

AN INTER-TASK REAL TIME DVFS SCHEME FOR MULTIPROCESSOR EMBEDDED SYSTEMS

Muhammad Khurram Bhatti¹, Cecile Belleudy², Michel Auguin³
{¹bhatti, ²belleudy, ³auguin}@unice.fr

^{1, 2, 3}LEAT Research Laboratory, University of Nice-Sophia Antipolis –CNRS, Nice, France

Abstract

In this paper¹, we have addressed energy-efficient scheduling of real time applications intended to be executed on multiprocessor systems. Our proposed technique, called Deterministic Stretch-to-Fit (DSF) technique, is based on inter-task real time dynamic voltage and frequency scaling (RT-DVFS). It mainly comprises of three components. Firstly, we propose an online algorithm to reclaim energy by adapting to the variations in actual workload of target application tasks. Secondly, we extend our online algorithm with an adaptive and speculative speed adjustment mechanism. This mechanism anticipates early completion of future task instances based on the information of their average workload. Thirdly, we propose a one-task extension technique for multi-task multiprocessor systems. No real time constraints of target application are violated while applying our proposed technique. Simulation results show that our online slack reclamation algorithm alone gives up to 53% gains on energy consumption and our extended speculative speed adjustment mechanism, along with the one-task extension technique, gives additional gains, reaching a theoretical low-bound on the scalable frequency and voltage.

Keywords: Real time systems, Scheduling, RT-DVFS, Energy-efficient scheduling, Power consumption.

I. INTRODUCTION

Energy efficiency in embedded systems is achieved by actively changing the power consumption profile of system components. Over the past decades, this change in profile has been achieved by putting system components into low power/energy states whenever they are idle. Such techniques are referred to as Dynamic Power Management (DPM) techniques and are well studied and practiced in early as well as modern embedded systems. State-of-the-art on DPM techniques can be found in [24, 25, 26, and 27]. In the recent past, the processor technology has much evolved and permits to have various voltage levels and multiple operating frequencies, such as in XScale® technology-based embedded processors [9] and Transmeta Crusoe [8], allowing the design of highly flexible systems. The choice of operating voltage and frequency in such highly flexible systems has direct impact on processors' speed and consequently, on the ordering and execution of tasks. Thus, scheduling techniques and voltage/frequency selection mechanisms need to be addressed together. Doing so provides the opportunities for achieving energy-efficient systems through scheduling. Recently emerged real time dynamic voltage and frequency scaling (RT-DVFS) technique is one of the most promising techniques in

energy-efficient scheduling. Real time applications exhibit large variations in their actual execution time and thus, often finish much earlier than their estimated worst-case execution time [6, 22]. RT-DVFS-based techniques exploit these variations in *actual* workload for dynamically adjusting voltage and frequency in order to reduce power consumption of processors. The challenge for these techniques, however, is to preserve the feasibility of schedule and provide deadline guarantees. In [1], authors have broadly classified RT-DVFS techniques into intra-task and inter-task strategies. In intra-task strategies, available (dynamic) slack time is reallocated inside the same task. These techniques often require insertion points in application's code to measure the evolution of a task over its execution time. Drawback of such techniques is the complexity and cost of insertion points for measurement [2, 3, and 4]. On the other hand, inter-task RT-DVFS techniques redistribute dynamic slack either among all ready tasks [1] or to immediate priority single ready task only at task boundaries. Fig.1 gives a perspective of how the execution of a task takes place under different RT-DVFS approaches. Parameters in Fig.1 like; r_i , d_i , AET, and WCET refer to release instance, deadline, actual execution time, and worst-case execution time of a task, respectively. Fig.1 (a) depicts a task's execution without RT-DVFS while Fig.1 (b) depicts an ideal stretch approach. An ideal stretch, however, is not feasible in practice because it is not possible to know a priori the exact amount of execution time a task may require. Fig.1 (c) and (d) depict stochastic scheduling-based and code instrumentation-based approaches, respectively. Both these approaches have opposite power consumption profiles. Indeed, the speed of processor increases with time in (c) while decreases in (d) as the knowledge of remaining execution time become clearer. Fig.1 (e) depicts the scenario where dynamic slack time is distributed to all ready tasks, i.e., processor's speed is globally reduced. This technique results in small values of slowdown factor for speed and eventually, smaller gains on energy consumption. This paper presents a novel inter-task RT-DVFS technique called Deterministic Stretch-to-Fit (DSF) technique. DSF technique permits a hard real time application to benefit from the same timeliness guarantees as provided by conservative (offline) schedulability analysis based on worst-case workload. This technique redistributes all dynamic slack time, produced by precedent task, to a single subsequent ready task in the appropriate priority order to maximize the slowdown of processor's speed such that the subsequent task, that consumes dynamic slack, does not execute beyond its worst-case execution boundary as depicted by Fig.1 (f). The idea to stretch a task's execution time up to its worst-case boundary is used earlier as well in single processor context. However, in multiprocessor systems, it is not trivial to apply simple stretching due to task migrations and preemptions. We provide an example in Section-IV to emphasis this argument. Rest of this paper is organized as follows. In Section-2, we briefly

¹ This work is supported by French project Pherma bearing reference ANR-06-ARFU06-003, Higher Education Commission of Pakistan, and the Secured Communicating Solutions (SCS) cluster of competitiveness.

discuss some related research work and describe the contribution of our research work in relation to the existing work. In Section-3, we provide our system model. In Section-4, we provide DSF technique in detail. In Section-5, we provide our experimental setup, simulation environment, and results. In Section-6, we conclude our paper.

