An Experimental Evaluation of Real-Time DVFS Scheduling Algorithms

Sonal Saha
ECE Dept., Virginia Tech, Blacksburg, VA, USA
sonal3@vt.edu

Binoy Ravindran
ECE Dept., Virginia Tech, Blacksburg, VA, USA
binoy@vt.edu

Abstract

We implement and experimentally evaluate the timeliness and energy consumption behaviors of fourteen state-of-the-art Real-Time Dynamic Voltage and Frequency Scaling (RT-DVFS) schedulers on two hardware platforms. The schedulers include CC-EDF, LA-EDF, REUA, DRA, and AGR1, among others, and the hardware platforms include the Intel i5 processor and the AMD Zacate processor. We implemented the schedulers in a real-time Linux kernel and measured their timeliness and energy consumption under a range of workloads including CPU-intensive, memory-intensive, mutual exclusion lock-intensive, and processor-underloaded and overloaded workloads. Our studies reveal that measuring the CPU power consumption as the cube of CPU frequency – as often done in the simulation-based RT-DVFS literature – ignores the idle state CPU power consumption, which is orders of magnitude smaller than the active power consumption. Consequently, power savings obtained by optimizing active power (i.e., RT-DVFS) is offset by completing tasks sooner by running at high frequency and quickly transitioning to the idle state (i.e., no DVFS). Thus, the active power consumption savings of the RT-DVFS techniques' revealed by our measurements are orders of magnitude smaller than their simulation-based savings reported in the literature.

Categories and Subject Descriptors C.3 [*Special-Purpose and Application-based Systems*]: Real-time and embedded systems; D.4.1 [*Operating Systems*]: Process Management–scheduling; D.4.7 [*Operating Systems*]: Organization and Design–real-time systems and embedded systems; D.4.8 [*Operating Systems*]: Performance–modeling and prediction

General Terms Design, Experimentation, Measurement

Keywords Real-time, Power, Energy, Dynamic voltage scaling, Power management, Operating systems

1. Introduction

A significant amount of research has been devoted to computer system power management at various levels of abstraction. For example, techniques such as clock gating [36] and low-power flip-flops [20] are hardware-level techniques that reduce active power during normal operation and reduce leakage power during sleep mode. Dynamic Voltage and Frequency Scaling (DVFS) [13, 37] and Dynamic Power management (DPM) [34, 27] are example techniques that optimize power consumption at the operating system-level. While DVFS involves adjusting the CPU voltage and frequency dynamically to reduce power consumption, DPM involves transitioning devices, including the CPU, to low-power/sleep states. Compiler-level power management techniques [16] optimize code to reduce execution time and memory accesses to save power. Application-level power management has also been studied. Examples include doing DVFS from the user space [26] by exchanging information between the user space and the OS, while maintaining soft real-time guarantees [42].

DVFS works on the following principle: In most of the CMOS-based modern processors, the maximum frequency of operation is dependent on the supply voltage and is given by [28]:

$$f = k \times \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{1}$$

Here, V_{dd} is the supply voltage, f is the clock frequency, V_t is the threshold voltage [28], and k is a constant. (1) can be rewritten as [8]:

$$f = a \times V_{dd} \tag{2}$$

where a is a constant. Thus, frequency has a linear relation with the supply voltage. When the CPU operates at a frequency f, its active power consumption, denoted P_{active}, is given by:

$$P_{active} = C_{ef} \times V_{dd}^2 \times f \tag{3}$$

Here, C_{ef} is the effective switch capacitance, V_{dd} is the supply voltage, and f is the clock frequency. By substituting (2) in (3):

$$P_{active} = \frac{C_{ef}}{a^2} \times f^3 \tag{4}$$

This, in turn, is equivalent to:

$$P_{active} = S_3 \times f^3, \tag{5}$$

where S_3 is a constant. Thus, the dynamic power consumption of a CMOS processor is directly proportional to the cube of CPU frequency [28].

Real-time DVFS (RT-DVFS) is a branch of DVFS, which involves reducing the CPU energy consumption by scaling the CPU frequency, while at the same time, ensuring that task time constraints are satisfied. Broadly, RT-DVFS techniques have two objectives: (i) Reduce energy consumption (through DVFS), and (ii) optimize task timeliness behavior through real-time resource management (i.e., real-time scheduling and synchronization). These two objectives may conflict, because reducing the frequency may increase task execution times, which is antagonistic to timeliness optimization. Most RT-DVFS scheduling algorithms consider satisfaction of time constraints as a "hard" constraint. They often utilize task slack times (i.e., time between task deadlines and worst-case

task execution times) to reduce the frequency (to the extent possible), and thus reduce energy consumption [29, 4, 42, 22].

A number of RT-DVFS algorithms [1, 19, 33, 6, 45, 10, 17, 44, 24, 25] have been developed in the past two decades. These algorithms have been extensively analyzed and their fundamental properties have been established – e.g., schedulable utilization bounds below which they meet all deadlines; conditions under which they satisfy energy budgets. These algorithms have also been experimentally studied using simulations [4, 41, 39, 43, 18], where the primary focus has been on understanding their normalized real-time and power consumption behaviors (e.g., normalized to no DVFS; normalized to highest frequency), and relative real-time/power trends. Only a very small fraction of these algorithms have been implemented and evaluated on real hardware platforms [29, 22, 42].

Simulation-based experimental studies have several advantages. First, it provides an effective way to evaluate the performance of an algorithm, especially to understand relative performance trends. Second, it is highly scalable and repeatable: a vast number of experimental settings can be constructed, and deterministically repeated, all programmatically. Additionally, it hides platform-specific issues through an abstract (often, discrete-event) simulation model, which significantly reduces development time.

However, simulation-based studies, especially those for RT-DVFS, have drawbacks. Unlike simulators used in the computer architecture community [38, 21, 30], there does not exist OS/platform simulators that have been rigorously evaluated and accepted by the (RT-DVFS) community as a whole. This has resulted in researchers developing their "home grown" simulators with many built-in assumptions (e.g., idle states of CPUs, number of idle states, transition overheads) that are not easy to verify against any particular hardware. Since the power savings of RT-DVFS techniques are highly platform- (and application-) dependent, this can potentially lead to incorrect conclusions. To illustrate this, we note that, many RT-DVFS works that have relied on simulation-based experimental studies have made the following assumptions:

(i) *A CPU has a continuous range of frequencies* [4, 41, 42]. However, actual CPUs do not have a continuous range, but discrete frequency steps. Some processors have a rich frequency set (e.g., 10), such as the Intel i5, and some have a smaller set (e.g., AMD Zacate has 3 steps; Via C7 has two steps). Thus, modeling a CPU with a continuous range of frequencies can give highly optimistic results, which may not hold on actual hardware.

(ii) *The CPU's idle state power consumption is negligible* [4, 41, 42]. These works only consider CPU's dynamic (or active) power consumption, which is assumed to be directly proportional to the cube of frequency, as illustrated by (5). In contrast, CPUs of most modern hardware have idle states (C states) and performance states (P states), and the CPU's power consumption is the summation of the power consumed in both these states [15]. Abstracting away this detail can lead to significantly optimistic, and sometimes, erroneous power savings, as our results show (Section 4).

(iii) *Non-CPU power consumption is insignificant.* In most of the RT-DVFS algorithms [4, 29, 42], only the CPU's power consumption is considered and the system level power consumption is ignored. The interaction between the CPU and other components (e.g., memory, bus, disk) is often ignored. This again can lead to erroneous power savings, as non-CPU devices' power consumption is *DVFS-independent*. Consequently, the overall power savings depend on the power profile of such devices and application workload characteristics – e.g., DVFS may prolong the CPU active state, which may potentially increase memory power consumption. In [3], Aydin *et. al.* have shown that below a particular speed, DVFS has a negative impact on the system-level energy consumption; in [35], Snowdon *et. al.* show that the optimal voltage and frequency setting is dependent on the system characteristics as well as the application.

