RenPar'19 / SympA'13 / CFSE'7
Toulouse, France, du 9 au 11 Septembre 2009

Controlling Energy Profile of RT Multiprocessor Systems by Anticipating Workload at Runtime[1]

Muhammad Khurram Bhatti, Muhammad Farooq, Cécile Belleudy, Michel Auguin

University of Nice-Sophia Antipolis –CNRS
LEAT, 250-Rue Albert Einstein, Bâtiment-4,
06560-Valbonne, France
{bhatti, muhammad, belleudy, auguin}@unice.fr

Abstract
Emerging trends in applications with the requirement of considerable computational performance and decreasing time-to-market have urged the need of multiprocessor systems. With the increase in number of processors there is an increased demand to efficiently control the energy and power budget of such embedded systems as well. Power management in embedded computing systems is achieved by actively changing the power consumption profile of the system by putting its components into such power/energy states which are sufficient to meet functionality requirements. Dynamic Power Management (DPM) strategies attempt to make decisions related to the choice of such states based on the available information at runtime. These strategies exploit the inherently present idleness (if any) in the application's behavior which is a priori unknown or non-stationary. This paper presents a novel DPM strategy called **As**sertive **D**ynamic **P**ower **M**anagement (AsDPM) for real time applications. It is based on the idle time extraction from application's behavior and clustering to make appropriate decision for state-transition of processors in a multiprocessor real time system. Experimental results show that conventional DPM approaches often yield suboptimal, if not incorrect, performance in the presence of real time constraints whereas, the AsDPM strategy gives better energy consumption performance under the same constraints by 10.40%. Also, it reduces the number of state transitions by 74.85% and 59.76% for EDF and LLF scheduling policies respectively.
Keywords –Energy efficient scheduling, idle time intervals, real time and multiprocessor systems, under-utilization, dynamic power management (DPM).

1. Introduction

Energy efficiency in modern day real time embedded systems has become a concerning issue, especially with the increased architecture complexity of multiprocessor system-on-chip (MP-SoC). Energy efficiency is achieved in both, mono and multiprocessor systems, by actively changing the power consumption profile of the system. This change in power consumption profile is performed in two ways: a) by putting system's components into such power/energy states which are sufficient to meet functionality requirements b) by dynamically changing the operating frequency and supply voltage of active components while guaranteeing functionality. These two approaches are referred as DPM (dynamic power management) and DVFS (dynamic voltage and frequency scaling) strategies in literature. Relative to DPM strategies, DVFS is recent in embedded real time systems because of the evolution of processor technology permitting to have multiple operating frequencies. However, the evolved processor technology significantly increases the impact of static-power consumption as well but DVFS techniques work only with the dynamic component of power consumption. Moreover, the number of frequency switching points supported by the modern-day processors is limited and frequency can not be reduced beyond a certain limit otherwise, the static power consumption significantly increases. Hence, the use of DPM strategies is still very interesting. This paper focuses on the DPM (dynamic power management) strategies. A DPM strategy exploits the inherently present idleness (if any) in the application's behavior which is a priori unknown or non-stationary. For this reason, a DPM strategy should be adaptive to the changes in application behavior. Whenever a DPM strategy (usually under the control of a scheduler) puts a component in an energy efficient state, bringing it

[1] This work is supported by project PHERMA bearing reference ANR-06-ARFU06-003 and Higher Education Commission (HEC) of Pakistan.

back to the active state may require additional energy and/or latency to service an incoming task. The input to a DPM strategy is an idle interval and the outcome is a decision whether to put a component (only processor in our case) into an energy efficient state or not? Since in an online schedule, the exact length of the idle interval is not known a priori therefore, there are several issues in coming to such decision intelligently based on this partial information. For instance, immediate shutdown—that is, shutdown as soon as an idle period is detected—may not save overall energy if the idle period is too short and the costs (temporal and energy) to recover a processor are greater than the gains in low power state. On the other hand, waiting too long to switch to low power state may not achieve the best possible energy reductions. Thus, there exists a need for effective (and efficient) decision making procedures to manage power consumption. [1, 2, 3, 4, 5, and 6]. We propose in this paper a novel DPM approach, called **As**sertive **D**ynamic **P**ower **M**anagement (AsDPM), to make such intelligent decisions on multiprocessor systems under the control of a global scheduler. AsDPM differs from the existing approaches in the way it exploits the inherent idleness in a schedule. The strategy is detailed in section-IV. However, it is important to note at this stage that AsDPM is not a scheduling algorithm itself. Rather it's a strategy to make an already schedulable task set (under a global scheduling policy) more energy efficient. Moreover, the scope of this paper is limited to the discussion of how AsDPM deals with the idle time extraction and clustering and it does not elaborate how a choice is made between various energy-efficient states supported by platform components. The remainder of this paper is organized as follows. In Section-II, we briefly describe some related research. In Section-III, we formally define the system model and some additional useful notations. In Section-IV we provide our strategy in detail and discuss its efficiency. In Section-V, we provide our experimental setup and simulation environment. In Section-VI, we discuss our results and in the Section-VII, we conclude our paper.

