. Sample Task Set

| Task | T=D (ms) | $C(s_{max})$ (ms) | $AET(s_{max})$ (ms) | $U_{WCET}(s_{max})$ | $U_{AET}(s_{max})$ |
|---|---|---|---|---|---|
| $\tau_1$ | 50 | 20 | 10, 20, 15, 12, 10, 10 | 0.4 | 0.256 |
| $\tau_2$ | 100 | 20 | 15, 10, 18 | 0.2 | 0.143 |
| $\tau_3$ | 150 | 15 | 12, 10 | 0.1 | 0.073 |
| Total | | | | 0.7 | 0.473 |

$\in \{0.2, 0.5, 0.7, 1.0\}$ without missing its deadline. The relative energy consumptions for executing the task at the different speed assignments are $E(0.2) = P(0.2) * 10/0.2 = 10.32$, $E(0.5) = P(0.5) * 10/0.5 = 6$, $E(0.7) = P(0.7) * 10/0.7 = 6.8$, and $E(1.0) = P(1.0) * 10 = 10$. The minimum energy consumption is indeed obtained for $s^*$, while it increases at both lower and higher speeds. One can see that the energy consumption of a task is a quadratic function with global minimum at $s^*$. Such an analysis minimizes only the energy consumption during the time intervals when tasks are executed, because it implicitly assumes a negligible power consumption during the CPU idle intervals.

The slack of a job refers to the CPU time that it does not use before its deadline. Hence, the *static slack* available to any job of a task $\tau_i$ can be computed offline as $slack_i = D_i - R_i$, where $R_i$ is the worst-case response time of $\tau_i$. At runtime, extra slack (referred to as *dynamic slack*) may become available when the job completes early, without consuming its WCET.

The DVFS solutions also can be classified as *intertask* and *intratask* algorithms. In intertask algorithms, when a job is dispatched, it is guaranteed to execute at the same speed level until it completes or is preempted by another (high-priority) job. When it resumes execution (after preemption), the scheduler may readjust its speed by considering the available slack at that time. The *intertask* algorithms form the majority of the current DVFS solutions, as it requires only the information about the WCET of the jobs and involves low runtime overhead. On the other hand, if the information about the execution time of the job is available, in particular its *probability distribution*, then there may be benefits in adjusting the job's speed *while* it is in progress, at well-determined points. This is the main idea behind the *intratask* algorithms [Xu et al. 2004, 2005; Shin et al. 2001], in which the job starts to execute at a low-speed level (relying on the fact that its early completion is more likely than the worst-case scenario), and then its speed is increased *gradually* at well-determined *power management points (PMPs)* as it continues to execute. Thus, for each task, a *speed schedule* is computed offline, showing what speed level will be assigned to its jobs during their execution, and at what point. The intratask algorithms aim at minimizing the *expected dynamic energy consumption*; however, they also require that the compiler generate code to enable the application to make system calls to the operating system at the well-determined PMPs during job execution, and they involve more overhead due to more frequent speed changes.

In this section, the task set shown in Table I is used as a running example. For each task, the *Actual Execution Time (AET)* of its jobs within the *hyperperiod* (defined as the least common multiple of all the task periods) is reported in the AET column. Note that the worst-case utilization of the task set at the maximum speed is equal to 0.7, whereas its average utilization is 0.473. For the sake of simplicity, in the examples, the speed scaling overhead and the power consumption in the idle state are considered negligible and the processor is assumed to have five discrete speed levels: 0.2, 0.4, 0.6, 0.8, and 1.0. The power function $P(s) = s^3$ is assumed when the processor is in the active state. In addition, it is assumed that the task execution times scale linearly with the processing speed.

Table II. Summary of DVFS Algorithms with Static Slack Reclaiming

| Algo. | Reference | Speed Set | $C(s)$ | Periodicity | Scaling Overhead | $P(s)$ | Scheduler | Complexity |
|---|---|---|---|---|---|---|---|---|
| YDS | Yao et al. [1995] | cont. | $C/s$ | aper. | no | $s^\alpha$ | EDF | $O(n \log^2 n)$ |
| SVS | Pillai-Shin [2001] | disc. | $C/s$ | per. | no | $\beta s^\alpha$ | EDF | $O(n)$ |
| SVS | Pillai-Shin [2001] | disc. | $C/s$ | per. | no | $\beta s^\alpha$ | RM | pseudo-poly. |
| AMMA | Aydin et al. [2001] | cont. | $C/s$ | per. | no | $\beta s^\alpha$ | EDF | $O(n^2 \log n)$ |
| ADZ | Aydin et al. [2006] | cont. | $x/s+y$ | per. | no | $P_{\text{ind}}+P_{\text{dep}}(s)$ | EDF | $O(n^3)$ |
| BBL | Bini et al. [2009] | disc. | $x/s+y$ | any | yes | op. modes | EDF/RM | $O(2^n)$ |
| AVR | Yao et al. [1995] | cont. | $C/s$ | aper. | no | $s^\alpha$ | EDF | $O(n)$ |
| QGF | Qadi et al. [2003] | cont. | $C/s$ | spor. | no | $\beta s^\alpha$ | EDF | $O(1)$ |

Table III. DVFS Algorithm Summary with Dynamic Slack Reclaiming

| Algorithm | Reference | Speed Set | Periodicity | Speed Scaling Overhead | $P(s)$ | Complexity |
|---|---|---|---|---|---|---|
| OLDVS | Lee and Shin [2004] | discrete | aperiodic | no | $\beta s^3$ | $O(1)$ |
| ZMu | Zhu and Mueller [2005] | continuous | periodic | no | $\beta s^3$ | $O(n)$ |
| OLDVS* | Gong et al. [2007] | discrete | aperiodic | no | $\beta s^3$ | $O(1)$ |
| LSP | Lawitzky et al. [2008] | discrete | sporadic | yes | $\beta s^3$ | $O(k)$ |
| BSDVFS | Bambagini et al. [2011] | discrete | periodic | yes | $\beta s + \gamma$ | $O(k)$ |
| BSDVFS* | Bambagini et al. [2011] | discrete | periodic | yes | $\beta s + \gamma$ | $O(k)$ |

Table IV. DVFS Algorithm Summary with Both Static and Dynamic Slack Reclaiming

| Algorithm | Reference | Speed Set | Scheduler | Complexity |
|---|---|---|---|---|
| cc-EDF | Pillai and Shin [2001] | discrete | EDF | $O(n)$ |
| cc-RM | Pillai and Shin [2001] | discrete | RM | pseudo-polynomial |
| LA-DVS | Pillai and Shin [2001] | discrete | EDF | $O(n)$ |
| DRA-OTE | Aydin et al. [2004] | continuous | EDF | $O(n)$ |
| AGR | Aydin et al. [2004] | continuous | EDF | $O(n)$ |

The overall energy consumption in a hyperperiod under the EDF scheduling policy is $E = 210$mJ and $E = 142$mJ considering the WCET and AET scenarios, respectively.

The rest of this section provides an overview of the most relevant DVFS algorithms, divided according to the type of slack they exploit: static slack (Section 4.1), dynamic slack (Section 4.2), or both (Section 4.3). The presented algorithms, classified according to the slack exploitation mechanism, are summarized in Tables II, III, and IV.

### 4.1. Static Slack Reclaiming

One of the first papers on DVFS-based energy-aware scheduling was by Yao et al. [1995]. The paper presented three algorithms by considering aperiodic tasks, continuous CPU speed, no speed scaling overhead, negligible power consumption during idle intervals, and task computation times inversely proportional to CPU speed ($C(s) = C/s$). The

first algorithm, hereafter referred to as YDS, consists of recursive identification of time intervals with maximum *computational density* (defined as the sum of CPU cycles of the tasks with arrival and deadline within the interval, divided by the length of the interval length). Specifically, the algorithm identifies the interval with the maximum intensity, sets the CPU speed to the intensity value for that interval, and is recursively reinvoked for the remaining execution intervals in the schedule. The offline algorithm is proved to be optimal and has an $O(n \log^2 n)$ complexity for $n$ aperiodic jobs. A second algorithm, executed online, considers jobs that may arrive dynamically. The algorithm recomputes the optimal schedule at each arrival time considering only the new and pending jobs. The third algorithm (AVR) sets the speed, for each instant, equal to the sum of density of those jobs whose arrival and deadline range contains the time instant under consideration. Although the complexity of AVR is lower than the previous optimal approaches, deadline misses may occur. In fact, since the speed is set equal to the sum of the worst-case utilization of the active jobs, the processor can be significantly slowed down when there are few active tasks, so the system may not terminate the jobs by their deadlines if additional tasks arrive.

Ishihara and Yasuura [1998] provided an analysis for synchronous frame-based real-time tasks (with identical release time and period), proving that under their assumed system model (no overhead and all tasks consume the same amount of energy), the energy is minimized when each job completes just at its respective deadline. That result implies that on a system with continuous speed/voltage, the total energy is minimized at the speed/voltage that reduces the idle time to zero. While that result is also implicit in the optimal YDS algorithm mentioned earlier, the main contribution of Ishihara and Yasuura [1998] is the derivation of an important property of the systems with discrete speed levels: when the system is constrained to use a finite set of speed/voltage, the energy is minimized by using the two speed/voltage values adjacent to the speed value that is optimal assuming a continuous range.

Aydin et al. [2001] proposed an optimal offline algorithm (abbreviated as AMMA in this survey) for selecting the running speed of periodic tasks with different energy features (e.g., due to the use of different system components, such as FPU). The paper's main contribution consists of showing that each task $\tau_i$ can execute at a constant speed $s_i$ whenever it is dispatched, without affecting the energy optimality. The paper also proposed an algorithm with complexity $O(n^2 \log n)$ to compute the optimal speed for each task, while preserving feasibility under EDF.

Aydin et al. [2006] proposed another algorithm (referred to as ADZ) considering periodic independent tasks with a more general computational model, where task execution time includes a speed-dependent portion and a constant part, and a more sophisticated power model with leakage power, frequency-dependent power (e.g., due to processing core), and frequency-independent power (e.g., due to the peripherals and memory) components. On the other hand, the speed range is continuous and the speed scaling overhead is neglected. The authors formulated a nonlinear optimization problem with convex constraints, where the objective is to minimize the overall energy consumption by finding the optimal speed for each task while guaranteeing their deadlines under EDF. The authors also showed that the problem can be solved in time $O(n^3)$ thanks to the Kuhn-Tucker optimality conditions for this class of convex problems. The analysis is enhanced by an online improvement that considers early task completions.

One of the earliest efforts that exclusively focused on sporadic tasks is Qadi et al. [2003]. The algorithm proposed in the paper, abbreviated as QGF in this survey, starts by running the task set at the lowest possible speed. When a new job is released, the speed is increased by the task utilization (WCET divided by the minimum interarrival time) only for an interval whose length is equal to the minimum interarrival time. The

algorithm was implemented in a $\mu$C/OS-II system and tested on a real platform while considering a continuous speed spectrum.