1.1 Contributions

In this paper, we implemented and measured the timeliness and power consumption behavior of fourteen RT-DVFS schedulers. The schedulers target single processor systems, and include Static Earliest Deadline First (Static-EDF) [29], Cycle Conserving EDF (CC-EDF) [29], Look-Ahead EDF (LA-EDF) [29], a variant of [22], REUA [40], Dynamic Reclaiming Algorithm (DRA), and Aggressive Speed Reduction Algorithm (AGR) [4], among others (Section 2). Static-EDF utilizes static slack for determining the CPU frequency, whereas CC-EDF, LA-EDF, DRA, and AGR utilize dynamic slack as well. LA-EDF, AGR, and DRA are more aggressive, compared to the other algorithms, in the sense that, they try to reduce the frequency as much as possible, while still satisfying task time constraints.

We implemented the schedulers in a Linux-based real-time kernel called ChronOS [12], and measured their real-time/power behaviors on two hardware platforms, including an Intel i5 processor and an AMD Zacate processor (Section 3).

We used a synthetic application to generate a broad range of workload conditions including CPU-intensive, memory-intensive, mutual exclusion lock-intensive, and processor-underloaded and overloaded workloads. We measured the actual CPU power by accounting for the power consumption in the active and idle states, and also the system power using a multimeter. We also measured the normalized CPU energy consumption[1], where the CPU power is considered to be proportional to the cube of the frequency, so as to compare with the simulated results of the algorithms.

Our studies reveal that measuring the CPU power consumption as the cube of CPU frequency can lead to incorrect conclusions (Section 4). In particular, it ignores the idle state CPU power consumption, which is orders of magnitude smaller than the active power consumption. Consequently, power savings obtained by exclusively optimizing active power (i.e., RT-DVFS) is offset by completing tasks sooner by running them at the highest frequency and quickly transitioning to the idle state (i.e., no DVFS). Thus, the active power consumption savings of the RT-DVFS techniques' revealed by our measurements are orders of magnitude smaller than their simulation-based savings reported in the literature.

Our studies also revealed important inconsistencies between previously reported power savings (in simulation-based studies) and actual power savings. For example, algorithms such as Static-EDF, CC-EDF, and [22], which have been reported to outperform Base-EDF (on power savings) do not actually do so. These algorithms outperform Base-EDF on normalized CPU energy. However, they consume only slightly lesser CPU power and system power as Base-EDF, or in some cases, even more. Aggressive energy saving algorithms such as LA-EDF, DRA, DRA-OTE, AGR1, and AGR2 do consume less actual CPU power and system power than Base-EDF. But their energy savings are not as high as reported in past simulation-based studies.

In the past, a small subset of RT-DVFS algorithms have been implemented and evaluated on actual hardware platforms. Example such works include [29, 42, 13, 22] (Section 5). However, these works also have largely ignored the idle state CPU power consumption, or have measured the CPU power as proportional to the cube of the frequency, which can lead to inaccuracies. Moreover, they haven't implemented a comprehensive set of RT-DVFS schedulers and experimentally evaluated them on a broad range of workloads, like our work has done.

[1] The normalized CPU energy is the same as the normalized CPU power, as the time factor gets cancelled in both the numerator and the denominator.

2. Candidate RT-DVFS Algorithms

Before describing the candidate RT-DVFS algorithms, we briefly discuss the rationale behind their design.

Two types of slack time are exploited by RT-DVFS algorithms:

(i) *Static slack*. This is the idle time interval due to low CPU demand of the application. For a periodic real-time application, the total CPU demand is the ratio of the task period to the worst-case task execution time (WCET), aggregated for all tasks. When the total CPU demand is less than 100%, then there exists time intervals during which the CPU idles. This is called the static slack, which can be utilized to scale the CPU frequency.

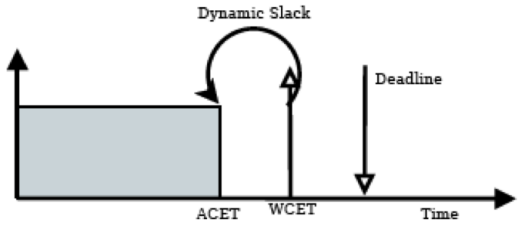

Figure 1. Dynamic Slack Example

(ii) *Dynamic slack*. This is the slack time that is available when the actual execution time of a task (ACET) is smaller than its (predicted) WCET, as shown in Figure 1. Dynamic slack time can only be obtained when the task completes—i.e., at run-time, in contrast to the static slack, which is known off-line, as task periods and WCETs are presumed to be known off-line for hard real-time applications. Once the dynamic slack becomes available, it can then be distributed to other eligible tasks by scaling their frequency, increasing their time budgets, and thus saving energy.

Most of the RT-DVFS algorithms differ in the way they estimate and utilize the static and dynamic slacks.

The candidate RT-DVFS algorithms that we selected for implementation can be classified based on their target task model as follows: (i) schedulers for independent underloaded task sets, (ii) schedulers for dependent underloaded task sets, and (iii) schedulers for overloaded task sets (both independent and dependent task sets). We briefly overview algorithms in each category. (Detailed descriptions are out of scope of this paper.)

2.1 Schedulers for Independent Underloaded Task Sets

2.1.1 Base-EDF

This is the basic EDF scheduler [14], which doesn't involve any frequency scaling and operates at the maximum frequency. The task with the earliest deadline is always dispatched for execution, and simply runs at the maximum frequency. We include this as a baseline case in the experimental study (similar to [29]).

Table 1. An Example Three Task Set

Task	WCET	ACET	Period
T_1	2	1.6	5
T_2	1	0.8	5
T_3	3	2.4	15

Figure 2 shows the Base-EDF schedule for the task set shown in Table 1.

2.1.2 Static-EDF

This scheduler, described in [29], uses the static slack estimation technique to scale the CPU frequency. The frequency is scaled based on the static utilization of the task set, i.e., $U = \sum_{i=1}^{n}(C_i/T_i)$, where C_i and T_i are the WCET and period of a task τ_i, respectively. (The task deadline is assumed to be equal to the task period.) All n tasks are run at the same frequency such that the utilization U of the processor at the scaled frequency becomes 1. We call this frequency, $S_{optimal}$. Static-EDF, thus ensures that no deadlines are missed, because $U \leq 1$, for which EDF guarantees no deadline misses [14].

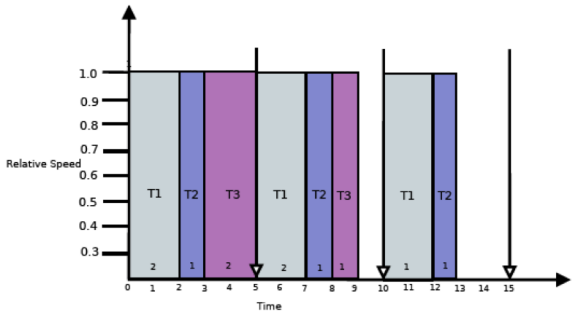

Figure 2. Base-EDF Schedule

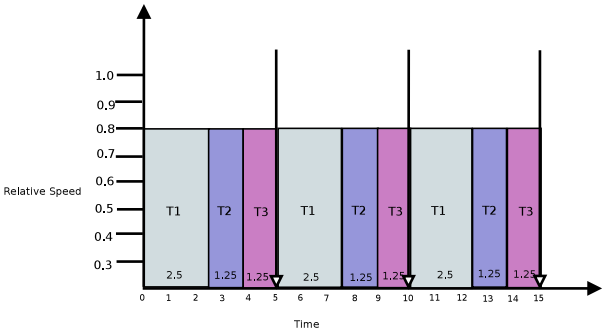

Figure 3. Static-EDF Schedule

Figure 3 shows the Static-EDF schedule for the task set shown in Table 1.