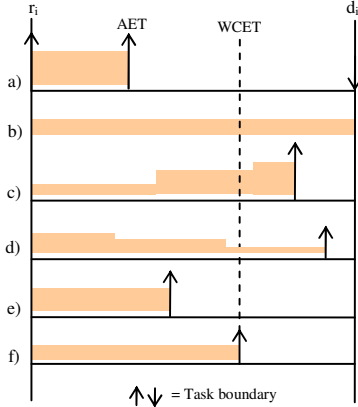


Fig. 1: Impact of different RT-DVFS approaches on the execution of real time tasks

II. RELATED WORK

Although RT-DVFS has become a major focus by real time and power-aware embedded systems research community in recent years, its proposition stems back as early as in 1994 when Weiser et al. first presented their work in [11]. Later on, numerous researchers have explored RT-DVFS on single-processor systems such as [2, 3, 4, 5, 6, 10, and 11]. However, fewer have considered the problem of applying RT-DVFS on multiprocessor platforms [15, 16, and 17]. For single processor systems, authors in [5] and [6] proposed inter-task RT-DVFS techniques for (single processor) optimal scheduling algorithms such as EDF and FPP. In [5], authors present a cycle-conserving approach in which cumulative resource utilization is recomputed at every scheduling event to rescale voltage and frequency appropriately. This approach however distributes dynamic slack time to all ready tasks which makes overall slowdown factor very small. Krishna and Lee proposed a power-aware scheduling technique using slack reclamation, but only in the context of systems with two voltage levels [18]. Hsu et al. described a performance model to determine a processor slow down factor under compiler control [19]. Based on the super-scalar architecture with similar power dissipation as the Transmeta Crusoe TM5400 [8], their simulation results show the potential of their technique. Mossé et al. proposed and analyzed several techniques to dynamically adjusting processor speed with slack reclamation in [20]. In multiprocessor systems, the problem of energy minimization by dynamic slack reclamation and dynamic speed adjustment for globally scheduled systems and independent task model is considered in [12]. Authors in this work consider dynamic slack sharing among multiple processors and reducing the speed globally. This work is extended in [13] to address the case of dependent tasks with AND/OR dependency constraints. In [14], Aydin et al. propose their solution for periodic hard real time tasks on identical multiprocessors with DVS support when only partitioned scheduling is used. Authors adopt the EDF scheduling policy and investigate the joint effect of partitioning heuristics on the energy consumption and the feasibility. For system-on-chip designs with two processors running at two different but fixed voltage levels, Yang et al. proposed a two-phase scheduling scheme that minimizes energy consumption under the time constraints by

choosing different scheduling options determined at compile time [17]. All these approaches, whether single or multiprocessor, achieve significant energy savings in their respective context. Our contribution in comparison to the existing work is described in the following.

A. Contributions

In this paper, we have proposed an inter-task dynamic voltage and frequency scaling technique for real time applications intended to be executed on multiprocessor platforms. Our contributions and differences with the previous work are presented in the following.

1. DSF technique is an *online* slack reclamation technique which addresses the issue of energy-aware *global* scheduling of periodic and independent tasks in multiprocessor hard real time systems unlike [14]. Our technique can also be applied in single processor systems, which is rather trivial.
2. Our proposed technique is *generic* in the sense that if a feasible schedule for a real time target application exists under worst-case workload using (optimal or non-optimal) global scheduling algorithms, the same schedule can be reproduced (using *actual* workload) with less power/energy consumption. DSF technique is indifferent to the applied scheduling algorithm. We demonstrate that the tasks will still meet their deadlines if the speed is reduced according to the rules we provide upon early completions. We achieve these objectives by exploiting only the *online* information of tasks. Unlike presented in [6], we do not keep track of the remaining execution times of all tasks in a static optimal schedule (in which each task instance presents its worst-case workload) rather, we construct an online canonical schedule ahead of the practical schedule.
3. Our technique permits the dynamic slack, produced by the precedent task, to be fully consumed by the subsequent (single) task at the appropriate priority level which is contrary to that of [5]. Such *greedy* allocation of slack allows DSF to largely varying the applicable voltage and frequency which eventually results in larger gains on energy consumption.
4. An *online, adaptive, and speculative* mechanism is proposed to anticipate early completion of tasks and aggressively slowdown processor speed. Apart from saving more energy, this speculative speed adjustment mechanism helps to avoid radical changes in operating frequency and supply voltage as well, which eventually results in less peak power consumption and longer battery life for portable embedded systems.
5. DSF technique can work for different processor technologies supporting discrete as well as continuous voltage and frequency scaling. Since processors which are able to operate on a (more or less) continuous voltage and frequency spectrum are fast becoming a reality [6, 23], therefore, we provide solution for systems that support both discrete as well as continuous dynamic voltage and frequency scaling.

III. SYSTEM MODEL & DEFINITIONS

A. Architecture model

Early research reports that processors alone can contribute up to 50% of total energy consumption of the system for computation-intensive workloads [6, 21]. Thus, we limit the scope of this paper to address processor-centric energy savings. We consider a multiprocessor platform composed of m identical processors and represented by $P = \{P_1, P_2, \dots, P_m\}$. We assume that the voltage and frequency can be varied on every processor independently and over a continuous spectrum between defined lower and upper bounds. This assumption can be easily lifted for processors with discrete operating frequencies. We consider that a statically specified

optimum speed (S_{max}) and corresponding maximum supply voltage (V_{dd}) and operating frequency (F_{ref}) are available. All other values for operating frequency and corresponding supply voltage are referred as F_{op} and V_{op} , respectively.