The problem of finding an optimal solution on a system with discrete speed levels was discussed in Bini et al. [2009] for a set of periodic or sporadic tasks under both EDF and Fixed-Priority (FP) scheduling policies. The authors provided a method (referred to as BBL) to compute the optimal speed offline (first assuming a continuous speed spectrum) and then introduced a speed modulation technique to achieve the target speed using two discrete values. The analysis selects the pair of available frequencies that minimize the energy consumption by also incorporating time and energy switching overheads. The execution time consists of a part that is speed dependent and another one that is not.

Yun and Kim [2003] proved that under fixed-priority assignments, the energy-aware scheduling problem with real-time constraints is NP-hard. Hence, the authors presented a Fully Polynomial Time Approximation Scheme (FPTAS), which, for each $\epsilon > 0$, guarantees an energy consumption that is greater than the optimal one at most by a factor of $1 + \epsilon$. Quan and Hu [2002] presented a deadline transformation algorithm for expressing the problem as a set of EDF-based problems, whose optimal schedules can be computed using the method proposed by Yao et al. [1995]. Since the transformation process is expensive, a more efficient strategy is also provided.

### 4.2. Dynamic Slack Reclaiming

In this section, we address the DVFS algorithms that exploit the dynamic slack, with a summary presented in Table III. All the algorithms here considered are based on EDF and assume that the computational times scale linearly with the speed ($C(s) = C/s$).

Lee and Shin [2004] proposed an algorithm (referred to as OLDVS) that accumulates the dynamic slack due to early completions and exploits it to decrease the CPU speed so that the current task is completed at the same time that it would complete in the schedule with the worst-case workload. The idea was improved in Gong et al. [2007] through the intratask algorithm OLDVS*, which divides each job execution in two parts: the first part is executed at a low-speed level and the speed is increased if it does not complete by the end of the first part. This approach implicitly assumes that the probability of completing the job in the first part is significantly higher than finishing in the second half. Both algorithms assume a discrete set of speeds, negligible power consumption during the idle intervals, and zero switching overhead.

Bambagini et al. [2011] extended the previous approaches by considering the switching overheads. More precisely, the enhanced algorithms (BSDVFS and BSDVFS*) check whether the dynamic slack is long enough to execute the next job at the desired speed, considering the overhead for switching to the new speed and then restoring the nominal speed (which guarantees the task set feasibility in the worst case). The two algorithms were implemented on a real embedded platform and the experiments showed also the negative impact of the leakage consumption on the overall energy figures.

Zhu and Mueller [2005] combined the DVFS mechanism with feedback control theory to save energy for periodic real-time task sets with uncertain execution times. Their approach, abbreviated as ZMu, uses a PID controller to compute the estimated execution time of the next job as a function of the difference between the actual and the expected execution time of the previous job of the same task. The plant in the closed control loop is represented by the EDF scheduler. The frequency/voltage selection is greedy, as it considers the estimated execution time for the running task and WCET for the others. Moreover, the frequency spectrum is assumed to be continuous and the speed scaling overhead is considered negligible. It is also assumed that the CPU uses the lowest speed level during the idle intervals.

Lawitzky et al. [2008] implemented an energy-saving algorithm (referred to as LSP) based on the Rate-Based Earliest Deadline (RBED) framework [Brandt et al. 2003],

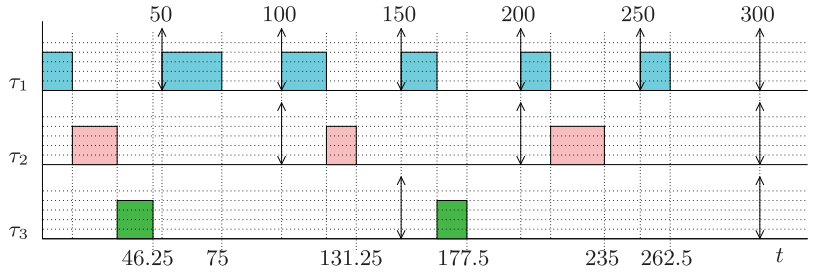



Fig. 4. SVS algorithm with early terminations.

which supports CPU time budget allocation and dispatching. The paper took speed scaling overhead into account and offers a system-wide view by considering not only the CPU but also bus and memory. The speed scaling overhead is automatically accounted within the CPU budget assigned to each task. In addition, the authors proposed to manage the static slack, which otherwise would be entirely allocated to non-real-time tasks. Their proposal consists of increasing the utilization values of real-time tasks to exploit the entire remaining static slack, even though the actual execution times are not changed. In such a way, at runtime, the overestimated utilization is automatically transformed into dynamic slack, which is in turn easily handled within the presented framework.

### 4.3. Dynamic and Static Slack Reclaiming

In this section, we overview the DVFS algorithms that reclaim both dynamic and static slack. The algorithms' main features are reported in Table IV. All the algorithms reported here consider periodic tasks whose computational times scale linearly with the speed ($C(s) = C/s$). Moreover, the speed scaling overhead is considered negligible and the power consumption is modeled by the function $P(s) = \beta \cdot s^3$.

Pillai and Shin [2001] proposed three algorithms considering both EDF and RM scheduling policies. The first approach, referred to as *Static Voltage Scaling (SVS)*, runs offline and exploits only the static slack: when the system starts, the running speed is set equal to the lowest available speed level that guarantees the task set feasibility. Figure 4 shows the SVS execution on our example task set with early terminations. The speed is set equal to 0.8, which is the slowest one higher than or equal to the worst-case utilization, 0.7, consuming 90.88mJ in a hyperperiod.

Then, the *cycle-conserving algorithm* (cc-EDF and cc-RM) is introduced. The algorithm, at every scheduling event, sets the running speed to the lowest level that guarantees timing constraints using the actual execution time for the completed jobs and the WCET information for future jobs. Notice that the cc-EDF algorithm generates a schedule identical to the SVS schedule if the actual workload is identical to the worst case. An instance of cc-EDF execution with early completions is reported in Figure 5, where the average execution speed is 0.684 and the overall dissipation is 70.58mJ.

The last proposed algorithm, called Look-Ahead RT-DVS (LA-DVS), runs only under EDF and aims at further reducing the running speed of the current (earliest-deadline) job as much as possible, while still guaranteeing the deadlines of other jobs. Hence, although the actual speed until the next deadline can be quite low, it may be necessary to execute future jobs at high-speed levels to meet their timing constraints, in case the current job takes (close to) its WCET. However, this side effect is significantly reduced thanks to frequent early task completions in practice. As shown in Figure 6, this algorithm can be considered proactive (in contrast to cc-EDF, which is reactive), in that it scales the speed down whenever possible and then, any task early termination

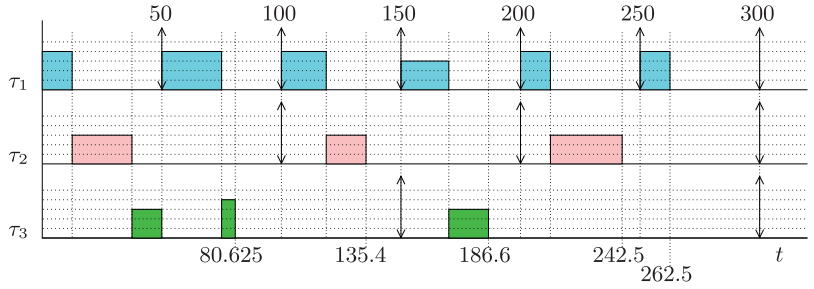



Fig. 5. cc-EDF algorithm with early terminations.

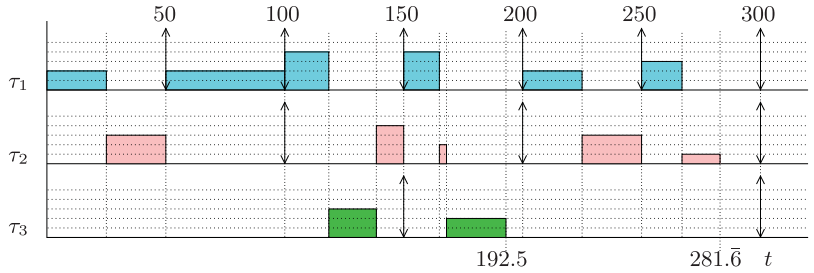
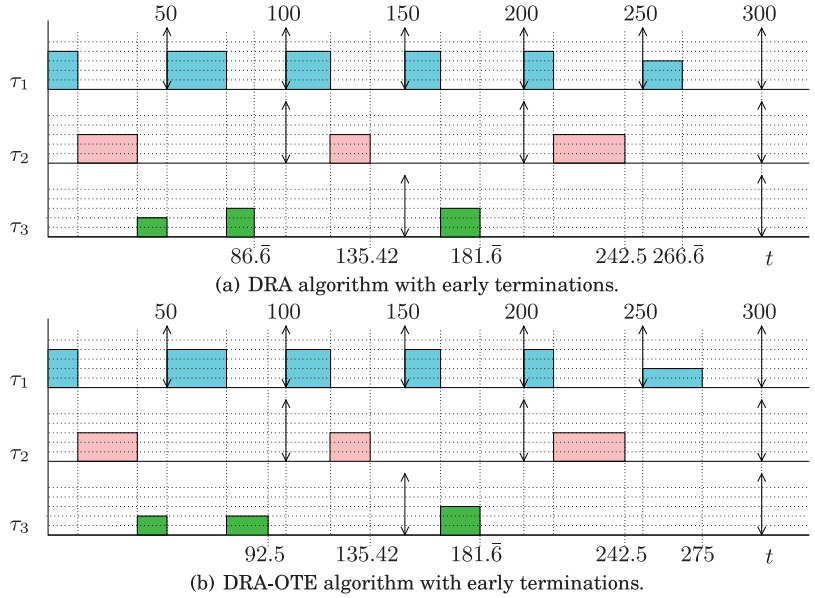



Fig. 6. LA-DVS algorithm with early terminations.

further improves the energy saving. The average execution speed is 0.518, which is slightly higher than the actual utilization, and the energy consumption is 50.04mJ.

Aydin et al. [2004] proposed three algorithms at increasing complexity and sophistication levels, for periodic real-time tasks. All the algorithms assume a continuous speed range and a negligible switching overhead. The first algorithm computes the running speed as the utilization of the task set (similar to SVS) and it is not changed at runtime. The algorithm works with all the scheduling algorithms that guarantee the full utilization of the processor while guaranteeing the feasibility, such as EDF and Least Laxity First (LLF).