2.1.3 CC-EDF

Cycle conserving EDF (CC-EDF) [29] utilizes the dynamic slack to scale the CPU frequency. When a task is released, it is assumed that the task will execute up to its WCET, and the frequency is set accordingly. However, on completion, if the actual execution time is lesser, then the extra unused cycles are transferred to the remaining tasks. As the remaining tasks now get more cycles than they require to execute, the frequency is scaled down.

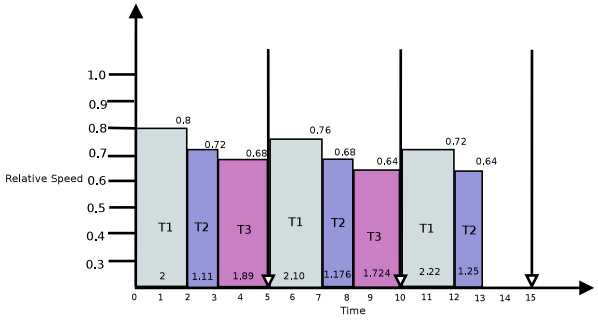

Figure 4. CC-EDF Schedule

Figure 4 shows the CC-EDF schedule for the task set shown in Table 1.

2.1.4 LA-EDF

Similar to CC-DF, Look-Ahead EDF (LA-EDF) [29] also reclaims dynamic slack to determine the task frequency. Here, the idea is to operate at the lowest possible frequency. At a scheduling event, the algorithm tries to do the minimum possible work before the next earliest deadline by "pushing" as much work as possible beyond that deadline, while making sure that all future deadlines are met, even if it has to run at a higher frequency in the future.

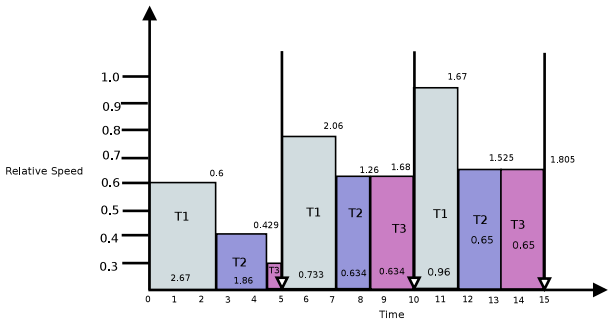

Figure 5. LA-EDF Schedule

Figure 5 shows the LA-EDF schedule for the task set shown in Table 1.

2.1.5 Snowdon-min

This is a subset of the RT-DVFS scheduler implemented by Lawitzky *et. al.* [22], which scales the frequencies of the CPU, the memory, and the bus. We do not consider scaling the memory frequency or the bus frequency (which is platform-specific), but only the CPU frequency, for a fair comparison. We call this subset, "Snowdon-min." In this algorithm, the amount of dynamic slack of a completed task is added to the budget of the next runnable task, and its frequency is scaled accordingly.

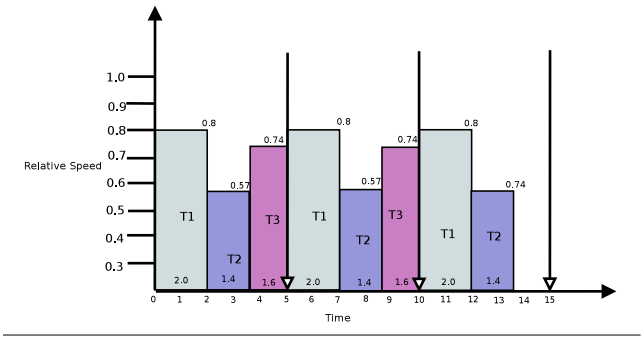

Figure 6. Snowdon-min Schedule

Figure 6 shows the Snowdon-min schedule for the task set shown in Table 1.

2.1.6 DRA

Dynamic Reclaiming Algorithm (DRA) is another dynamic slack reclaiming-based RT-DVFS algorithm [4]. In DRA, a data structure called an α queue is maintained. Whenever a task arrives, it pushes its WCET at the $S_{optimal}$ frequency in the α queue. Each element of the α queue is characterized by the deadline D_i of a task τ_i, and the remaining WCET of τ_i at the $S_{optimal}$ frequency, which is denoted as rem_i. The α queue is ordered according to EDF*. (EDF* is similar to EDF, except that, deadline ties are broken in favor of the task that arrived earlier.) With the progress of time, the rem_i field of the head of the α queue is subtracted from the elapsed time since the last scheduling event. If rem_i is smaller than the time elapsed, then after updating the rem_i of the head, the head is deleted. The update continues with the new head, till the entire elapsed time is exhausted. Whenever a task is selected for execution, the α queue is checked, and the rem_i field of all the α queue elements having a deadline lesser than or equal to τ_{best} is added to the rem_i of τ_{best}. (We will call the task with the earliest deadline as τ_{best} in all the remaining algorithms unless specified otherwise.) In this way, the unused slack time of all those tasks which completed earlier than the scheduled task is transferred to the scheduled task and the frequency is scaled accordingly.

2.1.7 DRA-OTE

The Dynamic Reclaiming Algorithm-One Task Technique (DRA-OTE) is an extension to the DRA algorithm [4]. This algorithm uses the concept of next task arrival time (or NTA). The next arrival time of any task instance in a system is known as NTA [19]. If there is only one task in the ready queue, and if its rem_i at $S_{optimal}$ is less than the time available till the NTA, then the frequency of execution of this task can be reduced to utilize the entire time till NTA. Thus, DRA-OTE exploits this extra slack to further decrease the frequency of execution.

2.1.8 AGR1 and AGR2

Agressive Speed Reduction 1 (AGR1) and Agressive Speed Reduction 2 (AGR2) [4] are also extensions of DRA. They are based on the idea that whenever there is more than one task in the ready queue and when all the tasks have to complete before the NTA, then the CPU time can be transferred among these tasks. For example, consider three tasks τ_1, τ_2, and τ_3 in the ready queue. Assume that all of them must complete before the NTA. If τ_1 has the earliest deadline, then it can obtain CPU time from τ_2 and τ_3, such that its speed can be reduced, while ensuring that all tasks complete before the NTA. This additional frequency scaling is done to the frequency obtained as a result of DRA-OTE.

In addition to dynamic slack reclaiming, AGR1 uses average workload information to predict the early completion of the future workloads and thus obtain extra slack to further reduce the frequency.

2.1.9 EUA

Energy-efficient Utility Accrual Algorithm (EUA) is an RT-DVFS algorithm that aims to maximize accrued timeliness utility per unit energy consumed, while reducing the total system power [40]. Unlike the previous works, the algorithm uses Martin's system-level energy consumption model for predicting task energy consumption toward determining frequency [28]. During underloads (i.e., $U \leq 1$), EUA accrues the maximum utility and scales the frequency in the same way as LA-EDF [29]. During overloads (i.e., $U > 1$), it accrues the maximum utility possible by running the tasks at the maximum speed. Schedule construction is done by examining tasks in the decreasing order of utility accrued per unit energy consumption. Tasks are tentatively included in a schedule, tested for schedule feasibility, and rejected if the schedule becomes infeasible.

