Quantitative Evaluation of Single and Multicore Real-Time DVFS Schedulers in Linux

[*TechnicalReport*]

Antonio Barbalace
ECE Virginia Tech
Blacksburg, VA, USA
antoniob@vt.edu

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

ABSTRACT

We report on our experience in implementing and evaluating nine, state-of-the-art single and multicore real-time dynamic voltage and frequency scaling (DVFS) schedulers on an embedded Linux platform. The algorithms include CC-EDF, LA-EDF, DRA, AGR, CVFS, and DR, among others, and the platform is a dual-core ARM Cortex-A9 MPCore processor/PandaBoard, running a real-time Linux kernel. Our evaluations reveal the effectiveness of the algorithms' power savings. However, due to non-negligible scheduling and voltage/frequency transition overheads, non ideal time accounting, and our integrated Linux scheduling environment, their timeliness behaviors deviate from the theoretical. We propose solutions and identify reduced, schedulable total task utilization bounds.

Categories and Subject Descriptors

D.4.8 [**Operating Systems**]: Performance—*measurements, operational analysis*; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Design, Measurement, Performance

Keywords

DVFS, real-time, scheduling, Linux, ARM, single-core, multicore, single clock domain

1. INTRODUCTION

On a modern processor, there are mainly two methods to reduce power consumption: dynamic power management (DPM) and dynamic voltage and frequency scaling (DVFS). DPM is a feature that allows to turn off specific parts of the processor, while other critical components (e.g., clock generator, timer) keep running. Turning off parts of the processor is accomplished gradually (following a state machine), and involve transitioning the processor to progressively deeper idle or sleep states. An entire core, the ALU, and different cache levels can be clocked out or turned off in different sleep/idle states. Deeper the sleep/idle state, greater is the latency incurred to transition the processor into that sleep/idle state, and the resulting power savings [25].

DVFS involves changing the speed of the processor by varying the clock frequency along with the supplied voltage. The processor frequency can be changed by modifying a set of clock multipliers, such change incurs a transition latency due to the waiting time for the phased locked loop (PLL) oscillator to lock on the new frequency. This transition latency depends on the from and to frequencies. The processor voltage can be changed by requesting a voltage transition to the (usually external) supply regulator. A communication between the processor and the supply regulator bounds the latency incurred in the transition. The frequency and voltage must be changed in a coordinated manner, because for each supply voltage, there is a maximum permitted frequency. During a voltage and frequency transition, the core/processor can continue executing.

Both DPM and DVFS have been extensively used in the real-time literature to obtain power savings, along with techniques that exploit *static* and *dynamic slack* of tasks. The static slack 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, which is called the static slack. The dynamic slack is the slack time that is available when the actual execution time of a task (AET) is smaller than its (predicted) WCET. 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. While DPM is used to put the processor in the sleep/idle state during idle times, DVFS is used to "stretch" the execution time of each task to reduce the idle time as much as possible [10, 9]. Both techniques are used in single and multicore systems. Note that stretching the real-time task set to completely reduce the idle time due to static and dynamic slack will not let other tasks (e.g., soft or non real-time) to complete.

Despite the extensive study of DPM and DVFS techniques in the real-time literature, vast majority of them have been evaluated using simulations [20, 1, 7, 12, 23]. Very few techniques have been implemented in an OS and evaluated through actual timeliness and power measurements. (See Section 2.)

Simulation-based studies have limitations. For example, most simulation results presented in the literature consider continuous voltage and frequency scaling ranges as well as instantaneous voltage and frequency transition time in the processor [20, 1, 7, 12, 23]. Additionally, all assume continuous time and precise timekeeping at the OS-level. Moreover, in the multicore domain, no core-wide synchronization issues have been considered in both the partitioned and global scheduling disciplines [8].

Contributions. We implement a suite of single-core and multicore real-time DVFS algorithms supporting a single voltage and frequency domain in a real-time Linux kernel, and experimentally evaluate their timeliness and power consumption on an embedded platform. Instead of considering the interplay of DPM and DVFS to obtain the maximum power savings, we focus only on DVFS algorithms. This choice was due to two important factors (often not considered in the literature): how to handle DVFS scheduling techniques in an integrated OS scheduling environment (Sections 3.1 and 3.3), and how to guarantee that the same code can run unmodified and efficiently on different platforms (portability). In general, without an accurate power model of the processor (which is platform specific), it is almost impossible to make the best power transition decision. The power model can be inferred by extensive measurements or from the processor specifications.

The real-time DVFS scheduling algorithms that we implemented belong to the earliest-deadline-first (EDF) family of algorithms, and include CC-EDF, LA-EDF, DRA, AGR, CVFS, and DR, among others. The algorithms are either single-core or multicore partitioned schedulers. (Since global real-time DVFS scheduling has been sparsely studied, we ignore that class.) All algorithms belong to the class of inter-task schedulers: DVFS decisions are made only at task boundaries (release/execution, preemption and termination times).

We used the ChronOS [11] real-time Linux kernel in the implementation. ChronOS integrates Ingo Molnar's `PREEMPT_RT` patch, which allows nearly all of the kernel to be preempted yielding superior interrupt service latencies, along with support for (single-core and) multicore real-time scheduling. Our platform is an OMAP4430 dual-core, ARM Cortex-A9 MPCore mounted on a PandaBoard (`pandaboard.org`). We chose the ARM platform due to its wide use in embedded systems. After overviewing related work (Section 2) and describing our implementation setup (Section 3), we report our experimental results (Section 4).

