Global Journal of Computer Science and Technology
Volume 11 Issue 10 Version 1.0 May 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
ISSN: 0975-4172 & Print ISSN: 0975-4350

A Three Phase Scheduling for System Energy Minimization of Weakly Hard Real Time Systems

By Smriti Agrawal, Rama Shankar, Ranvijay

Abstract- This paper aims to present a three phase scheduling algorithm that offers lesser energy consumption for weakly hard real time systems modeled with (𝕞𝕞, 𝕜𝕜) constraint. The weakly hard real time system consists of a DVS processor (frequency dependent) and peripheral devices (frequency independent) components. The energy minimization is done in three phase taking into account the preemption overhead. The first phase partitions the jobs into mandatory and optional while assigning processor speed ensuring the feasibility of the task set. The second phase proposes a greedy based preemption control technique which reduces the energy consumption due to preemption. While the third phase refines the feasible schedule received from the second phase by two methods, namely speed adjustment and delayed start. The proposed speed adjustment assigns optimal speed to each job whereas fragmented idle slots are accumulated to provide better opportunity to switch the component into sleep state by delayed start strategy as a result leads to energy saving. The simulation results and examples illustrate that our approach can effectively reduce the overall system energy consumption (especially for systems with higher utilizations) while guaranteeing the (𝕞𝕞, 𝕜𝕜) at the same time.

Keywords: *Dynamic power down, Dynamic voltage scaling, (𝕞𝕞, 𝕜𝕜) model, Preemption Control, Scheduling, Weakly hard real time system.*

GJCST Classification: *J.7*

Strictly as per the compliance and regulations of:

A Three Phase Scheduling for System Energy Minimization of Weakly Hard Real Time Systems

Smriti Agrawal[α], Rama Shankar[Ω], Ranvijay[β]

Abstract- This paper aims to present a three phase scheduling algorithm that offers lesser energy consumption for weakly hard real time systems modeled with $(\mathbb{m},\mathbb{k})$ constraint. The weakly hard real time system consists of a DVS processor (frequency dependent) and peripheral devices (frequency independent) components. The energy minimization is done in three phase taking into account the preemption overhead. The first phase partitions the jobs into mandatory and optional while assigning processor speed ensuring the feasibility of the task set. The second phase proposes a greedy based preemption control technique which reduces the energy consumption due to preemption. While the third phase refines the feasible schedule received from the second phase by two methods, namely speed adjustment and delayed start. The proposed speed adjustment assigns optimal speed to each job whereas fragmented idle slots are accumulated to provide better opportunity to switch the component into sleep state by delayed start strategy as a result leads to energy saving. The simulation results and examples illustrate that our approach can effectively reduce the overall system energy consumption (especially for systems with higher utilizations) while guaranteeing the $(\mathbb{m},\mathbb{k})$ at the same time.

Keywords: Dynamic power down, Dynamic voltage scaling, $(\mathbb{m},\mathbb{k})$ *model, Preemption Control, Scheduling, Weakly hard real time system.*

I. Introduction

Real time applications are usually composed of set of tasks that interact with each other by exchanging messages. These tasks and their corresponding messages are often invoked repeatedly and are required to complete their services by respective deadlines. Examples of such applications include process control automated manufacturing system and delivery of audio/video frames in multimedia [1]. In process control automated manufacturing system finishing beyond deadline can have a catastrophic effect whereas it may be annoying but acceptable without much loss in case of multimedia applications. An application with catastrophic effect is defined as hard real time whereas degraded performance application is soft real time in nature. Besides these hard and soft deadlines, multimedia application such as video conferencing is being referred to as weakly hard real time where missing of some tasks to complete by deadlines degrade the quality of result however, result is acceptable. For example, in real time transmission of digitized motion video; a source (e.g., a video camera) generates a stream of video frames at a rate of say 30 frame/sec from which at least 24 frames/sec are needed to visualize the movement of the image [17]. When transmitting such frames if sufficient processing power and network bandwidth are available then a high quality video (receiving 30 frames/sec at destination) can be projected whereas degraded but acceptable quality of image is received. In case at least 24 frames/sec reach at the destination within deadline then desired quality is received. For weakly hard real time systems the assurance of minimum acceptable quality result is attained by imprecise concept [17, 18] or by $(\mathbb{m},\mathbb{k})$ model [11]. In imprecise concept a frame has to be received at destination (may be full or portion of it) while a partially received frame is considered as dropped frame in $(\mathbb{m},\mathbb{k})$. That is, all frames are required to be received for imprecise computation whereas certain frames may be dropped to maintain the minimum quality in $(\mathbb{m},\mathbb{k})$ constraints. To ensure a deterministic quality of service (QoS) to such systems, Hamdaoui and Ramanathan [1] used the $(\mathbb{m},\mathbb{k})$ model in which, out of $\mathbb{k}$ consecutive task instances any $\mathbb{m}$ instances must meet their respective deadlines. The $(\mathbb{m},\mathbb{k})$ model scatters the effect of $\mathbb{m}$ deadline misses over a window of $\mathbb{k}$ which is different from accepting low miss rate in which a series of frames may be lost in a burst load leading to intolerant behavior in terms of missing a portion. Besides guaranteeing for QoS in terms of $(\mathbb{m},\mathbb{k})$ designer of real time system has to take care of minimization of energy especially for portable devices.

Energy-aware computing has been realized as one of the key area for research in real time systems [20]. Energy-driven scheduling algorithms have been developed to reduce system's energy consumption while satisfying the timing constraints [2, 3, 4, 5, 6, 19, 20, 21, 22, 25] are applicable for system having frequency dependent component (speed of the system varies with variation in its operating frequency) as resource. They will be able to reduce energy for system having frequency dependent components only. Besides frequency dependent component many systems have frequency independent components such as memory where above energy-driven voltage scheduling algorithms are inadequate.

For the systems having frequency dependent component energy consumption decreases with

About[α]- Department of IT, JB Institute of Engineering and Technology, Hyderabad, India.

About[Ω]- Department of CSE, Motilal Nehru National Institute of Technology, Allahabad India.

About[β]-Department of CSE, Motilal Nehru National Institute of Technology, Allahabad India.

reduction in operating frequency and vice-versa. The energy consumption may increase while frequency is being reduced for the system having both frequency dependent as well as independent components. This is because on reducing the frequency, components which are frequency independent may be forced to be active for longer duration leading to more energy consumption. Authors [7, 8, 9, 10] revealed that the frequency dependent component (processor core) consumes around 30% of total energy while frequency independent (memory and peripherals devices) account for the remaining 70% of energy consumption. Thus, the energy consumption of the frequency independent components plays a crucial role in overall energy consumption of a system. Group of researcher [6, 28, 29, 32] are focused for minimization of system energy (energy required by frequency dependent and independent component) rather than minimization of processor energy only. We use the term frequency dependent component to refer a processor and frequency independent for memory or peripheral devices. The three common techniques used for minimization of system energy are dynamic voltage scaling (DVS), dynamic power down (DPD) and Preemption control (PC) which will be discussed in the following subsection.

Dynamic Voltage Scaling (DVS), is based on adjusting the processor voltage and frequency on-the-fly [12, 13] as energy requirement depends on operating frequency as well as voltage. The DVS attempts to reduce the processor speed to the extent it is possible, to obtain reduction in energy consumption. The speed of a frequency dependent component is said to be reduced if it is either operating at lower voltage or frequency. The task execution time increases with the reduction in processor speed leading to the following consequences:

- a release may miss its deadline while it is feasible at higher speed.
- the longer execution time will be able to decrease the energy consumption of the processor whereas the system energy may be increased
- frequency independent components remain active for longer time and increase the energy consumption.
- longer execution time implies more losses in energy due to leakage current [44].

However, the task execution times do not always scale linearly with the processor speed [13, 14, 15, 16, 23, 26] because system may have some components (memory and peripheral devices) which do not scale with the operating frequency. Thus, DVS may not be efficient (further reduction in the speed would increase the energy consumption) when the system energy is considered. To solve this problem, authors [27, 29, 30, 31] suggested a lower bound (critical speed which balanced the energy consumption between the processor and peripheral devices to minimize the system energy) on the processor speed to avoid the negative impact of the DVS. Niu and Quan [11] used a combined static/dynamic partitioning strategy for $(\mathbb{m},\mathbb{k})$ model to reduce the processor energy and are not efficient for system energy. Beside the DVS energy minimization approach authors [35, 36] suggested to switch off the system (power down) rather than scale down the speed to reduce the energy requirement which is discussed briefly in next subsection.

Dynamic Power Down (DPD) is switching to sleep mode (least power mode) of the unused components since the workload is not constant at all times. Although leaving a component (frequency dependent or independent) in idle/active state consumes power but switching to sleep mode too often may also be counter productive due to heavy context switching overheads. Thus, the DPD technique strives to balance the active and the sleeping time of the components.

Authors [32, 34, 35] used DPD to switch the processor and the peripheral devices into sleep mode based on threshold (minimum time for which the component may sleep for positive energy saving) value to save energy for both hard and soft real time systems. The Niu and Quan [36] proposed a DPD based scheduling method to reduce the system energy consumption for weakly hard real-time systems with $(\mathbb{m},\mathbb{k})$ constraints. The reduction in energy consumption achieved by the DPD technique would increase with the enlargement of the idle slot length. The increment in the length of the idle slot can be achieved by the preemption control technique which is discussed in the following sub-section.

Preemption Control (PC) is allowing a lower priority job to continue execution even when a higher priority job is ready such that none miss their deadline. When a job starts execution on the processor then the associated devices are switched to active state in which they remain till it completes. Thus, if a lower priority job is preempted by the higher priority job then the associated components remain active and consume energy for the time for which the job is preempted. This extra consumption in the energy can be reduced by delaying the higher priority job if possible and completing the lower priority job in the meanwhile (laxity of the higher one). Moreover, each time a job is preempted the context of the job needs to be saved and to be restored when it resumes. This context saving and retrieval would incur an overhead both in terms of time and energy. Thus, reducing number of preemptions reduces the response time of the job and undue energy dissipations due to preemption overhead, longer response time. Agrawal et. al. [29] proposed a preemption control technique where the lower priority job is forced to execute at higher speed levels and complete before the arrival of a higher priority one. The authors themselves say that such a policy may not always lead to energy saving performance.