The second algorithm *(Dynamic Reclaiming Algorithm, DRA)* uses a queue structure called $\alpha$-queue where each element contains the deadline and the remaining execution time $rem_i$ of task $\tau_i$. When a task arrives, its absolute deadline and execution time at the optimal speed are inserted in the $\alpha$-queue. At every scheduling event, the $rem_i$ field of the $\alpha$-queue's head is decreased by the amount of the elapsed time since the last event. In other words, the $\alpha$-queue represents the ready queue in the worst-case schedule at that specific time. Once a new job is about to be scheduled, its remaining execution time is summed with the $rem_i$ values in $\alpha$-queue whose deadlines are less than or equal to the task in question, and then the speed is scaled accordingly. This procedure enables the current job to reclaim the dynamic slack of already completed higher-priority jobs, while still ensuring it does not complete later than the instant when it would complete in the worst-case schedule. The algorithm is improved by incorporating the One Task Extension (DRA-OTE) technique, which, when there is only one task in the ready queue and its worst-case completion time at the current speed falls earlier than next scheduling event, slows the speed down to let the task terminate at the next event. The schedule produced by DRA, associated with our running example, is reported in Figure 7(a), giving an average speed of 0.67. In addition, the OTE feature further improves the performance, reducing the average speed down to 0.625, as depicted




(a) DRA algorithm with early terminations.

(b) DRA-OTE algorithm with early terminations.

Fig. 7. DRA and DRA-OTE algorithms.

in Figure 7(b). During the hyperperiod, DRA and DRA-OTE consume 68.88mJ and 65.48mJ of energy, respectively.

The third algorithm, *Aggressive Speed Reduction - AGR 1*, relies on the idea that when all the ready tasks have deadlines earlier than the next task arrival time, the computational budget can be exchanged among those tasks without affecting the feasibility. Specifically, in such a situation, the algorithm reduces the speed of the current job by allocating some of the CPU time of other low-priority ready tasks. This approach may force other pending tasks to execute at very high-speed levels to meet their deadlines in some execution scenarios. To mitigate this, another algorithm (AGR-2) is proposed, which limits the extent of the slowdown for the current task by considering the information about the average case workload.

Saewong and Rajkumar [2008] presented a framework for fixed-priority periodic real-time tasks and batch (non-real-time) tasks. Specifically, the objective is to minimize the energy consumption while providing enough computational capability to guarantee reasonable response times to batch tasks without missing any deadline. The first algorithm, called *Background Preserving (BG-PRSV)*, increases the system frequency to execute the incoming batch tasks, while the second, denoted as *Background on Demand (BG-OND)*, alternates the execution between a *normal mode* and a *turbo mode* (the latter using a higher frequency) according to the pending batch workload. The speed selection at the scheduling point involves the analysis of the available slack (both static and dynamic) to find the lowest possible frequency that still meets the deadlines. The proposed algorithms assume a discrete frequency range, negligible power consumption at idle state, and cubic power function. Moreover, the effect of the limited number of speed levels on the algorithm performance has been studied in Saewong and Rajkumar [2003]. Several solutions have been proposed and implemented for different types of architectures and fixed-priority periodic tasks:

—*Sys-Clock* (for systems with considerable speed/voltage scaling overhead): a single (system-wide) frequency is computed and kept constant until the workload changes.

Table V. Summary of Offline DPM Algorithms

| Algorithm | Reference | Periodicity | Dynamic Slack | Scheduler | Offline Complexity |
|---|---|---|---|---|---|
| HSCTB1 | Huang et al. [2009b] | aperiodic | no | EDF | pseudo-polynomial |
| HSCTB2 | Huang et al. [2009a] | aperiodic | implicit | EDF/FP | pseudo-polynomial |
| RHS | Rowe et al. [2010] | periodic | implicit | RM | O(1) |
| ES-RHS | Rowe et al. [2010] | periodic | implicit | RM | O(1) |

—*PM-Clock* (for systems with low-speed/voltage scaling overhead): for each task a separate frequency is computed and the speed is adjusted at each context switch.
—*Opt-Clock*: a nonlinear offline optimization problem formulation is used to determine the optimal speed for each task to minimize the overall energy consumption.
—*DPM-Clock*: the dynamic slack is managed at runtime and is assigned to the next job in its entirety.

## 5. UNIPROCESSOR DPM ALGORITHMS

DPM-based energy management algorithms are based on the principle of putting the processor to low-power (sleep) states at runtime. A main problem involved in DPM research is to make sure that the transitions are beneficial in terms of energy savings, because as explained in Section 2.1, there is a minimum time interval (called the *break-even time*) that amortizes the time and energy overhead associated with each transition. In fact, a common technique is to use the *task procrastination* technique, which postpones the execution of the ready jobs as much as possible by exploiting the system slack at that time, thereby compacting busy periods and yielding long idle intervals. By doing so, the number of runtime transitions and overhead are also reduced. On the other hand, utmost care must be taken to avoid the violation of the timing constraints in real-time systems when employing the procrastination technique.

In recent years, the DPM-based techniques have received more attention compared to the DVFS-based schemes, which previously dominated the research area. There are several reasons for this trend. First, with increased scaling in CMOS technology, DVFS is able to save a smaller amount of energy by reducing the dynamic energy consumption. On the other hand, DPM techniques have been motivated by the rising impact of leakage power in modern computing platforms, as highlighted by Kim et al. [2003]. In addition, new processors are equipped with multiple low-power states, each offering different energy and overhead characteristics, and give increased runtime flexibility.

Moreover, DPM techniques can also mitigate some problems associated with the DVFS technique as reported in the literature, including reliability degradation and increased preemption overhead. For instance, Zhang and Chakrabarty [2003] and Zhu et al. [2004] report that scaling down the voltage and frequency has a negative impact on the system reliability, as it may increase the rate of transient faults by several orders of magnitude. Another side effect of DVFS techniques has been identified by Kim et al. [2004] as increased number of preemptions, which leads to higher runtime overhead and higher energy consumption.

The rest of this section introduces the most common DPM approaches proposed in the literature. The offline and online DPM algorithms are discussed in Sections 5.1 and 5.2, respectively. The respective summaries of the main features of the algorithms are presented in Tables V and VI.

All the algorithms discussed in this section consider the break-even times for the CPU explicitly in their analysis. Even though some papers consider only a single low-power state, we note that their approach can be easily extended to systems with

Table VI. Summary of Online DPM Algorithms

| Algorithm | Reference | Periodicity | Dynamic Slack | Scheduler | Online Complexity |
|---|---|---|---|---|---|
| LC-EDF | Lee et al. [2003] | periodic | implicit | EDF | $O(n)$ |
| LC-DP | Lee et al. [2003] | periodic | implicit | RM | $O(1)$ |
| ERTH | Awan and Petters [2011] | sporadic | explicit | EDF | pseudo-polynomial |

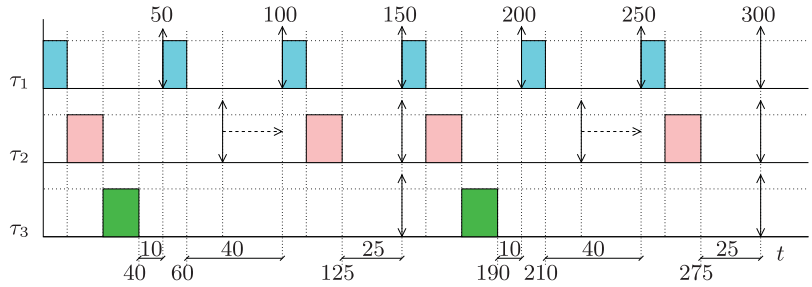
multiple low-power states by exploiting the "deepest" inactive state with break-even time shorter than or equal to the length of the available idle interval.

### 5.1. Offline DPM Algorithms

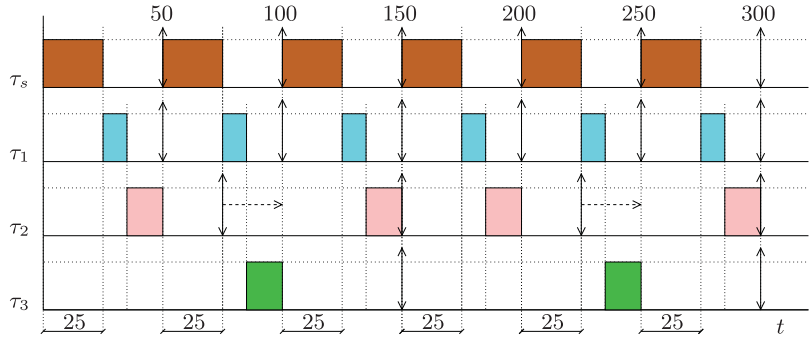
Huang et al. [2009b] proposed an offline analysis technique to devise a periodic scheme that defines active and sleep phases for event streams. The analysis computes the duration of the phases assuming that event arrivals are described using *Real-Time Calculus* [Thiele et al. 2000]. During the active phase, the execution takes place at the maximum speed. The sleep intervals that result from this approach are typically shorter and more frequent than those obtained through the procrastination algorithms. The algorithm does not consider task early terminations in the analysis. Huang et al. [2009a] proposed an online algorithm that procrastinates job executions by considering the pattern of arrivals observed in recent history and the ones estimated by the analysis through the Real-Time Calculus. Unlike the first algorithm, dynamic slack is implicitly exploited by the work-conservative nature of the algorithm. The algorithms are referred to as HSCTB1 and HSCTB2, respectively. Standby and sleep states are considered, assuming a negligible and nonnegligible transition overhead, respectively.

Rowe et al. [2010] presented two techniques to harmonize task periods with the aim of clustering task executions (i.e., to combine processor idle times whenever possible). The framework assumes a system without the DVFS feature. The first algorithm, *Rate-Harmonized Scheduler (RHS)*, introduces the concept of *harmonizing period ($T_H$)*. The scheduler is notified by the task arrivals only at the integer multiples of the harmonizing period. The harmonized period is computed as a function of the shortest period. For instance, if the effective arrival time is at 3.5 and the harmonizing period is 1, then the scheduler considers this arrival only at time 4. Since all the arrivals are considered at integer multiples of the harmonizing period, if there is no task to execute, then the processor can be put in sleep state until the next period. The approach considered fixed-priority tasks whose priorities are assigned by the Rate Monotonic policy. Although the exact schedulability can be checked by evaluating the worst-case response time through the Time Demand Analysis, the utilization bound for schedulability reduces to 0.5, in the general case. The second algorithm, called *Energy-Saving RHS (ES-RHS)*, introduces a new task with period equal to $T_H$ (highest priority). Its computation time is evaluated by considering $T_H$ and the spare utilization. The new task enables putting the processor to sleep state when it is invoked and when its computational budget is longer than or equal to the break-even time. The main advantage of ES-RHS with respect to RHS is that the idle times generated by task early terminations extend the sleep interval in the next period. In such a way, multiple short idle intervals can be combined to a single long interval, giving an advantage over RHS. Two low-power states are taken into account, idle and sleep, considering a short and long break-even time, respectively.