B. Application model

We consider that a real time application is being modelled as a finite set of n independent and recurring tasks which is referred as $TS = \{Ta_1, Ta_2, \dots, Ta_n\}$. Each member task is characterized by at least a quadruplet (r_i, C_i, d_i, T_i) where the elements of quadruplet refer to release time instance, worst-case execution requirement, relative deadline, and periodicity of a task, respectively. Moreover, a task may have runtime parameters such as; it may finish its execution at best-case (B_i), worst-case (C_i), or actual-case (AET_i) execution times (Fig.2). Later in this paper, we shall often refer AET_i and C_i of a task Ta_i , while running at a particular frequency F , as $AET_i@F$ and $C_i@F$, respectively. We consider that relative deadline of a task is equal to its period ($d_i=T_i$) unless stated otherwise. All tasks are preemptive and support migration as in case of SMP architectures. We consider both synchronous and asynchronous task models. Fig.2 represents graphically all task parameters. Individual utilization of a task is given by $u_i = C_i / T_i$ and cumulative utilization of task set is given by Eq.1.

$$u_{sum}(TS) \stackrel{def}{=} \sum_{T_i \in TS} u_i \quad (1)$$

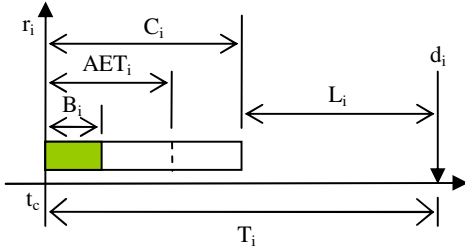


Fig.2: Static & dynamic parameters of a real time periodic task

C. Power and energy model

Power consumption of a processor, represented as a function of speed (S) in variable speed settings, can be decomposed into static and dynamic components which relate to supply voltage (V_{op}), operating frequency (F_{op}), and leakage-current (I_q) through an approximate relation given by eq.2.

$$Pwr(S) = \alpha C_{eff} V_{op}^2 f_{op} + I_q V_{op} \quad (2)$$

Here, α and C_{eff} refers to the switching activity and effective load capacitance, respectively. The first addend in eq.2 corresponds to the amount of dynamically dissipated power caused by switching circuitry and second addend models statically dissipated power caused by leakage current. RT-DVFS techniques mainly deal with dynamic component of power dissipation. We also do not treat the case of processor shutdown at any stage due to idle time and therefore, do not deal with static power dissipation. If a task occupies a processor during an execution interval of $[t_1, t_2]$ then the energy consumed by the processor during this time interval is given by eq.3.

$$E(t_1, t_2) = \int_{t_1}^{t_2} Pwr(S(t)) dt \quad (3)$$

Eq.2 and eq.3 clearly demonstrate that, while processors are functional, striking power/energy savings can be achieved if operating voltage and frequency are reduced simultaneously, at the expense of increased latency. We assume that (V_{op}, F_{op}) can be

scaled over a continuous spectrum between (V_{min}, F_{min}) , values that correspond to minimum processor speed (S_{min}) necessary to keep system functional, and maximum allowed (V_{dd}, F_{ref}) that correspond to maximum speed (S_{max}). Each processor has a *current speed* (\hat{S}) at runtime such that $(S_{min} \leq \hat{S} \leq S_{max})$. We refer to (V_{op}, F_{op}) and processor *speed* interchangeably throughout this paper. We consider that the overhead for changing voltage/speed is not negligible. However, it is incorporated in the worst-case workload of every task.

D. Notations

In the following, we define some notations that are used throughout this paper.

- **Scheduling Event** is the arrival/release, termination, and deadline (where deadline is not equal to period) of a job instance of a task.
- **Canonical schedule** (S^{can}) is a static optimal schedule in which each task instance presents its worst-case workload under any given scheduling algorithm.
- **Practical schedule** (S^{pra}) is an online schedule in which each task instance presents variations to its worst-case workload. These variations are bounded by best-case (B_i) and worst-case (C_i) execution times of a task.
- **Dynamic Slack** is the difference between worst-case workload and actual workload exhibited by a task at runtime. It can be known only when a task terminates its job.

IV. DETERMINISTIC STRETCH-TO-FIT (DSF) TECHNIQUE

Timeliness guarantees for real time applications are provided through static schedulability analysis considering the worst-case workload of application tasks. This is a very conservative analysis because during execution, real time tasks exhibit large variations in their actual execution time (AET) and thereby generate dynamic slack [22]. Before providing the details of our proposed technique, we would like to emphasize that leaving this dynamic slack either unused or slowing down blindly the processor's speed is not a safe approach, especially for hard real time applications.

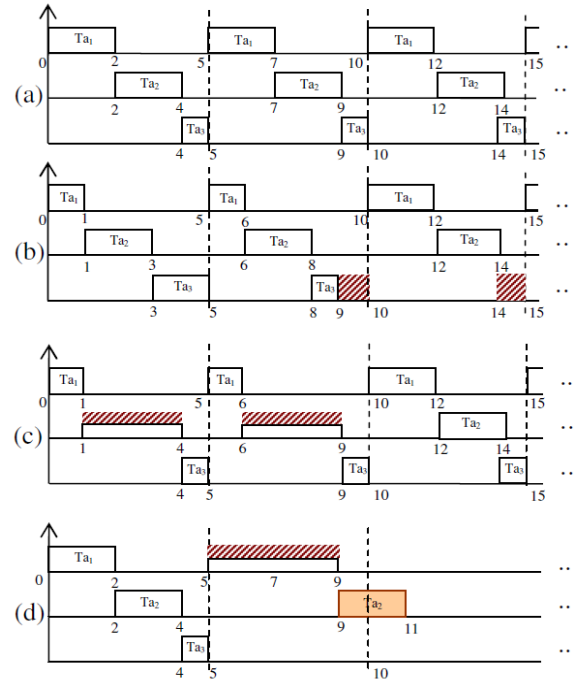


Fig.3: Execution of tasks in canonical and practical schedules

We give a simple example of three tasks being scheduled on single processor using earliest deadline first (EDF) algorithm. Quadruplet values for these tasks are (0,2,5,5), (0,2,5,5), and (0,3,15,15), respectively. The u_{sum} is equal to 1.0. Fig.3 (a) shows the canonical execution of tasks with no task generating dynamic slack. Fig.3 (b) shows that, for first and second job, task Ta_1 finishes early (at time instance 1 and 6, respectively) and generates two units of slack time. Early termination of Ta_1 causes entire schedule to advance by two time units and the slack time is lost eventually at time 9 and 14 in the absence of RT-DVFS. In Fig.3(c), on the other hand, the slack time produced by Ta_1 is immediately consumed by task Ta_2 between time 1-4 and 6-9, respectively, causing no affect on the rest of the schedule in the presence of RT-DVFS. However, Fig.3 (d) depicts another case where, it is not always beneficial to redistribute slack time blindly. In Fig.3 (d), tasks Ta_1 and Ta_2 execute with their worst-case workload but task Ta_3 finishes earlier at time 5, generating two units of slack time. If this slack is immediately consumed by next priority task Ta_1 at time 5, then task Ta_2 will not respect its deadline at time 10. The shaded areas in Fig.3 show the distribution of slack time.