It is observed if only DPD is applied on a system then based on the threshold the components would be allowed to switch into sleep state and gain the energy reduction. Although, increasing the length or accumulating the idle slots further reduces the energy by DPD; DPD technique itself does not suggest any method to do so. While DVS would lower the assigned speed to each job and increase its execution time which in turn increases its response time. An increment in response time of a job not only increases the energy consumption by the associated components which remain active for longer time but also due to additional preemption which may occur. On the other hand, preemption control at the assigned speed may not be able to reduce the response time and/or number of preemptions. To address the shortcoming of each (DVS, DPD and PC) and to enhance the overall reduction in system energy consumption we suggest a judicious combination of all the above techniques.

Table 1: Symbol Table

$\mathbb{e}_{p,i}$	Computation required by the frequency dependent components of task τ_i
$\mathbb{e}_{d,i}$	Computation required by the frequency independent components of task τ_i
rel_i^j	Release time of a job τ_i^j, i.e., $rel_i^j = j * p_i$
D_i^j	Absolute deadline of a job τ_i^j, i.e., $D_i^j = j * p_i + d_i$
ft_i^j	Finish time of a job τ_i^j
s_{ci}	Critical speed of the processor for the task τ_i
s_{ai}	Speed of the processor assigned to the task τ_i
s_{ai}^j	Speed of the processor assigned to the job τ_i^j
a_i^j	Frequency independent component a_i is associated with task τ_j
$E_{dslp,i}^j$	Energy consumed per unit time by the device a_i associated with task τ_j in sleep state
$E_{dact,i}^j$	Energy consumed per unit time by the device a_i associated with task τ_j in active state
thd_i	DPD threshold of the device a_i
E_{pidle}	Energy consumed per unit time by the processor in the idle state
E_{pslp}	Energy consumed per unit time by the processor in the sleep state
E_{pi}	Energy consumed per unit time by the processor when running at a speed s_i ($E_{pi} = Cs_i^3$ where C is constant)
thp	DPD threshold of the processor
L	MK_hyperperiod
$\mathfrak{R}$	Preemption Overhead is context switching time required when a higher priority preempts a lower priority task
$E_{\mathfrak{R}}$	Energy consumed during each preemption

The length of the idle slot can be enhanced by selecting better speed level for DVS (suggested in third phase) or reducing the response time by PC (suggested in second phase) or delaying the execution of a job (suggested in third phase). The priorities are assigned based on the earliest deadline first (EDF) policy in which the job whose absolute deadline is lower has higher priority. The number of preemptions for different jobs of a task may vary as the earliest deadline first scheduling is dynamic at task level and arrival of mandatory jobs depends on the partitioning strategy. Thus, a job level DVS view would increase its efficiency (suggested in third phase). On the other hand, increasing the speed of few jobs (selected based on the greedy technique suggested in phase-2) could reduce energy consumbed by lower priority job with longer execution as well as preemption overhead. Recently, two groups of researcher Agrawal et. al. [29] and Niu and Quan [37] have used a two phase approach for system energy minimization for weakly hard real time system with $(\mathbb{m},\mathbb{k})$ constraints. The authors have suggested a combination of DVS, DPD and PC techniques however, neither have they taken into the account the preemption overhead nor do they balance the effects of the three techniques.

In this paper we aim to minimize the system energy consumption for weakly hard real time system modeled with $(\mathbb{m},\mathbb{k})$ constraints using a fine balance of DVS, DPD and PC. The reduction in energy consumption is achieved at both, task as well as job level for which we adopt a three phase approach. In the first phase the task level view of the system is taken. The feasibility to each task in the set is ensured keeping in account the preemption overhead. While the second and third phase adopts job level view. A greedy based preemption control technique is proposed at the job level in the second phase. It is further refined in the third phase by adjusting the speed assigned to a job and accumulation of idle slots by delayed start to effectively balance the three approaches.

The rest of the paper is organized as follows; the next section provides a system model followed by section III which presents our new approach along with algorithm. The simulation results are enlisted in section IV whereas section V concludes the work.

II. System Model

This paper aims to minimize the system energy consumption for a system having independent periodic task set $T = \{\tau_1, \tau_2, \tau_3 \ldots \tau_n\}$ that assures minimum QoS defined by $(\mathbb{m},\mathbb{k})$. The priority of a job is assigned based on the earliest deadline first policy. The system consists of two types of components namely, frequency dependent (processor) and frequency independent (memory and peripheral devices). The following considerations are made:

1. The frequency independent components are represented by set $A = \{a_1, a_2, a_3 \ldots a_N\}$ where a_i represents a memory or peripheral device. The power management policies reported in [29, 37, 39] used only two states (active and sleeping) for a frequency independent component and there is no

recourse conflicts. Same consideration is taken in this work.

2. The frequency dependent components (DVS processor) can operate at $\mathcal{N}+1$ discrete voltage levels, i.e., $V=\{v_{slp}, v_1, v_2, v_3 \ldots v_{\mathcal{N}}\}$ where each voltage level is associated with a corresponding speed from the set $S=\{s_{slp}, s_1, s_2, s_3 \ldots s_{\mathcal{N}}\}$.The speed s_1 is the lowest operating speed level measure at voltage v_1 whereas maximum speed $s_{\mathcal{N}}$ at the voltage level $\mathrm{v}_{\mathcal{N}}$. A processor can lie in one of the three possible states namely active, idle and sleep. In the active state the processor can run at any of the speed levels between s_1 to $s_{\mathcal{N}}$, while in the idle state and sleep state it will function at speed s_1 and s_{slp} repectively.
3. Each task $\tau_i \in T$ has attributes $< e_i(s_j), p_i, d_i, \mathbb{m}_i, \mathbb{k}_i >$ where $e_i(s_j)$, p_i and d_i are the computation time at the speed s_j, period and relative deadline respectively. We assume that the task relative deadline is conservative [38] i.e. $d_i \leq p_i$ which is same as considered in [29, 37]. Beside these temporal characteristics minimum QoS requirement is represented by a pair of integers $(\mathbb{m}_i, \mathbb{k}_i)$, such that out of $\mathbb{k}_i$ consecutive release of τ_i at least $\mathbb{m}_i$ releases must meet their deadline.

The symbols used in this paper are summarized in the table1 while the terms used are discussed in the next subsection.

a) *Terms Used*

MK_hyperperiod (L): It can be defined as the point after which all the task in the set are in phase and $(\mathbb{m}, \mathbb{k})$ pattern for each task is restarted i.e. the situation at time $\mathrm{t}=0$ is restored, mathematically,$L = LCM((\mathbb{k}_i * p_i)\ where\ i = 1, 2 \ldots n)$ where LCM stands for least common multiple.

Response time $\left(R_i^j(s)\right)$ of a job τ_i^j: It is the sum of the time requirement of the job τ_i^j and higher priority preempting jobs. Mathematically, $R_i^j(s) = e_i(s) + \sum_{\forall \tau_h^x \in H_{(i,j)}}(e_h(s_{ah}^x) + \Re)$where $H_{(i,j)}$ is the set of mandatory jobs preempting τ_i^j during the time $\left(rel_i^j, rel_i^j + R_i^{j,\gamma-1}(s_k)\right)$. The equation $R_i^j(s)$ is an iterative equation which can be solved using different iterations represented by $\gamma = 0, 1, 2 \ldots \infty$. For the first iteration $R_i^{j,0}(s) = e_i(s)$. The iterative equation $R_i^{j,\gamma}(s)$ terminates when either of the two conditions is satisfied: a) value of the two consecutive iteration is same i.e., $R_i^{j,\gamma-1}(s) = R_i^{j,\gamma}(s)$ or b) value exceeds its relative deadline i.e., $R_i^{j,\gamma}(s) > d_i$.

DPD Threshold (th): In DPD policy a component is switched to a sleep state on the occurrence of idle slot to save energy. For such a switching the system has to save the state of the task at the beginning and restore the saved status at the end of sleep state (switching from sleep state to active state). These two activities incur an overhead called the DPD overhead. To have a positive energy saving the component should not be switched to sleep state for duration (t) less than the DPD threshold th which can be estimated as follows:

Energy consumed by a component when it remains idle during idle slot t is $E_{idle}\,t$ (1)

Energy consumed in sleep state during t

Energy consumed by the component to go into sleep state is $E_{save} * t_{save}$ and to awake is $E_{wake} * t_{wake}$ where E_{save} is the energy per unit time to save the context, E_{wake} is the energy per unit time to retrieve the context and t_{save}, t_{wake} are the time the component needs to save and wake during context switch respectively. Thus, the component can sleep for time $(t - t_{save} - t_{wake})$ and consume energy at a rate E_{slp}. Hence, the energy consumed for sleep state of duration t would be

$$E_{save}\,t_{save} + E_{wake}\,t_{wake} + E_{slp}\,(t - t_{save} - t_{wake}) \quad (2)$$

To attain a positive energy gain the energy consumed by switching to sleep state (as measured in equation (2)) should be less than that consumed in the idle mode (as measure in equation (1)), i.e., (2)<(1)

$$\Rightarrow E_{save}\,t_{save} + E_{wake}\,t_{wake} + E_{slp}\,(t - t_{save} - t_{wake}) < E_{idle}\,t$$

In worst case when no energy gain is measured (equation (1)=(2)) then the threshold th can be estimated

$$th = \left(E_{save}\,t_{save} + E_{wake}\,t_{wake} - E_{slp}\,(t_{save} + t_{wake})\right)/\left(E_{idle} - E_{slp}\right) \quad (3)$$

The threshold of each component can be estimated by equation (3).

Substituting the value of $E_{save}\,t_{save} + E_{wake}\,t_{wake}$ in terms of th in the equation (2) we get,

$$th\left(E_{idle} - E_{slp}\right) + E_{slp}\,(t_{save} + t_{wake}) + E_{slp}\,(t - t_{save} - t_{wake}) \quad (3a)$$

Energy saved by switching to sleep state would be the difference between equations (1) and (3a).

$$diff = E_{idle}\,t - th\left(E_{idle} - E_{slp}\right) + E_{slp} = (t - th)\left(E_{idle} - E_{slp}\right)$$

If $(t > th)$then the energy gain $(diff)$ would be positive hence, energy consumed in switching to sleep state and remain in it for t units of time would reduce energy consumption and hence, is advisable to switch to sleep state.

For $(t = th)$the energy consumed to remain idle or sleep are same.