An example for the RHS and ES-RHS algorithms is presented in Figures 8(a) and 8(b), respectively, considering three tasks with overall utilization $U = 0.5$: $\tau_1$ ($C_1 = 10$ms and $T_1 = 50$ms), $\tau_2$ ($C_2 = 15$ms and $T_2 = 75$ms), and $\tau_3$ ($C_3 = 15$ms and $T_3 = 150$ms). All the algorithms harmonize periods with respect to $T_H = T_1 = 50$ms. ES-RHS introduces the new task $\tau_s$, characterized by a period $T_s = T_H$ and execution




(a) RHS algorithm in the worst case.




(b) ES-RHS algorithm in the worst case.

Fig. 8. RHS and ES-RHS algorithms.

time $C_s = (1 - U) \cdot T_s = 25$ms. We assume a system with the following power characteristics: $P(1.0) = 1.0$ W, $P_\sigma = 0.2$ W, $B_\sigma = 6$ms, and $E_\sigma = 6$mJ. Both algorithms generate feasible schedules with frequent state transitions. We observe that RHS is not always able to exploit the sleep state during idle intervals, leading to an overall energy consumption of 208mJ. On the other hand, ES-RHS manages to exploit the sleep state more effectively, consuming 198mJ of energy during the hyperperiod.

### 5.2. Online DPM Algorithms

Lee et al. [2003] proposed two leakage control algorithms for procrastinating task executions as long as possible, to prolong and compact idle intervals, both under dynamic (LC-EDF) and fixed (LC-DP) priority scheduling. Both algorithms assume periodic tasks with periods equal to the deadlines and a system without DVFS feature. The main idea behind the algorithms is to compute at each job arrival the maximum time the job can be delayed without missing its deadline. Under EDF scheduling, whenever the CPU becomes idle, LC-EDF computes the maximum time duration $\Delta_k$ that the task with the earliest arrival time ($\tau_k$) can be delayed by using the following equation:

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

Then, the system is put to the low-power state (procrastinated) for $\Delta_k$ time units. If another higher-priority task $\tau_j$ with absolute deadline shorter than the $\tau_k$'s deadline arrives before the end of the procrastination interval, the procedure is executed again, by considering the length of the idle interval already elapsed, $\delta_k$, and obtaining the new

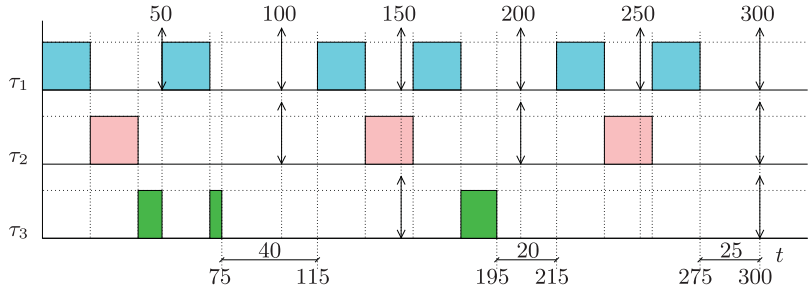



Fig. 9. LC-EDF algorithm in the worst case.

value of the procrastination interval $\Delta_j$ through the following equation:

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

For fixed-priority systems, the authors resort to the *dual priority* scheme [Davis and Welling 1995] in order to compute the length of the procrastination interval. More precisely, the additional sleep time is computed as the minimum *promotion time* $Y_i$ (relative deadline minus the worst-case response time) among the tasks in the lower run queue. The promotion time of each task is computed statically as the difference between its relative deadline and the worst-case response time, derived from Time Demand Analysis. The main limitation of such an approach is that it requires a dedicated hardware to implement the algorithms and manage sleep and wake-up operations. Although task early terminations are not directly involved in the analysis, the work-conserving (nonidling) nature of the algorithms can indirectly incorporate the dynamic slack at runtime.

As an example, consider the task set with parameters $C_1 = 10$ms, $T_1 = 50$ms, $C_2 = 10$ms, $T_2 = 100$ms, $C_3 = 7.5$ms, and $T_3 = 150$ms. The processor's power characteristics are characterized by the following parameters: $P(1.0) = 1.0$W, $P_\sigma = 0.2$W, $B_\sigma = 15$ms, and $E_\sigma = 6$mJ. The schedule that is generated by the LC-EDF algorithm is shown in Figure 9. We observe that there are three idle intervals in the schedule, lasting for 40, 20, and 25ms, respectively. In addition, the overall energy dissipation is 236mJ.

Awan and Petters [2011] proposed an algorithm under EDF, called *Enhanced Race-to-Halt (ERTH)*, which targets dynamically monitoring and accumulating both static and dynamic slack, in order to apply the DPM technique effectively. The authors considered sporadic tasks with different criticality (hard, soft real-time, and best effort) and a processor model with several low-power states. Essentially, the algorithm uses a single counter to keep track of both static and dynamic slack. When the system is idle, the processor is put to the deepest low-power state with break-even time not exceeding the amount of the existing slack at that time. Similarly, if there are some ready tasks, and the amount of available slack is longer than or equal to the break-even time, then the processor is switched off as long as possible without causing any deadline miss. On the other hand, if the amount of slack is less than the break-even times, the processor executes the current workload at the maximum speed and then attempts to switch to a sleep state when idle.

## 6. INTEGRATED DVFS-DPM ALGORITHMS FOR UNIPROCESSOR SYSTEMS

This section considers the algorithms that use both DVFS and DPM techniques. Specifically, these *integrated* algorithms exploit both speed scaling and low-power states to maximize energy savings, unlike the techniques that use only one feature.

Table VII. Summary of Integrated Algorithms That Compute Speed Scaling Factors at Design Time (Offline)

| Algorithm | Reference | Scheduler | Online Complexity |
|---|---|---|---|
| CD-DVS-P | Jejurikar et al. [2004] | EDF | $O(1)$ |
| CD-DVS-P1 | Jejurikar and Gupta [2004] | FP | $O(1)$ |
| CD-DVS-P2 | Jejurikar and Gupta [2004] | DP | $O(1)$ |
| OSS | Chen and Kuo [2006] | RM | $O(n\log(n))$ |
| VOSS | Chen and Kuo [2006] | RM | $O(n\log(n))$ |
| BBMB | Bambagini et al. [2013] | FP | $O(1)$ |

Table VIII. Summary of Integrated Algorithms That Compute Speed Scaling Factors at Runtime (Online)

| Algorithm | Reference | Dynamic Slack | Scheduler | Online Complexity |
|---|---|---|---|---|
| DVSLK | Niu and Quan [2004] | implicit | EDF | pseudo-polynomial |
| FPLK | Quan et al. [2004] | implicit | FP | $O(1)$ |
| DSR-DP | Jejurikar and Gupta [2005a] | explicit | EDF | $O(1)$ |

The simplest solution consists of exploiting a feature when the other is not applicable at a specific point during the execution. For example, if the available slack is shorter than the processor's break-even time and the jobs cannot be procrastinated, then the system may choose to scale speed to reduce energy, while meeting deadlines. Conversely, if there is ample slack at runtime, it is possible that the speed to exploit all the available slack is less than the critical speed $s^*$. In that case, the system can scale the speed only up to $s^*$ and then the processor can be put in sleep state during the remaining idle interval. However, more sophisticated techniques give the same importance to the two techniques, with the objective of compacting idle intervals together (to make better use of DPM) and using speed scaling (DVFS) at appropriate levels to reduce the dynamic energy.

We examine the integrated algorithms in two sections: in Section 6.1, we overview the algorithms that make the speed scaling decisions offline, and in Section 6.2, we consider those that compute the speed scaling factors online. The main features of the offline and online algorithms are summarized in Tables VII and VIII, respectively.

As a running example in this section, we consider a periodic task set with three tasks and the following parameters: $C_1 = 10$ms, $T_1 = 50$ms, $C_2 = 10$ms, $T_2 = 100$ms, $C_3 = 7.5$ms, and $T_3 = 150$ms. For the sake of simplicity, computational times are supposed to scale linearly with the speed ($C(s) = C/s$). Let us consider the following power characteristics: $P(s^*) = 1.0$W, $P_\sigma = 0.2$W, $B_\sigma = 15$ms, and $E_\sigma = 6$mJ. We assume that the critical speed for this system is $s^* = 0.5$ and the task set is feasible under Rate Monotonic (i.e., fixed) priority assignments at such a speed. Since idle intervals are usually shorter than the break-even time (15ms), it is not possible to switch to the sleep states during them, leading to an overall energy dissipation of 253mJ.

### 6.1. Offline Speed Scaling

This section discusses a number of algorithms that statically assign speed scaling factors to individual tasks during the design (i.e., offline) phase and, at runtime, exploit low-power states to further reduce energy dissipation. Moreover, all these algorithms are designed for periodic real-time tasks and do not explicitly consider dynamic slack.

Jejurikar et al. [2004] proposed an approach (CS-DVS-P) based on the critical speed analysis and task procrastination for periodic preemptive tasks executed under the EDF scheduling policy. Offline, the algorithm first computes the lowest speed (higher than or equal to the critical speed, $s^*$) that guarantees the task set feasibility. Then, the maximum amount of time ($Z_i$) each job of task $\tau_i$ can spend in the sleep state within its

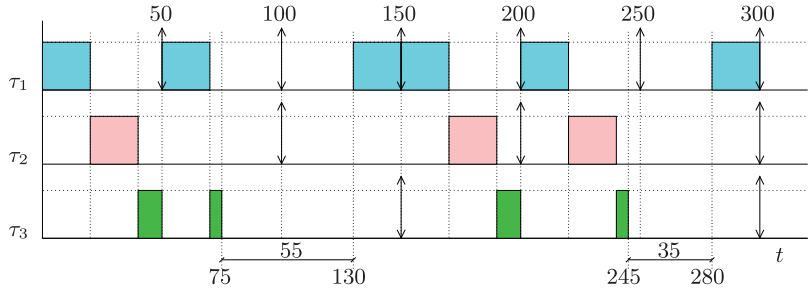



Fig. 10. CS-DVS-P1 algorithm in the worst case.

period without leading to any deadline miss is evaluated using the following equation:

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

At runtime, when there is no pending job, the processor is put in a low-power sleep state (as deep as justified by the break-even time and available slack) until the next job arrival. When a job arrives and the processor is still in sleep mode, an external controller continues to keep the processor in sleep state for an additional time period computed as the minimum of the remaining time to wake up and the precomputed delay of the newly arriving job.