A. Dynamic Slack Reclamation (DSR) algorithm

Our dynamic slack reclamation algorithm is based on detecting the early termination of tasks with respect to their worst-case execution times. Since it is not possible to know a priori the exact amount of actual workload of a running task until it terminates, therefore, DSR algorithm determines the amount of slack only once a task is terminated by comparing worst-case and actual-case workload of terminating task. Based on the amount of available slack, if any, the algorithm determines how much a processor could be slowed down to execute next dispatched (appropriate priority) task such that the dispatched task does not execute *beyond* its worst-case boundary defined in canonical schedule. In a multiprocessor system, producing and keeping a priori (and then following) an entire canonical schedule for a given task set is clearly impractical in the absence of tie-breaking rules and preemptive and migrating task models. To this aim, we produce an online canonical schedule ahead of practical schedule that mimics the canonical execution of tasks. We present our criterion for producing online optimal schedule in section-IV (C). At this stage, we assume that the canonical schedule (S^{can}) for given task set is known a priori. Algorithm-1 in Fig.4 presents the pseudo-code for online slack reclamation algorithm. DSR algorithm exploits the fact that different scheduling events have different impact on an application's schedule. For example, a termination event may produce dynamic slack but does not increase concurrent utilization of platform and therefore, only updates the priorities of remaining ready tasks. A release event, on the other hand, increases concurrent platform utilization and may cause preemptions. Algorithm-1 demonstrates that, for a feasible task set, S^{can} is obtained a priori and all tasks are sorted according to their priority order under concerned scheduling algorithm. Highest priority m -tasks are assigned to m -processors for execution at statically specified maximum frequency (F_{ref}) and dynamic slack is initialized to zero. Note that DSR updates current speed (s)

available for Ta_2 before its worst-case termination in S^{can} shall occur and hence, its execution can be slowed down by $SR = (t_{av}/C_2@F_{ref})$. Fig.5 (c) shows that task Ta_2 does not execute beyond its worst-case boundary ($C_2@F_{ref}$) defined in S^{can} at \hat{s} and therefore, the conservative schedulability analysis for both tasks hold.

B. Online canonical schedule

For single processor systems, a statically constructed canonical schedule (S^{can}) remains the same for all executions if all tasks execute with their worst-case workload. The same is not true for multiprocessor systems, especially in the absence of tie-breaking rules and preemptive and migrating tasks. Thus, it is impractical to produce and keep entire canonical schedule offline. To this aim, we produce an online canonical schedule ahead of practical schedule that mimics the canonical execution of future tasks using the periodicity of tasks. For illustration purpose, we can safely say that a multiprocessor global scheduling algorithm maintains at least three types of sorted task queues: a running tasks' queue (RuTQ), a ready tasks' queue (ReTQ) and a general purpose tasks' queue (TQ) for all non-ready and non-running tasks (Fig.6).

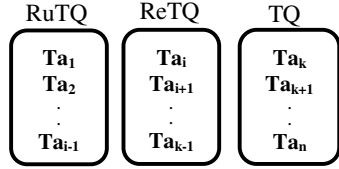


Fig.6: Sorted task queues

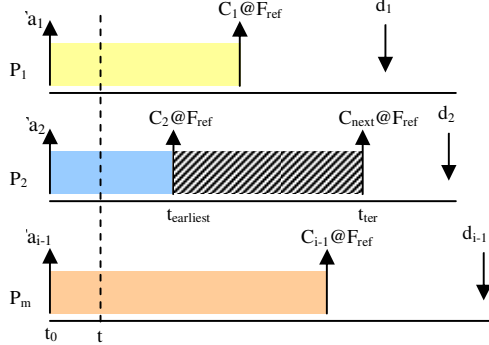


Fig.7: Online canonical schedule ahead of practical schedule Since it is not possible to know a priori the early termination of any task, therefore, in order to construct an online canonical schedule, we make the conservative assumption that currently running tasks will terminate with their worst-case workload. However, if later on a running task terminates early, that does not affect scheduling. Let us consider a multiprocessor system with m processors and n tasks to be scheduled under a global scheduling algorithm (such that $n \geq m$). At any time instance t , there can be at most m -highest priority tasks in RuTQ and the rest of $(n-m)$ tasks should be present in either ReTQ or TQ. From the sorted ReTQ and TQ queues, the next highest priority task (Ta_{next}) can be identified using eq.4 where Ta_i and Ta_k represent the highest priority tasks in ReTQ or TQ respectively. Based on the knowledge of worst-case workload of running tasks at time instance t , we can also identify the earliest possible time instance ($t_{earliest}$) at which task (Ta_{next}) can start execution (eq.5) on any particular processor.

$$Ta_{next} = \text{Max_Pr io_Task} \{Ta_i, Ta_k\} \quad (4)$$

$$t_{earliest} = t + \min_{j=1}^m \{C_j^{rem}\} \quad (5)$$

Fig.7 shows that task Ta_2 running on processor P_2 will terminate earlier than other running tasks using its worst-case workload and hence $t_{earliest}$ for Ta_{next} is previewed to occur on processor P_2 . Similarly, the worst-case termination for Ta_{next} can be known using Eq.6. It is sufficient to determine at most m next priority tasks ahead of practical schedule to construct online canonical schedule in a multiprocessor system.