When $(t < th)$then it is recommended to remain in the idle state rather than to switch to sleep state in which it would consume more energy.
Thus, the energy consumed by a component for an idle slot of (t)would be:

$$\varepsilon(t) = \begin{cases} E_{idle}\, t & 0 \le t \le th \\ E_{idle}\, th + E_{slp}(t - th) & t > th \end{cases} \quad (4)$$

Critical speed of the task (s_{ci}): The DVS technique advocates that reduction in the speed of the frequency dependent component would reduce the energy consumption. This may not be true when the system is having both frequency dependent and independent components because lower speed leads to longer execution time for which the frequency independent components would remain active and consume energy. That is, on reduction in speed, the system energy consumption first decreases then it starts increasing incase speed is further reduced. The speed at which system energy requirement is least for a task is called the critical speed. Each task in the system has its own critical speed because its computation demand and set of associated components may differ. It can be determined as follows:

Consider a task τ_i with computation time $e_i(s) = \mathbb{e}_p/s + \mathbb{e}_d$ where $\mathbb{e}_p$ and $\mathbb{e}_d$ are the computation requirement for frequency dependent component at the speed s and independent component respectively. Then the energy consumed by the task τ_i at speed s would be

$$E_i(s) = e_i(s) * \left(E_{p\omega} + \sum^{k\in A} E^i_{dact,k}\right)$$

where ω is the speed index of s, i.e., $s_\omega = s$. Thus, $E_{p\omega}$ is the rate of energy consumption of the processor at speed $s(or\ s_\omega)$, $\sum^{k\in A} E^i_{dact,k}$ is the total energy consumption rate of all the frequency independent devices associated with task τ_i.

In [11, 13] authors have used energy model where energy consumed by the processor is directly proportional to the cube of the operating speed i.e., $E_p \propto s^3$ hence, $E_p = Cs^3$ where $s \in S$ and C is the proportionality constant.

As the task energy consumption function, $E_i(s)$ is a strictly convex function over speed s it can have a single speed at which energy consumption could be minimum, this can be estimated by setting its first derivative to zero followed by the second derivative to be positive.
Thus, taking the first derivative of $E_i(s)$ with respect to s as

$$\frac{\partial E(s)}{\partial s} = \left(-\left(\mathbb{e}_p/s^2\right)\left(Cs^3 + \sum^{k\in A} E^i_{dact,k}\right)\right) + \left(\left(\mathbb{e}_p/s + \mathbb{e}_d\right)(3Cs^2)\right) = 0$$

$$\frac{\partial E(s)}{\partial s} = 3C\mathbb{e}_d s^4 + 2Cs^3\mathbb{e}_p - \mathbb{e}_p \sum^{k\in A} E^i_{dact,k} = 0 \quad (5)$$

By Descartes' Rule of Signs [43], there is only one positive root of the equation since the sign between two consecutive terms changes only once. This root is referred to as the critical speed of the task τ_i represented as s_{ci}.

In the following subsection we discuss the various methods for partitioning the jobs into mandatory and optional. The partitioning problem is NP-hard [40] hence, various heuristic techniques (Red_Pattern, Even_Pattern, Rev_Pattern, Hyd_Pattern, Mix_Pattern) can be used which are discussed below:

Deeply Red-Pattern (Red_Pattern): This pattern was proposed by Koren & Shasha [41]. Mathematically, this can be described as

$$\pi_i^j = \begin{cases} 1, & 0 \le j \bmod \mathbb{k}_i < \mathbb{m}_i \\ 0, & otherwise \end{cases} \quad j = 0, 1, \ldots . \mathbb{k}_i - 1$$

When π_i^j is 1, release τ_i^j is mandatory while it is optional in case 0 is assigned to π_i^j. We refer this pattern as Red_Pattern. Advantage of applying this pattern to a task set for energy minimization is that it aligns the optional jobs together so that a component has a better opportunity to switch into sleep state to save energy. For a task whose critical speed is higher than or equal to the highest possible speed $(s_\mathcal{N})$ the operating speed should never be scaled down. Assigning Red_Pattern to such a task helps to extend the idle interval for switching to sleep state. However, for a task whose critical speed is lower than $s_\mathcal{N}$ Red_Pattern overloads the system leading to large size busy intervals and need more energy to be feasible.

Evenly Distributed Pattern (Even_Pattern): Ramanathan [42] used evenly distributed pattern in which the first release is always mandatory and the distribution of mandatory and optional is evenly i.e., alternating. Mathematically, this can be described as

$$\pi_i^j = \begin{cases} 1, & if\ j = \left\lfloor \left\lceil \frac{j * \mathbb{m}_i}{\mathbb{k}_i} \right\rceil * \frac{\mathbb{k}_i}{\mathbb{m}_i} \right\rfloor \\ 0, & otherwise \end{cases} \quad for\ \ j = 0, 1, \ldots \mathbb{k}_i - 1$$

We refer it to as Even_Pattern.

Reverse Evenly Distributed Pattern (Rev_Pattern): This pattern is a reverse of the Even_Pattern, hence the first release is always optional and the distribution of mandatory and optional is alternating. Mathematically:

$$\pi_i^j = \begin{cases} 0, & if\ j = \left\lfloor \left\lceil \frac{j * (\mathbb{k}_i - \mathbb{m}_i)}{\mathbb{k}_i} \right\rceil * \frac{\mathbb{k}_i}{(\mathbb{k}_i - \mathbb{m}_i)} \right\rfloor \\ 1, & otherwise \end{cases} \quad j = 0, 1, \ldots \mathbb{k}_i - 1$$

This pattern was first proposed by Niu & Quan [11] and we refer it as Rev_Pattern.

Hybrid Pattern (Hyd_Pattern): This pattern was proposed by [11] in which instead of assigning same pattern to all the tasks in the task set, they assigned different type of patterns (Red_Pattern or Even_Pattern) to each task. For example, task τ_1 is partitioned into mandatory and optional according to Red_Pattern while τ_2 and τ_3 could be assigned Red_Pattern or Even_Pattern. Thus, yielding 2^n possible combination of pattern assignment where n is the number of the tasks in the task set.

Mixed Pattern (Mix_Pattern): The hybrid pattern allows a task in the task set to be scheduled by Red_Pattern or Even_Pattern. In both cases at least the first release of each task is mandatory (if not more e.g. $(\mathbb{m}, \mathbb{k}) = \{(3,5), (4,7)\}$ first two releases of both the task are mandatory with the Hyd_Pattern) and are in phase hence, will overload the system, forcing it to be feasible with high energy requirement. Therefore, to improve the performance of Hyd_Pattern authors [29] suggested a mixed pattern (Mix_Pattern) which combines the Hyd_Pattern with the Rev_Pattern yielding 3^n possible combination of pattern assignment. By including the Rev_Pattern the Mix_Pattern would give a task fairer chance to execute at lower speed assignment (the second release of both the task in the above example would be mandatory while the first may or may not be so. Since the second release of a task would usually be out of phase with the other releases and will not overload the system as hybrid pattern does). Thus, Mix_Pattern is the superset of all the above suggested patterns. In this paper we would use Mix_Pattern.

In the following section we propose the energy minimization technique for the weakly hard real time system which was modeled in this section.

III. Three-Phase Energy Minimization Technique

This work is refinement of the two phase approach suggested by Agrawal et. al. [29]. In the first phase authors estimate the critical speed for each task and use a static partitioning strategy called Mix_Pattern. Based on the critical speed and the mandatory job distribution authors assigned the speed to each task such that the task set is feasible. While in phase two the authors suggested a preemption control strategy. They suggested increasing the speed of the lower priority job so that it can complete before preemption. However, the reduction in energy due to preemption control may be less than the energy consumed to fit the lower priority job in the slack of the higher priority job, i.e., the technique may be counter productive. In such cases they suggest to execute at the assigned speed as was done by Niu and Quan [37].

In this paper we suggest a three phase technique for system energy minimization. In the first phase we generate a feasible schedule which assigns the speed closest to the critical speed to all the tasks partitioned by Mix_Pattern. In the second phase, we refine the preemption control technique suggested by Agrawal et. al. [29], Niu and Quan [37] after locating their pitfalls. Further, in the third phase we measure the idle slots available on either side of a job execution window. Based on which we adjust the speed of the job or delay the starting of a job so as to combine the two slot. In the following subsection we illustrate the three phases.

Phase-1: Task Level Feasibility and Speed Assignment

In this phase we first estimate the critical speed of each task according to the equation (5). Further, the jobs of each task are marked mandatory/optional according to Mix_Pattern and speed closest to the critical speed on which the task set is feasible is assigned. The algorithm for speed_fitting as suggested by [29] can be stated below.

// Greedy approach based speed fitting algorithm

Algorithm speed_fitting(task set T)

Begin

1. For all the task $\tau_i \in T$
   Do
   a. Compute the critical speed for each task s_{ci}
   b. Initialize ω_i with the speed index of s_{ci}
   c. Assign $s_{ai} = s_{\omega_i}$ (which is same as s_{ci})
   Repeat
2. While (not feasible)
   Do
   a. For all task $\tau_i \in T$
   Do
   i. If $(\omega_i < N)$
   1. Compute

$$\nabla_i = \left(\left(E_{p\omega_i+1} e_i(s_{\omega_i+1})\right) - \left(E_{p\omega_i} e_i(s_{\omega_i})\right)\right) m_i \Big) / p_i k_i$$

   Else
   1. $\nabla_i = \infty$
   Repeat
   b. Select a task τ_i with smallest ∇_i
   c. If $(\omega_i < N)$
   i. $\omega_i = \omega_i + 1$
   ii. $s_{ai} = s_{\omega_i}$
   iii. Goto step 2
   Else
   i. Goto step 2b. to select next smallest ∇_i
   Repeat

End

In the following subsection we describe the job level second phase.

Phase-2: Modified Preemption Control Technique

The feasible schedule generated after speed_fitting for the task set T in the first phase may not be optimal in terms of energy consumption. To further reduce the energy consumption in this phase we suggest a greedy based preemption control followed by speed adjustment and delayed start in third phase.

When a job is scheduled on the processor then the associated devices are switched to active state in which they remain till it completes. Thus, if a lower priority job is preempted by the higher priority job then the associated device remain active and consume energy for the time for which the job is preempted. This extra consumption in the energy can be reduced by

delaying the higher priority job if possible and completing the execution of the lower priority job in the meanwhile (laxity).