Our contributions include solutions to the following, real-time DVFS scheduler implementation problems:

- how to **account and track elapsed time** in single and multicore architectures. Most of the past efforts count processor cycles instead of considering time intervals, which eases the computation logic, but is not easily applicable on modern hardware and software. (Sections 3.2 and 4.1.)
- how to **deal with transition times** in single and multicore real-time DVFS scheduling. We show that transition times are non-negligible overheads. We propose a worst case accounting technique for bounding the new available total utilization. (Sections 4.1 and 4.3.)
- how to deal with **discrete voltage and frequency levels**. We tested the same application with different DVFS transition levels, measuring and comparing power savings. (Section 4.2.)
- how to redesign **multicore DVFS schedulers to reduce synchronization overheads for changing processor speeds**. We propose a transactional based approach where only the latest core involved in the speed changing procedure will commit the speed change. Every other core will abort without retry. (Section 4.3.)

To the best of our knowledge, our work is the first effort at integrating a suite of single and multicore real-time DVFS schedulers in a general purpose OS such as Linux, and measuring the schedulers' actual power and timeliness behaviors on a community supported embedded hardware such as the PandaBoard. Our implementation is publicly available at `chronoslinux.org`.

2. RELATED WORK

One of the earliest real-time DVFS (single-core) scheduler is due to Pillai and Shin [20]. Their basic algorithm, called static EDF (S-EDF), exploits static slack, and another variant called, cycle conserving EDF (CC-EDF), exploits dynamic slack. They also present an algorithm called Look-Ahead EDF (LA-EDF), which reclaims dynamic slack. Aydin *et al.* [1] presented another approach for single-core DVFS EDF scheduling called, dynamic reclaiming algorithm (DRA). DRA was extended with the one task executing (OTE) technique of Shin and Choi [24] for fixed priority algorithms. The resulting algorithm, DRA-OTE, was further extended with a speculative technique based on average task usage, called aggressive speed reduction technique (AGR) [1]. The same idea of adopting the average task usage to scale down the processor speed was also adopted by Chiu *et al.* in their aggressive look-ahead EDF (ALA-EDF) [7]. Lawitzky *et al.* described a DVFS EDF algorithm [14] that extends two slack reclaiming algorithms of Lin and Brandt [17]. We refer to an implementation of this algorithm as SNOWDON, as in [22].

However, these works evaluate the algorithms using simulations. Implementations have been done, but are very few: Lawitzky *et al.*'s algorithm [14] was implemented and evaluated. (They also presented ways to account for transition times and to avoid time measurement in processor cycles.) However, the work did not report any actual timeliness and power measurements. Snowdon *et al.* develop power management techniques and present implementation-based measurements in a body of work [26, 30, 27, 29, 28, 6, 16, 15]. However, these works do not consider task timing constraints and real-time OS issues. Performance degradation, feasibility, and usability questions are addressed by Isci *et al.* [13] through an implementation-based study, but the work also targets non real-time systems.

In the multicore DVFS literature, DPM is usually considered, because on large core count machines, putting cores to sleep/idle state can provide greater power savings than by using DVFS alone. One of the first real-time DVFS scheduler that considers multicore architectures (with one clock domain) and EDF is the coordinated voltage and frequency scaling (CVFS) technique [12], which is an adaptation of CC-EDF for multicore partitioned scheduling. Seo *et al.* [23] further extended CC-EDF, called dynamic repartitioning (DR). While CVFS assumes an offline optimal partition of the tasks, DR does not do so, and dynamically migrates tasks to balance workloads across cores. Aydin and Yang also analyze different offline partitioning methods in [2]. Zhu *et al.* [21, 31] develop a DVFS scheduler for frame-based task sets, considering different voltage and frequency islands. Bhatti *et al.* consider per core voltage and frequency scaling [3], and extend DRA for multicores with a technique called deterministic stretch-to-fit (DSF). However, all these works focus on simulation-based evaluations.

3. IMPLEMENTATION SETUP

3.1 Power Management

Conventionally, operating systems enter sleep/idle states after staying in the idle state for a predefined time interval, during which the processor runs the idle task, wasting energy. If the idle intervals occur intermittently and their length is short, the system can experience performance degradation, and in general, the power savings may not be significant. To improve power savings in the idle task, hardware vendors use NOP instructions that consume far less amount of power then normal instructions (e.g., 20% [18]). In the Linux kernel port for the ARM architecture, the idle task mainly consists of the recently introduced wait for interrupt (WFI) instruction [19]. Once issued, this instruction moves the processor to standby where only the wakeup logic is clocked, improving the power savings. Deeper sleep/idle states require the usage of *CPUidle* governors that are architecture dependent, which attempts to "guess" the best sleep/idle state to transition the core to, obtaining power savings.

DPM techniques are available in Linux. Additionally, DVFS techniques can be implemented via *CPUfreq* governors, with estimations of processor usage used to scale the frequency up or down. Both *CPUidle* and *CPUfreq* governors consider transition times.

3.2 Time Keeping

Time keeping in general purpose OSes is not required to be accurate (especially if this means special hardware, or synchronization-costly). Time keeping is usually required for tracing or profiling and other debugging purposes; the only real requirement is monotonicity along all cores. Linux maintains monotonicity (`current_kernel_time()`) along different cores and subsequent time calls. However, this does not mean that the clock is always timely updated. Timer interrupts can experience delays, due to non-preemptable code running concurrently. The system clock is usually not fine-grained, which does not help in precise estimation of task execution times. In contrast, real-time DVFS scheduling presumes coherent clock along all cores, monotonicity, and high resolution.