2.2 Schedulers for Dependent Underloaded Task Sets

2.2.1 HS

High Speed (HS) is an RT-DVFS algorithm for task sets with non-preemptible blocking sections [43]. The algorithm determines a static speed, called high speed, to ensure that no deadlines are missed when the tasks are scheduled with EDF in the presence of non-preemptible blocking sections. Non-preemptible blocking

sections are modeled as a special case of the Stack Resource Policy (SRP) [5], in which there is one resource that is shared by all the tasks. When tasks are scheduled with EDF under SRP, then feasibility is given by $\forall k, 1 \leq k \leq n \sum_{k=1}^{n} C_i/D_i + B_k/D_k \leq 1$, where B_k is the maximum blocking time of task τ_i. Thus, a static speed H can be selected as follows (which ensures that all deadlines are met): $\forall k, 1 \leq k \leq n \sum_{k=1}^{n} C_i/D_i + B_k/D_k \leq H$.

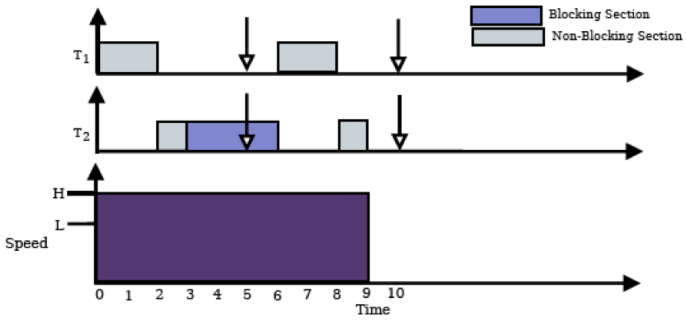

Figure 7. HS Schedule

Figure 7 shows the HS schedule for the task set in Table 1.

2.2.2 DS

In the HS algorithm, the processor can operate only at one constant speed. Dual Speed (DS) is an improvement over HS by operating at two speeds and thereby aiming to save more energy [43]. The two speeds include the static high speed calculated by HS (H), and the $S_{optimal}$ speed calculated by Static-EDF (L). The processor operates at the high speed only if a task is blocked by a lower priority task, for a duration which is till the completion time of the blocking task. At all other times, the processor operates at the low speed, thus saving energy, while making sure that no deadlines are missed.

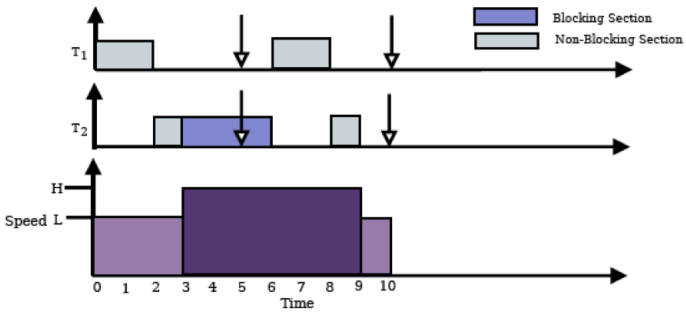

Figure 8. DS Schedule

Figure 8 shows the DS schedule for the task set in Table 1.

2.2.3 USFI_EDF

The algorithm, Uniform Slowdown with Frequency Inheritance-EDF (USFI-EDF) [18] allows any real-time resource access protocol (e.g., PIP [32], PCP [32], SRP [5]) for task synchronization. (Since we use EDF for scheduling in our experimental studies, we choose SRP, as it works with both fixed and dynamic priority scheduling.) The algorithm does frequency inheritance i.e., a low priority job is allowed to inherit the frequency of the highest priority job that it blocks to minimize deadline misses.

2.3 Schedulers for Overloaded Task Sets

2.3.1 REUA

REUA [41] is an extension of the EUA algorithm [40] for task models with dependencies. It extends EUA in the following way: a task's dependent tasks (i.e., dependencies arising due to transitive mutual exclusion) is first determined by following the chain of resource request and ownership. A task and its dependents are then considered together in determining the (aggregated) utility accrued per unit energy consumption for executing the task and its dependents. Potential deadlocks are detected by running a cycle detection algorithm and resolved by aborting the task with the least utility. Schedule construction is done as before: a task is examined (together with its dependents) in the decreasing order of the (aggregated) utility accrued per unit energy consumption, tentatively included in a schedule (together with the dependents), tested for schedule feasibility, and rejected (together with the dependents) if the schedule becomes infeasible. When there are no dependencies, REU defaults to EUA.

3. Experimental Environment

3.1 Platform

We implemented the fourteen candidate RT-DVFS algorithms in the ChronOS real-time Linux kernel [12]. Our evaluation was conducted on two hardware platforms. The first platform is an ASUS laptop with the Intel i5 processor and uses the Enhanced Intel Speedstep technology for scaling the processor voltage and frequency on-the-fly. The processor has 10 frequencies (see Table 2). The driver used is acpi_cpufreq. The second platform is an AMD Zacate mini-ITX Motherboard [2]. This processor operates at 3 frequencies, including 800 MHz, 1.28 GHz, and 1.6 GHz, and uses the powernow! technology for DVFS. The driver used is powernow_k8.

3.2 Test Application

To evaluate the schedulers, we extended and used the test application in [11, 12]. This application allows a configurable task set description, including task parameters such as WCET, period, deadline, ACET, and critical section length. The application creates each task as a thread. Each instance of a periodic task is automatically created to use ChronOS *scheduling segments* (i.e., code segments subject to time constraints) [12], with the configured time constraints. Scheduling segments trigger scheduling events in ChronOS, which invoke the (RT-DVFS) scheduler. Depending upon the workload chosen, a scheduling segment either burns the CPU for its ACET or WCET (in case of CPU-intensive workload), or does heavy memory accesses during its ACET or WCET (in case of memory intensive workload).

3.3 Timeliness Measurements

We measure task timeliness by measuring the deadline satisfaction ratio (DSR). DSR is defined as the ratio of the number of tasks that met their deadlines to the total number of task releases.

3.4 Power Measurements

The ACPI specifications [15] define the following idle and performance states of the CPU:

P state

This is the CPU's performance (or active) state. The number of P states supported by a CPU is CPU-specific. The processor operates at a different frequency/voltage pair in different P states, and thus consumes different amounts of power in each state. The lower the P state is, the higher the frequency the CPU operates in, and thus consumes more power. The P states, the

frequency of operation, and the power consumed in each P-state of the Intel i5 processor is given in Table 2.

C0 state
This is the CPU's operating state. The CPU is operating in one of the P-states when in this state. The power consumed in this state is dependent on the P-state in which it is operating in.

C1 state
This is the first idle state. In this state, only the CPU's main internal clocks are halted (via software). In Intel i5, the CPU consumes 1000mW in this state.

C2 state
In this state, only the CPU's main internal clocks are halted (via hardware) and the CPU takes longer time to wake up from this state. For Intel i5, the power consumed in this state is 500mW.

C3 state
In this state, most parts of the processor is stopped, such as caches. As a result, the processor is no longer cache-coherent in this state, and takes longer time to wake up from this state. The power consumed in this state for Intel i5 is 300mW.

Table 2. P-states in Intel i5 processor

P-state	Frequency (MHz)	Power (watts)
P0	2400	25000
P1	2399	25000
P2	2266	23316
P3	2133	21689
P4	1999	20116
P5	1866	18531
P6	1733	17021
P7	1599	15517
P8	1466	14068
P9	1333	12640
P9	1199	11250

System power measurement. We measured the total system power using the Fluke 289 RMS multimeter [9], which is a high resolution multimeter (resolution is 0.5 mA). The multimeter has an averaging mechanism, wherein it can obtain the average of the current measurements over an interval of time (while it is measuring). We made a slit in the cord of the power supply, and attached the multimeter in series, to measure the current. We obtained the rms value of the average current and multiplied it with the rms voltage, which is 120 V, to obtain the average power consumed in that time interval.