2. Related Work

From the design point of view of energy-efficient real time systems, DPM strategies are well studied and the most implemented for long time. The studied techniques include predictive strategies [10], stochastic-modeling-based strategies [11], session clustering strategies [12], online strategies [13], and adaptive learning-based strategies [4]. Also Lu et al. present in [12] a quantitative comparison of various existing strategies. Authors in [1, 2, and 9] classify DPM strategies into two main categories: a) *predictive schemes* and b) *stochastic optimum control schemes*. Predictive schemes attempt to predict the timing of the occurrence of future idle interval in the system and schedule state-transition (usually to a single lower power state) based on these predictions. The chief characteristic of stochastic approaches is the construction (and validation) of a mathematical model of the system that lends itself to a formulation of a stochastic optimization problem. Then strategies to guide the system power profile are devised that achieve the most power savings in presence of the uncertainty related to system inputs. Most of the mentioned strategies are predictive in the sense that they use a sequence of past idle period lengths to predict the length of future idle interval. Also, they make assumptions on the application's characteristics and the probabilistic distribution of idle time intervals. While several useful and practical techniques have been developed using the predictive and stochastic optimum control schemes, it is difficult to develop bounds on the quality of the results without extensive simulations and/or model justification [1]. Our approach differs with the existing approaches in the sense that it is not predictive. However, it is also based on the adaptive learning of past idleness behavior of the application but the learned information is not used for the prediction of future idle intervals rather; it is used to maintain a (temporal) threshold value to make appropriate decisions for state-transition. Moreover, we don't make any assumptions on the application characteristics and the probabilistic distribution of idle intervals. Our strategy extracts the idleness in a schedule and clusters it on some processors to better exploit it by putting them into more energy-efficient states.

3. System Model and Notations

Application Model –We consider a finite set of n real time, independent, and recurring tasks referred by $\tau = \{\tau_1, ..., \tau_n\}$. Each task in τ is characterized by at least a quadruplet (r_i, C_i, d_i, T_i) where; r_i refers to the arrival or release time, C_i refers to the worst-case execution requirement, d_i refers to the relative deadline, and T_i refers to the periodicity of task *i*. Moreover, a task may have runtime parameters like for example; it

may finish its execution at its best-case (B_i), worst-case (C_i) or actual-case (AET_i) execution times. We consider that the relative deadline of a task is equal to its period (d_i=T_i) unless stated otherwise. The task set is pre-emptive and supports task migration (as in case of SMP multiprocessor architectures). Tasks in the system may release at different instances but once released they are recurrent, i.e., they are asynchronous. Fig.1 represents all task parameters. The utilization u_i of a task i is the ratio of its execution requirement to its period, i.e. $u_i = C_i / T_i$ and the total utilization $u_{sum}(\tau)$ of a task set is given as follows:

$$u_{sum}(\tau) \overset{def}{=} \sum_{\tau_i \in \tau} u_i$$

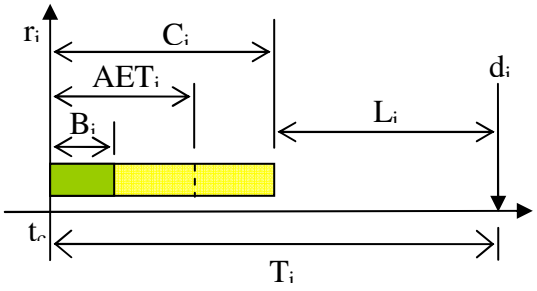


Fig.1: Task's parameters

We consider in this document a multiprocessor platform composed of m identical processors upon which a real time task set τ is scheduled. Processor set is referred as $\pi = \{\pi_1, ..., \pi_m\}$ and they all run at reference (nominal) frequency F_{ref}. Task set τ is scheduled under a global scheduling algorithm.

Power and Energy Model –We consider that a processor can be in one of the k-power states denoted by {S_1… S_k}. The power consumed while the processor is in a particular state j is denoted by α_j and the time it remains in this state is denoted by ψ_j. The states are ordered such that $\alpha_j > \alpha_k$ as long as $j < k$. Thus, state S_1 is the *active* state with the highest power consumption. Let the transition energy between two states j and k be given by γ_{j-k} and transition time between any two states S_j and S_k by ψ_{j-k}. Usually, the energy needed and time spent to go from a higher (power) state to a lower (power) state is negligible, whereas the converse is not true. Thus, we simplify the model by considering only the time and power necessary to power up the system, i.e., from any state j to the *active* state only. Furthermore, we consider that a processor can only do a transition to the active state when powering up and never transition to an intermediate higher-powered state. As a result, we only need the time and total power consumed in transitioning up from each state j to the *active* state S_1. The power consumed in transitioning from any state j to the *active* state (S_1) is denoted by β_j.

4. Assertive Dynamic Power Management Strategy

Dynamic power management is the process of controlling power consumption profile of a processor by efficiently changing its operational state. In real time systems, an energy efficient strategy is constrained by the hard timing requirements to ensure deadline guarantees. For real time applications, being modeled as periodic and independent tasks, the worst-case (offline) under-utilization can be known a priori based on the cumulative utilization $u_{sum}(\tau)$. During execution, this under-utilization appears in the form of idle intervals. These idle intervals are non-stationary and occur randomly (at runtime) with a variable length on different processors. A conventional DPM strategy exploits these idle intervals whenever they arrive on processor(s), i.e. it waits for the idleness to occur during execution before deciding about changing the energy profile. AsDPM, on the other hand, extracts this distributed idleness and clusters it on some processor(s). Doing so permits to increase the length of idle interval and allows some k-processors to be in energy efficient state during these intervals while other (m-k) processors execute the existing and upcoming jobs of ready tasks. This strategy is based on clairvoyance as it relies on runtime information available about the current and anticipated (future) workload of the system to determine the platform resource requirement between any two scheduling events. AsDPM does not intervene with the priority order specified for tasks by the

scheduling algorithm. Moreover, we make no assumptions about the form of the distribution governing the length of idle intervals which allows our strategy to adapt at runtime to the non-stationary idle intervals in various application models. We consider Earliest Deadline First (EDF) global in detail to demonstrate how AsDPM works in conjunction with the scheduler. To compare results, we will use AsDPM in conjunction with Least Laxity First (LLF) global in experiments (only) as well. As stated earlier as well, AsDPM is not a scheduling algorithm itself. Rather it's a technique which works as a subordinate of global scheduler to make an already schedulable task set more energy efficient. For example, if a particular task set is not schedulable under EDF-global, applying AsDPM provides no guarantees that it will become schedulable. However, a schedulable task set under EDF-global remains schedulable under EDF-global with AsDPM and gives better performance from the point of view of energy consumption.