3.3 ChronOS Linux

ChronOS Linux [11] is a real-time multiprocessor scheduler extension of the `PREEMPT_RT` Linux kernel branch. It runs on 32/64bit x86 and ARM processors. The reference Linux version used in this paper is 3.0.10. ChronOS offers an integrated scheduler for critical, hard, soft, and non real-time tasks in Linux. Figure 1 illustrates ChronOS scheduling environment. Example critical system tasks include time keeping and multiprocessor management, which have a higher

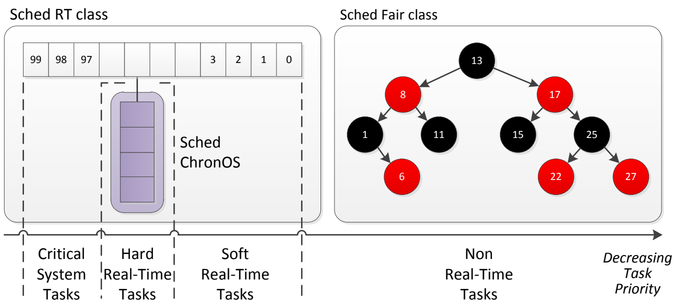

Figure 1: ChronOS Linux integrated scheduling environment. Critical, hard, soft and non real-time tasks must be scheduled with different guarantees.

priority than hard real-time tasks. Hard real-time tasks are managed by the ChronOS scheduler; all ChronOS tasks have the same `SCHED_FIFO`/`SCHED_RR` Linux priority value. Tasks with lesser priorities (down to zero) are considered soft real-time tasks. Non real-time tasks are scheduled by Linux CFS. (A similar integrated scheduling approach is used in RBED [5].) In this integrated scheduling environment, tasks running with higher priority then hard real-time tasks must be carefully considered. In our real-time DVFS setup, interrupt handlers are running as critical system tasks.

ChronOS allows implementation of both single and multicore schedulers. Multicore schedulers can run concurrently on any processor (i.e, the partitioned scheduling approach), or globally synchronizing every processor. We have implemented a suite of different single-core scheduling policies including S-EDF, CC-EDF, LA-EDF, SNOWDON, DRA, DRA-OTE, AGR1 and AGR2. Additionally, we have implemented the partitioned versions of S-EDF, CC-EDF, CVFS and DR by means of a global scheduler that runs concurrently or by globally synchronizing all cores.

To change the frequency speed, we interface the scheduler with the *CPUfreq* module by creating a new in-kernel "governor" that accomplishes the required scaling, which provides portability across different architectures (see Figure 2). A governor is a kernel object that decides when and how to change the processor frequency and command the change. ChronOS schedulers run in the middle of the Linux scheduler code that operates with interrupts disabled. Hence, frequency changes, which may require interrupts to be disabled or to hold a global lock, cannot be issued inside the ChronOS scheduler. A post schedule processing callback commands the frequency change, before the control returns to the dispatched user space task.

3.4 Measurement Framework

To record the power consumption of the device under test (i.e., the PandaBoard), we measure its instantaneous current consumption while using a constant voltage supply. Figure 3 shows a schema of our measurement setup. All the current data were logged by the *data logging workstation*, which is directly connected by an optical serial cable with the measurement equipment (AKTAKOM AM-1118). The *control workstation* sequentially loads every test on the PandaBoard and timestamps the recorded data on the remote *data logging workstation*. A post processing stage was required to correlate the whole data.

To obtain the power measurements and the deadline sat-

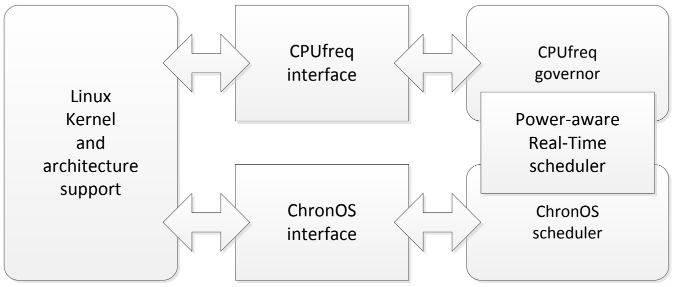
Linux Kernel and architecture support
CPUfreq interface
CPUfreq governor
Power-aware Real-Time scheduler
ChronOS interface
ChronOS scheduler

Figure 2: To provide portability, a DVFS real-time scheduler interfaces with the ChronOS API for choosing the next task to execute and with the *CPUfreq* subsystem for frequency scaling.

isfaction ratio (DSR), we run a synthetic application. This application, originally developed in [11], can run different workload types that focus on CPU usage, memory usage, cache trashing, pipeline flush, etc. We adopted the CPU intensive workload for the evaluation reported here, because it reasonably mimics a computational data filtering application. We ran the single-core tests using a 5-task set with a hyperperiod of 15 seconds. The same task set was extended for multicore tests.

4. EXPERIMENTAL RESULTS

4.1 Transition Time

Despite that all our candidate algorithms belong to the EDF family (which is theoretically optimal for single-cores), we observed that, they never gave a DSR of 1.0 (i.e., all deadlines met) on a single-core machine. However, they gave a DSR of 1.0 when running at the highest frequency, after disabling DVFS scheduling. This happened at any total utilization and AET/WCET ratio, and jeopardizes the schedulers' (hard real-time) timeliness assurance. Nevertheless, their power savings were found to be effective.

Figure 4 shows the board power consumption (Watts) and the DSR. Each data point in the figure is an average of 9 runs, with an average absolute deviation of $\approx 20mW$ for the power consumption and ≈ 0.05 for the DSR. The non smoothness of the curves is due to i) the low sample rate of our measurement device, which only gives us approximate measurements, and ii) the non ideal behavior of the synthetic application that only approximately executes for the

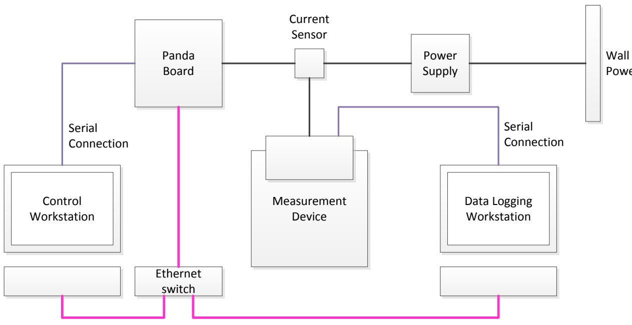

Figure 3: Measurement setup. We measured the full PandaBoard power consumption.

required amount of time. In Figure 4, within each column, the task set's total utilization is changing (at the fixed steps 0.2, 0.4, 0.6, 0.8, and 1.0). Within each column, the ratio of tasks' AET to WCET is fixed. In the rightmost column, where the ratio of AET to WCET is 1.0, power savings is entirely due to static slack, and all curves, except that of classical EDF (which is non power-aware) overlap.