Normalized and actual CPU power measurement. We used the CPUpower tool [23] to obtain the normalized and the actual CPU power measurements. When an user application is fed as input, CPUpower tool gives information about the average frequency in the active state as well as the percentage of time spent in the respective performance (P states) and idle (C states) states for the duration of the application execution.

Normalized CPU energy consumption. We calculate the normalized CPU energy as $(f_{avg}/f_{max})^3$, where f_{avg} is the average frequency for the duration of the application execution. f_{avg} is obtained from the Freq field of the CPUpower tool's output.

Actual CPU power. We calculate the actual CPU power as the sum of the power consumed in the idle state and the active state. Thus, for Intel i5, actual CPU power = active power + C1 power + C2 power + C3 power, where:

- *active power*= (% of time spent in C0) × (power consumed in the corresponding P-state)
- *C1 power*= (% of time spent in C1) × 1000mW
- *C2 power*= (% of time spent in C2) × 500mW
- *C3 power*= (% of time spent in C3) × 300mW

4. Experimental Results

4.1 Timeliness

Figures 9 and 10 show the DSR of the schedulers, on Intel i5, for a 5 task set with deadlines and periods in the range $[500ms, 5000ms]$ and per-task utilization in the range $[0.1, 0.4]$. The figures show the DSR at increasing CPU utilization for ACET= 0.4WCET and ACET= 0.7WCET, respectively. We observe that all schedulers meet all deadlines during underloads. During overloads, we observe that aggressive algorithms such as AGR1 and AGR2 miss deadlines at a faster rate than others. REUA performs quite well during overloads; for the ACET= 0.7WCET case, it meets the most number of deadlines.

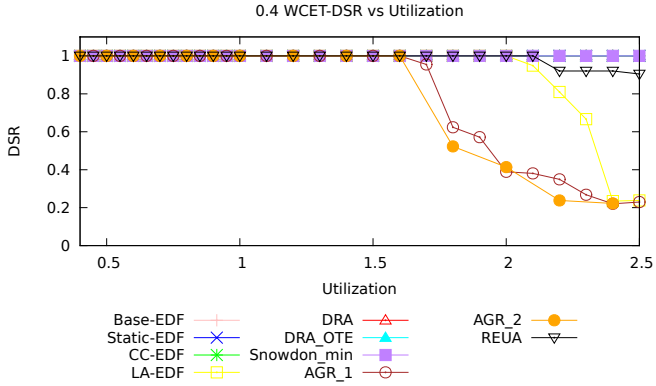

Figure 9. DSR vs. CPU Utilization for a 5 task set, at 0.4 WCET, on Intel i5

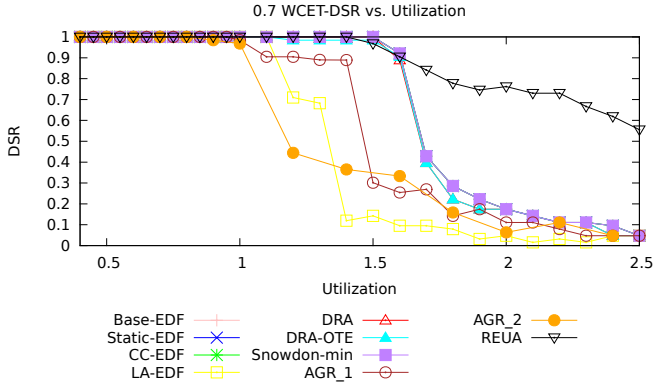

Figure 10. DSR vs. CPU Utilization for a 5 task set, at 0.7 WCET, on Intel i5

DSR results for other ACET cases are omitted here for brevity, and can be found in [31]. They confirm the trend that aggressive algorithms such as AGR1 and AGR2 miss deadlines at a faster rate than others during overloads, when ACET becomes closer to WCET. Moreover, REUA continues to perform well during overloads.

4.2 Energy Consumption vs. CPU Utilization on Intel i5

Figures 11 and 12 show the normalized CPU energy and the actual CPU power, respectively, on the Intel i5, for the same task set as that in Section 4.1. The figures show the the normalized CPU

energy and the actual CPU power, as the CPU utilization is varied from 50% to 250% at ACET = 0.7WCET. We conduct this experiment, because it helps us understand the energy consumption of the schedulers when the offered load increases while the slack remains the same.

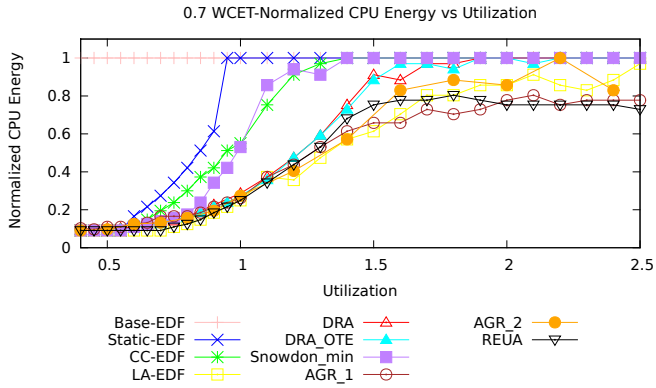

Figure 11. Normalized CPU Energy vs. CPU utilization for a 5 task set, at ACET = 0.7WCET, on Intel i5

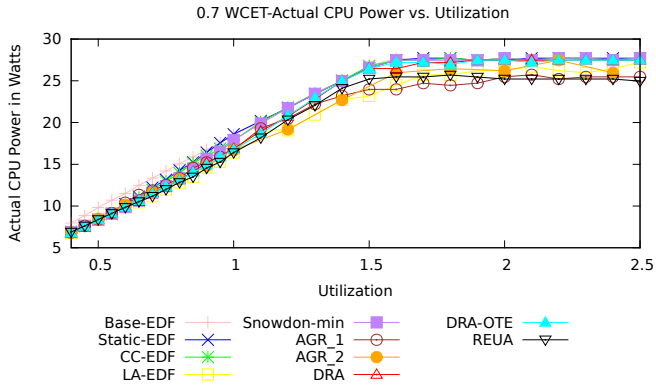

Figure 12. Actual CPU Power vs. CPU utilization for a 5 task set, at ACET = 0.7WCET, on Intel i5

We observe that the normalized CPU energy consumption is different from the actual CPU power consumption. The relative energy savings of the schedulers (relative to Base-EDF) observed from the normalized CPU energy savings plot, is much higher than that observed from the actual CPU power consumption plot. Even though the normalized CPU energy consumption of the schedulers is consistent with their theoretical results, their actual CPU power consumption shows significantly less energy savings (compared to Base-EDF).

This is because, the total CPU power consumption depends on the power consumed in the active and the idle state of the CPU. For Base-EDF, the CPU runs at a higher frequency for a shorter duration, whereas for algorithms such as LA-EDF, the CPU runs at a lower frequency, but for a shorter duration. If the energy efficiency of the CPU's idle states is high, then the power saved by running at a lower frequency for a longer duration is offset by running at a higher frequency for a shorter duration, and transitioning to the idle state sooner.

The idle state power consumption of Intel i5 is in the range of 0.3W to 1W, while the active state power consumption is in the range of 11.25W to 25W. Due to the high energy efficiency of the idle states, we therefore do not obtain significant energy savings from the RT-DVFS algorithms.

Results for other ACET cases (0.2 WCET, 0.4 WCET, etc.) and task set cases are omitted here for brevity; they show consistent trends, and can be found in [31].