The higher priority preempting job can be delayed up to its laxity available so that it does not miss its deadline. This laxity can be estimated as follows: $laxity_h^x = D_h^x - R_h^x(s_{ah}^x)\ \forall \tau_h^x \in H_{(i,j)}$ where $H_{(i,j)}$ is the set of mandatory jobs which preempts τ_i^j such that $rel_h^x > rel_i^j$. Hence, the time available for execution non-preemptively by the lower job would be

$$Ta_i^j = min\left(min_{\forall \tau_h^x \in H_{(i,j)}:(rel_h^x + laxity_h^x) < t_{curr}}\ (rel_h^x - tcurr + laxityhx, Dij - tcurr \right. \quad (6)$$

where t_{curr} is the current time when no higher job is available then $rel_h^x = \infty$. If the time available is sufficient to complete the job τ_i^j non-preemptively as suggested by [37] then we do so. However, when more than one higher priority jobs preempt a single lower priority job then approach suggested in [37] may fail to finish the lower priority job earlier. This is due to the fact that once a higher job finishes and another higher priority job is available in the ready queue then it would be scheduled as it has priority higher than the incomplete preempted job. This can be observed from the example 1.

Example1: Consider a task set $T = \{\langle e_i(s_{ai}),\ p_i,\ d_i\rangle: \langle 15, 25, 25\rangle, \langle 25, 100, 100\rangle\}$. When scheduled without preemption control then the response time of the lower priority job τ_2^1 after being preempted by τ_1^2 and τ_1^3 would be 70 refer figure 1a. However, as illustrated by figure 1b (obtained by utilizing the concept of preemption control used in [37]) the response time of job τ_2^1 remains 70 whereas the number of preemptions is reduced from 2 to 1. This is because τ_2^1 is unable to complete in slack of τ_1^2 which completes at time 50 after which the scheduler schedules the higher priority job τ_1^3 since; no job is being preempted so no preemption control is applied.

Thus, we refine the preemption control approach suggested in [37] without varying the speed as *modified preemption control at assigned speed (MPCAS)*. Here a lower priority job may be allowed to restart even when higher priority job is ready, provided feasibility of the higher priority is assured. The effectiveness of this approach is seen in figure 1(c) where the response time of the job τ_2^1 is reduced to 55 from 70. The proposed MPCAS approach is given as below:

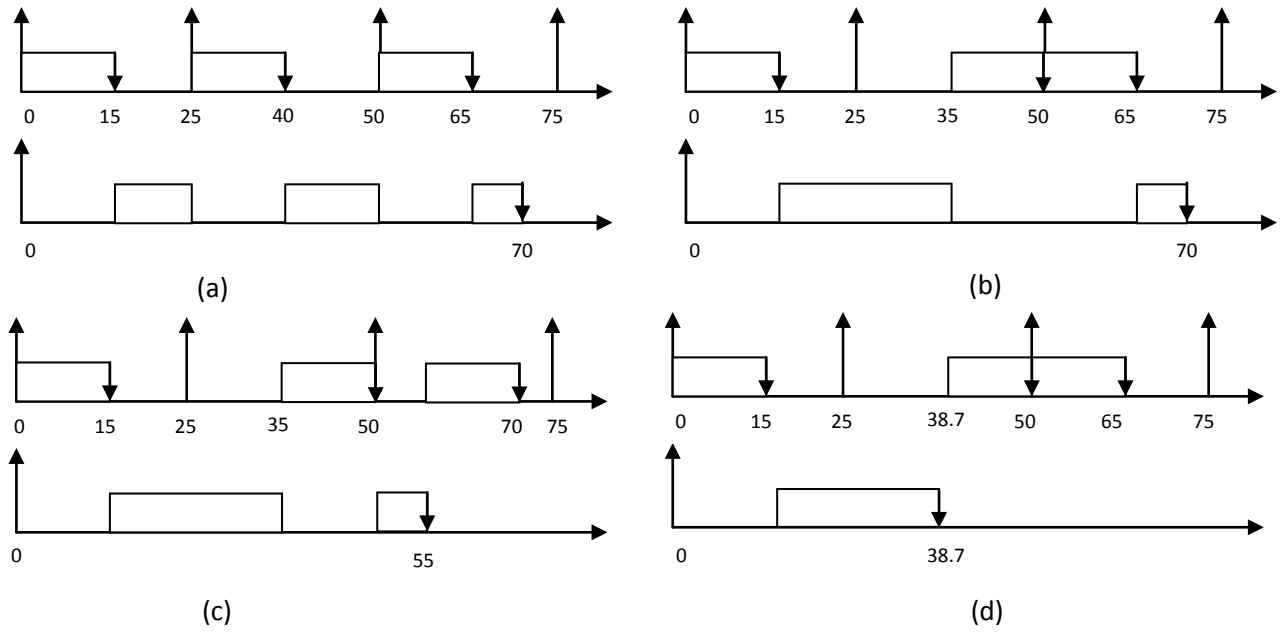


Figure 1: Schedule for task set T of example 1: (a) Uncontrolled Preemption (b) Preemption Control as suggested by [37], (c) Preemption Control by proposed MPCAS algorithm (d) Preemption Control by the

higher priority is assured. The effectiveness of this approach is seen in figure 1(c) where the response time of the job τ_2^1 is reduced to 55 from 70. The proposed MPCAS approach is given as below:

// Preemption control at the assigned speed

//Algorithm MPCAS (task set T)

Begin

1. Set the $t_{curr} = 0$ // the current time
2. For all jobs in one MK_hyperperiod

Do

a. if(incomplete_queue is empty)
   i. if(ready_queue is empty)
      1. wait for a job to arrive in it
      2. Update t_{curr}
   ii. Let τ_i^j be read from the ready_queue
   iii. Estimate the time available Ta_i^j
   iv. If $\left(Ta_i^j \geq e_i(s_i^j)\right)$
      1. Execute τ_i^j non-preemptively for $e_i(s_i^j)$
      2. Update $t_{curr} = t_{curr} + e_i(s_i^j)$
      3. $e_i(s_i^j) = 0$
      4. Goto step 2a.
   v. Else
      1. Execute τ_i^j non-preemptively for Ta_i^j
      2. $e_i(s_i^j) = e_i(s_i^j) - Ta_i^j + \Re$
      3. Insert τ_i^j into incomplete_queue based on its priority
      4. Update $t_{curr} = t_{curr} + Ta_i^j$
      5. Goto step 2a.
b. Else
   i. Let τ_i^j be read from the incomplete_queue
   ii. Estimate the time available Ta_i^j
   iii. If $\left(Ta_i^j \geq e_i(s_i^j)\right)$
      1. Execute τ_i^j non-preemptively for $e_i(s_i^j)$
      2. Update $t_{curr} = t_{curr} + e_i(s_i^j)$
      3. $e_i(s_i^j) = 0$
      4. Goto step 2a.

      Else
      1. Insert τ_i^j into incomplete_queue based on its priority
      2. Goto step 2.a.i.

Repeat

End

The MPCAS algorithm would reduce the response time of the lower priority job (τ_2^1 would finish at time 55 for the example) so the associated devices have better opportunity to switch to sleep state and save energy according to DPD. However, when component's DPD threshold is large than this reduction in response time may not be sufficient to allow the associated components to sleep and save energy. Agrawal et. al. [29] increase the speed of the lower priority job and hence, reduce its execution time so that it can fit in the slack available before it could be preempted (speed of the job τ_2^1 would be increased such that it would finish by 35 in the example). The authors themselves state that this may be counter productive. That is, increment in energy consumption by executing the lower priority job at higher speed is more than the energy reduction gained due to early switching to sleep state for some components. To overcome this drawback we suggest a speed refinement for the preempted lower priority job as well as preempting higher priority jobs. This speed combination is predicted by greedy based preemption control (GBPC) which utilizes right and left idle slot (refer figure 2 and definition 1, 2, 3, 4) of the processor and the devices.

Definition 1: The device left idle slot of a job τ_i^j it is the time when the previous job of the same task $\left(\tau_i^{j-1}\right)$ finishes and relinquishes the resource $\alpha_{ld_i}^{\ j}$ to the time $\beta_{ld_i}^{\ j}$ when this job τ_i^j is scheduled for the first time.

Definition 2: The device right idle slot of a job τ_i^j is the time when this job $\left(\tau_i^j\right)$ finishes and relinquishes the resource $\alpha_{rd_i}^{\ j}$ to the time $\beta_{rd_i}^{\ j}$ when the next job of the same task $\left(\tau_i^{j+1}\right)$ is scheduled for the first time.

Definition 3: The processor left idle slot of a job τ_i^j starts when all the jobs in the ready queue finishes $\alpha_{lp_i}^{\ j}$ to the time job τ_i^j to its released $\left(\beta_{lp_i}^{\ j}\right)$ to the empty queue. Mathematically, $\delta_{lp_i}^{\ j} = max\left(0, \left(\beta_{lp_i}^{\ j} - \alpha_{lp_i}^{\ j}\right)\right)$

Definition 4: The processor right idle slot of a job τ_i^j is the time when it finishes and relinquishes the processor $\alpha_{rp_i}^{\ j}$ to the time any other job is scheduled on it $\beta_{rp_i}^{\ j}$.

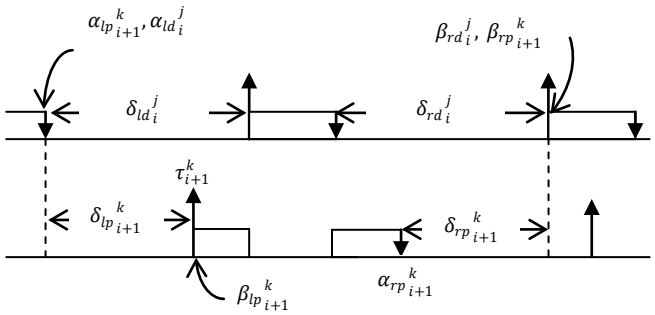


Figure 2: Left and right idle

Thus, the left idle time for the device and the processor are $\delta_{ld_i}^{\ j} = \left(\beta_{ld_i}^{\ j} - \alpha_{ld_i}^{\ j}\right)$ and $\delta_{lp_i}^{\ j} = \left(\beta_{lp_i}^{\ j} - \alpha_{lp_i}^{\ j}\right)$ respectively whereas $\delta_{rd_i}^{\ j} = \left(\beta_{rd_i}^{\ j} - \alpha_{rd_i}^{\ j}\right)$, $\delta_{rp_i}^{\ j} = \left(\beta_{rp_i}^{\ j} - \alpha_{rp_i}^{\ j}\right)$ are the right idle slots. As there is no resource conflict for frequency independent component so whenever a task is started first time all the associated resources are activated and remain to be so till the job completes. Thus, the left as well as the right idle time for any two frequency independent

component associated with the same task are always same. In the following subsection we estimate the energy consumption required during the idle slots by the device and the processor:

Energy estimation of device a_k^i during the idle slots: Consider an idle slot $\left((idle = \delta_{ld_i}{}^j)\ or\ (idle = \delta_{rd_i}{}^j)\right)$ for a device a_k^i associated with a job τ_i^j. If idle time is greater than its threshold thd_k then the device would switch into sleep state otherwise it would remain active. Thus, the energy consumed $\left(\varepsilon_{dk}^i(idle)\right)$ by the device as can be derived from the equation (4) as

$$\varepsilon_{dk}^i(idle) = \begin{cases} E_{dact,k}^i idle & 0 \le idle \le thd_k \\ E_{dact,k}^i thd_k + E_{dslp,k}^i(idle - thd_k) & idle > thd_k \end{cases} \quad (6)$$

Energy estimation for processor during the idle slots: For a processor idle slot $\left(\left(idle = \delta_{lp_i}{}^j\right)\ or\ \left(idle = \delta_{rp_i}{}^j\right)\right)$, if this idle time is greater than threshold of the processor thp then the processor would switch to sleep state otherwise remain in idle state. Thus, the energy consumption rate for the processor can be estimated from the equation (4) is

$$\varepsilon_{pi}^j(idle) = \begin{cases} E_{pidle}\ idle & 0 \le idle \le thp \\ E_{pidle}\ thp + E_{pslp}\ (idle - thp) & idle > thp \end{cases} \quad (7)$$

In the next subsection we estimate the energy consumed by the frequency dependent and independent components during job execution.