Note that frequency scaling saves more power then the idle task: DRA-OTE gives a power savings of $\approx 350mW$ (on the whole board). The power that can be saved exploiting the dynamic slack is shown in the other columns in Figure 4. Smaller the total utilization, smaller is the power savings due to dynamic slack, and greater is the power savings due to static slack. The power savings due to dynamic slack lie under the S-EDF curve, and greater the total utilization, greater is the power savings due to dynamic slack. Peak savings are $\approx 350mW$ using the best performing algorithms (DRA-OTE and AGRs) outperforming the idle task (which moves the processor to the idle state).

We note that the power trends are extremely similar to those in the original works (which were largely based on simulation studies). What is not considered in those works is the penalty due to scheduler and voltage/frequency scaling, which results in deviations from predicted execution times. Such deviations manifest also for classical EDF, but only when the total utilization is close to 1.0: the rightmost plot in Figure 4 shows the DSR for total utilization 1.0, and AET/WCET ratio of 1.0 for a DSR of 0.9365. This demonstrates that scheduler overheads and non accurate accounting of critical system events/tasks (which have higher priority than ChronOS tasks) matter in an integrated scheduling environment when the task set fully utilizes the processing resources. (See also Section 4.3.)

Different algorithms expose different penalty patterns. S-EDF and CC-EDF are very close to EDF, and exhibit low DSR degradations. DRA, DRA-OTE, and AGR1 are worse at lower AET/WCET ratios, while AGR2 is worse at higher AET/WCET ratios. LA-EDF exhibits the worst DSR completing on time the 60% of the jobs at the maximum total utilization.

We also note that, there is no explicit correlation between power savings and DSR (i.e., it is not true that lower DSR means greater power savings).

Accounting transition times. To solve this problem, we account for speed transition in the WCET at task subscription. Such accounting is done in the kernel space because is the *CPUfreq* module that gives the worst case transition time. Lawitzky *et al.* indicate that, in single-core systems, each task's job WCET must be augmented by two transition times [14]. We found that this is correct, and extended the same assumption for multicore partitioned scheduling with one clock domain. (See Section 4.3.)

Adding the transition time reduces the total utilization of the task set. This is a function of the number and characteristic of the tasks and the duration of the transition time. In particular, for short task durations and high count task sets, the total available utilization is evidently reduced. The maximum total utilization for EDF is reduced in the single-core case to: $u_{tot} = 1 - 2t_{trans} \sum_1^n \frac{1}{p_i}$.

Transition avoidance. Besides accounting for transition times, we also added a transition avoidance check at each rescheduling point, similar to [14]. A speed reduction transition is avoided when the dynamic slack time earned is less

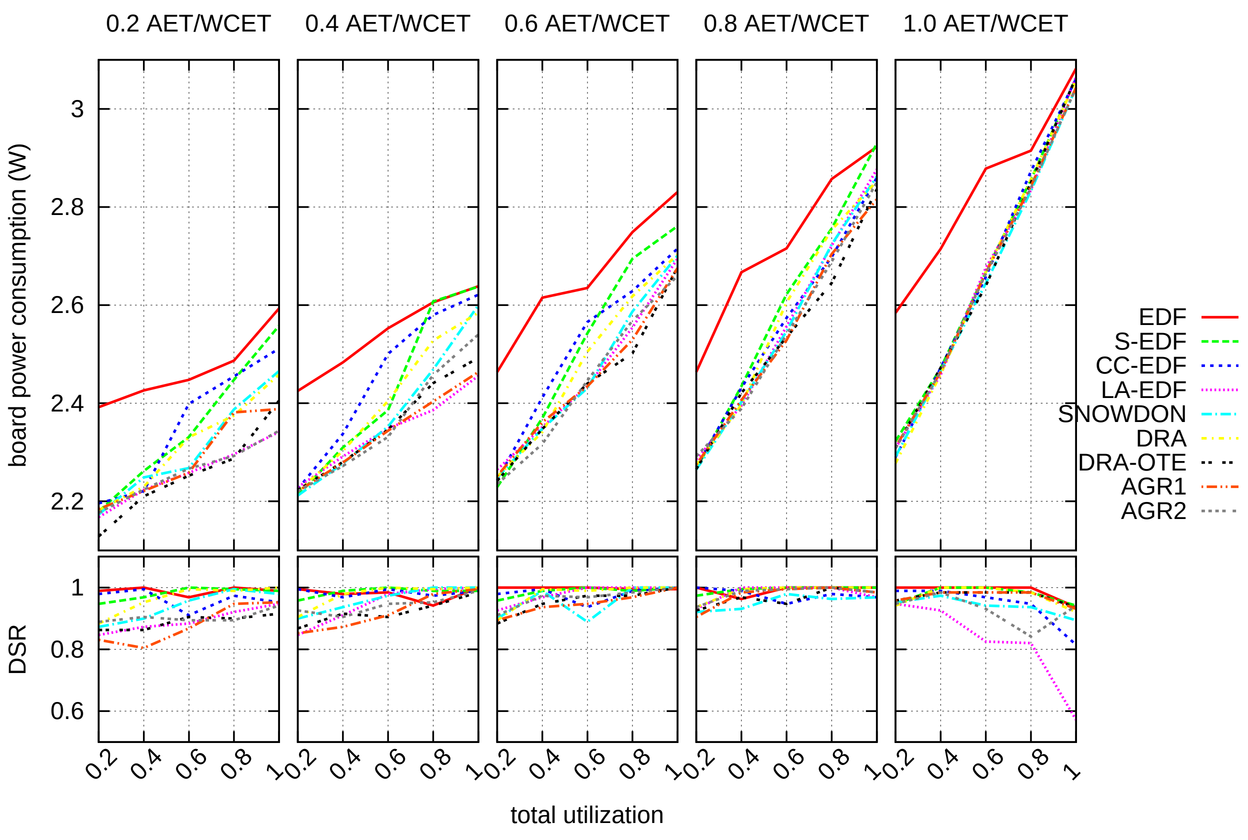

Figure 4: Board power consumption (top) and deadline satisfaction ratio (bottom) of single-core DVFS algorithms. Hard real-time tasks in ChronOS cannot miss their deadline.