4.3 Energy Consumption vs. ACET on Intel i5

Figures 13 and 14 show the normalized CPU energy and the actual CPU power, respectively, on Intel i5, when the ACET is varied from 0.1WCET to 1.0WCET, for a fixed CPU utilization of 70%. The task set is the same as that in Section 4.1. We conduct this experiment, because it helps us understand the energy consumption when the slack varies and thus the impact of RT-DVFS (recall that RT-DVFS techniques exploit slack to save energy).

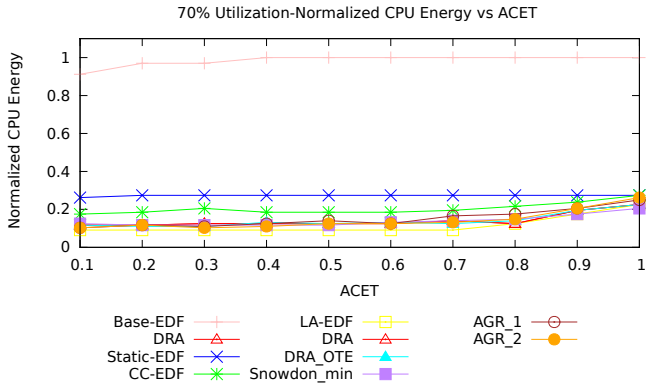

Figure 13. Normalized CPU Energy vs. ACET for a 5 task set, at 70% CPU Utilization, on Intel i5

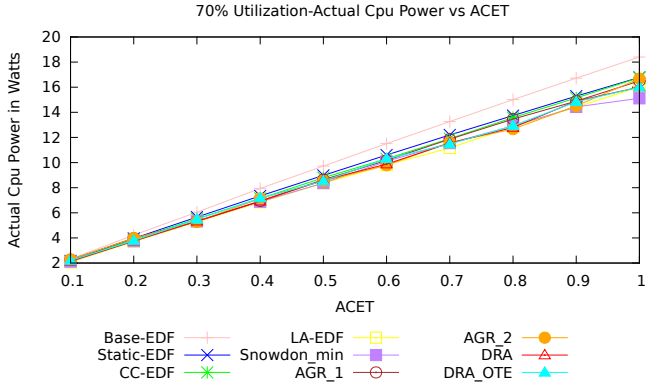

Figure 14. Actual CPU Power vs. ACET for a 5 task set, at 70% CPU Utilization, on Intel i5

We observe that, as the ACET increases, the actual power consumption increases much steeply than in the case of the normalized CPU energy consumption. This is because, with the increase in ACET, the dynamic power consumption increases due to the increase in frequency, as well as due to the increase in the time spent in the active state. The normalized CPU energy plots capture the dynamic power increase due to the increase in frequency, but not due to the increase in time. Both these parameters are captured by the actual CPU power consumption plots.

Results for other CPU utilizations and task set cases show consistent trends, and can be found in [31].

4.4 Energy Consumption on AMD Zacate

Figure 15 shows the normalized CPU energy consumption on the AMD Zacate, as the CPU utilization is varied from 50% to 250% at ACET = 0.7WCET (same case as in Section 4.2). Note that, the curves in this figure are not as smooth as that for the Intel i5 in Figure 11. This is due to the lesser number of frequency steps (i.e., 3) of AMD Zacate (Intel i5 has 10). The relative normalized CPU energy savings of the schedulers (relative to Base-EDF) on AMD is much lower than that on Intel i5, as can be observed from Figures 11 and 15 (due to the same reason).

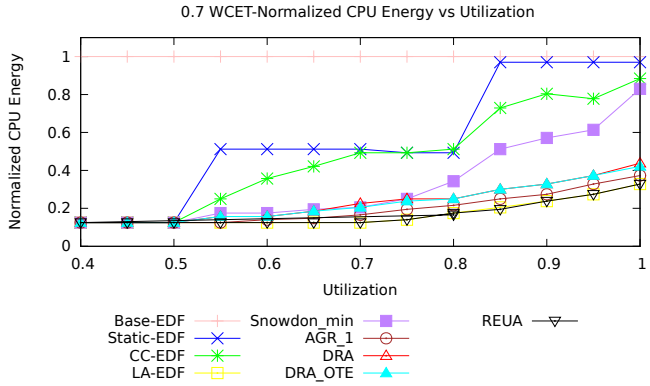

Figure 15. Normalized CPU Energy vs. CPU utilization for a 5 task set, at ACET = 0.7WCET, on AMD Zacate

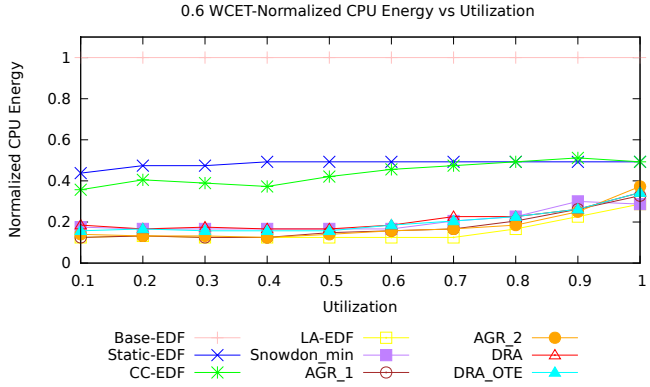

Figure 16. Normalized CPU Energy vs. ACET for a 5 task set, at 70% CPU Utilization, on AMD Zacate

Figure 16 shows the normalized CPU energy consumption when the ACET is varied from 0.1WCET to 1.0WCET, for a fixed CPU utilization of 70% (same case as in Section 4.3). The trends are similar to that of the Intel i5 case (similar reasons as in Section 4.3).

4.5 Energy Consumption of Dependent Task Sets

Recall that the schedulers HS, DS, USFI-EDF, and REUA allow dependent tasks. We now measure their energy consumption. We use a 10 task set with deadlines and periods in the range $[400ms 20000ms]$ and per-task utilization in the range $[0.010.4]$. The tasks share a single lock-guarded shared resource. A task critical section is equal to 20% of its WCET.

Figures 17 and 18 show the normalized CPU energy consumption and the actual CPU power consumption, respectively. Similar

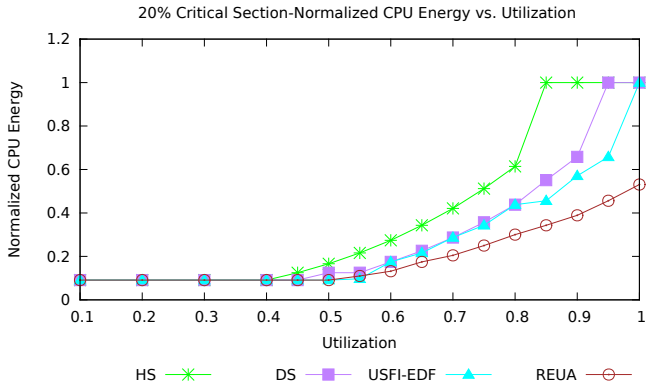

Figure 17. Normalized CPU Energy vs. CPU utilization for a 10 task set, with critical section equal to 20% of WCET

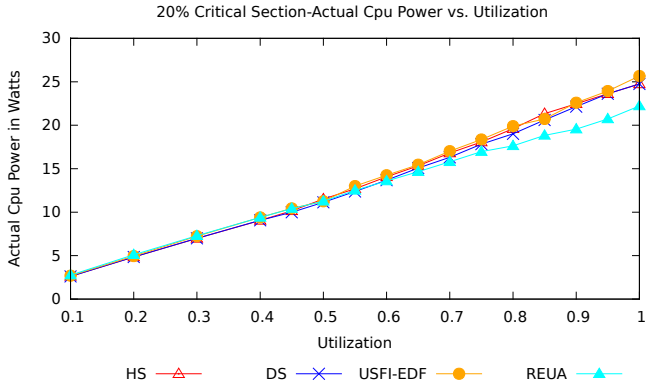

Figure 18. Actual CPU Power vs. CPU utilization for a 10 task set, with critical section equal to 20% of WCET

to the schedulers for independent task sets, the normalized CPU energy consumption and the actual CPU power consumption is different for these schedulers as well. The normalized energy consumption of DS, USFI_EDF, and REUA is very low when compared to that of HS. REUA performs the best. However, from Figure 18, we observe that the actual CPU power consumption of these schedulers is similar.