Energy estimation for response time $\left(R_i^j(s_i^j)\right)$ of a job τ_i^j: When a job τ_i^j starts execution then all the associated frequency independent devices are switched to active state in which they remain till it completes. The frequency dependent components work at the assigned frequency for this task as well as other jobs preempting it. Thus, the total energy consumed by a job τ_i^j during its response time

$$E_i^j(s_i^j) = R_i^j(s_i^j)\sum^{k\in A} E_{dact,k}^i + E_{p\omega_i} e_i(s_i^j) + \sum_{\forall \tau_h^x \in H_{(i,j)}}{}^x \left(e_h(s_{ah}) E_{p\omega_h} + E_{\Re}\right) \quad (8)$$

where ω_i, ω_h are the speed index of s_i^j, s_{ah}^x and $H_{(i,j)}$ is the set of mandatory jobs preempting job τ_i^j.

The energy consumed by the frequency independent component during $R_i^j(s_i^j)$ would be $R_i^j(s_i^j) * \sum^{k\in A} E_{dact,k}^i$. Whereas, $\left(E_{p\omega_i} * e_i(s_i^j)\right)$ will be the processor energy consumed by the job τ_i^j alone. Further, job τ_i^j would be preempted by $\forall \tau_h^x \in H_{(i,j)}$ during its execution window $\left(rel_i^j, rel_i^j + R_i^j(s_i^j)\right)$. A preempting mandatory job τ_h^x would execute at its assigned speed of s_{ah}^x, therefore it will consume energy $e_h(s_{ah}^x) * E_{p\omega_h}$. However, each time a job preempts τ_i^j it would incur energy overhead $E_{\Re}$ for accessing the memory which is frequency independent [24]. Thus, total energy consumed during response time of job τ_i^j is given in equation (8). Further, energy is also consumed during the idle slots.

Total energy consumption during the left idle, response time and right idle slots

Thus, the energy consumed by the device during the left (and right) idle slot of a job τ_i^j would be $\varepsilon_{di}^j(\delta_{ld_i}{}^j)$ (and $\varepsilon_{di}^j(\delta_{rd_i}{}^j)$) (refer equation (6)) while the energy consumption by the processor during the left (and right) idle slot would be $\varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right)$ (and $\varepsilon_{pi}^j\left(\delta_{rp_i}{}^j\right)$) (refer equation (7)). The energy consumed during the execution of the job would be $E_i^j(s_i^j)$ as estimated in the equation (8). Thus, the energy consumption of the job τ_i^j along with its left and the right idle slots is

$$\varepsilon_i^j(s_i^j) = \varepsilon_{di}^j(\delta_{ld_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right) + E_i^j(s_i^j) + \varepsilon_{di}^j(\delta_{rd_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{rp_i}{}^j\right) \quad (9)$$

After estimating the energy we now discuss the technique for greedy based preemption control. If the lower priority job is preempted then the preemption control at the assigned speed is done (using PCAS algorithm) and the energy is estimated for the preempting higher priority jobs and the preempted lower priority job. If the lower priority job is still preempted by one or more, higher priority jobs then the response time of the lower priority can be further reduced. The reduction in the response time of the lower priority job can be achieved by increasing the speed of either the higher priority bottleneck job τ_h^x (such that $Ta_i^j = \left((rel_h^x - t_{curr}) + slack_i^j\right)$) or the preempted lower priority job. The choice between the two is made based on the minimum increment in energy, i.e., $min\left(\Delta E_h^x(s_h^x), \Delta E_i^j(s_i^j)\right)$ where

$$\Delta E_h^x(s_h^x) = e_h\left(s_{\omega_h+1}\right)\left(E_{p\omega_h+1} + \sum^{k\in A} E_{dact,k}^h\right) - e_h\left(s_{\omega_h+1}\right)\left(E_{p\omega_h} + \sum^{k\in A} E_{dact,k}^h\right)$$

(ω_h is the speed index of s_h^x) similarly, estimate $\Delta E_i^j(s_i^j)$. The speed of the chosen job is incremented and the energy is estimated. The process of further reduction in response time of the lower priority job is repeated and the energy for different combinations is estimated till either a) the lower priority job is no longer preempted; b) all the jobs are assigned maximum available speed level. The speed combination which requires minimum energy is assigned and the schedule is updated.

Further, energy minimization is achieved by improving the schedule obtained in phase-2.

Phase-3: Speed Adjustment and Delay Start (SADS)

After assigning speeds to each task in the phase-1 ensuring feasibility followed by the reduction in

energy consumption by preemption control in the second phase. This phase will adjust the speed assigned (increase or decrease) and accumulate the idle slot (delay start a job if possible) to reduce the energy consumption. In this phase detail analysis of the preemption controlled schedule is done where right and left idle slot (refer figure 1 and definition 1, 2, 3, 4) of the processor and the device are re-estimated. After estimating the energy we now propose the new technique for improvement namely, speed adjustment and delay start which we finally combine to provide overall reduction in energy. The next subsection discusses the speed adjustment.

Speed adjustment

In the phase-1 the feasibility of the task set was to be ensured by assigning the speed at the task level, while the phase-2 increases the speed of some jobs to decrease loss in energy due to preemption. In this phase the speed is adjusted by considering each job separately to reduce the energy consumption based on the left and right idle slots. The philosophy for this approach is that speed fitting was done at the task level to make all the jobs feasible. Executing job at higher speed may favor switching to sleep state by more components (sleeping for more time) in some cases while executing at lower speed may favor the idea of DVS. Thus, depending on the left and the right idle slots we estimate the optimal speed for each job which may be different from that of the task. In the next subsection we measure the energy consumption at the job level after adjusting the speed.

Energy estimation of a job after adjusting the speed

i. Energy estimation at speed s_{ai}^j: The energy consumed by the device during the left (or right) idle slot would $\varepsilon_{di}^j(\delta_{ld_i}{}^j)$ (or $\varepsilon_{di}^j(\delta_{rd_i}{}^j)$) (refer equation (6)) while the energy consumption by the processor during the left(or right) idle slot would be $\varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right)$ (or $\varepsilon_{pi}^j\left(\delta_{rp_i}{}^j\right)$) (refer equation (7)). The energy consumed during the execution of the job would be $E_i^j(s_{ai}^j)$ as estimated in the equation (8). Thus, the energy consumption of the job τ_i^j along with its left and the right idle slots is

$$\varepsilon_i^j(s_{ai}^j) = \varepsilon_{di}^j(\delta_{ld_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right) + E_i^j(s_{ai}^j) + \varepsilon_{di}^j(\delta_{rd_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{rp_i}{}^j\right)$$

ii. Energy estimation at speed $s_x < s_{ai}^j$: In a scenario where some of the components may not be able to switch to the sleep state (depending on their thresholds) then executing the job at a lower speed (s_x) than the assigned speed (s_{ai}^j) may save the processor energy. But this execution is subject to the availability of the right idle slot since this reduction in speed will force longer response time. This extra time can be measured as $\Delta_i^j(s_x) = R_i^j(s_x) - R_i^j(s_{ai}^j)$ for completion which will reduce the right idle slot by the same amount.

In case the value of $\Delta_i^j(s_x) > \delta_{rp_i}{}^j$ indicating that the right idle slot is not long enough, hence, the lower speed (s_x) cannot be assigned. Otherwise energy consumption will be

$$\varepsilon_i^j(s_x) = \varepsilon_{di}^j(\delta_{ld_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right) + E_i^j(s_x) + \varepsilon_{di}^j\left(\delta_{rd_i}{}^j - \Delta_i^j(s_x)\right) + \varepsilon_{pi}^j\left(\delta_{rp_i}{}^j - \Delta_i^j(s_x)\right)$$

iii. Energy estimation at speed $s_y > s_{ai}^j$: When some components are unable to switch to sleep state then if a job executes at a higher speed then it will complete earlier. This would improve the possibility to switch the components into sleep state and increase the sleeping time of the already sleeping components. The time thus saved is $\Delta_i^j(s_y) = R_i^j(s_y) - R_i^j(s_{ai}^j)$ which will increase length of the right idle slot. Hence, the total energy consumption will be

$$\varepsilon_i^j(s_y) = \varepsilon_{di}^j(\delta_{ld_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right) + E_i^j(s_y) + \varepsilon_{di}^j\left(\delta_{rd_i}{}^j - \Delta_i^j(s_y)\right) + \varepsilon_{pi}^j\left(\delta_{rp_i}{}^j - \Delta_i^j(s_y)\right).$$

Thus, in general the energy estimated at any speed s can be stated as:

$$\varepsilon_i^j(s) = \varepsilon_{di}^j(\delta_{ld_i}{}^j) + \varepsilon_{pi}^j\left(\delta_{lp_i}{}^j\right) + E_i^j(s) + \varepsilon_{di}^j\left(\delta_{rd_i}{}^j - \Delta ijs + \varepsilon pij\delta rpij - \Delta ijs\right) \quad (10)$$

Where $\Delta_i^j(s) = R_i^j(s) - R_i^j(s_{ai}^j)$

In the next subsection we discuss the technique for accumulation of idle slots by delaying the task execution window.