or equal to the transition cost. These two simple techniques improved the algorithm DSR to 1: Figure 6 shows the DSR and power improvements of CC-EDF by introducing transition time accounting and transition avoidance.

Transition timestamping. Since we cannot rely our estimation on processor cycles, we require precise timekeeping. To calculate the exact amount of WCET consumed by each task's job, it is necessary to trace and record every frequency transition. We timestamp the speed transition before it actually happens. A more precise timestamping, in an architecture-independent manner, is future work.

4.2 Discrete Voltage and Frequency Levels

Vast majority of new processors support a broad range of synthesizable frequencies, but not voltages. The Linux port for the OMAP4430/PandaBoard currently supports only four voltage and frequency levels (we call this set of levels, *original*). Without manipulating the voltage levels, we extend the number of available frequencies to 10 and 35 to understand whether the power savings improve (we named these sets of frequencies *standard* and *many*). In most of the results presented, we used the *standard* set.

Figure 5 reports the mapping between algorithm's required utilization and adopted frequency (for each frequency set, *original*, *standard*, and *many*) and voltage.

Figure 5 shows the power and DSR of CC-EDF on a single-core, with the original voltage/frequency setting and with the augmented ones (with and without considering transition times). For the rest of the paper, we focus on CC-EDF as the reference algorithm.

From Figure 6, it is clear that having many frequency steps improve power savings. These savings are not more than $100mW$ comparing *original* and *many* in the optimal situation, when the *original* curve is the farthest from the *many* curve. Thus, more scaling points improve the power savings. However, as indicated in [4], width modulation of fewer scaling points nevertheless obtains good power savings.

Figure 6 also shows the DSR, with and without considering the transition times. We notice that considering the transition times always gives good DSR results. When transition times are not considered, power savings are still obtained, but the behavior is not deterministic.

Exploiting the frequency difference. When using discrete levels of voltages, the chosen voltage is always the upper bound that can guarantee that the processor runs safely at the specified frequency. The difference between the exact voltage that map to the requested frequency and the voltage value chosen is quadratically proportional to the amount of further power that can be saved, providing a rich set of available voltages. Even though there are a broad range of synthesizable frequencies, it is unlikely that there exist one for each utilization value requested by the scheduler. As shown in Figure 5, there are different buckets of frequencies running at the bucket's maximum value. The result is that, the current task is running at a frequency greater than the one requested by the scheduler. We observed that the expected finishing time of the task at the two different frequencies is different (AET or WCET), and this time difference