Results for other critical section lengths, number of locks, and task sets show consistent trends, and can be found in [31].

4.6 System Power on AMD Zacate

We measured the total system power for the 5 task set, for ACET=0.6 WCET, on AMD Zacate. Figures 19 and 20 show the system power for CPU intensive and memory-intensive workloads, respectively. We observe that Base-EDF consumes the maximum power and LA-EDF, the least. The power consumption of the other schedulers lie between these two extremes. Note that the difference between the scheduler power consumption is not significant. This is because of the high energy efficiency of AMD Zacate's idle states.

We also observe that the power consumed for the memory-intensive workloads is similar to that for the CPU-intensive case. This illustrates that, memory power consumption is DVFS-independent.

Results for other ACET cases, fixed-CPU utilization cases, and task sets show consistent trends, and can be found in [31].

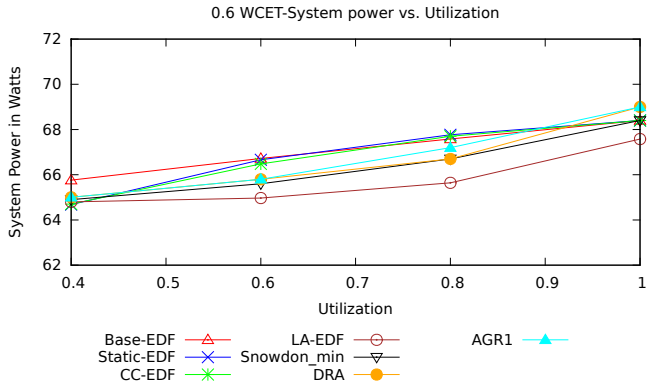

Figure 19. Total System Power vs. CPU Utilization for a 5 task set (CPU intensive workload), ACET = 0.6WCET, on AMD Zacate

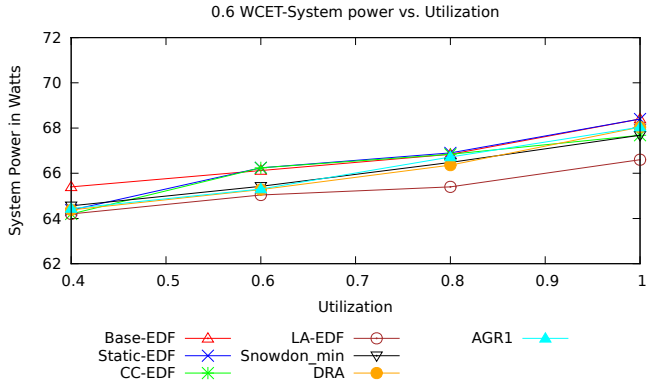

Figure 20. Total System Power vs. CPU Utilization for a 5 task set (memory intensive workload), ACET = 0.6WCET, on AMD Zacate

5. Related Work

Even though RT-DVFS has been a highly active of research [1, 19, 33, 6, 45, 10, 17, 44, 24, 25], very few of the algorithms have been implemented and evaluated on actual hardware platforms.

One of the earliest such efforts is by Pillai and Shin [29]. They devised five RT-DVFS algorithms, which reduce the energy consumption while ensuring 100% deadline satisfaction ratio, implemented them in the Linux 2.2.16 kernel, and measured real-time/power behaviors on an AMD K6-2+ processor with power-now! technology. They found that EDF-based schedulers outperform RMA based ones. (Among the EDF based ones, Static-EDF utilizes the static slack, whereas CC-EDF and LA-EDF reclaim the dynamic slack to scale the CPU frequency.) Additionally, they concluded that the energy savings of RT-DVFS algorithms are highly dependent on the voltage and frequency settings available on the hardware, and not much on the number of tasks or on the energy efficiency of the CPU's idle states. In contrast, our results show that the savings are, actually, highly dependent on the energy efficiency of the idle states. [29] is also limited to processor underloads (i.e., the offered task load is less than 100% of processor capacity), independent tasks (i.e., no synchronization), and CPU-intensive (no memory-intensive) workloads. In contrast, our workloads cover all these cases.

Yuan *et. al.* [42] designed and implemented a statistical DVFS scheduler in an OS called the Grace-OS. Similar to other simulation-based studies of DVFS algorithms, they also assume that CPU power is proportional to the cube of the frequency, and measure actual CPU energy accordingly. However, as mentioned before, this may lead to inaccuracies, as the actual CPU power consumption differs and depends on the energy consumed in the CPU's active and idle states.

Grunwald *et. al.* [13] implemented DVFS algorithms developed by Weiser *et. al.* [37] in the Linux kernel, running on the Itsy pocket computer. They measured the system power consumption and concluded that the algorithms did not give significant energy savings. The algorithms studied were not real-time.

Snowdon *et. al.* [22], designed and implemented a DVFS algorithm, which aims to reduce system-level power. They develop a time/energy model, and use it to calculate the time and the energy required by an application at a particular CPU, memory, and bus frequency. They integrate this model with the RBED real-time scheduler [7], which then uses the time-energy model to select a feasible set of frequencies that minimizes the total energy consumption while satisfying hard real-time constraints. They implement this scheduler in the OKL4 microkernel on the Gumstix platform with the XScale PXA255 processor, and measure the system power consumption. They conclude that, the good performance of a microkernel is dependent on the small overhead of context switches, and a DVFS scheduler introduces large overheads due to the scheduler complexity. The work does not report energy measurements or real-time measurements.

6. Conclusions

Our work reveals that the actual power savings of RT-DVFS scheduling algorithms (that we studied) are orders of magnitude smaller than their (largely simulation-based) savings previously reported in the literature. To illustrate this, Tables 3 and 4 show the normalized CPU energy consumption and the actual CPU power consumption of the schedulers (relative to Base-EDF), respectively, at 70% CPU utilization for the cases when the ACET is 0.3, 0.6, and 0.9 of the WCET. We observe that algorithms such as Static-EDF, CC-EDF, and Snowdon-min reduce normalized CPU energy consumption by 65% to 80% over Base-EDF, for the 0.6 WCET case, for example. However, we observe (in Table 4) that their actual energy savings is only in the 10%–12% range. These savings are smaller than the savings previously reported for these algorithms [29, 4, 22, 40, 41].

Aggressive energy saving algorithms such as LA-EDF, DRA, DRA-OTE, AGR1, and AGR2 do perform better: they reduce normalized energy consumption by 84%–87% over Base-EDF (for the 0.6 WCET case). However, their actual energy savings is only ≈15%, which is again smaller than the previously reported savings [29, 4, 22, 40, 41].