Delay Start Technique

In this part of the third phase we aim to assemble the idle slots fragmented on the two sides of a job by delaying its execution if the schedule permits i.e. shift the job execution towards its deadline. This may enable the associated components to sleep or sleep for longer time to save energy. A job may delay its execution up to its deadline so as to be feasible. But extending the job up to its deadline may force the up coming job to miss their deadlines. Thus, a job would be allowed to consume only the processor right idle slot so that it may not push the upcoming jobs. Thus, a job τ_i^j may shift up to $min\left(D_i^j, \beta_{rp_i}{}^j\right)$, without missing its own deadline or modifying the schedule of the subsequent jobs. Hence, a delay will move the task execution by an amount $\theta_i^j(s) = min\left(D_i^j, \beta_{rp_i}{}^j\right) - R_i^j(s)$. In the next subsection we measure the energy consumption at the job level after delaying its execution and adjusting its speed.

Energy estimation of a job with delayed start

i. **Energy estimation due to delayed start at assigned speed s_{ai}^{j}:** Delaying a job τ_i^j would shift it towards right will elongate the left idle slot of the components hence provide better opportunity to the components to switch to sleep state. Thus, when execution of a job is shifted then its left idle slot will increase by $\theta_i^j(s_{ai}^j)$ while the right idle slot will decrease by the same. The energy consumption of the job τ_i^j along with its left and the right idle slots is

$$\varepsilon_i^j(s_{ai}^j) = \varepsilon_{di}^j\left(\delta_{ld_i}^{\ j} + \theta_i^j(s_{ai}^j)\right) + \varepsilon_{pi}^j\left(\delta_{lp_i}^{\ j} + \theta_i^j(s_{ai}^j)\right) + E_i^j(s_{ai}^j) + \varepsilon_{di}^j\left(\delta_{rd_i}^{\ j} - \theta_i^j(s_{ai}^j)\right) + \varepsilon_{pi}^j\left(\delta_{rp_i}^{\ j} - \theta_i^j(s_{ai}^j)\right).$$

ii. **Energy estimation due to delayed start at speed $s_x < s_{ai}^j$:** In a scenario where some of the components may not be able to switch to the sleep state (depending on their thresholds) then executing the job at a lower speed (s_x) than the assigned speed (s_{ai}^j) may save energy and reduce length of the right idle slot. Further, delaying the job would add the remaining right idle slot to the left idle slot, hence save energy. The energy consumption will be

$$\varepsilon_i^j(s_x) = \varepsilon_{di}^j\left(\delta_{ld_i}^{\ j} + \theta_i^j(s_x)\right) + \varepsilon_{pi}^j\left(\delta_{lp_i}^{\ j} + \theta_i^j(s_x)\right) + E_i(s_x) + \varepsilon_{di}^j\left(\delta_{rd_i}^{\ j} - \theta_i^j(s_x)\right) + \varepsilon_{pi}^j\left(\delta_{rp_i}^{\ j} - \theta_i^j(s_x)\right).$$

iii. **Energy estimation due to delayed start at speed $s_y > s_{ai}^j$:** When some components are unable to sleep in the left idle slot generated after accumulation with speed s_{ai}^j then increasing the speed would reduce the response time. Thus, a combination of higher speed and shift would elongate the left idle slot to provide room for switching into the sleep state. Hence, the energy consumption will be

$$\varepsilon_i^j(s_y) = \varepsilon_{di}^j\left(\delta_{ld_i}^{\ j} + \theta_i^j(s_y)\right) + \varepsilon_{pi}^j\left(\delta_{lp_i}^{\ j} + \theta_i^j(s_y)\right) + E_i^j(s_y) + \varepsilon_{di}^j\left(\delta_{rd_i}^{\ j} - \theta_i^j(s_y)\right) + \varepsilon_{pi}^j\left(\delta_{rp_i}^{\ j} - \theta_i^j(s_y)\right).$$

Thus, in general the energy estimated at any speed s and shifting can be stated as

$$\varepsilon_i^j(s) = \varepsilon_{di}^j\left(\delta_{ld_i}^{\ j} + \theta_i^j(s)\right) + \varepsilon_{pi}^j\left(\delta_{lp_i}^{\ j} + \theta_i^j(s)\right) + E_i(s) + \varepsilon_{di}^j\left(\delta_{rd_i}^{\ j} - \theta_i^j(s)\right) + \varepsilon_{pi}^j\left(\delta_{rd_i}^{\ j} - \theta_i^j(s)\right) \quad (11)$$

Combining adjusting the speed and delayed start concept

Finally, combining the two concepts the speed adjustment (equation (10)) and delayed (equation (11)) for considering each job for improvement individually we get

$$\varepsilon_i^j(s,\mathcal{s}) = \varepsilon_{di}^j\left(\delta_{ld_i}^{\ j} + \mathcal{s}\theta_i^j(s)\right) + \varepsilon_{pi}^j\left(\delta_{lp_i}^{\ j} + \mathcal{s}\theta_i^j(s)\right) + E_i^j(s) + \varepsilon_{di}^j\left(\delta_{rd_i}^{\ j} - \mathcal{s}\theta_i^j(s) - \mathcal{s}'\Delta_i^j(s)\right) + \varepsilon_{pi}^j\left(\delta_{rd_i}^{\ j} - \mathcal{s}\theta_i^j(s) - \mathcal{s}'\Delta_i^j(s)\right) \quad (12)$$

Where $s \in S$ set of speed levels available, $\mathcal{s}$ is a binary number which has a value 1 if a shift operation is made and $\mathcal{s}'$ is its complement.

Thus, for minimum energy consumption of a job τ_i^j must be assigned a speed s and delayed start operation $\mathcal{s}$ such that $\varepsilon_i^j(s,\mathcal{s})$ is minimum. Since, the left idle slot of job is same as the right idle slot of the previous job. Adjusting the speed/delay starting one job will affect the previous job's idle slots. Hence, by iterating the third phase further reduction in energy is achieved. The proposed Speed Adjustment and Delay Start can be stated as SADS algorithm.

The effectiveness of proposed three phase algorithm can be seen from the example 2 and table 2.

Example 2: Consider a task set $T = \{\langle e_i = (\mathbb{e}_{p,i}, \mathbb{e}_{d,i}), p_i,\ d_i\rangle: \langle(250,5),25,25\rangle, \langle(210,18),100,100\rangle\}$ to be scheduled on a DVS processor which can operate at speed $S = \{10, 25, 30, 35, 37, 40, 105\}$ where its threshold $thp = 10$. The device pool $A = \{a_1,\ a_2\}$ consists of two devices such that device a_1 is associated with task τ_1 and a_2 with τ_2 have attributes as $< a_k^i, thd_i, E_{dact,k}^i,\ E_{dslp,k}^i >: < a_1^1,\ 10,\ 54687,\ 0.0 >, < a_2^2,\ 30,\ 262285,\ 0.0 >$. The preemption overhead is $\mathfrak{R} = 5$ and energy it consumes for preemption is $\mathrm{E}_{\mathfrak{R}} = 8568937$.

The critical speed (s_{c1}, s_{c2}) as estimated from equation (5) would be 25 and 30 respectively. The MK_hyperperiod will be 100.

// Speed adjustment and delay start based third phase algorithm

//Algorithm SADS(task set T)

//input is the feasible schedule generated after speed fitting and GBPC after phase-2

Begin

1. While (no further reduction in energy)

   Do

   a. For each job $\tau_i^j \in \mathbb{M}$ where $\mathbb{M}$ is the set of mandatory jobs in the task set T arriving during any MK_hyperperiod (L)

      Do

      i. Estimate the left and the right idle time for device $\left(\delta_{ld_i}^{\ j}, \delta_{rd_i}^{\ j}\right)$and the processor $\left(\delta_{lp_i}^{\ j}, \delta_{rp_i}^{\ j}\right)$ according to definitions 1, 2, 3 and 4.

      ii. Assign speed to job τ_i^j as $s_{ai}^j = s_{ai}$ and shifting as $\mathcal{s}_i^j = 0$

      iii. Assign $min = \varepsilon_i^j(s_{ai}^j, \mathcal{s}_i^j)$ according to the equation (12)

      iv. For every speed $s \in S$

      Do

1. Estimate $\varepsilon_i^j(s,0)$ according to the equation (12)
2. If $\left(min > \varepsilon_i^j(s,0)\right)$
   a. Update $min = \varepsilon_i^j(s,0)$
   b. Update $s_{ai}^j = s$ and shifting as $\mathcal{s}_i^j = 0$
   End if
3. Estimate $\varepsilon_i^j(s,1)$ according to the equation (12)
4. If $\left(min > \varepsilon_i^j(s,1)\right)$
   a. Update $min = \varepsilon_i^j(s,1)$
   b. Update $s_{ai}^j = s$ and shifting as $\mathcal{s}_i^j = 1$
   End if
   End for
End for
2. Estimate the total energy for a MK_hyperperiod (L)
End while
End

In the following section we present the results obtained by implementation of the approach discussed in this section.

IV. Simulation Results

This section compares the performance of our proposed three phase scheduling algorithm (in which we apply greedy based preemption control, speed adjustment and delayed start) referred to as GBSADS with the higher speed preemption control (HSPC) approach suggested by [29]. All simulation results are computed on a DVS processor with operating speed level set as $S = \{0, s_1, s_2, s_3 \ldots s_{10}\}$ where s_i is a uniform random number generated in the interval $[10, 200]$. We consider ten types of devices with multiple instances forming a pool of devices. For a task, devices are randomly selected from this pool. Rate of energy consumption for a device is computed based on the energy required by the processor at the maximum speed, i.e., $E_{dact,i} = \text{Þ}E_{p10}$ where Þ is a uniform random number in the range [0.1, 20]. The task set $T = \{\tau_1, \tau_2, \tau_3 \ldots \tau_n\}$ with $(\mathbb{m}, \mathbb{k})$ utilization U (i.e. $\sum \mathbb{m}_i e_i / p_i \mathbb{k}_i$ a uniform random number in the range (0, 1]). The preemption overhead and energy required during preemption are uniform random number in the range (0, 1] and (0, 100] respectively. Similar type of considerations where taken in [29]. The other parameters are summarized in the table 3.

The key parameter, measured for simulation is energy consumed during one MK_hyperperiod. The result reported is the average value of results obtained for hundred task sets. The following section deals with the variation in energy with component threshold, task set utilization and device to processor energy proportionality constant.