EDF global –Earliest Deadline First (EDF) scheduling policy was originally defined for mono-processor systems and it is an optimal policy in that context. However, when the definition of monoprocessor system extends in a straightforward manner into identical multiprocessor systems, the definition of EDF as a multiprocessor scheduling policy is not the same. EDF uses, for execution at each instant in time, the jobs (of tasks) which have the smallest deadline. The utilization guarantees for a periodic task set with implicit deadlines under EDF or any other static-priority multiprocessor scheduling algorithm cannot be higher than $(m+1)/2$ for an m-processor platform [7, 8]. We preserve the definition of EDF as global scheduling policy for identical multiprocessor systems except that a released job may be deferred (delayed) from execution unless its urgency level is critical.

Approach –AsDPM determines the minimum platform resources (number of active processors working at nominal frequency F_{ref}) needed to fulfill the execution requirement of released jobs at runtime. The strategy uses the notion of anticipated laxity of tasks to avoid undue pessimism and still ensuring deadline guarantees. Deadline guarantees are provided for all ready tasks between any two scheduling events. AsDPM takes into account the anticipated laxities of all released tasks and analyzes that if the tasks are scheduled according to the given priority order (by scheduler) then is it possible to delay the execution of some low priority tasks in order to use the same number of processor(s) which are currently executing higher priority task(s). If so, then these low priority tasks are deferred till the next scheduling event and remaining processors are either put in low power state (if they were already in active state) or at least no further processor is activated. Otherwise, if a task cannot be deferred then a processor is put into active state (S_1) to meet the execution requirement. Notice that AsDPM is an admission control technique which decides when a ready task enters in running tasks' queue of the scheduler. Without this admission control, all ready tasks are executed as soon as there are enough computing resources available in the system leading to poor possibilities of putting some processors in low power state. The decision related to state-transition is taken based on the Worst-Case Execution Time (WCET) requirement of released tasks. However, we will demonstrate in later sections that AsDPM is optimist and takes full advantage in case where tasks execute with their Actual Execution Time (AET).

Nomenclature –Standard events on which a scheduler is called are the release and termination of a task (release of a task also represents deadline event in case where d_i=T_i). These events will be referred as Standard Scheduling Events (SSE) in the later discussion. AsDPM is an admission control technique and it lets a ready task to enter in running tasks' queue based on a test called Laxity Bottom Test (LBT). Laxity is a run time parameter of a task showing its urgency to execution. LBT is performed at every SSE for all ready tasks. AsDPM uses a notion of anticipated laxity to perform LBT. The anticipated laxity (l_i^c) of a task is a measure of how long it can be delayed from execution in the presence of higher priority tasks (than itself) on a certain processor. Anticipated laxity of a task does not represent its real laxity (L_i) rather; it is always smaller than or equal to the real laxity of a task ($l_i^c \leq L_i$). A negative value of anticipated laxity for task i is interpreted as if the scheduler continues to execute the released tasks with the same platform resources (number of currently active processors) then at least this job of task i will miss its deadline in future and therefore, additional processor(s) must be activated to meet execution requirements. AsDPM also uses the notion of different task queues (fig.2):

i. **Task Queue (TQ):** This queue contains all tasks at any time instance. Tasks contained by all other derived queues are always members of TQ.

ii. **Released Task Queue (ReTQ):** This queue contains tasks which are released but not executing on any processor. When released, a task is immediately put in this queue.

iii. **Running Task Queue (RuTQ):** This queue contains tasks which are currently running on active processors (m_{act}). When a task τ_i is executing on a processor π_j, it is represented as $\pi_j[\tau_i]$.

iv. **Deferred Task Queue (DeTQ):** This queue contains tasks which were already released on last scheduling event but deferred from execution despite ($m_{act} < m$). When a task τ_i is deferred from execution, it is virtually assigned (for execution in future) to an active processor π_j and represented as $\tau_i[\pi_j]$. This virtual task assignment is valid up to the next SSE and performed to avoid pessimism in computing. Let's refer the tasks being in DeTQ at any SSE as a sub set τ^d.

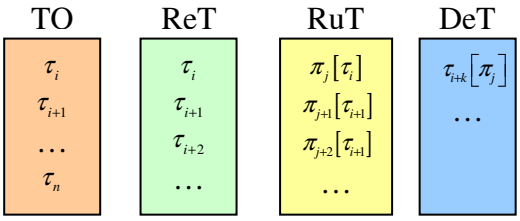


Fig.2: Different task queues

All task queues are sorted according to the priority order. AsDPM does not intervene in the priority order specified by the scheduler so it may be an important information for the reader that global scheduler works with ReTQ and RuTQ only. DeTQ and TQ are invisible or transparent for the scheduler. However, AsDPM visualizes all task queues (Fig. 3).