Jejurikar and Gupta [2004] extended the algorithm to fixed-priority (CS-DVS-P1) and dual-priority (CS-DVS-P2) systems. With respect to the original algorithm given in Jejurikar et al. [2004], only the computation of the $Z_i$ values is different, leaving the online step the same. Moreover, the authors showed that the dual-priority scheduler is able to guarantee longer $Z_i$ values than the fixed-priority scheduler. The resulting schedule of CS-DVS-P1 for the task set under analysis is reported in Figure 10 while executing the task set at the critical speed $s = s^* = 0.5$. The promotion times are $Y_1 = 30$ms, $Y_2 = 60$ms, and $Y_3 = 75$ms. The schedule has two idle intervals lasting for 55 and 35ms, respectively. In a hyperperiod, the overall energy dissipation is 234mJ.

Chen and Kuo [2006] showed that the DPM part of the algorithm proposed by Jejurikar and Gupta [2004] may lead to deadline misses; thus, they proposed two solutions to avoid them, *Online Simulated Scheduling (OSS)* and *Virtual OSS (VOSS)*. Both algorithms consider periodic independent tasks for fixed-priority systems where priorities are assigned according to the Rate Monotonic policy. Initially, all tasks are assigned the lowest speed that still guarantees the feasibility, subject to the lower bound of critical speed. OSS runs when the ready queue is empty and simulates the execution of tasks that arrive earlier than the earliest absolute deadline, accounting for their idle time. Then, the arrivals of those tasks are delayed for the relative accounted time, while the processor is put in sleep mode until the first job arrival (if and only if the available idle time is longer than the break-even time). VOSS enhances OSS by combining the online simulation with the virtual blocking time. Specifically, in the simulation phase, the algorithm considers as arrival time the value of $r_{i,k} + Z_i$, where $Z_i$ represents the maximum blocking tolerance that each task can afford without causing deadline misses. $Z_i$ is computed offline through the response time analysis. In this way, the arrivals of the tasks taken into account result in further delays than those provided in OSS, leading to longer sleep intervals. The complexity of the online step is due to the simulation phase, which is $O(n \cdot \log(n))$, while the offline computation of the virtual blocks has pseudo-polynomial complexity. The OSS execution is illustrated in Figure 11, while VOSS provides a schedule equivalent to CS-DVS-P1's (Figure 10). Note

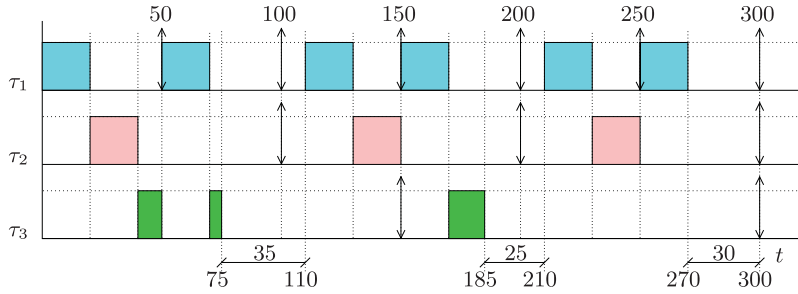



Fig. 11. OSS algorithm in the worst case.

that OSS leads to three idle intervals of length 35, 25, and 30ms, while VOSS compacts them in two longer intervals of length 55 and 35ms. In both cases, the critical speed is used for all the tasks, since it yields a feasible schedule. The energy consumption within the hyperperiod is 237mJ and 234mJ, respectively.

Bambagini et al. [2013] proposed an algorithm for fixed priority tasks, hereafter abbreviated as BBMB, which exploits the limited preemptive scheduling model [Buttazzo et al. 2013] to further reduce energy consumption with respect to the fully preemptive model. More precisely, the algorithm consists of an offline and an online step. At design time, when the nonpreemptive regions are computed, the lowest feasible speed (not lower than the critical speed $s^*$) and the minimum value among all task blocking tolerances are both evaluated. The blocking tolerance for a task is the maximum time interval during which a task can be blocked by lower-priority tasks. At runtime, when the system is idle, 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.

### 6.2. Online Speed Scaling

In this section, we consider integrated schemes that make both DPM and DVFS decisions at runtime to reduce the energy consumption of periodic real-time tasks.

Jejurikar and Gupta [2005a] extended the algorithm in Jejurikar et al. [2004] to explicitly consider task early terminations on dynamic priority systems. The algorithm is called *Dynamic Slack Reclamation with Dynamic Procrastination (DSR-DP)*. The first improvement consists of collecting unused computation times (dynamic slack) in a *Free Run Time (FRT)* list, which also includes information on the priority of the task that generated it. To prevent any deadline miss, each job can only use the dynamic slack generated by tasks with higher or equal priority. Such additional CPU time is partially exploited to slow down the processor speed while the job is executing and also to extend the time spent in sleep state. Specifically, the slack distribution algorithm primarily uses the additional slack to scale the speed down, and if the critical speed is reached, the residual time is used to extend the sleep interval.

Niu and Quan [2004] proposed the DVSLK algorithm, which reduces both leakage and dynamic power consumption, rather than focusing on a single component. The algorithm considers periodic tasks scheduled by EDF. When there are ready tasks, the algorithm selects for each task the speed that minimizes the power consumption due to both static and dynamic energy. Conversely, when the system is idle, the algorithm computes the latest starting time to put the system in a low-power state and postpone task execution as long as possible without leading to a deadline miss. Since all the jobs within the next busy period are considered, the complexity is pseudo-polynomial.

Quan et al. [2004] proposed an enhanced version (FPLK) for systems with fixed-priority tasks. The algorithm has a significantly lower complexity than DVSLK.

Basically, at design time, the algorithm computes the latest activation time for each task, and then, when the processor is idle, it is put to the low-power state until the first job arrival time augmented by the precomputed delay time. Although the delays computed at design time are pessimistic, the online complexity is constant.

Irani et al. [2007] introduced two techniques for dynamic speed scaling with and without low-power states: DSS-S and DSS-NS. DSS-NS is based on using mostly speed scaling, while DSS-S executes the workload at the maximum speed to maximize the use of the low-power states. Both the $P(s)$ and $P(s)/s$ functions are assumed to be convex and the scheduler implements the EDF policy. An offline algorithm for DSS-S and two online solutions for DSS-S and DSS-NS were presented. The main idea behind the offline algorithm is to procrastinate tasks and execute them at a speed no lower than the critical speed. Under the assumptions of convexity, the proposed offline algorithm achieves an approximation ratio of 3 with respect to the optimal solution. However, the overheads due to the speed scaling and state transition are not taken into account.

## 7. DVFS MULTIPROCESSOR ALGORITHMS

Resource management for multiprocessors has been an active research area in real-time systems for decades. When (approximately around 2003) the ever-increasing power densities presented the so-called *power wall* challenge to the designers, it became clear that further increasing the clock frequency was not sustainable and further performance improvements would have to be provided through multicore systems. This has been accompanied by gradually increasing research activity in real-time systems to extend the existing energy-aware scheduling results to multiprocessor platforms.

Most of the existing work in this area considers *homogeneous* multiprocessor systems, although in recent years some efforts have been carried out to generalize the results to heterogeneous systems with different characteristics. Similarly, most of the early papers focused exclusively on dynamic power consumption and implicitly ignored the static power while applying the DVFS technique. Gradually, the research community incorporated the static power into the power management frameworks in various ways.

In terms of the DVFS models, early papers adopted settings with a set of processing cores where the voltage and frequency of each processor can be configured independently. These algorithms can be divided between those fixing a constant frequency for each core (i.e., per-CPU DVFS) and those assigning a frequency for each task (i.e., per-task DVFS), as done for single-core algorithms. Algorithms belonging to the first group are characterized by the computation of a single frequency for each processor that is used independently of the currently running task. The other group consists of algorithms that determine a speed for each task and adapt the CPU frequency depending on the running task. More recently, considering the implementation complexity of the underlying hardware platforms, researchers started to explore the implications of having a common voltage/frequency level shared by multiple cores (also known as the *voltage island* framework).

In addition, the multicore platforms have driven the creation of new programming models to exploit the full computational power provided by their architecture. Some authors considered those programming paradigms while defining the task model. To express code parellelism, some authors extended the classical sporadic task model by considering tasks described by directed acyclic graphs (DAGs).

### 7.1. Per-CPU DVFS Multicore Algorithms

This section discusses the algorithms computing a set of fixed frequencies, one for each core. All the algorithms consider task sets composed of periodic tasks with implicit deadlines. Table IX summarizes the main characteristics of the presented algorithms.

Table IX. DVFS Algorithm Summary for Multicore Platforms with Per-Core Frequencies

| Algorithm | Reference | Scheduler | P(s) | Speed Set | Switch Overhead | Complexity |
|---|---|---|---|---|---|---|
| Reservation | Aydin-Yang [2003] | part. EDF | $\beta s^3$ | cont. | 0 | $O(nm)$ |
| LA+LTF | Chen et al. [2006] | part. EDF | $\beta s^3+\gamma$ | cont. | 0 | $O(n)$ |
| LA+LTF+FF | Chen et al. [2006] | part. EDF | $\beta s^3+\gamma$ | cont. | $E_{sw}$ | $O(n)$ |
| AMBFF | Zeng et al. [2009] | part. EDF/FP | measured | disc. | measured | $O(nmk)$ |
| GMF | Moreno-DeNiz [2012] | global U-LLREF | $\beta s^3$ | disc. $\delta$_step | 0 | poly. |

In one of the earliest DVFS-based multiprocessor real-time scheduling papers, Aydin and Yang investigated the problem of partitioning periodic real-time tasks on a homogeneous multiprocessor platform with the eventual objective of applying DVFS on each processor separately [Aydin and Yang 2003]. They considered only dynamic power and ignored runtime overheads. One contribution of the paper is to show that in the settings where each processor can be fully utilized (e.g., through EDF scheduling algorithm), the most balanced partitioning is also the most energy-efficient one. Even though partitioning a set of real-time tasks is known to be intractable for multiprocessor systems, they showed that the problem of computing the most energy-efficient partitioning is NP-hard in the strong sense, even for trivially schedulable task systems with total utilization not exceeding 1. They also experimentally analyzed the behavior of the well-known partitioning heuristics Worst-Fit, Best-Fit, and First-Fit and concluded that the Worst-Fit heuristic generally outperforms the others.