Table 2: Energy estimation for a MK_Hyperperiod of the task set T for the example 2

s_{a1}^2	s_{a1}^2	s_{a1}^3	s_{a1}^4	s_{a2}^1	ft_1^2	Energy	Remark
25	25	25	25	30	100	56393661	Uncontrolled Preemption technique with DVS and DPD
25	25	25	25	30	75	33473217	Preemption control as suggested by [37]. Reduction in energy consumption is 40.6%.
25	25	25	25	105	35	36900380	Preemption control by increasing the speed of the lower priority job as suggested by [29].
Phase 2: GBPC							
25	25	25	25	30	60	29538942	Performing preemption control at the assigned speed (ASPC). This incapable of preventing preemption but reduces the response time. Reduction in energy from [37] 11.7% and 47.6% from uncontrolled preemption technique.
25	30	25	25	30	58.3	29126514	Increasing the speed of τ_1^2 based on the $\Delta E_1^2(s_1^2) = 31757.1,\ \Delta E_2^1(s_2^1) = 91715$. Preemption could not be prevented but the energy consumption is decreased.
25	30	25	25	35	57.3	29219229	$\Delta E_1^2(s_1^2) = 94063.1,\ \Delta E_2^1(s_2^1) = 91715$. Increasing the speed of τ_2^1
25	30	25	25	37	57	29312320	$\Delta E_1^2(s_1^2) = 94063.1,\ \Delta E_2^1(s_2^1) = 92790$. Increasing the speed of τ_2^1
25	35	25	25	37	55.8	29092841	$\Delta E_1^2(s_1^2) = 94063.1,\ \Delta E_2^1(s_2^1) = 185809$. Increasing the speed of τ_1^2
25	37	25	25	37	55.5	29076167	$\Delta E_1^2(s_1^2) = 62511, \Delta E_2^1(s_2^1) = 185809$. Increasing the speed of τ_1^2
25	40	25	25	37	38.7	16205293	$\Delta E_1^2(s_1^2) = 98151,\ E_2^1(s_2^1) = 185809$. Increasing the speed of τ_1^2. Preemption is avoided. Reduction in energy consumption by 51.59% from [29, 37] and 71.3% from uncontrolled preemption is received.
Phase -3: SADS							
Delaying job τ_1^4 for 10 units						15648423	Reduction of 53.3% from [29, 37] and 72.3% from uncontrolled preemption is received.

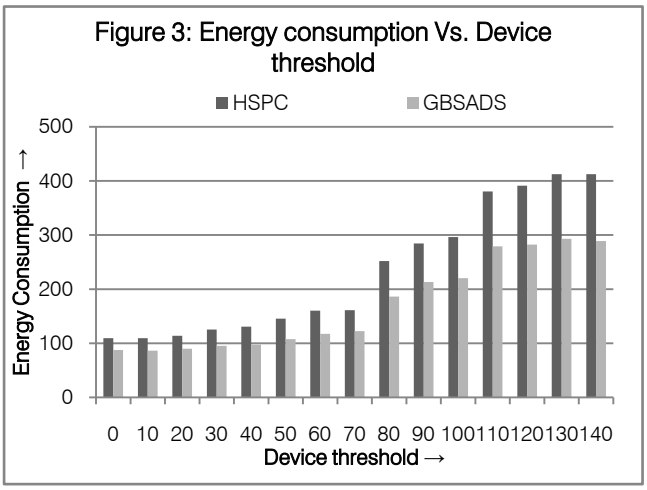
Effect of component threshold on Energy consumption: The value of the threshold of a component indicates the length of the idle slot for which the component will consume same energy in active state as it would do so in sleep state. Thus, as the threshold increases the requirement for long idle slots increases in absence of which energy consumption increases. However, increment in threshold will affect the energy requirement up to a certain value (length of the longest idle slot) beyond which no component would switch to sleep state, so any further increment in the threshold will not increase the energy consumption of the system. The effect of the increment in threshold for frequency independent and dependent components can be seen in the figure 3 and figure 4 for which the value of $U = [0.5,0.6]$ and $\text{Þ} = 10$.

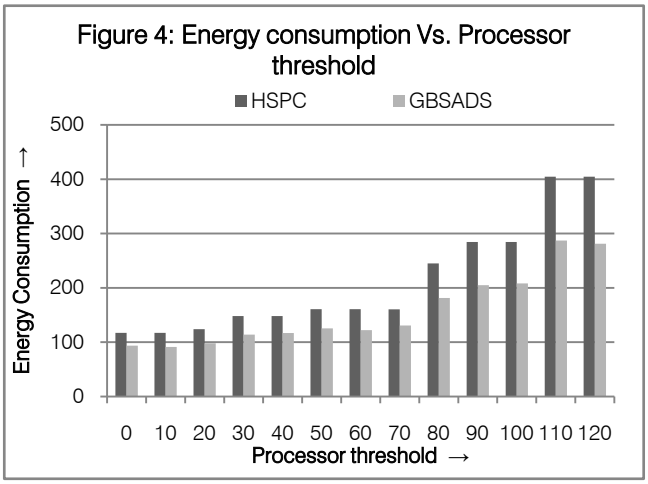
Table 3: Simulation Parameters

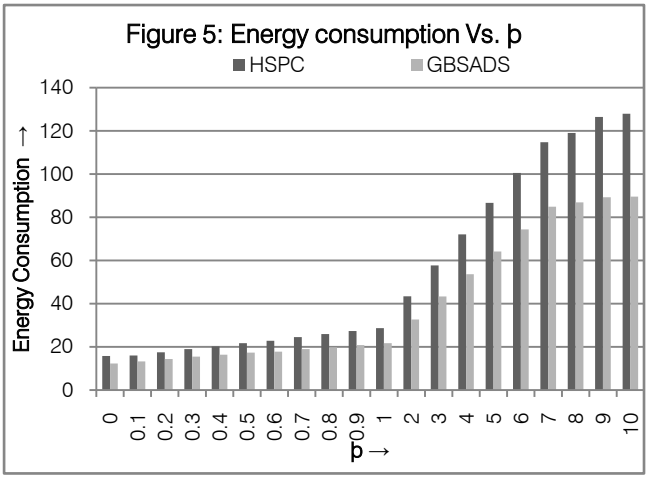
Parameter	Condition	Range
UTh Utilization Threshold	Is assigned	0.01
u_i Utilization	If $U - \sum u_{i-1} \geq UTh$ the select a uniform random number	$(0, U - \sum u_{i-1}]$
	If $U - \sum u_{i-1} < UTh$ then assign	$u_i = U - \sum u_{i-1}$
e_i worst case execution time	select a uniform random number	(0,100]
p_i period	select a uniform random number	(0,1000]
d_i deadline	select a uniform random number	$[e_i, p_i]$
k_i	Is a random integer selected uniformly	[1,10]
m_i is the number of mandatory jobs in k_i	Assigned a value	$\lfloor u_i p_i k_i / e_i$
thp processor threshold	select a uniform random number	[0, 200]

The effect of the variation of the device threshold is shown in the figure 3. When the device threshold is lower (0-80) it can be observed that the energy consumption by GBSADS approach is almost 23% lower than that of the HSPC approach, while this reduction in the energy consumption is more prominent (approximately 32%) at higher values of the threshold range (90-140). Beyond 130 it is constant due to the fact that at lower threshold value both GBSADS and HSPC control preemption around the assigned speed. But as this threshold increases the shorter idle slots become inadequate to switch the device into sleep state, the greedy based preemption control in second phase and delay start done in the third phase of the GBSADS approach assembles these idle slots efficiently and hence, provide better opportunity to switch the device into sleep state. Similar trends are seen for the variation in the processor threshold (refer figure 4) in which we get an overall gain of approximately 30%.

Effect of rate of processor to device energy (Þ) on Energy consumption: The rate of energy consumption by a frequency independent component is a constant. This constant could be less than the rate of the energy consumption in the processor (for processor dominant system Þ ≤ 1) while for device dominant systems this would be greater than one. This variation in the ratio (0.1-10) for both processor and device dominant systems is observed in the figure 5 for which task sets of utilization U=[0.5, 0.6]. For lower value of the ratio (0.1-1) processor dominated system the GBSADS approach saves approximately 20% of the energy and this saving increase up to 26% for device dominant systems. A sudden rise in the energy consumption is observed for a value of Þ = 2 which indicates the dominance of the devices energy consumption and as more devices are added to such a system this rise is even more prominent. At lower level the DVS approach is more prominent due to the fact the processor energy consumption is dominant, the HSPC approach applies DVS and high speed preemption control which would be inadequate. On the other hand, GBSADS approach applies the concept of DVS





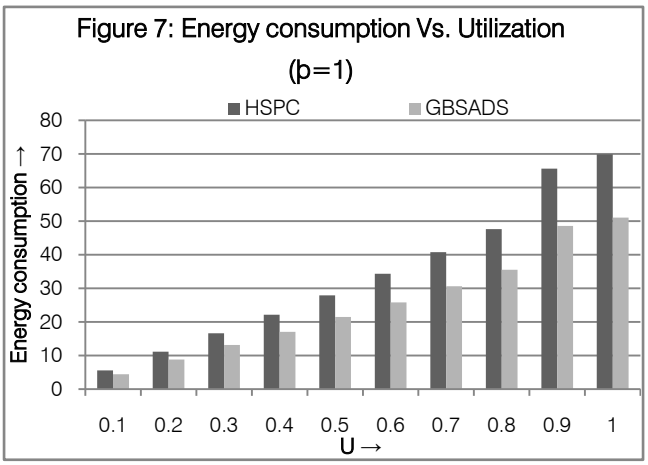


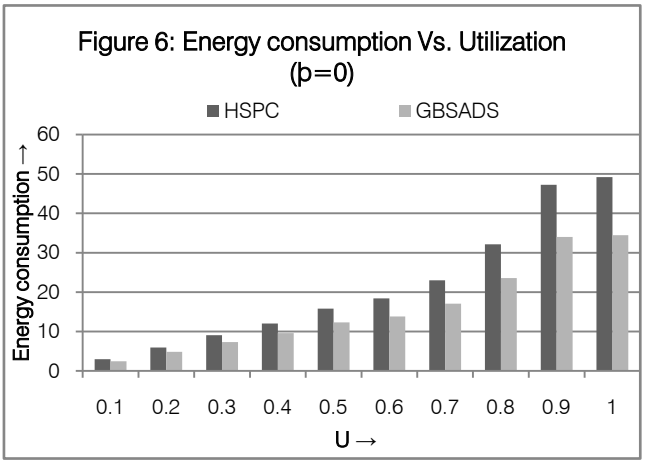
at three levels (speed assignment, greedy based preemption control and speed adjustment) thus, a gain of 20% is received. However, at the higher ratio the device energy consumption is dominant and hence DVS is less effective compared to the DPD technique. The GBSADS approach is able to accumulate the idle slots efficiently as it

does delayed start along with speed adjustment while controlling the preemption based on greedy approach whereas HSPC only controlled preemption.