$$t_{ter} = t_{earliest} + C_{next} \quad (6)$$

C. One Task Extension (OTE) technique

As presented in [6] and [10], one can further slow down processors when there is only one task left in the ready task's queue and its worst-case completion time at \hat{s} does not extend beyond the next scheduling event (next arrival/closest deadline of any task). OTE technique can be used in conjunction with any scheduling policy. In a single processor system, let us suppose that Ta_k is the last remaining ready task in ReTQ at time instance t and the earliest possible release instance of any other task present in TQ is $t_{earliest}$ such that $t_{earliest} > t$. If $(Y = t_{earliest} - t - C_k > 0)$ at \hat{s} then the execution of Ta_k can be further slowed down to consume additional Y time units, implied that the lower bound on speed (S_{min}) is respected. OTE can be extended to multiprocessor systems as shown in algorithm-2 (Fig.8) using the same criterion. If there are at most m tasks left in ready task's queue (ReTQ) then the available execution time for each ready task can be extended up to the earliest time instance representing a scheduling event (deadline of task or next arrival) as shown by lines 2-6. Processor's speed can then be updated using SR (line 8).

Algorithm-2

```

1-Foreach scheduling event
2-  If (number of tasks in ReTQ ≤ m) then
3-    Determine earliest release instance ( $r_{next}$ ) from TQ
4-    Foreach task in ReTQ
5-      If  $C_i \leq r_{next}$  then
6-         $t_{available} \leftarrow \text{Min}(d_i, r_{next})$ 
7-         $SR \leftarrow (t_{available}/C_i)$ 
8-        Update  $\hat{s}$  w.r.t. SR
9-      End if
10-    End
11-  End if
12-End

```

Fig.8: One task extension (OTE) for multiprocessor systems

D. Effect of early completion on schedulability of tasks

Proposition-1: If all tasks in a feasible task set meet their deadlines in canonical schedule then the early termination of some or all tasks using their actual workload does not affect the schedulability of tasks under same scheduling algorithm.

To support our proposition, we consider the analysis of Pillai and Shin in [5] for single processor systems. Consider a feasible task set with priorities ascending towards small task indices such that; $Ta_i > Ta_{i+1} > Ta_{i+2} > \dots > Ta_{n-1} > Ta_n$. Under an optimal scheduling algorithm, none of these tasks shall miss deadline in the presence of higher priority released task. Let us consider that Ta_i and Ta_{i+1} are released at the same time ($r_i = r_{i+1}$). Lower priority task Ta_{i+1} shall meet its deadline while Ta_i and Ta_{i+1} terminate with their worst-case workload as shown by eq.7. Clearly, if the higher priority task Ta_i finishes earlier than its worst-case (generating e_i units of dynamic slack) then the lower priority task Ta_{i+1} will still meet its deadline and there is no effect on its schedulability as shown in Eq.8.

$$\{r_{i+1} + C_{i+1} + C_i\} \leq d_{i+1} \quad (7)$$

$$\{r_{i+1} + C_{i+1} + (C_i - e_i)\} \leq d_{i+1} \quad (8)$$

Generalizing Eq.7 and Eq.8, we can say that no lower priority task Ta_n misses its deadline if some or all higher priority tasks finish earlier than their statically specified worst-case workload as shown by Eq.9.

$$\left\{ r_n + C_n + \sum_{j=1}^{n-1} (C_j - e_j) \right\} \leq d_n \quad \text{for } e \geq 0 \quad (9)$$

For multiprocessor systems, we divide tasks into two subsets. First subset comprises of all tasks present in TQ and the second comprises of all tasks present in $ReTQ$ and $RuTQ$. Let us say that $u_l(TS)$ and $u_r(TS)$ are the cumulative utilizations of first and second subsets respectively. Eq.10 shall always hold for a feasible task set where α_x refers to the schedulability bound of any global scheduling algorithm X.

$$u_{sum}(TS) = \sum_{i=1}^k u_l(TS)_i + \sum_{j=k+1}^n u_r(TS)_j \leq \alpha_x \quad (10)$$

Similar to single-processor case, even if a task terminates earlier than its worst-case workload such that $(C_{i+k} - e) < C_{i+k}$ as shown in eq.11, the task set remains schedulable.

$$\sum_{i=1}^k u_l(TS)_i = C_1/T_1 + C_2/T_2 + \dots + C_{(k-1)}/T_{(k-1)} + (C_k - e_k)/T_k \quad (11)$$

Generalizing Eq.11, such that some or all tasks terminate earlier than their worst-case workload, demonstrates that eq.10 will still hold and the task set remains schedulable with often changed order of task execution.

E. Online Speculation Mechanism (OSM)

Algorithm-3

```

1-Foreach scheduling event
2-   Foreach task i
3-     If task is terminated then
4-       Obtain AET for current instance k of task  $Ta_i$ 
5-       Update AvAET
            $(AvAET)_i \leftarrow \left( \sum_{j=1}^{k-1} (AET)_j + (AET)_k \right) / k \quad \forall k$ 
6-       Assign  $C_i^{specu} \leftarrow (AvAET)_i$ 
7-     Endif
8-   End
9-End

```

Fig.9: Online speculative speed adjustment mechanism (OSM)