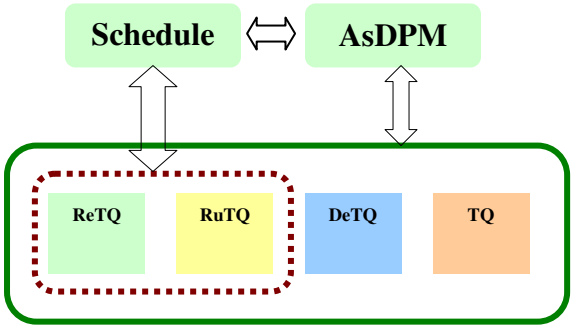


Fig .3: Scheduler's view of task queues

Working Principle – The working principle of AsDPM is based on the clairvoyance from the point of view of platform resource requirements. At any SSE, the runtime information about the current workload of scheduler (remaining execution requirement of all released tasks) is available. Based on this information, AsDPM anticipates whether it is possible to defer low priority tasks or not. Consequently, it decides the minimum number of processors sufficient to provide deadline guarantees of all released tasks till the next SSE. In other words, AsDPM ensures the availability of minimum platform resource adaptively to execute the application at runtime while respecting temporal constraints. Since AsDPM strategy works in conjunction with a global scheduler therefore, we initially consider EDF as our global scheduling policy to elaborate the working principles of AsDPM. Upon the occurrence of an SSE, all released tasks are immediately put in ReTQ and scheduler is called. The scheduler sorts ReTQ according to EDF priority order (task with smaller deadline has higher priority). Algorithm-1 demonstrates the working mechanism of AsDPM in conjunction with EDF-scheduler. From the description in Agorithm-1, it is evident that AsDPM strategy controls the admission of tasks into RuTQ through LBT. In the following we explain how LBT is performed. Every task has a runtime parameter called laxity (L_i) which is a measure of its urgency to execution relative to its deadline, e.g. a task with L_i=0 is the most urgent in the sense that if it is not executed at this stage, it will miss its deadline. Laxity of task τ_i at time instance t_c is given by Eq.1.

$$L_i = d_i - \left(t_c + C_i^{rem}\right) \tag{1}$$

Here, C_i^{rem} refers to the remaining execution requirement of task τ_i . AsDPM uses the notion of anticipated laxity (l_i^c) which is the offered laxity by a task in the presence of all higher priority tasks (than itself). Anticipated laxity is always smaller than or equal to the real laxity of a task ($l_i^c \leq L_i$).

Algorithm-1: Working mechanism of AsDPM

@ Standard Scheduling Event (SSE) occurred @ t_c

- *Sort* ReTQ *by scheduler's priority order*
- *Move the highest priority task from* ReTQ *to* RuTQ
- *Perform LBT for all tasks remaining in* ReTQ

```
for (i=1; i≥ size (ReTQ); i++)
    if l_i^c ≥ 0 then
        Move τ_i to DeTQ
    else
        m_act = m_act+1
        Move back all tasks from DeTQ to ReTQ
        Move highest priority task from ReTQ to RuTQ
        break
    end if
```

- *Reiterate LBT until* ReTQ *becomes empty*
- *Execute tasks present in* RuTQ

Since we consider a multiprocessor system and there may be multiple higher priority tasks currently executing on different processors therefore, the remaining execution requirement (C_i^{rem}) of these tasks can be equal or less than their worst-case execution requirement (C_i). Hence, the computation of anticipated laxity must take into account the particular processor on which it is computed. Moreover, a higher priority task may not be currently executing but present in DeTQ. Thus, while computing the anticipated laxity of a task, all higher priority tasks (than itself) from RuTQ and DeTQ must be considered. Eq.2 shows how this anticipated laxity is computed.

$$l_i^c = d_i - \left(t_c + C_c^{rem} + C_i^{rem} + \sum_{\tau_k \in \tau^d, \tau_k[\pi_j]} C_k^{rem} \right) \tag{2}$$

Here, C_c^{rem} refers to the remaining execution requirement of the higher priority task which is currently executing on a processor π_j, C_i^{rem} is the remaining execution requirement of the task τ_i itself and C_k^{rem} is the remaining execution requirement of a higher priority task which is already in DeTQ. From the system model, we know that whenever a task is deferred, it is virtually associated to a particular processor by the notation of $\tau_i[\pi_j]$ for future execution. This virtual task-processor affinity helps in reducing the pessimism from computations. Eq.2 shows that the sum of the remaining execution requirement of only those tasks in τ^d shall be considered which have their affinity for the same processor π_j while computing anticipated laxity. This is because once a task is deferred with a virtual affinity to a particular processor, there is no need to consider the execution requirement (C_k^{rem}) of all other tasks present in τ^d. Hence, the contributing factor to the computation of anticipated laxity l_i^c of task τ_i on processor π_j is only the collective C_k^{rem} of those tasks having the same processor affinity. Doing so helps in avoiding repetitive consideration of the same (deferred) tasks in computing l_i^c on different processors and eventually the value of l_i^c is not pessimist. The LBT checks the value of l_i^c before moving a task to either RuTQ or DeTQ. If the value of l_i^c is positive ($l_i^c \geq 0$) then it is interpreted as if the system continues to run with same number of active processors until next SSE, this task will meet its deadline and therefore, it can be deferred. Otherwise, resource augmentation is required to satisfy the workload requirements. To do so, another processor is activated ($m_{act} = m_{act}+1$), the

highest priority task from resorted ReTQ is moved to RuTQ for execution on this newly activated processor, and LBT is reiterated on the remaining tasks in ReTQ until it becomes empty. The number of iterations is of the linear complexity order *(g+1)* where, *g* refers to the number of *activations* of processors being performed at current SSE. Between any two SSE, all tasks are assured their deadline guarantees. Example-1 helps in understanding LBT.