Chen et al. [2006] presented some approximation algorithms to solve the Leakage-Aware Multiprocessor Energy-Efficient Scheduling (LAMS) problem that aims at minimizing the energy required to schedule a set $T$ of periodic tasks partitioned over $m$ identical processors having a continuous range of frequencies $[s_{min}, s_{max}]$ and requiring an energy overhead $E_{sw}$ to switch to the inactive mode. The first proposed algorithm is called LA+LFT and works under the hypothesis of negligible energy overhead to switch back and forth from the active mode ($E_{sw} = 0$). It sorts tasks in nonincreasing order and assigns each of them to a core using the Largest Task First (LTF) strategy. The speed of each core is set to the sum of the utilizations of the tasks assigned to it and every core is switched to the inactive mode as soon as it becomes idle. Then, a second algorithm is presented for nonnegligible switching overheads ($E_{sw} \neq 0$) that performs a second phase in which tasks are reassigned in order to avoid cores to execute at a speed lower than the critical one. The new algorithm, called LA+LFT+FF, produces a partitioning using fewer cores than the available ones, switching the unused cores to inactive for the whole hyperperiod. Finally, they proposed an extension for both algorithms that provides further energy reduction by procrastinating the insertion of an activated task to the ready queue in order to merge the idle intervals and decrease the number of state switches.

Zeng et al. [2009] proposed an algorithm called Adaptative Minimal Bound First-Fit (AMBFF) that is a polynomial-time heuristic that partitions a set of periodic tasks over a multicore platform and sets CPU speeds. The focus of the paper is the use of a more realistic platform model where each speed is selected from a finite set of available frequencies and the power consumption of each one is not computed using a mathematical model but extracted from a lookup table containing the results of a profiling phase on the real hardware platform. The algorithm works on the reduction of the static energy consumption assigning as many tasks as possible to a core with the First-Fit (FF) heuristic, while the reduction of the dynamic energy is pursued dynamically setting the bound for the heuristic to the values of the discrete speeds. The complexity of the algorithm is $O(nmk)$, where $n$ is the number of tasks, $m$ is the number of cores in the platform, and $k$ is the number of available frequencies.

Table X. DVFS Algorithm Summary for Multicore Platforms with Per-Task Frequencies

| Algorithm | Reference | Scheduler | P(s) | Speed Set | Complexity |
|---|---|---|---|---|---|
| GSSR | Zhu et al. [2003] | global NP | $\beta s^3$ | discrete | polynomial |
| FLSSR | Zhu et al. [2003] | global NP | $\beta s^3$ | discrete | polynomial |
| SPA2 | Lu and Guo [2011] | partitioned | $\beta s^3$ | continuous | polynomial |
| PHD | Lu and Guo [2011] | partitioned | $\beta s^3$ | continuous | polynomial |
| FFDH rigid | Xu et al. [2012] | partitioned | $(P_i, s_i)$ | discrete | $O(kn)$ |
| FFDH mold. | Xu et al. [2012] | partitioned | $(P_i, s_i)$ | discrete | $O(kmn^2)$ |
| DVFS-DPM | Chen et al. [2013] | time triggered | $P_{dyn}(s)+P_{sta}$ | discrete | MILP solving |

Unlike the other approaches previously presented in this section, the algorithm proposed by Moreno and De Niz [2012], namely, Growing Minimum Frequency (GMF), produces an optimal DVFS assignment. It considers periodic tasks that will be scheduled using the U-LLREF algorithm, which is an extension of the LLREF scheduling policy [Cho et al. 2006] allowing the task set to be executed on uniform multiprocessors with the cores running at different speeds. The approach considers uniform multiprocessors with a set of discrete frequencies evenly separated by a frequency step $\delta$. The overhead due to the speed change is avoided because the frequency is fixed offline and the static power consumption is considered negligible; thus, no DPM support is provided. The GMF algorithm starts by sorting the tasks in nonincreasing order of utilization and all the frequencies are set to the minimum one. The algorithm evaluates a set of $i$ tasks running on $i$ cores, where the values of $i$ start from 1 and grow up to the number of processors ($m$). The utilization of the $i$ tasks is compared with the sum of the $i$ frequencies and, in case it is greater, the slowest core is incremented with steps $\delta$ till the cores can accommodate the task set. If all the $i$ cores reach the maximum frequency before satisfying the condition, then the task set is infeasible; otherwise, $i$ is increased by one. When the value of $i$ reaches the number of cores $m$, the last round is executed considering all the $n$ tasks. This methodology allows maintaining the complexity of the algorithm polynomial.

### 7.2. Per-Task DVFS Multicore Algorithms

This section presents DVFS algorithms for uniform multicore platforms that exploit the flexibility of the power management infrastructure to assign a frequency computed offline for each task. Table X summarizes the main characteristics of the presented algorithms

Zhu et al. [2003] proposed one of the first approaches for multicore systems that considers a large number of characteristics of the platforms and the typical applications running on it. In particular, it proposes the support for a frame-based task set with a common implicit deadline among all tasks, and precedence constraints in tasks' execution. The task set is scheduled in a global fashion and tasks are executed nonpreemptively. Regarding the energy model, the paper considers a negligible static power and thus proposes algorithms addressing only the DVFS method, which is compatible with the weight of dynamic power with respect to the static one when the paper was written. Nevertheless, the authors added some remarks about when it is possible to put the system in sleep state, thus enabling the integration of a DPM mechanism. The proposed approaches are first presented for a continuous frequencies range and negligible switching overhead, and then extended to address a more realistic scenario of discrete speeds and nonnegligible frequency switch overhead. The first proposed algorithm is called Global Scheduling with Shared Slack Reclamation (GSSR) and is invoked every time a new frame starts or when a task ends its execution on a processor. It computes the minimum speed used to execute the selected task without missing the deadline for the current frame. This mechanism automatically includes in the

computation all the slack from tasks that have already terminated the job for the current frame, while maintaining a polynomial complexity. Then, the authors propose the First-order Scheduling with Shared Slack Reclamation (FLSSR), which extends GSSR by considering the precedence constraints between tasks of the same frame. This is obtained by taking into account tasks that are released but not yet in the ready queue due to precedence constraints and executing the algorithm also when a task is unblocked and enters the ready queue.

Lu and Guo [2011] proposed two DVFS algorithms to deal with energy management in multiprocessor platforms. The authors assume to deal with split tasks, that is, tasks consisting of subtasks that must be executed sequentially but that can be allocated to different cores. Such a task model is useful to overcome the performance limitation of partitioned scheduling approaches in the presence of tasks with a high utilization factor. Split tasks are periodic with implicit deadlines and are scheduled using Deadline Monotonic. Energy is saved by applying DVFS before or after partitioning split tasks. The resulting algorithms extend the approaches called SPA2 proposed by Guan et al. [2010] and PDMS_HPTS_DS (PHD) presented by Lakshmanan et al. [2009].

Xu et al. [2012] proposed an algorithm to deal with the problem of energy management on multicore platforms of parallel tasks. The authors address frame-based tasks with an implicit deadline. For each task, the level of parallelism can be fixed (rigid task) or decided at each job activation (moldable task). In terms of energy model, the paper considers processors with a set of discrete frequencies with a lookup table to store the power consumption related with each speed. For both types of task, an integer linear programming (ILP) formulation and an efficient heuristic are proposed to find a valid solution. In the case of rigid tasks, the heuristic has two steps: in the first step, tasks are allocated using an efficient level-packing algorithm (e.g., First Fit Decreasing Height, Best Fit Decreasing Height, Next Fit Decreasing Height), and then the proposed algorithm iterates through the available frequencies for all cores till it finds the set minimizing the total energy consumption. The complexity of the algorithm is $O(kn)$, where $n$ is the number of tasks and $k$ is the number of available frequencies. Dealing with moldable tasks requires computing the level of parallelism for each task, and this is done through a heuristic able to reduce the complexity from exponential to linear with respect to the number of tasks ($n$) and cores ($m$). For each possible solution, the algorithm for the rigid task is executed; thus, the total complexity of the approach is $O(kmn^2)$.

Chen et al. [2013] presented an approach based on mixed integer linear programming (MILP) to perform an optimization of DVFS and DPM at the same time. The approach manages groups of applications, each one composed by a set of tasks with precedence constraints described using a direct acyclic graph (DAG) characterized by a common implicit deadline. The considered energy model takes into account the different sources of power consumption, a set of discrete frequencies, and time/energy overhead. In particular, the relative dynamic power consumption is computed using the model proposed by Martin and Siewiorek [2001]. Regarding the static power consumption, the model takes into account the time $t_{sw}$ and the energy $E_{sw}$ to switch between active and sleep mode, also computing the break-even time $t_{BET}$ to discriminate when it is worth performing such a switch. The main contribution of the paper is the characterization of the idle intervals for the MILP formulation, thus optimizing both DVFS and DPM. To reduce the search space for the solver, the authors presented a technique called "Execution Windows Analysis," able to reduce the set of tasks composing an idle interval to those that are actually present in the interval under analysis. The algorithm produces as output the time-triggered schedule for the application together with the execution frequency for each task.

Table XI. DVFS Algorithm Summary for Multicore Platforms with Global Frequency

| Algorithm | Reference | Scheduler | P(s) | Speed Set | Complexity |
|---|---|---|---|---|---|
| CVFS* | Devadas and Aydin [2010] | partitioned EDF | $\beta s^3 + \gamma$ | cont. | $O(m)$ |
| SFA | Pagani and Chen [2014] | partitioned EDF | $\beta s^\alpha$ | cont. | $O(m)$ |
| *milp* | Gerards et al. [2014] | time triggered | $\beta_1 s^\alpha + \beta_2 s + \gamma$ | cont. | MILP solver |
| *milp* | Srinivasan and Chatha [2007] | partitioned | $(P_i, s_i)$ | disc. | MILP solver |
| LPPWU | Srinivasan and Chatha [2007] | partitioned | $\beta s^\alpha$ | cont. | polynomial |

## 8. MULTIPROCESSOR DVFS ALGORITHMS BASED ON VOLTAGE ISLANDS

The platform flexibility exploited by the algorithms presented in the previous section is not without cost. Herbert and Marculescu [2007] showed that the hardware complexity needed to provide independent DVFS to each core exceeds the advantages in terms of energy reduction in modern VLSI architectures. On the other hand, having a single frequency for all cores does not allow exploiting a significant part of the unused energy, as shown in Funaoka et al. [2008]. A tradeoff solution adopted in current multicore platforms is to use *voltage islands*, which are groups of cores sharing the same voltage and frequency. This section presents some energy management algorithms that use single-frequency and single-voltage islands. Table XI summarizes the main characteristics of the presented algorithms.