In this section, we extend our proposed dynamic slack reclamation algorithm with a *speculation* mechanism for soft real time tasks which often coexist with hard real time tasks. Whenever, under a constant speed, the actual workload of tasks exhibit large variations, starting a task with the assumption that it will execute with its worst-case workload can be too conservative. Instead, making a *speculation* that current instance of a task will most probably execute with actual workload lesser than the worst-case helps in maintaining lower processor speeds. Our online speculative mechanism is based on the *average* execution behavior of soft real time tasks. Through runtime profiling of tasks, we maintain a history of *actual* workload exhibited by each task during all past executions. This actual workload is then averaged to achieve a speculative execution time for future executions. Use of speculative workload helps avoiding a radical change in \hat{s} and improves energy savings. However, this speculative move might shift the task's worst-case completion time to a point later than the one in S^{can} under an actually high workload. If this pessimistic scenario turns out to be true, processor's speed should be increased later to

guarantee feasibility of future tasks. Algorithm-3 (Fig.9) represents our speculative mechanism. We can notice that a separate *speculative* workload (C_i^{specu}) is maintained for all soft real time tasks based on the *average actual execution time* (AvAET) of every task. When a task terminates, the value of AvAET is updated using all previously completed job instances of that task (referred by k) which have occurred until this point in time. In the presence of our speculative mechanism, algorithm-1 does not execute a soft real time task with S_{max} (line 25 of algorithm-1) rather; it uses C_i^{specu} of the task to figure out an appropriate value for $\hat{s} < S_{max}$ and therefore, saves more power and energy.

V. EXPERIMENTAL SETUP & RESULTS

We have performed our simulations using a freeware tool called STORM (Simulation TOol for Real-time Multiprocessor Scheduling) [7]. STORM permits to model target applications in the form of tasks with user-specified characteristics such as periodicity, deadlines, worst-case execution time, data dependency/precedence constraints, release instances, and variations in actual-execution time etc. Furthermore, hardware architecture models can be characterized using temporal and energy consumption parameters of real world systems. We have used the parameters of PXA270 in our simulations which is an XScale® technology-based processor for embedded computing [9]. PXA270 consumes 925-mwatts in running state while 260-mwatts in Idle state. All results on energy consumption characteristics are scaled between 0.0 and 1.0 with respect to the non-optimized maximum values. Supply voltage and frequency is varied on a continuous spectrum for the presented results. We have considered two target applications. Our first application is an H.264 video decoder. We consider H.264 because it is a real world computation extensive application used to produce high quality video stream. Our second application is based on a synthetic task set. Use of synthetic application allows us to introduce user-specified variations, such as variations in the number of tasks and cumulative utilization etc, in the characteristics of tasks and analyze performance for potentially different target applications. We simulate both applications using EDF global scheduling algorithm on a multiprocessor platform.

A. H.264 video decoder application

H.264 is a high compression rate algorithm relying on several efficient techniques to extract spatial (within a frame) and temporal dependencies (between frames). We have considered simulating decoder end of H.264 standard using a pipelined model of tasks. Table-1 presents the task set of H.264 video decoder which is modelled into periodic tasks. Inter-task temporal dependencies are extracted by asynchronous execution of tasks. Cumulative resource utilization of task set is $u_{sum}(TS) = 1.90$, i.e. EDF global scheduling algorithm theoretically requires at least three processors ($m=3$) to schedule this task set without any deadline miss. A single frame of H.264 decoder is 150ms long with presented task set. We have performed our simulations for fifty consecutive frames.

Table-1: Pipelined model of H.264 video decoder tasks

Task Name	Release (r _i)	WCET (C _i)	Period (T _i)	Deadline (d _i)	Priority level
TG	0	2	15	15	1
SI	15	3	15	15	1
RE-1	30	17	30	30	2
RE-2	30	17	30	30	2
RE-F	60	8	30	30	2
LI	90	3	30	30	2
RA	120	2	30	30	2

B. Synthetic task set application

Our synthetic application is composed of six synchronous and independent tasks (table-2). We have varied the number of tasks and the cumulative utilization of the task set to evaluate the performance of our DSF technique.

C. Simulation results

We present our simulation results and analysis by introducing variations in BCET/WCET ratio, the number of tasks, and the cumulative utilization of target application.

Table-2: Synthetic application tasks

Task Name	Release (r_i)	WCET (C_i)	Period/ Deadline ($T_i=d_i$)	Priority level
Ta ₁	0	30	80	1
Ta ₂	0	40	100	2
Ta ₃	0	50	120	3
Ta ₄	0	50	150	4
Ta ₅	0	80	200	5
Ta ₆	0	100	250	6

1) Effect of BCET/WCET ratio

Theoretically, RT-DVFS algorithms work on the assumption that at runtime, a real time application exhibits large variations in its actual-case and worst-case workload. Our experimental results confirm these assumptions. Fig.10 and Fig.11 depict the normalized energy consumption of our target applications as a function of BCET/WCET ratio with fixed cumulative utilization of $u_{sum}=1.90$ and number of tasks ($n=6$). We have varied BCET/WCET ratio from 20% to 100% with AET having a uniform probability distribution function as suggested in [6]. We make the following observations on our simulation results.

1. When BCET/WCET ratio = 1, i.e. all tasks exhibit their worst-case execution time, no change in the energy profile occurs with or without DSF technique which is obvious to understand as there is no dynamic slack generated by any task, which could be reclaimed. However, for BCET/WCET ratio < 1, remarkable variations in energy consumption can be seen in both, H.264 decoder (Fig.10) and synthetic task set applications (Fig.11).
2. Simulation results in Fig.10 and Fig.11 show that the rate of decrease in energy consumption is larger for BCET/WCET ratio >0.5. For BCET/WCET ratios <0.5, energy savings continue to increase but the relative percentage of improvement is rather small. This is because the expected workload of target application starts converging to 50% with increasing ratio. We observe up to 44.85% savings for BCET/WCET ratio >0.5 and 42.50% savings for BCET/WCET ratio <0.5.
3. We observe that our dynamic slack reclamation algorithm alone can achieve up to 53% energy savings for BCET/WCET ratio less than or equal to 0.5. With online speculation mechanism (OSM) and one-task extension (OTE) techniques in place, we achieve up to 56% energy savings. We predict that the contribution of OSM shall be much more significant for soft real time tasks and that of OTE for aperiodic tasks.