Example-1: Let's consider three tasks to schedule. τ_1, τ_2, and τ_3 are in descending EDF priority order. Fig.4 depicts that, at time instance t_c, the anticipated laxity of task τ_2 in the presence of τ_1 and that of τ_3 in the presence of τ_1 and τ_2 is positive on processor π. Hence, τ_2 and τ_3 can meet their deadlines after the execution of τ_1 and therefore, can be deferred. Tasks τ_2 and τ_3 are shown dotted because this is the scheduler's view at current SSE with their virtual assignment on π which may change upon the arrival of next SSE.

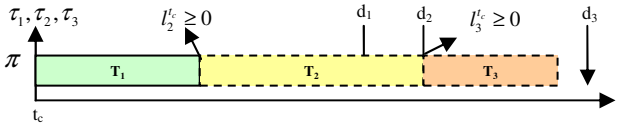


Fig.4: Task deferring and scheduler's view at current SSE.

The strength of AsDPM strategy lays in the extraction and clustering of idle time intervals from a schedule. In example-2, we demonstrate how effectively AsDPM can extract this idleness from the schedule.

Example-2: Let's consider three periodic independent and synchronous tasks (τ_1, τ_2, and τ_3) to be scheduled on two processors (π_1 and π_2). The quadruplet values for task τ_1, τ_2, and τ_3 are $(0,3,8,8)$, $(0,6,10,10)$ and $(0,4,16,16)$ respectively. Fig.5 (a) represents the EDF global schedule for these tasks. It can be noticed that, on run time, EDF generates some random idle intervals (gray dashed area) because of the under-utilization present in the schedule. Fig.5 (b) shows the same schedule again where AsDPM has extracted these idle time intervals from processor π_1 and clustered them on processor π_2. These clustered idle time intervals can be used to put π_2 in an energy efficient state for long time while π_1 continues to execute released jobs at the nominal frequency.

Transition Penalties–Transition costs in terms of energy consumed (γ_{j-k}) and time taken (ψ_{j-k}) for recovery from any state *j* to the *active* state (S_1) is not negligible in the presence of temporal constraints in real time systems. A DPM strategy should take into account the contribution of transition costs while computing the energy consumed in a particular state (Eq.3). To model these penalties in AsDPM strategy, we consider that the energy consumed in transition from a more energy-efficient state S_j is greater than the energy consumed in transition from a lesser energy-efficient state S_k ($\gamma_{j-1} > \gamma_{k-1}$), and the same is true for temporal penalty ($\psi_{j-1} > \psi_{k-1}$).

$$\text{Energy (per state)} = \alpha_j \times \psi_j + h * \gamma_{j-1} \tag{3}$$

Here, 'h' is the total number of transitions performed from state *j* to *active* state. Power consumed in transition from any state *j* to *active* state S_1 is given by Eq.4.

$$\beta_{j-1} = \frac{\gamma_{j-1}}{\psi_{j-1}} \tag{4}$$

In AsDPM, thanks to the anticipated laxity, we can know a priori that the number of processors currently in *active* state may not be sufficient to meet future deadlines for ready tasks and therefore, a processor should be immediately recovered from an energy-efficient state *j* to the *active* state (S_1) to share the system's workload. However, a processor will take ψ_{j-1} time units to recover. Since, the anticipated laxity of a task does not represent its real urgency; a newly activated processor has time to recover before executing a ready task. In other words, the recovery time ψ_{j-1} of a processor can be masked with the remaining execution requirement C_c^{rem} of currently executing task(s). However, there are cases in which this time masking is not so evident.

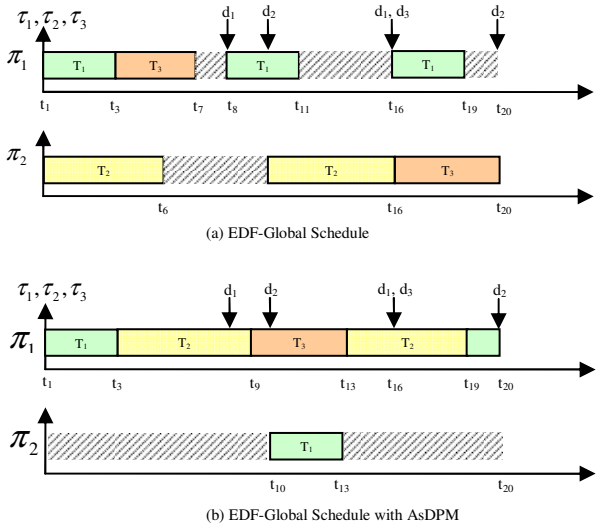


Fig.5: Idle time extraction and clustering using AsDPM

Case-1: *Processor's recovery time from an energy-efficient state j is greater than the offered laxity of a task* τ_i *at its release (*$\psi_{j-1} > L_i$ *).*

At an SSE, A situation may arise in which a task is released such that its anticipated laxity goes negative while there is no intermediate priority task between itself and the currently running tasks, i.e. $\sum_{\tau_i \in \tau^d, \tau_i[\pi_j]} C_i^{rem} = 0$ .In this case another processor must be immediately put in active state. But if $C_c^{rem} < \psi_{j-1}$ and $L_i < \psi_{j-1}$ are true then recovery time can not be masked with the currently executing task and the released task will miss its deadline in any case. To avoid such a case, the anticipated laxity must be computed while considering the maxima between C_c^{rem} and ψ_{j-1} (Eq.5).

$$C_c^{rem-\max} = \max\left(C_c^{rem}, \psi_{i-1}\right) \tag{5}$$

Hence, Eq.2 transforms into Eq.6.

$$l_i^c = d_i - \left(t_c + C_c^{rem-\max} + C_i^{rem} + \sum_{\tau_k \in \tau^d, \tau_k[\pi_j]} C_k^{rem} \right) \tag{6}$$

Moreover, we constraint our task model such that the static laxity of any task shall always be greater then or equal to the processor's recovery time from the deepest energy-efficient state ($L_i \geq \psi_{j-1}$). We will discuss how to remove this constraint in future work.