Devadas and Aydin [2010] presented an approach to reduce power consumption on a chip-multiprocessor (CMP) characterized by multiple sleep states and a single-frequency DVFS common to all cores. The proposed solution consists of an offline and an online phase. The first part computes the optimal number of processing cores and the allocation of tasks to cores. The runtime support dynamically recomputes the actual working frequency and determines which idle cores can be temporally put in a sleep state without jeopardizing timing constraints. The proposed algorithms deal with a set of periodic tasks with implicit deadlines and schedule them in a partitioned way using preemptive EDF on each core. Regarding the power model, the authors consider a cubic dynamic power consumption and the static one. To determine the number of active cores and the task-to-core allocation, the authors present three different algorithms based on the Worst-Fit Decreasing (WFD) allocation. Sequential-Search (SS) explicitly computes the energy cost obtained by WFD for each value of the number of cores, from the minimum feasible one up to $m$, and selects the one with the lowest energy. Instead, Greedy Load Balancing (GLB) and Threshold-based Load Balancing (TLB) execute SS only one time for $m$ cores and then reduce the number of active cores by shifting some tasks to other cores. The authors then present the Coordinated Voltage and Frequency Scaling (CVFS) algorithm that recomputes the working frequency at each scheduling point and core state transition setting it equal to the minimum among the actual utilizations of all active cores and the global critical frequency, which is the minimum frequency below which the benefits of DVFS are overwhelmed by the impact of static power consumption. An extension that exploits task early completions is also proposed, namely, CVFS*, and the complexity of both algorithms is shown to be $O(m)$.

Pagani and Chen [2014] proposed an algorithm called Single Frequency Approximation (SFA) that is executed after task partitioning and computes the minimum fixed frequency that guarantees the task set schedulability. The approach considers a set of periodic tasks with implicit deadlines that have been statically partitioned among the $m$ available cores and are scheduled using the Earliest Deadline First (EDF) algorithm on each core. The energy model considers both static and dynamic power consumption and can be integrated with most of the single-core DPM algorithms managing nonnegligible time and energy overhead to switch from active to sleep states. The algorithm

sets the working speed as the minimum among the utilizations of the $m$ groups of tasks partitioned to the $m$ cores. Tasks' early completions can be handled by the approach proposed by Devadas and Aydin [2010] for the CVFS* extension. The paper focuses on the analysis of the SFA approach in terms of approximation factor under different hypotheses with respect to the optimal energy consumption in the hyperperiod.

Gerards et al. [2014] presented a new approach to determine the optimal clock frequencies and the schedule to minimize energy consumption. Their algorithm considers frame-based tasks with a common implicit deadline and manages precedence constraints between tasks. The approach considers both static and dynamic power consumption with negligible overhead to switch from active to sleep state. The paper first analyzes the energy consumption as a function of the level of parallelism, then shows how to compute the optimal frequencies for a given schedule. Later on, the authors present the weighted makespan criterion, which is used to compute both the schedule and the frequencies at the same time.

Srinivasan and Chatha [2007] proposed an approach based on an MILP formulation to optimize energy consumption using DVFS, DPM, and loop unrolling. The authors presented three more variants of the initial formulation in order to reduce its complexity in exchange for a small worsening in the quality of the solution. The approach considers tasks with precedence constraints modeled as a direct acyclic graph (DAG). The energy model includes discrete frequencies and multiple sleep states with a state graph diagram modeling time and energy costs for every transition. The paper also proposes one heuristic to find an approximated solution in polynomial time using some deterministic approaches or a simulated annealing one. The proposed methods are tested using some multimedia application as a testbench.

## 9. RELATED PROBLEMS

This survey primarily focused on scheduling algorithms that target minimizing energy on uniprocessor and multiprocessor hard real-time systems. In this section, we briefly discuss some other problems with additional objectives and related research efforts.

An interesting problem is related to the joint scheduling of real-time and non-real-time tasks, where the goal is to minimize the overall energy consumption while guaranteeing real-time constraints and reducing the response time of non-real-time tasks. As in the case of real-time tasks, reducing response times of non-real-time tasks, in general, conflicts with the energy-saving objective. Aydin and Yang [2004] investigated the impact of speed scaling decisions on the responsiveness of non-real-time tasks and energy consumption while still meeting the timing constraints of hard real-time tasks. Saewong and Rajkumar [2008] proposed to exploit the available slack time to execute non-real-time tasks at the maximum speed to minimize their response time. Huang et al. [2014] formally formulated the minimization problem as a convex program, integrating DVFS with the EDF-VD scheduling technique, to show how the solution space can be reduced. Then, an optimal algorithm was provided.

Another problem investigated in the literature concerns energy management for soft real-time tasks. The problem has been addressed in the context of both single-core [Sharma et al. 2003; Wu et al. 2007] and multicore [Wang and Lu 2008; Chen et al. 2011] platforms.

The energy-aware scheduling of tasks that share resources accessed in nonpreemptive fashion is another important problem that was addressed in Lee et al. [2007] and Jejurikar and Gupta [2005b].

A more general energy-aware coscheduling problem includes both the CPU and devices in the analysis. Devices are typically considered speed independent, providing low-power states and requiring nonpreemptive access [Devadas and Aydin 2008; Yang et al. 2007]. Other authors considered the problem of coscheduling tasks and messages

[Yi et al. 2009; Marinoni et al. 2011]. The problem has also been addressed in the context of multicore systems [Gerards and Kuper 2013].

## 10. CLOSING REMARKS

This survey presented an overview of the state-of-the-art algorithms that addressed energy-aware scheduling in real-time systems on uniprocessor and multiprocessor platforms. Besides the relevance of each individual solution, the survey showed how the real-time support has evolved to address new capabilities and challenges due to the technological innovations.

The presented algorithms have been classified based on their primary energy management technique (DVFS or DPM) and the basic assumptions made on the system/power model. In general, DVFS algorithms are more effective on processors in which a significant energy reduction can be achieved at low clock frequencies, whereas DPM algorithms work better on systems in which the energy consumption is primarily reduced by putting the processor in low-power states as long as possible and then executing the workload at the maximum speed. Hybrid approaches that combine both techniques have also been discussed.

Considering that the effectiveness of the discussed algorithms depends on a large number of assumptions and characteristics of the hardware and software components, ranking the presented solutions based on a quantitative evaluation would be misleading, incomplete, and unfair. On the other hand, considering a number of different platforms, as already done in Saha and Ravindran [2012] and Bambagini et al. [2014], would be very limiting and not exhaustive, because some solutions are tailored on very specific hardware features. For such reasons, we opted for a more general assessment of the basic methodologies, providing a set of best practices that can serve as a guideline for the users in the selection of the most suitable approach.

Intuitively, DVFS algorithms should be considered whenever the power model can be described by a convex function of the speed or, more generally, whenever the energy consumed by executing a task at a lower speed for a longer time is less than executing it at a higher speed for a shorter time (possibly by exploiting the remaining idle time in a low-power state). In addition, processors with high static power consumption may favor algorithms that exploit low-power states instead of speed scaling, as speed scaling only affects the dynamic component of the consumed power.

Even on platforms whose power model follows a cubic function of the speed, the frequency granularity and the switching overhead may make aggressive DVFS algorithms less competitive. Many DVFS algorithms consider either continuous or coarse granularity speed ranges, and working with a smaller speed set may force the selection of a higher frequency, with the side effect of higher energy consumptions. Although actual processors can take advantage of coarse granularity speed ranges, the speed scaling overhead is still significant. More precisely, since the switching overhead is generally proportional to the difference between the actual and the new speed, aggressive algorithms that attempt to exploit the deepest speed as soon as possible may waste a significant part of the available slack time. To avoid this, designers should check the set of available frequency/voltage levels and verify that the available idle time is sufficiently higher than the scaling overhead.

Aggressive DVFS algorithms that reclaim most of the dynamic slack should be used only when WCETs are much longer than the average execution times, requiring the user to derive an execution profile of the application tasks. If the available slack is not expected to be abundant, the performance of DVFS algorithms becomes very close to that achievable by simply exploiting static slack. In addition, the runtime overhead of algorithms with high computational complexity may further exacerbate this issue.

Conversely, DPM algorithms may work poorly if the processor break-even time is generally longer than the available slack time. In this case, any DPM algorithm could exploit only shallow low-power states, whereas a simple DVFS algorithm that exploits only static slack (e.g., by setting the lowest feasible speed at the system start time) could be more effective. Since this kind of situation cannot by detected easily, verifying that the break-even time is no less than the static slack of the task with the shortest period may provide a safety guard.

When dealing with multicore systems, the selection between voltage-island-based DVFS and per-core DVFS depends on the features of the available hardware. Also, the choice of the specific energy-aware algorithm depends on the characteristics of the task set. Algorithms designed for independent periodic tasks cannot support parallel programming paradigms, where tasks are subject to precedence constraints. In such cases, energy-aware approaches providing support for DAG tasks can make a better exploitation of the platform features. Those algorithms that produce a time-triggered schedule based on tasks' arrival times can fit very well with periodic applications, such as classical control systems, but they are unsuitable for applications including sporadic tasks characterized by a high variability in the arrival rates.

## REFERENCES


Muhammad Ali Awan and Stefan M. Petters. 2011. Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In *Euromicro Conference on Real-Time Systems (ECRTS'11)*.

Hakan Aydin, Vinay Devadas, and Dakai Zhu. 2006. System-level energy management for periodic real-time tasks. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*.

Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. 2001. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the 13th IEEE Euromicro Conference on Real-Time Systems (ECRTS'01)*.

Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. 2004. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers* 53, 5 (May 2004), 584–600.

Hakan Aydin and Qi Yang. 2003. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*. IEEE, 9–pp.

Hakan Aydin and Qi Yang. 2004. Energy - Responsiveness tradeoffs for real-time systems with mixed workload. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*.

Mario Bambagini, Marko Bertogna, and Giorgio Buttazzo. 2014. On the effectiveness of energy-aware real-time scheduling algorithms on single-core platforms. In *Proceedings of the 19th Conference on Emerging Technologies and Factory Automation (ETFA'14)*.

Mario Bambagini, Marko Bertogna, Mauro Marinoni, and Giorgio C. Buttazzo. 2013. An energy-aware algorithm exploiting limited preemptive scheduling under fixed priorities. In *Proceedings of the 8th IEEE International Symposium on Industrial Embedded Systems (SIES'13)*.

Mario Bambagini, Francesco Prosperi, Mauro Marinoni, and Giorgio C. Buttazzo. 2011. Energy management for tiny real-time kernels. In *Proceedings of the IEEE International Conference on Energy Aware Computing (ICEAC'11)*.

Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *Transactions on Very Large Scale Integration Systems* 8, 3 (2000), 299–316.

Enrico Bini, Giorgio C. Buttazzo, and Giuseppe Lipari. 2009. Minimizing CPU energy in real-time systems with discrete speed management. *ACM Transactions on Embedded Computing Systems* 8, 4 (July 2009), 31:1–31:23.

Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. 2003. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*.

Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. 2013. Limited preemptive scheduling for real-time systems. A survey. *IEEE Transactions on Industrial Informatics* 9, 1 (2013), 3–15.

Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. 1995. Low power CMOS digital design. *IEEE Journal of Solid State Circuits* (1995), 473–484.

Gang Chen, Kai Huang, and Alois Knoll. 2013. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions on Embedded Computing Systems* 13, 3s (June 2013), 111:1–111:21.

Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. 2006. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*.