2) Effect of number of tasks

To examine the effect of the number of tasks on the energy profile of an application, we have performed simulations by doubling and tripling the number of tasks for fixed value of BCET/WCET ratio (=0.6) and cumulative utilization ($U_{sum} = 2.324$). Results are depicted in Fig.12. We observe that, increasing the number of tasks slightly increases the energy savings (up to 8.86%). This is because the main determinant of variations in energy consumption is the actual workload and with increased number of tasks, there are potentially more opportunities for tasks to generate dynamic slack and therefore, more possibilities to

reclaim energy by frequency and voltage scaling. However, Fig.12 shows that doubling or tripling the number of tasks changes the energy consumption by a small margin.

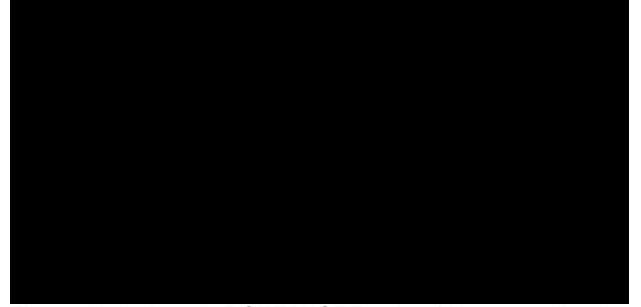


Fig.10: Variations in BCET/WCET ratio of H.264 application

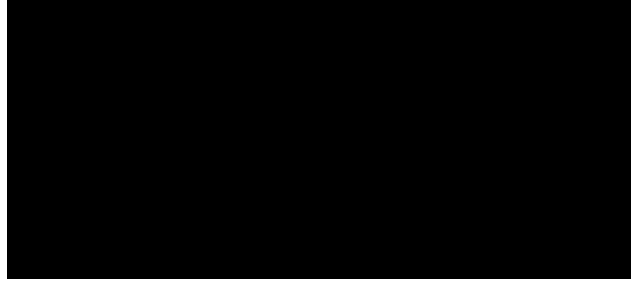


Fig.11: Variations in BCET/WCET ratio of synthetic task set

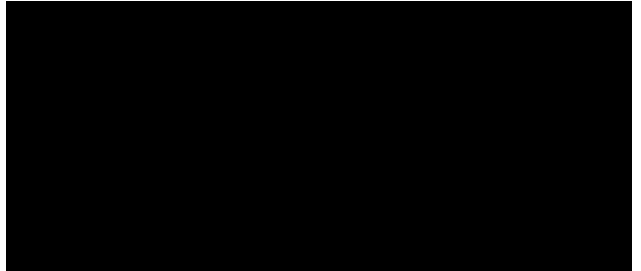


Fig.12: Variations in the number of application tasks

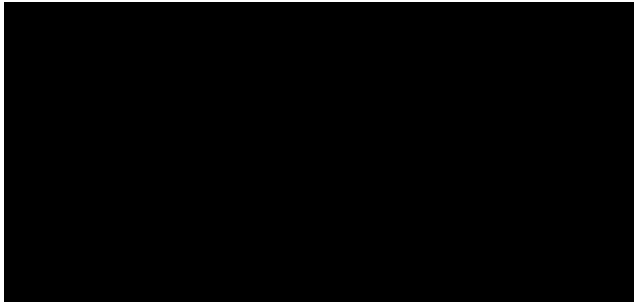


Fig.13: Variations in cumulative utilization

3) Effect of cumulative utilization

We have varied the cumulative utilization of tasks for our synthetic applications between 50% (lower workload) and 100% (maximum workload) for fixed values of BCET/WCET ratio (=0.6) and the number of tasks ($n=6$). Fig.13 shows our simulation results. We observe that the energy trends plotted in Fig.13 resembles closely with each other. This is due to the fact that BCET/WCET ratio is fixed. In addition, we observe that the difference in energy savings for various percentage of cumulative utilization is more or less similar. Our remark on this observation is that, lesser workload inherently favours more energy savings on fixed platform architectures and the opportunities for RT-DVFS algorithms really

lie in variations in the actual workload. In the presence of static speed optimization mechanism, the variation in cumulative utilization does not affect energy consumption profile of processors as they are always charged to maximum or 100%.

VI. CONCLUSIONS AND FUTURE PERSPECTIVE

In this paper, we have addressed the problem of energy-efficient scheduling for real time applications which are intended to be executed on multiprocessor systems. We have proposed a novel RT-DVFS technique, called Deterministic Stretch-to-Fit (DSF) technique, which is based on inter-task voltage & frequency scaling and it comprises of an online dynamic slack reclamation algorithm (DSR), a speculative speed adjustment mechanism (OSM), and a one-task extension schemes (OTE). DSF is a generic technique in the sense that if a feasible schedule for a real time target application exists under worst-case workload using (optimal or non-optimal) global scheduling algorithms, the same schedule can be reproduced (using actual workload) with less power/energy consumption. We have shown that if dynamic slack is reclaimed in such a way that no task finishes its execution later than the completion time in canonical schedule then we can guarantee all deadlines with less energy consumption. Moreover, we have shown that producing and keeping entire canonical schedule offline is impractical in multiprocessor systems. We have proposed a scheme to produce an online schedule ahead of practical schedule to mimic the canonical execution of tasks. Our simulation results show that DSR algorithm alone gives up to 53% gains on energy consumption. Furthermore, OSM and OTE schemes give additional gains on peak power, battery life, and overall energy consumption, reaching a theoretical low-bound on the scalable frequency and voltage. Notice that our reported results are based on the assumption that the operating frequency and supply voltage of processor can be scaled *continuously*. In the existing processors, however, these parameters can only be changed in predefined discrete steps. DSF technique can be easily adapted to discrete operating frequencies and supply voltages by adapting to the nearest values of the estimated ones. However, we predict that the gains will be slightly marginalized because it would not be possible to exploit all available dynamic slack in certain cases. As a future perspective, we plan to implement our proposed DSF technique with multiprocessor partitioned scheduling algorithms and integrate it into a system-level energy optimization framework which will permit to apply either RT-DVFS or DPM techniques (or both) depending on the characteristics of target application.