Case-2: *The highest priority task in ReTQ offers real (online) laxity <*ψ_{j-1} *at any SSE.*

Let's consider that, on the previous SSE, certain tasks were selected to execute while others were deferred and then during execution an intermediate priority task τ_x (having priority lesser than all currently executing tasks and some deferred tasks) is released. Release of τ_x is another SSE and AsDPM anticipates the laxities of all tasks according to the new execution requirement of tasks. Suppose that an already deferred task τ_y (having lesser priority then τ_x) does not pass LBT. A request to recover another processor (from energy-efficient state) will be sent by AsDPM. The scheduler will select the next highest priority task from ReTQ to be executed on newly activated processor. But the real laxity the highest priority deferred task is lesser than its static laxity at runtime (as this task was deferred on previous SSE and real laxity continues to decrease) so there is no guarantee that its real laxity is still greater than the recovery time of new processor (ψ_{j-1}) or not. To avoid this situation, the real laxity of highest priority deferred task is compared with ψ_{j-1} and if the resultant value is positive then this task is assigned to newly activate processor. Otherwise, next priority deferred task is checked for the same. Fig.6 (a) shows that τ_2 is the highest priority deferred task

and τ_4 is the lowest priority deferred task. Both tasks offer positive anticipated laxities at time t_c. Fig.6 (b) shows that the intermediate priority task τ_3 is released at time instance 't_{c+1}'. We can notice that τ_4 does not pass LBT at the release of task τ_3 and transition time ψ_{j-1} of π_2 is greater than the real laxity of τ_2. Therefore, τ_3 is placed on π_2 and τ_2 remains on π_1. Doing so helps to avoid priority inversion of tasks and the deadlines are still respected.

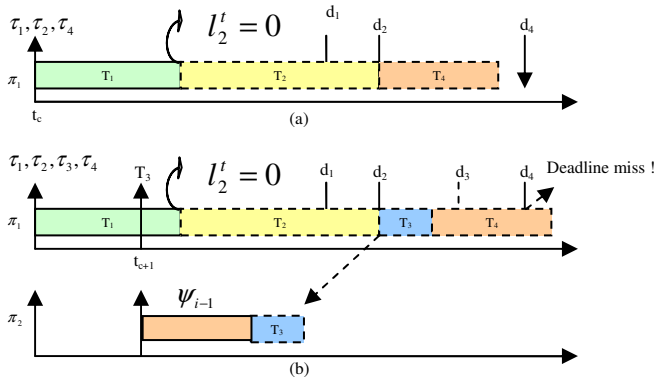


Fig.6: Impact of processor's activation time on the schedule

Energy-efficient state-transitions –AsDPM strategy is adaptive to the evolution of workload during runtime, i.e. at every scheduling event, it analyzes the current execution requirement of all released tasks and consequently decides on the number of processor sufficient to satisfy this demand while no task shall miss its deadline. AsDPM does not have prior knowledge of the impact of upcoming SSE, i.e. it does not know whether the execution requirement will increase or decrease on the next SSE. This uncertainty may cause AsDPM to become potentially over-optimist in the sense that it over-extracts the idleness in the beginning and require more processors to meet tasks' deadlines later. This over-extraction of idleness may have two drawbacks: a) the next scheduling event is too close and state-transition cost is higher then gains achieved by transition b) some processors remain too long in low-power state such that the deferred task(s) miss deadlines. This shortcoming can be overcome if lossy idleness is not extracted from the schedule. The scheduler is permitted to execute some tasks during lossy idle intervals. An important question which may arise here is that, how to decide that an idle interval will result in a lossy state-switching? To answer this question, we introduce lemma-1.

Lemma-1: *AsDPM does not extract lossy idleness from the schedule and therefore, a state-transition is always energy-efficient.*

The proof that every state transition is energy efficient is based on two parameters: a) knowledge of instances where the platform resource demand may increase and b) adaptive learning of the idleness behavior of an application. To elaborate further, we distinguish between the types of SSEs. A release event (of a task) is different from a termination event (of a task) such that, whenever a task terminates, it never increases the execution requirement of the system and consequently the platform resources. However, a release event increases the execution requirement of the system which may or may not result in an increased demand for platform resource. A release event will at least require the same number of active processors (m_{act}) as on the previous scheduling event. Therefore, knowledge of the occurrence of next release instance is helpful to determine the length of idle interval. To do so, AsDPM strategy uses the information available in sorted Task Queue, ReTQ and RuTQ. From system model we know that tasks are periodic and d_i=T_i which means the next deadline of a task is also a reference to the release of its next job. Hence, it is sufficient to compare the deadlines of top-most tasks in TQ, ReTQ, and RuTQ to determine next release instance. Eq.4 refers to the next release instance.

$$r^{next} = \min\left(d_q, d_u, d_e\right) \tag{4}$$

Here, d_q is deadline of the top-most task in TQ, d_u is deadline of the task on top of RuTQ, and d_e is deadline of the top-most task in ReTQ. This r^{next} refers to the earliest event which can potentially cause an

augmentation in the platform resource. Based on the arrival pattern of r^{next}, a threshold on the minimum (non-lossy) length of idle interval is learned. On the termination event of a task, AsDPM determines the next r^{next}, compares the length of time with learned threshold, and decides whether to transition a processor to energy-efficient state or not. Hence, AsDPM never perform a transition which is not energy-efficient.