Jian-Jia Chen, Kai Huang, and Lothar Thiele. 2011. Power management schemes for heterogeneous clusters under quality of service requirements. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC'11)*. 546–553.

Jian-Jia Chen and Chin-Fu Kuo. 2007. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*.

Jian-Jia Chen and Tei-Wei Kuo. 2006. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. *SIGPLAN Notices* 41, 7 (June 2006).

Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. 2006. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*.

Robert Davis and Andy J. Welling. 1995. Dual priority scheduling. In *Proceedings of the 16th IEEE International Real-Time Systems Symposium (RTSS'05)*.

Vinay Devadas and Hakan Aydin. 2008. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *Proceedings of the 8th ACM International Conference on Embedded Software (EMSOFT'08)*.

Vinay Devadas and Hakan Aydin. 2010. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Proceedings of the International Green Computing Conference (GREENCOMP'10)*.

Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki. 2008. Energy-efficient optimal real-time scheduling on multiprocessors. In *Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC'08)*.

Marco E. T. Gerards, Johann L. Hurink, and Jan Kuper. 2014. On the interplay between global DVFS and scheduling tasks with precedence constraints. *IEEE Transactions on Computers* 64, 6 (2014), 1742–1754.

Marco E. T. Gerards and Jan Kuper. 2013. Optimal DPM and DVFS for frame-based real-time systems. *ACM Transactions on Architecture and Code Optimization* 9, 4 (January 2013), 41:1–41:23.

Min-Sik Gong, Yeong Rak Seong, and Cheol-Hoon Lee. 2007. On-line dynamic voltage scaling on processor with discrete frequency and voltage levels. In *Proceedings of the 2007 International Conference on Convergence Information Technology (ICCIT'07)*.

Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. 2010. Fixed-priority multiprocessor scheduling with Liu and layland's utilization bound. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*. 165–174.

Sebastian Herbert and Diana Marculescu. 2007. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'07)*.

Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzo. 2009a. Adaptive dynamic power management for hard real-time systems. In *Proceedings of the Real-Time Systems Symposium (RTSS'09)*.

Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzo. 2009b. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th IEEE International Conference on Decision and Control (CDC'09)*.

Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. 2014. Energy efficient DVFS scheduling for mixed-criticality systems. In *Proceedings of the 14th International Conference on Embedded Software (EMSOFT'14)*.

Sandy Irani, Sandeep Shukla, and Rajesh Gupta. 2007. Algorithms for power savings. *ACM Transactions on Algorithms* 3, 4 (Nov. 2007), 37–46.

Tohru Ishihara and Hiroto Yasuura. 1998. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'98)*.

Ravindra Jejurikar and Rajesh Gupta. 2004. Procrastination scheduling in fixed priority real-time systems. In *Conference on Languages, Compilers and Tools for Embedded Systems (LCTES'04)*.

Ravindra Jejurikar and Rajesh Gupta. 2005a. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the Conference on Design Automation Conference (DAC'05)*.

Ravindra Jejurikar and Rajesh Gupta. 2005b. Energy aware non-preemptive scheduling for hard real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*.

Ravindra Jejurikar, Cristiano Pereira, and Rajesh K. Gupta. 2004. Leakage aware dynamic voltage scaling for real time embedded systems. In *International Conference on Design Automation Conference (DAC'04)*.

Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztián Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. 2003. Leakage current: Moore's law meets static power. *Transactions on Computers* 36, 12 (Dec. 2003), 68–75.

Taewhan Kim. 2006. Application-driven low-power techniques using dynamic voltage scaling. In *Proceedings of the Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*.

Woonseok Kim, Jihong Kim, and Sang Lyul Min. 2004. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the Symposium on Low Power Electronics and Design (ISLPED'04)*.

Karthik Lakshmanan, Ragunathan Rajkumar, and John Lehoczky. 2009. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS'09)*. 239–248.

Martin Lawitzky, David C. Snowdon, and Stefan M. Petters. 2008. Integrating real-time and power management in a real system. In *Operating Systems Platforms for Embedded Real-Time Applications*.

Cheol-Hoon Lee and Kang G. Shin. 2004. On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS 04)*.

Jaewoo Lee, Kern Koh, and Chang-Gun Lee. 2007. Multi-speed DVS algorithms for periodic tasks with non-preemptible sections. In *Embedded and Real-Time Computing Systems and Applications*.

Yann-Hang Lee, Krishna P. Reddy, and C. Mani Krishna. 2003. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*.

C. L. Liu and James W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20, 1 (Jan. 1973), 46–61.

Junyang Lu and Yao Guo. 2011. Energy-aware fixed-priority multi-core scheduling for real-time systems. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'11)*.

Mauro Marinoni, Mario Bambagini, Francesco Prosperi, Francesco Esposito, Gianluca Franchino, Luca Santinelli, and Giorgio C. Buttazzo. 2011. Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems. In *Proceedings of the 16th IEEE International Conference on Emerging Technologies & Factory Automation*.

Thomas L. Martin and Daniel P. Siewiorek. 2001. Non-ideal battery and main memory effects on CPU speed-setting for low power. *IEEE Transactions on VLSI Systems* 9, 1 (2001), 29–34.

Sparsh Mittal. 2014. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology* (Jan. 2014), 47:1–47:31.

Bren Mochocki, Xiaobo Sharon Hu, and Gang Quan. 2007. Transition-overhead-aware voltage scheduling for fixed-priority real-time systems. *ACM Transactions on Design and Automated Electronics Systems* 12, 2 (April 2007).

Gabriel A. Moreno and Dionisio De Niz. 2012. An optimal real-time voltage and frequency scaling for uniform multiprocessors. In *Proceedings of the 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'12)*.

Siva G. Narendra and Anantha P. Chandrakasan. 2010. *Leakage in Nanometer CMOS Technologies*. Springer.

Linwei Niu and Gang Quan. 2004. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'04)*.

Santiago Pagani and Jian-Jia Chen. 2014. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems* 13, 5s (September 2014), 158:1–158:25.

Padmanabhan Pillai and Kang G. Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. *ACM SIGOPS Operating Systems Review* 35, 5 (October 2001).

Ala' Qadi, Steve Goddard, and Shane Farritor. 2003. A dynamic voltage scaling algorithm for sporadic tasks. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*.

Gang Quan and Xiaobo Hu. 2002. Minimum energy fixed-priority scheduling for variable voltage processor. In *Proceedings of the International Conference on Design, Automation and Test in Europe (DATE'02)*.

Gang Quan, Linwei Niu, Xiaobo Sharon Hu, and Bren Mochocki. 2004. Fixed priority scheduling for reducing overall energy on variable voltage processors. In *Real-Time Systems Symposium (RTSS'04)*.

Anthony Rowe, Karthik Lakshmanan, Haifeng Zhu, and Ragunathan Rajkumar. 2010. Rate-harmonized scheduling and its applicability to energy management. *IEEE Transactions on Industrial Informatics* 6, 3 (2010), 265–275.

Saowanee Saewong and Raj Rajkumar. 2008. Coexistence of real-time and interactive & batch tasks in DVS systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*.

Saowanee Saewong and Ragunathan (Raj) Rajkumar. 2003. Practical voltage-scaling for fixed-priority RT-systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*.

Sonal Saha and Binoy Ravindran. 2012. An experimental evaluation of real-time DVFS scheduling algorithms. In *Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR'12)*.

Kiran Seth, Aravindh Anantaraman, Frank Mueller, and Eric Rotenberg. 2003. FAST: Frequency-aware static timing analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS'03)*.

Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian Lu. 2003. Power-aware QoS management in web servers. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*. 63.

Dongkun Shin, Jihong Kim, and Seongsoo Lee. 2001. Intra-task voltage scheduling for low-energy, hard real-time applications. *IEEE Journal on Design & Test* 18, 2 (March 2001), 20–30.

Dimitrios Soudris, Christian Piguet, and Costas Goutis. 2002. *Designing CMOS Circuits for Low Power*. Springer.

Krishnan Srinivasan and Karam S. Chatha. 2007. Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints. *VLSI Journal Integration* 40, 3 (April 2007), 326–354.

Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. 2000. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'00)*, Vol. 4.

Leping Wang and Ying Lu. 2008. Efficient power management of heterogeneous soft real-time clusters. In *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSSC'08)*. 323–332.

Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. 2007. Utility accrual real-time scheduling under the unimodal arbitrary arrival model with energy bounds. *IEEE Transactions on Computers* 56, 10 (Oct. 2007), 1358–1371.

Huiting Xu, Fanxin Kong, and Qingxu Deng. 2012. Energy minimizing for parallel real-time tasks based on level-packing. In *Proceedings of the 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'12)*.

Ruibin Xu, Daniel Mossé, and Rami Melhem. 2005. Minimizing expected energy in real-time embedded systems. In *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT'05)*.

Ruibin Xu, Daniel Mossé, and Rami Melhem. 2007. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Transactions on Computer Systems* 25, 4 (Dec. 2007), 449–456.

Ruibin Xu, Chenhai Xi, Rami Melhem, and Daniel Moss. 2004. Practical PACE for embedded systems. In *Proceedings of the 4th ACM international Conference on Embedded Software (EMSOFT'04)*.

Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. 2007. Preemption control for energy-efficient task scheduling in systems with a DVS processor and Non-DVS devices. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*.

Frances Yao, Alan Demers, and Scott Shenker. 1995. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*.

Jun Yi, Christian Poellabauer, Xiaobo Sharon Hu, Jeff Simmer, and Liqiang Zhang. 2009. Energy-conscious co-scheduling of tasks and packets in wireless real-time environments. In *Proceedings of the 15th IEEE Symposium on Real-Time and Embedded Technology and Applications (RTAS'09)*.

Han-Saem Yun and Jihong Kim. 2003. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Transactions on Embedded Computing Systems* 2, 3 (Aug. 2003), 393–430.

Gang Zeng, Tetsuo Yokoyama, Hiroyuk. Tomiyama, and Hiroaki Takada. 2009. Practical energy-aware scheduling for real-time multiprocessor systems. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'09)*.

Ying Zhang and Krishnendu Chakrabarty. 2003. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'03)*.

Baoxian Zhao and Hakan Aydin. 2009. Minimizing expected energy consumption through optimal integration of DVS and DPM. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'09)*.

Dakai Zhu and Hakan Aydin. 2009. Reliability-aware energy management for periodic real-time tasks. *IEEE Transactions on Computing* 58, 10 (2009), 1382–1397.

Dakai Zhu, Rami Melhem, and Bruce R. Childers. 2003. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems* 4, 7 (2003), 686–700.

Dakai Zhu, R. Melhem, and D. Mosse. 2004. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'04)*.

Yifan Zhu and Frank Mueller. 2005. Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling. *Journal on Real-Time Systems* 31 (December 2005).

Received July 2014; revised April 2015; accepted July 2015