REFERENCES

- [1] NAVET, N., GAUJAL, B. Ordonnancement temps réel et minimisation de la consommation d'énergie, Chapter-4 in System Temps Réel –Volume 2, June 2006.
- [2] MOSSÉ D., AYDIN H., CHILDERS B., MELHEM R., 2000. Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications, Workshop on Compiler & Operating Sys for Low-Power.
- [3] SHIN Y., CHOI K. System-Level Power Optimization of Embedded Systems, Rapport n° SNU-EE-TR-2000-3, School of Electrical Engineering, Seoul National University, 2000.
- [4] SHIN Y., CHOI K., SAKURAI T. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors, International Conference on Computer Aided Design, p. 365-368, 2000.
- [5] PILLAI P., SHIN K. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, ACM Symposium on Operating Systems Principles, p. 89-102, 2001.
- [6] AYDIN H., MELHEM R., MOSSÉ D., MEJIA-ALVAREZ P. Power-Aware Scheduling for Periodic Real-Time Tasks, IEEE Transactions on Computers, vol. 53, n°5, p. 584-600, 2004.
- [7] <http://storm.rts-software.org>, 2009.
- [8] <http://www.transmeta.com>
- [9] <http://www.intel.com>.
- [10] SHIN, Y., CHOI, K. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In Proceedings of 36th Design Automation Conference (DAC '99), pp. 134-139, 1999.
- [11] WEISER, M., WELCH, B., DEMERS, A., SHENKER, S. Scheduling for Reduced CPU Energy. In the Proceedings of First Symposium on Operating Systems Design and Implementation, 1994.
- [12] ZHU, D., MELHEM, R., CHILDERS, B. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. In the proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS), 2001.
- [13] ZHU, D., ABOUGHAZALEH, N., MOSSÉ, D., MELHEM, R. Power aware scheduling for AND/OR graphs in multi-processor real time systems. In the proceedings of the International Conference on Parallel Processing (ICPP'02), 2002.
- [14] AYDIN, H., YANG, Q. Energy-aware partitioning for multiprocessor real-time systems. In the proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03), Workshop on Parallel and Distributed Real-Time Systems, 2003.
- [15] GRUIAN, F. System-Level Design Methods for Low-Energy Architectures Containing Variable Voltage Processors. In the proceedings of Workshop Power-Aware Computing Systems, Nov. 2000.
- [16] SHRIVER, P.M., GOKHALE, M.B., BRILES, S.D., KANG, D., CAI, M., MCCABE, K., CRAGO, S.P., SUH, J. A Power-Aware, Satellite-Based Parallel Signal Processing Scheme. chapter 13, Power Aware Computing, Plenum/Kluwer Publishers, pp. 243-259, 2002.
- [17] YANG, P., WONG, C., MARCHAL, P., CATTHOOR, F. DESMET, D., VERKEST, D., LAUWEREINS, R. Energy-Aware Runtime Scheduling for Embedded-Multiprocessor SoCs. IEEE Design and Test of Computers, vol. 18, no. 5, pp. 46-58, 2001.
- [18] KRISHNA, C.M., LEE, Y.H. Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems. In the proceedings of Sixth IEEE Real-Time Technology and Applications Symp., May 2000.
- [19] HSU, C.H., KREMER, U., HSIAO, M. Compiler-Directed Dynamic Frequency and Voltage Scheduling. In the proceedings of Workshop Power-Aware Computing Systems, Nov. 2000.
- [20] MOSSE', D., AYDIN, H., CHILDERS, B. MELHEM, R. Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications. In the proceedings of Workshop Compiler and OS for Low Power, Oct. 2000.
- [21] POUWELSE, J., LANGENDOEN, K., SIPS, H. Dynamic Voltage Scaling on a Low-Power Microprocessor. In the proceedings of Seventh Int'l Conf. Mobile Computing and Networking (MOBICOM), July 2001.
- [22] ERNST, R., YE, W. Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification. In the proceedings of Int'l Conf. Computer-Aided Design (ICCAD '97) pp. 598-604, 1997.
- [23] NAMGOANG, W., YU, M., MEG, T. A High Efficiency Variable-Voltage CMOS Dynamic DC-DC Switching Regulator. In the proceedings of IEEE Int'l Solid-State Circuits Conf., pp. 380-391, 1997.
- [24] S. IRANI, SANDEEP SHUKLA, and RAJESH GUPTA. Online Strategies for Dynamic Power Management in Systems with Multiple Power-Saving States. ACM Transactions on Embedded Computing Systems, Vol. 2, No. 3, August 2003, Pages 325-346.
- [25] BHATTI, M. K., FAROOQ, M., BELLEUDY, C., AUGUIN, M., MBAREK, O. Assertive Dynamic Power Management (AsDPM) Strategy for Globally Scheduled RT Multiprocessor Systems. In the proceedings of 19th International Workshop, PATMOS, Delft, The Netherlands, 2009.
- [26] IRANI, S., SHUKLA, S., and GUPTA, R. Competitive analysis of dynamic power management strategies for systems with multiple power saving states. In Proceedings of the Design Automation and Test Europe Conference, 2002.
- [27] LOPEZ, J. M., DIAZ, J. L., GARCIA, M., GARCIA, D. F. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In Proc. 12th Euromicro Conf. Real-Time Systems, pages 25-33, 2000.