5. Experimental Setup

Our experimental setup is based on simulations being performed using a freeware tool STORM[2] (Simulation TOol for Real-time Multiprocessor Scheduling) [14]. The simulator permits to model software applications in terms of periodic (dependent or independent) tasks. A multiprocessor architecture (with real world architectural models) can be instantiated upon which these tasks execute using various global scheduling policies. An important aspect of our simulation environment is that we use the abstract models of existing architectural parameters, i.e. the application is constrained with real temporal and power consumption parameters. For the simulation results presented in this paper, we have used the parameters of Intel's XScale® technology-based processor PXA270 (Table-2) for embedded computing with the parameters presented in Table-2. All timing characteristics are given in milliseconds. Our example task set[3] is composed of six periodic tasks ($n=6$) with the quadruplet values of Table-1. All results are provided while considering state-transitions between S_1 and S_4. The total utilization is given by $u_{sum}(\tau) = \sum_{\tau_i \in \tau} u_i = 2.20$. Since this task set is schedulable on three processors under EDF ($m=3$). Therefore, under EDF-AsDPM, we simulate this task set on maximum three processors for simulation time equal to the hyper-period of tasks. The hyper-period of task set is 1200 time units and all tasks execute with their worst-case execution requirement unless stated otherwise.

6. Simulation Results and Performance Analysis

We have observed various performance parameters to determine the impact of AsDPM on the schedules generated by EDF-global and LLF-global. The extraction and clustering of idle intervals from various processors is quite evident. Table-5 shows the percentage of time each processor is busy over the entire simulation time. Results show that AsDPM has extracted the idleness (or under-utilization) from π_1 and π_2 to cluster it on π_3 (Table-5). Another important amelioration in performance is that AsDPM reduces the number of state-transitions (Table-3) on all processors collectively by 74.85% and 59.76% for EDF-global and LLF-global respectively. Referring to Table-3 again, the example task set consumes 10.38% less average power and 10.40% less energy while scheduled under EDF-AsDPM. Similarly, it consumes 9.71% less average power and 9.70% less total energy over the hyper-period while scheduled under LLF-AsDPM. To elaborate further our simulation environment, we have provided real simulation traces of tasks on every processor. Fig.8 (a) presents a simulation trace showing the distribution of idleness of all tasks on three processors for simulation time 50ms under EDF and Fig.8 (b) shows the simulation traces generated with EDF-AsDPM for the same duration. Fig.7 shows that number of active processors during execution remain less than theoretical maximum (m=3) for most of time. At runtime, it is often the case that tasks do not execute always with their worst-case execution time (WCET). As mentioned earlier in section-IV, AsDPM is optimistic and takes advantage of this characteristic at runtime. To demonstrate this feature, we have simulated the same task set with the actual execution time (AET) of every task. Upon every new release of a task, we randomly generate a value of AET which ranges between its WCET and not less than 60% of WCET. Table-4 shows the results obtained. We notice that the gains on energy and average power for EDF increase up to 15.86% and 15.79% respectively and for LLF they increase up to 17.58% and 17.60% respectively. Moreover, the number of state-transitions reduces by 74.53% 66.35% for EDF and LLF respectively.

[2] STORM is developed by IRCCyN in the PHERMA project.

[3] Simulations were performed with multiple example task sets. However, only one example is provided here to demonstrate performance ameliorations.

Table-1: Task set

Task	r_i	C_i	d_i	T_i
T_1	0	6	16	16
T_2	0	8	20	20
T_3	10	8	24	24
T_4	10	8	30	30
T_5	16	16	40	40
T_6	20	20	50	50

Table-2: Parameters of *XScale*® PXA270 processor.

State	α* (mw)	ψ *(ms)
S_1 (Running)	925	
S_2 (Idle)	260	0.001
S_3 (Standby)	1.70	11.28
S_4 (Sleep)	0.16	136

* Values at F_{op}=624 MHz and V_{dd}=1.55Volts

Table-3: System-level performance with WCET of tasks

Performance	EDF	EDF-AsDPM	LLF	LLF-AsDPM
Average power(W)	2.224	1.993	2.224	2.008
Total Energy(J)	2.671	2.393	2.671	2.412
Number of state-transitions	167	42	169	68

Table-4: System level performance with AET of tasks

Performance	EDF	EDF-AsDPM	LLF	LLF-AsDPM
Average power(W)	1.906	1.605	1.917	1.580
Total Energy(J)	2.289	1.926	2.303	1.898
Number of state-transitions	216	55	211	71

Table-5: Occupancy of processors with WCET of tasks

Processor occupancy (%)	EDF	EDF-AsDPM	LLF	LLF-AsDPM
π_1	86	99	83	99
π_2	73	94	75	90
π_3	56	20	58	26