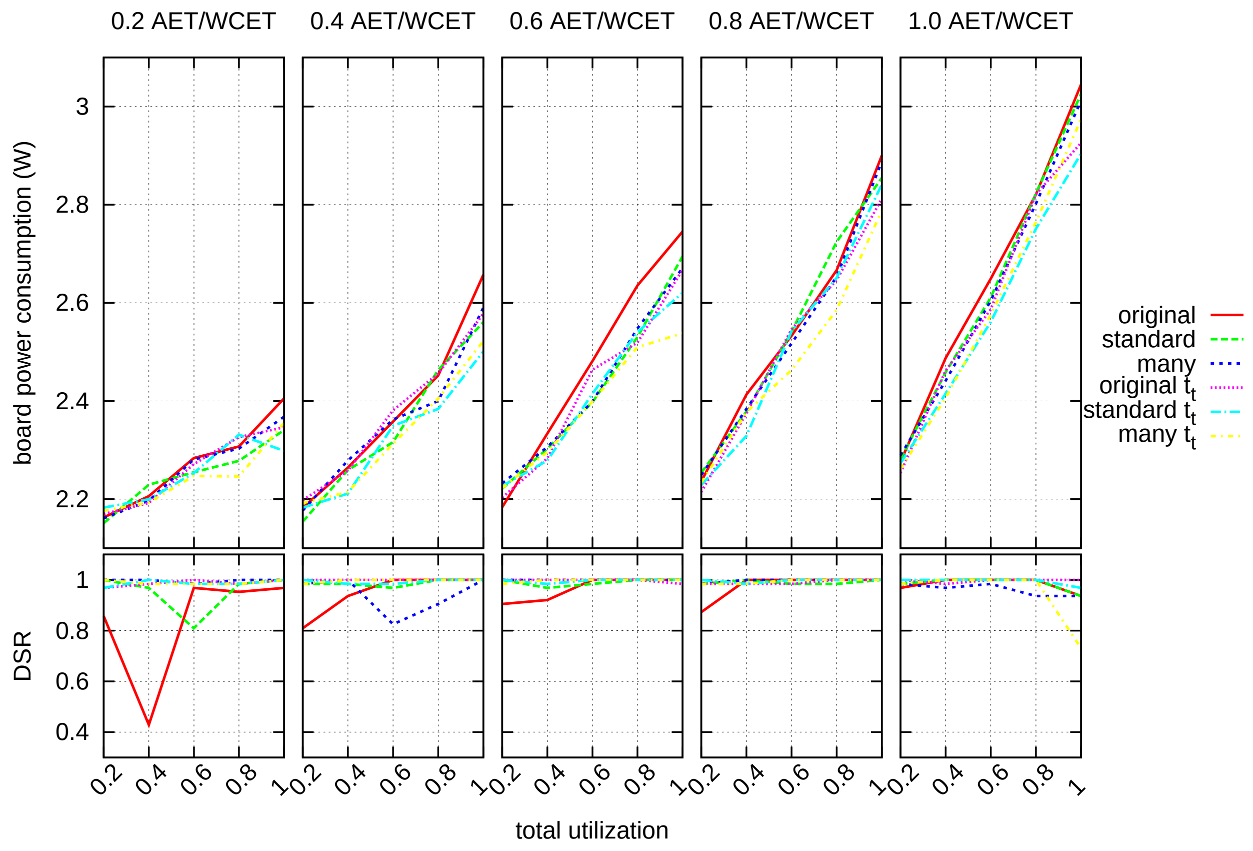

Figure 6: Board power consumption and DSR of the CC-EDF algorithm at different sets of discrete frequency levels (*original*, *standard* and *many*), with and without considering transition times (*original*, *standard*, *many*, *original* t_t, *standard* t_t, and *many* t_t).

can be used to further slow down the processor for the next task. A similar idea is used in the CVFS algorithm. This technique can be used in single and multicore algorithms. We apply this technique in single-core CC-EDF, and show the resulting power improvements in Figure 7. The result is that CC-EDF's net power is not always improved using this technique. A $50mW$ power improvement is shown in the 0.8 AET/WCET column. Also, note that, without considering transition times, DSR reduces.

4.3 Multicore DVFS Scheduling

Due to the single clock domain restriction of the PandaBoard (newest product in the embedded market), the only two relevant multicore DVFS schedulers in the literature include CVFS [12] and DR [23]. We implemented both. Both CVFS and DR are partitioned scheduling algorithms, which allow different partitioning schemes to be used (we generated them offline). Both of them take advantage of a balanced partition. DR dynamically migrates tasks to balance the load across cores, temporarily breaking the partition. CVFS tracks the difference between the requested frequency and the actual frequency at which the processor is running. In a multicore environment with a single clock domain, the maximum frequency requested among every core is selected as the running frequency. Coordination between cores is required.

Concurrency control. Partitioned algorithms running in a multicore processor with a single clock domain suffer from the problem of how to access shared resources, which include clock domain registers, the global list of all frequency changes, their timestamps, and per-core desired frequency value arrays. We developed a transactional protocol that avoids conflicts when cores concurrently access these resources. The protocol detects concurrent accesses eagerly, and speculatively aborts the oldest requesting core. Even though the protocol should reduce per task transition times, we could not measure that evidence on our dual-core processor. In an M-core partitioned environment with a single clock domain, which must be serially accessed, every task's job is subject to $2M$ transition times. This worst case happens when no one core is able to detect a conflict early, and each core reduces its frequency. The reduced utilization per core can be recalculated as $2Mt_{trans}$ (instead of $2t_{trans}$). The maximum total utilization for partitioned EDF is reduced in the multicore case to $u_{tot} = (M-1)/2 - 2Mt_{trans}\sum_1^n \frac{1}{p_i}$.

In Figure 8, we compare board power consumption of CC-EDF and CVFS accounting for transition times with a total utilization $U_t = 1$ and $U_t = 2$. In general, CVFS saves more power then CC-EDF, around $100mW$ with $U_t = 1$ and $200mW$ with $U_t = 2$. The (dynamic) power consumption of the processor increases to circa $3.7W$, fully loading both cores. A rough estimation of the power consumption of the chip is that each core, fully loaded, consumes $800mW$. The