Table 3. Normalized CPU Energy Consumption

Scheduler	0.3 WCET	0.6 WCET	0.9 WCET
Static-EDF	0.373	0.373	0.373
CC-EDF	0.250	0.250	0.328
LA-EDF	0.125	0.125	0.238
Snowdon-min	0.157	0.185	0.238
DRA	0.166	0.166	0.262
DRA-OTE	0.148	0.166	0.262
AGR1	0.140	0.166	0.262
AGR2	0.140	0.166	0.262

This discrepancy can be explained as follows: the CPU's energy consumption is given by $E_{cpu} = P_{idle} * t_{idle} + P_{active} * t_{active}$, where P_{idle} and P_{active} are the idle state and the active state power consumption of the CPU, respectively, and t_{idle} and t_{active}

Table 4. Actual CPU Power Consumption

Scheduler	0.3 WCET	0.6 WCET	0.9 WCET
Static-EDF	0.935	0.92	0.914
CC-EDF	0.912	0.89	0.906
LA-EDF	0.872	0.852	0.862
Snowdon-min	0.879	0.877	0.864
DRA	0.872	0.85	0.89
DRA-OTE	0.873	0.853	0.89
AGR1	0.87	0.854	0.863
AGR2	0.87	0.854	0.863

are the time spent in the idle and active states, respectively. If $P_{idle} \ll P_{active}$ then it makes more sense to keep t_{idle} high and t_{active} low. However RT-DVFS algorithms (almost always) minimize the idle time. In modern processors, $P_{idle} \ll P_{active}$ e.g., Intel i5's power consumed in the active state ranges from 11.25W to 25W, whereas the energy consumed in the idle state ranges from 0.3W to 1W. Thus, by reducing the CPU frequency and increasing t_{active}, the energy savings obtained by the reduction of active power consumption can be offset by completing tasks sooner, by running them at the highest frequency and transitioning to the idle state earlier, so that t_{active} decreases.

Therefore, the energy efficiency of RT-DVFS algorithms is highly dependent on the relative power consumption of the active and the idle states, which in turn, is dependent on the characteristics of the platform used. In both the platforms used by us, the energy efficiency of idle states is very high and consequently, the power savings obtained, was not high. This might be the case with most modern processors.

We also observe that the performance of an RT-DVFS algorithm is highly dependent on the number of frequency steps available on the processor.

Our RT-DVFS scheduler implementations (in the ChronOS real-time Linux kernel) are publicly available at `chronoslinux.org`.

References

[1] T. A. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. In *ECRTS*, pages 165–174, 2004.

[2] AMD. AMD fusion mini-itx motherboard, 2011. `http://www.gigabyte.com/press-center/news-page.aspx?nid=963`.

[3] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *RTSS*, pages 313–322, 2006.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, 53:584–600, May 2004.

[5] T. P. Baker. A stack-based resource allocation policy for realtime processes. In *RTSS*, pages 191–200, 1990.

[6] E. Bini, G. Buttazzo, and G. Lipari. Minimizing cpu energy in real-time systems with discrete speed management. *ACM Trans. Embed. Comput. Syst.*, 8:31:1–31:23, July 2009.

[7] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. In *RTSS*, pages 396–, 2003.

[8] A. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. In *IEEE Journal of Solid-State Circuits*, volume 27, pages 473–484, April 1992.

[9] F. Corporation. Fluke 289 true-rms industrial logging multimeter. Available: `http://www.fluke.com/fluke/usen/digital-multimeters/fluke-289.htm?PID=56061`, Last accessed 2012.

[10] F. Dabiri, A. Vahdatpour, M. Potkonjak, and M. Sarrafzadeh. Energy minimization for real-time systems with non-convex and discrete operation modes. In *DATE*, pages 1416–1421, 2009.

[11] M. Dellinger. An experimental evaluation of the scalability of real-time scheduling algorithms on large-scale multicore platforms. Master's thesis, Virginia Tech, Blacksburg, VA, USA, 2011. Available `http://scholar.lib.vt.edu/theses/available/etd-05122011-142219/`.

[12] M. Dellinger, P. Garyali, and B. Ravindran. ChronOS Linux: a best-effort real-time multiprocessor Linux kernel. In *DAC*, pages 474–479, 2011.

[13] D. Grunwald, C. B. Morrey, III, P. Levis, M. Neufeld, and K. I. Farkas. Policies for dynamic clock scheduling. In *OSDI*, pages 6–6, 2000.

[14] W. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.

[15] HP, Intel, et al. Advanced configuration and power interface specification, 2011. `http://www.acpi.info/spec.htm`.

[16] C.-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *PLDI*, pages 38–48, 2003.

[17] C.-M. Hung, J.-J. Chen, and T.-W. Kuo. Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element. In *RTSS*, pages 303–312, 2006.

[18] R. Jejurikar and R. Gupta. Energy-aware task scheduling with task synchronization for embedded real time systems. In *CASES*, pages 164–169, 2002.

[19] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *RTAS*, pages 219–, 2002.

[20] U. Ko and P. T. Balsara. High-performance energy-efficient d-flip-flop circuits. *IEEE Trans. Very Large Scale Integr. Syst.*, 8:94–98, February 2000.

[21] J. Larus. SPIM: A mips32 simulator, 1990. `http://pages.cs.wisc.edu/~larus/spim.html`.

[22] M. P. Lawitzky, D. C. Snowdon, and S. M. Petters. Integrating real-time and power management in a real system. In *Workshop on Operating System Platforms for Embedded Real-Time Applications*, Prague, Czech Republic, 2008.

[23] Linux Kernel Mailing List Post. Cpupowerutils. `http://lwn.net/Articles/433002/`, 2011.

[24] S. Liu, Q. Qiu, and Q. Wu. Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In *DATE*, pages 236–241, 2008.

[25] S. Liu, Q. Wu, and Q. Qiu. An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems. In *DAC*, pages 782–787, 2009.

[26] X. Liu, P. Shenoy, and M. D. Corner. Chameleon: Application-level power management. *IEEE Transactions on Mobile Computing*, 7:995–1010, August 2008.

[27] Y.-H. Lu, L. Benini, and G. De Micheli. Power-aware operating systems for interactive systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 10:119–134, April 2002.

[28] T. L. Martin. *Balancing batteries, power, and performance: system issues in CPU speed-setting for mobile computing*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[29] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, pages 89–102, 2001.

[30] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[31] S. Saha. An experimental evaluation of real-time DVFS scheduling algorithms. Master's thesis, Virginia Tech, Blacksburg, VA, USA, 2011. `http://scholar.lib.vt.edu/theses/available/`

etd-09122011-125316/unrestricted/Saha_S_T_2011.pdf.

[32] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39:1175–1185, September 1990.

[33] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *DAC*, pages 438–443, 2001.

[34] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *MobiCom*, pages 11–19, 2000.

[35] D. C. Snowdon, S. Ruocco, and G. Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Workshop on Power Aware Real-time Computing*, Sep 2005.

[36] A. G. M. Strollo, E. Napoli, and D. De Caro. New clock-gating techniques for low-power flip-flops. In *ISLPED*, pages 114–119, 2000.

[37] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, 1994.

[38] Wind River. Wind River Simics, 2012. http://www.windriver.com/products/simics/.

[39] H. Wu, B. Ravindran, and E. D. Jensen. On bounding energy consumption in dynamic, embedded real-time systems. In *ACM SAC*, pages 933–934, 2006.

[40] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. CPU scheduling for statistically-assured real-time performance and improved energy efficiency. In *CODES+ISSS*, pages 110–115, 2004.

[41] H. Wu, B. Ravindran, E. D. Jensen, and P. Li. Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems. In *ACM EMSOFT*, pages 64–73, 2004.

[42] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP*, pages 149–163. ACM Press, 2003.

[43] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *RTSS*, pages 235–, 2002.

[44] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *DAC*, pages 628–631, 2005.

[45] J. Zhuo and C. Chakrabarti. Energy-efficient dynamic task scheduling algorithms for DVS systems. *ACM Trans. Embed. Comput. Syst.*, 7:17:1–17:25, January 2008.