7. Conclusions

This paper presented a novel strategy for dynamic power management in real time multiprocessor systems called **As**sertive **D**ynamic **P**ower **M**anagement (AsDPM). AsDPM determines the minimum platform resources (the number of active processors working at nominal frequency F_{ref}) needed to fulfill the execution requirement of released jobs at runtime. The strategy extracts the distributed idleness from some of the processors and clusters it on the others. Doing so permits to increase the length of idle interval and allows some *k*-processors to be in energy efficient state during these intervals while other (*m-k*) processors execute the existing and upcoming jobs of ready tasks. AsDPM relies on runtime information available about the current and anticipated (future) workload of the system to determine the platform resource requirement between any two scheduling events. The strategy does not intervene with priority order specified for tasks by the scheduling algorithm and it makes no assumptions about the form of the distribution governing the

length of idle intervals, which allows our strategy to adapt at runtime to the non-stationary idle intervals in various application models. Experimental results show that using AsDPM in conjunction with EDF and LLF as global scheduling policies optimizes system's energy consumption by 10.40% and 9.70% respectively. In addition, AsDPM reduces the number of state-transitions by 74.85% and 59.76% while used in conjunction with EDF and LLF as global scheduling policies respectively. In future work, we have previewed to deal with other task models like aperiodic and sporadic tasks. We are currently working to relax the constraint of $L_i \geq \psi_{j-1}$ discussed in section-IV. Since in periodic task set, the information related to release events in known a priori therefore, programming some virtual events in advance to activate processor(s) earlier for those tasks having this constraint ($L_i < \psi_{j-1}$) will be helpful in avoiding deadline misses. Moreover, making AsDPM strategy intelligent enough to choose the most appropriate energy-state for clustered idle time intervals at runtime is an interesting aspect to be introduced in future.

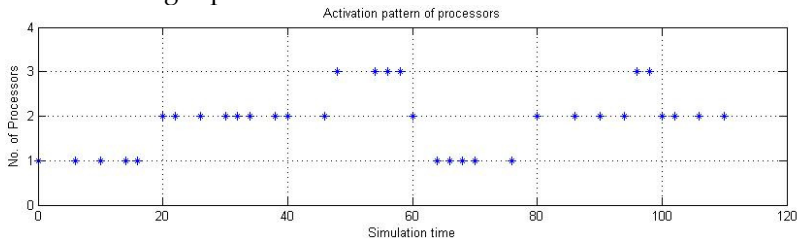
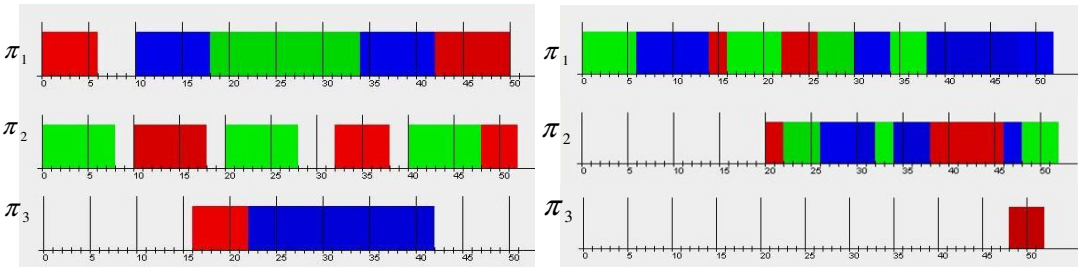


Fig.7: Evolution of the number of active processors during execution (measurements are taken at SSEs)



(a) (b)

Fig.8: Idle time distribution in EDF and EDF-AsDPM schedules

References

[1] 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.

[2] BENINI, L. and DE MICHELI, G. Dynamic Power Management: Design Techniques and CAD Tools. Kluwer Dordrecht, 1998.

[3] BENINI, L., DE MICHELI, G., and MACII, E. Designing low-power circuits: Practical recipes. IEEE Circuits and Systems Magazine 1, 1 (March), 6–25, 2001.

[4] CHUNG, E. Y., BENINI, L., BOGLIOLO, A., and DE MICHELI, G. Dynamic power management for non-stationary service requests. In Proceedings of the Design Automation and Test Europe 1999.

[5] 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.

[6] SHUKLA, S. and GUPTA, R. A model checking approach to evaluating system level power management for embedded systems. In Proceedings of IEEE Workshop on High Level Design Validation and Test (HLDVT01) (November). IEEE Press, New York, 2001.

[7] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In Proc. 22nd IEEE Real-Time Systems Symposium, pages 193–202, London, UK, Dec. 2001.

[8] J. M. Lopez, J. L. Diaz, M. Garcia, and D. F. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In Proc. 12th Euromicro Conf. Real-Time Systems, pages 25–33, 2000.

[9] BENINI, L., BOGLIOLO, A., AND DE MICHELI, G. A survey of design techniques for systemlevel dynamic power management. IEEE Transactions on Very Large Scale Integration (TVLSI) Systems 8, 3, 299–316, 2000.

[10] HWANG, C.-H., ALLEN, C., AND WU, H. A predictive system shutdown method for energy saving of event-driven computation. In Proceedings of the IEEE/ACM International Conference on Computer Aided Design. 28–32, 1996.

[11] BENINI, L., BOGLIOLO, A., PALEOLOGO, G., AND DEMICHELI, G. Policy optimization for dynamic power management. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18, 6, 813–833, 1999.

[12] LU, Y. AND DE MICHELI, G. Adaptive hard disk power management on personal computers. In Proceedings of the Great Lakes Symposium on VLSI, 1999.

[13] RAMANATHAN, D., IRANI, S., AND GUPTA, R. K. Latency effects of system level power management algorithms. In Proceedings of the IEEE International Conference on Computer-Aided Design, 2000.

[14] http://storm.rts-software.org