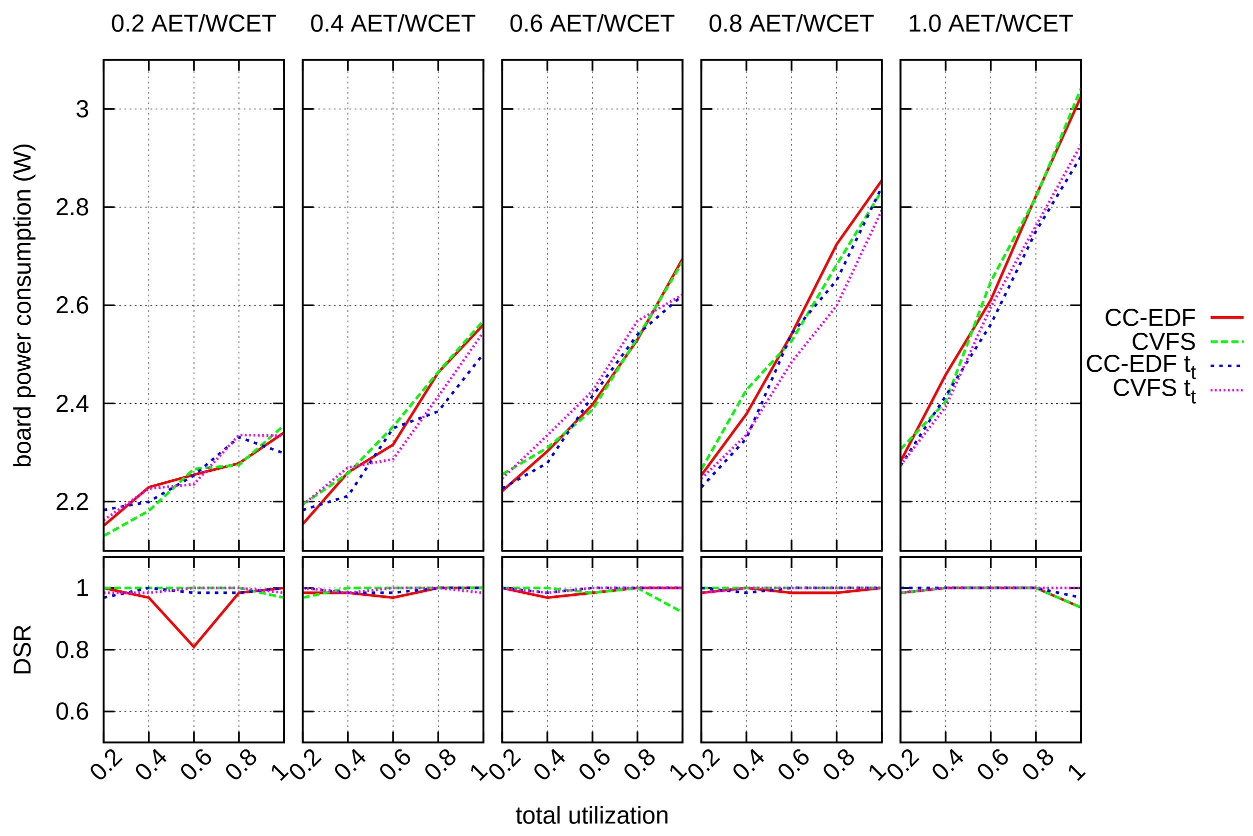

Figure 7: CC-EDF algorithm's board power consumption and DSR, with and without the CVFS optimization for single-core, and with and without transition times (CC-EDF, CVFS, CC-EDF t_t, and CVFS t_t).

rest of the power consumption ($2.1W$) is due to the CPU's static power consumption and that of other on-chip and off-chip devices.

In Figure 8, it is also evident that the DSR is not 1, despite considering the transition times and using our technique that reduces the waiting time for changing the processor speed. With trace-based debugging (i.e., using `ftrace`), we realized that, this is due to our integrated scheduling environment. Since ChronOS Linux is running atop the `PREEMPT_RT` patch, different non preemptable sections of Linux code can deviate our scheduler. We did not observe this in the single-core experiments, as the work was migrated to the other core.

Comparing Figures 8 and 7, we notice that the total power consumption when running the same load on one CPU or on two CPUs (with load balanced) consumes similar power. With DR, we obtain similar power and DSR results as CVFS for these workloads (where few migrations are required).

Stop the World. Instead of processor cores concurrently changing the clock, an alternative is to adopt a "stop the world" approach by globally synchronizing scheduling: the scheduling code runs on a single processor and all the other processors spin-await a decision. This does not require the previously described infrastructure. However, it increases the execution overhead, and also, the transition times are difficult to estimate and bound in the worst case.

5. CONCLUSIONS

Our major conclusion is that real-time DVFS scheduling can save power, but will likely degrade DSRs. After incorporating expedients from practice, the DSR degradation translates into reduced schedulable utilization bounds. When the utilization bound is significantly degraded, DVFS scheduling is not effective, as the processor idle state will yield similar power savings. (This is also the conclusion in [16].) When transition times are short (compared to task periods), power savings is larger, and the utilization bound is not seriously degraded. This is valid for both single and multicores.

The low core count of our designated processor did not let us show the benefits of our transactional approach for accessing shared resources (needed to make a speed transition). With emerging high core-count embedded hardware, we expect benefits from this approach.

6. REFERENCES

[1] H. Aydin, R. Melhem, et al. Power-aware scheduling for periodic real-time tasks. *Computers, IEEE Transactions on*, 53(5):584 – 600, 2004.

[2] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *IPDPS*, page 9 pp., 2003.

[3] M. Bhatti, C. Belleudy, and M. Auguin. An inter-task real time DVFS scheme for multiprocessor embedded systems. In *DASIP*, pages 136 –143, oct. 2010.

[4] E. Bini, G. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *ECRTS*, pages 3–10, 2005.

[5] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft

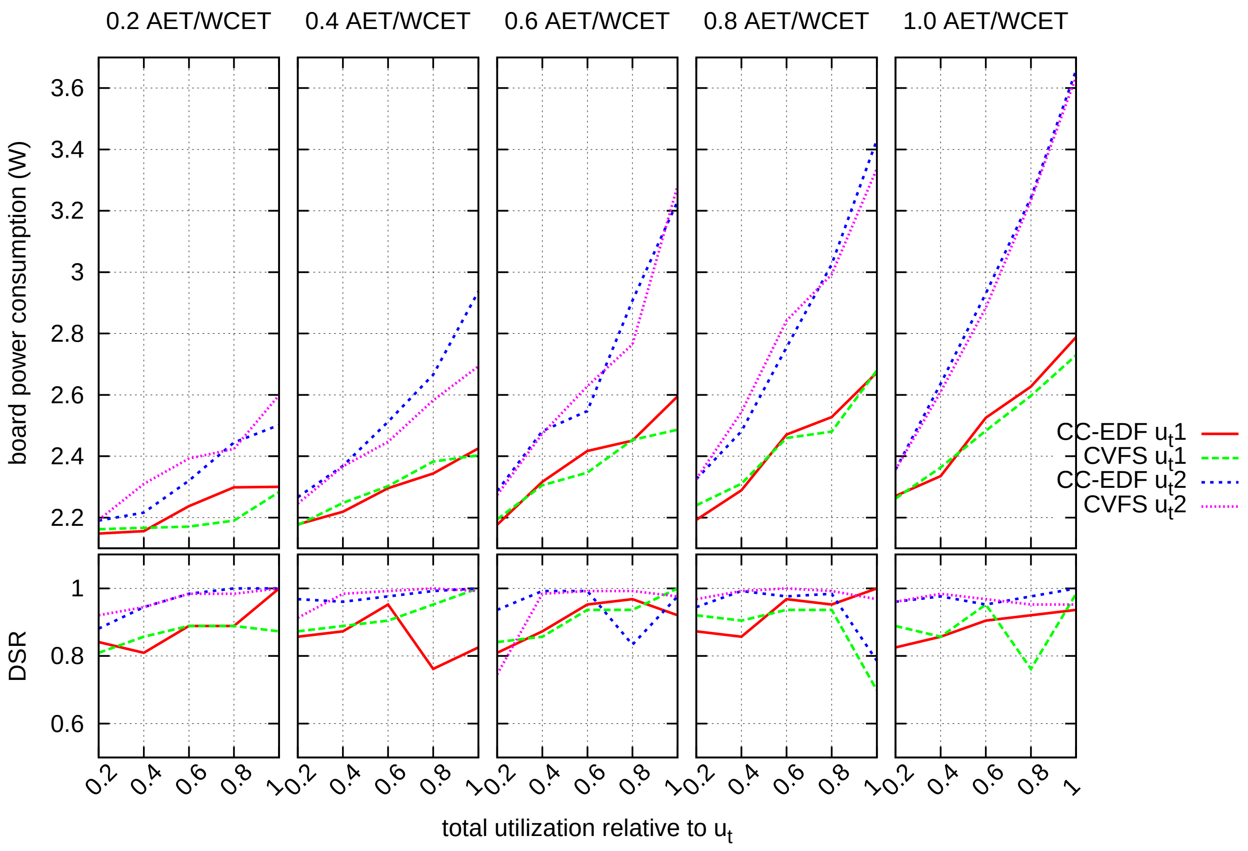

Figure 8: CC-EDF and CVFS running on dual core, considering transition times, with total utilization $U_t = 1$ (u_t1) and $U_t = 2$ (u_t2).

real-time, and non-real-time processes. In *RTSS*, pages 396 – 407, 2003.

[6] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX ATC*, pages 21–21, 2010.

[7] L.-Y. Chiou, H.-E. Lim, and Y.-S. Chen. Aggressive look-ahead earliest deadline first algorithm. In *TENCON*, pages 1 –4, 2007.

[8] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, Oct. 2011.

[9] P. de Langen and B. Juurlink. Leakage-aware multiprocessor scheduling for low power. In *IPDPS*, page 8 pp., 2006.

[10] P. de Langen and B. Juurlink. Trade-offs between voltage scaling and processor shutdown for low-energy embedded multiprocessors. In *SAMOS*, pages 75–85, 2007.

[11] M. Dellinger, P. Garyali, and B. Ravindran. Chronos linux: A best-effort real-time multiprocessor linux kernel. In *DAC*, pages 474 –479, 2011.

[12] V. Devadas and H. Aydin. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Int'l. Green Computing Conf.*, pages 61 –72, 2010.

[13] C. Isci, A. Buyuktosunoglu, et al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO*, pages 347–358, 2006.

[14] M. P. Lawitzky, D. C. Snowdon, and S. M. Petters. Integrating real time and power management in a real system. In *OSPERT*, 2008.

[15] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *HotPower*, Oct 2010.

[16] E. Le Sueur and G. Heiser. Slow down or sleep, that is the question. In *USENIX ATC*, pages 16–16, 2011.

[17] C. Lin and S. A. Brandt. Improving soft real-time performance through better slack reclaiming. In *RTSS*, pages 410–421, 2005.

[18] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *ISLPED*, pages 76–81, 1998.

[19] L. Pieralisi. Consolidating linux power management on arm multiprocessor systems, 2011.

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

[21] X. Qi and D. Zhu. Power management for real-time embedded systems on block-partitioned multicore platforms. In *ICESS*, pages 110–117, 2008.

[22] S. Saha and B. Ravindran. An experimental

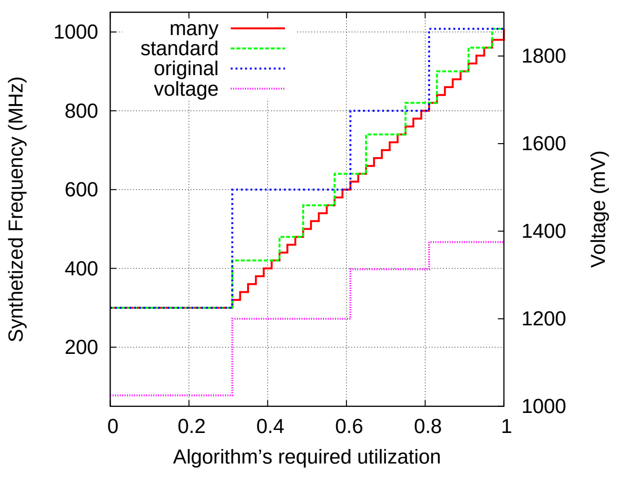

Figure 5: Three different sets of frequency levels (4 levels in the *original* setup, 10 levels in the *standard* setup, and 35 in the setup called *many*), and a single set of voltage levels.

evaluation of real-time dvfs scheduling algorithms. In *SYSTOR*, 2012.

[23] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE TPDS*, 19(11):1540 –1552, nov. 2008.

[24] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *DAC*, pages 134–139, 1999.

[25] A. Sinkar and N. S. Kim. Analyzing potential power reduction with adaptive voltage positioning optimized for multicore processors. In *ISLPED*, pages 189–194, New York, NY, USA, 2009. ACM.

[26] D. C. Snowdon, E. Le Sueur, et al. Koala: a platform for os-level power management. In *EuroSys*, pages 289–302, 2009.

[27] D. C. Snowdon, S. M. Petters, and G. Heiser. Power measurement as the basis for power management. In *OSPERT*, 2005.

[28] D. C. Snowdon, S. M. Petters, and G. Heiser. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *EMSOFT*, pages 84–93, 2007.

[29] 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*, 2005.

[30] D. C. Snowdon, G. van der Linden, et al. Accurate run-time prediction of performance degradation under frequency scaling. In *OSPERT*, 2007.

[31] D. Zhu, R. Melhem, and B. R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE TPDS*, 14(7):686–700, jul 2003.