Effect of system utilization on the energy consumption: The energy consumption is measured as the system utilization increases for different values of Þ. The value of Þ indicates the dominance of the device energy consumption on the overall energy consumption of the system (higher its value more the system is device dominated). It can be observed from all the following figures (6, 7 and 8) that when the utilization is high (0.8-1) then the reduction in the energy consumption is substantial. This is because for such utilizations the system is overloaded hence, the speed assignment for the feasibility in the first phase is at higher speeds. The HSPC approach does not slow the once assigned speed while GBSADS approach may reduce the speed assigned to the out of phase jobs substantially leading to reduction in the energy consumption. Besides speed adjustment it also delays the start and controls the preemption of lower priority jobs to accumulate the idle slots favoring the sleeping off the components.

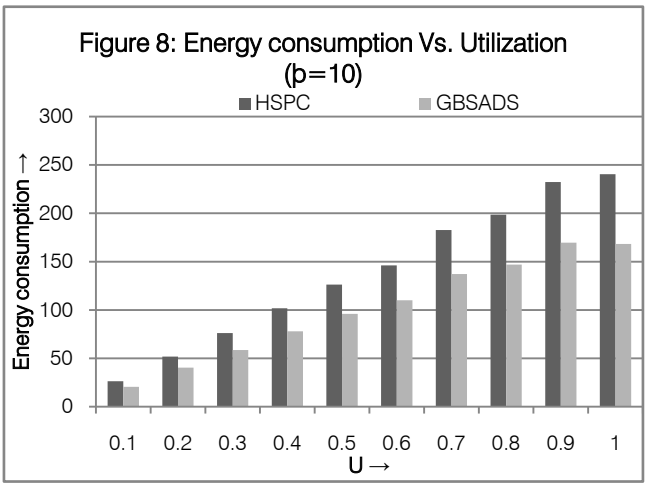
Effect of only processor energy consumption (when no devices are attached Þ=0): When no frequency independent components are associated with the system then the effect of the utilization on the system energy consumption can be seen in the figure 6. For lower utilization (0.1-0.3) the GBSADS approach consumes around 18% less energy while this reduction improves up to around 24% for medium utilizations (0.4-0.7) and still further up to approximately 30% for higher utilizations. For lower utilizations the speed assigned by both approaches in first phase is close to the critical speed and hence, energy saving by GBSADS is only due to the delayed start in the third phase which accumulates the fragmented idle slots and favor the processor to switch into the sleep mode (or sleep for longer time). For task sets with higher utilization, the speed assigned to a task in the first phase are generally higher than its critical speed due to overloading of the system by both the approaches. For reducing the energy consumption the HSPC approach the execution of the preempted jobs at either higher or at the same assigned speed. Executing preempted jobs at higher





speed of such systems having no devices attached would be counter productive while execution at the assigned speed would not incur any reduction in energy. On the other hand, GBSADS would adjust the speed (may reduce the assigned speed) taking into the account the thresholds and the idle slots in the second and the third phase so as to balance the impact of DVS, DPD and PC techniques.

When the device to processor rate of energy consumption is comparable Þ=1: The effect of the utilization on the overall energy consumption can be seen from the figure 7. The trend of the energy consumption is similar to that observed in the figure 6. But for higher utilizations the reduction in the energy



consumption is less (approximately 26%) as compared to 30% in figure 6. This is due to the fact; when higher speeds are assigned in the phase-1 reducing the speed by the GBSADS approach increases the response time of a job which would in turn force the devices to remain active for longer period and consume energy hence, lower gain is observed when compared to system comprising of frequency dependent components only.

When the device to processor rate of energy consumption is ten times (device dominated) Þ=10: The effect of utilizations on the energy consumption of a device

dominated system can be observed in the figure 8 which is similar to the trend seen in figure 6 and 7 in which at higher utilizations GBSADS approach performs better than HSPC approach.

V. Conclusion

In this paper we presented a three phase scheduling algorithm which minimizes the system energy consumption for weakly hard real-time system while maintaining the (m,k) guarantee. The system consists of a DVS processor (capable of operating at various frequencies) and frequency independent peripheral devices. We proposed a three phase scheduling algorithm where in the first phase a mixed pattern based partitioning is used to determine the mandatory and optional jobs of a task and assign speed levels to ensure the feasibility of the task set.

However, the major contribution of the work lays in the second and third phase which analyses and refines the first phase schedule at job level. In the second phase we formulated a greedy based preemption control technique which adjusted the speed of the preempted/preempting jobs based on the laxity to further reduce the energy consumption. The third phase focused on accumulation of the idle slots through utilizing the concept of delay start and speed adjustment. The speed adjustment is a method of assigning an optimal speed to individual job based on the availability of idle slot on the either side of the execution window of a job and the threshold of the components. While delayed start technique delays the execution of a job up to its available slack time to assimilate the idle slots fragmented on the either side of a job's execution window. The effectiveness of the proposed algorithm has been discussed through examples and extensive simulation results.

The proposed three phase scheduling algorithm is compared with [29] where the authors have adopted similar scenario. The simulation results indicate that the three phase scheduling algorithm consumes approximately 30% less energy for task set at higher utilizations (0.8-1) while it is 24% better for lower utilization systems (0.1-0.7). The reduction in the energy consumption is 30% for higher values of the threshold of a component while lesser improvement is observed approximately 23% for lower threshold value. The proposed algorithm was targeted for device dominant systems for which it performed 26% better. However, the simulations indicate that the approach is valid for processor dominant systems as well for which an improvement of about 20% is received. Thus, the proposed algorithm is capable of performing better in the system/process energy constrained systems when the system is overloaded (utilization is high) or the threshold of the components are high.

References Références Referencias

1. M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," IEEE Trans. Compute., vol. 44, no. 12, pp. 1443–1451, Dec. 1995.
2. D. Moss´e, H. Aydin, B. Childers, and R. Melhem, "Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications", Workshop on Compiler and OS for Low Power, 2000.
3. Q. Qiu, Q. Wu and M. Pedram, "Dynamic Power Management in a Mobile Multimedia System with Guaranteed Quality-of-Service", ACM/IEEE Design Automation Conference, pp. 834-839, 2001.
4. G. Qu and M. Potkonjak, "Power Minimization Using System-Level partitioning of Applications with Quality of Service Requirements", IEEE/ACM International Conference on Computer-Aided Design, pp. 343-346, 1999.
5. G. Quan and X. Hu, "Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors", 38th IEEE/ACM Design Automation Conference, pp. 828-833, 2001.
6. S. Hua and G. Qu, "Energy-Efficient Dual-Voltage Soft Real-Time System with (m, k)-Firm Deadline Guarantee", CASES'04, Washington, DC, USA pp 116- 123, September 22–25, 2004.
7. L. Doherty, B. Warneke, B. Boser, and K. Pister, " Energy and performance considerations for smart dust" International Journal of Parallel Distributed Systems and Networks, 2001.
8. Douglis, F., Krishnan, P., and Marsh, B.Thwarting, "The power-hungry disk", proceedings of the Winter USENIX Conference, pp. 292-306, 1994.
9. M. A. Viredaz and D. A. Wallach. Power evaluation of a handheld computer. IEEE Micro, pp. 66–74, 2003.
10. J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R.Wang., "Modeling hard-disk power consumption", FAST '03, pp. 217–230, 2003.
11. L. Niu., G. Quan, "Energy minimization for real time systems with (m, k)- guarantee" IEEE Tans. On Very large scale integrated (VLSI) systems, vol. 14, no. 7 July 2006.
12. M. Weiser, B. Welch, A. Demers and S. Shenker., "Scheduling for Reduced CPU energy", USENIX Symposium on Operating Systems Design and Implementation, 1994.
13. H. Aydin, V. Devadas, D. Zhu, "System-level Energy Management for Periodic Real-Time Tasks", Proceedings of the 27th IEEE

International Real-Time Systems Symposium (RTSS'06).
14. E. Bini, G.C. Buttazzo and G. Lipari, "Speed Modulation in Energy-Aware Real-Time Systems", in Proc. of the 17th Euromicro Conference on Real-Time Systems (ECRTS), 2005.
15. K. Choi, R. Soma and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times", in Proceedings of Design, Automation and Test in Europe, 2004.
16. K. Seth, A. Anantaraman, F.Mueller and E. Rotenberg., "FAST: Frequency-Aware Static Timing Analysis", in Proc. of the 24th IEEE Real-Time System Symposium, 2003.
17. X. Huang and A. M. K. Cheng, "Applying Imprecise Algorithms to Real-Time Image and Video Transmission", Real-Time Technology and Applications Symposium, Chicago, USA pp. 96-101, 15-17 May 1995.
18. Xiao Chen and Albert Mo Kim Cheng, "An Imprecise Algorithm for Real-Time Compressed Image and Video Transmission" Sixth International Conference on Computer Communications and Networks, Proceedings, Las Vegas, NV, USA , pp. 390-397.
19. A. F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in Proc. AFCS, pp. 374–382, 1995
20. L. Niu and G. Quan, "Energy-Aware Scheduling for Real-Time Systems With (m; k)-Guarantee", Dept. Comput. Sci. Eng., Univ. South Carolina, Tech. Rep. TR-2005-005, 2005.
21. G. Bernat and A. Burns, "Combining (n;m)-hard deadlines and dual priority scheduling," in Proc. RTSS, Dec. 1997, pp. 46–57
22. W. Kim, J. Kim, and S. L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack analysis," in Proc. DATE, pp. 788, 2002
23. S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority Real-time Systems", in Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03), 2003.
24. Y. Zhang and K. Chakraborty, "An Unified approach for fault-tolerance and dynamic power management in fixed-priority real-time embedded systems", IEEE Transactions on Computer- Aided Design of Integrated Circuit and Systems, Vol. 25, No. 1, January 2006.
25. H. Aydin, R. Melhem, D. Moss´e and P. Mejia-Alvarez, "Dynamic and Aggressive Power-Aware Scheduling Techniques for Real-Time Systems", in Proceedings of the 22nd IEEE Real-time Systems Symposium (RTSS'01), 2001.
26. X. Fan, C. Ellis, and A. Lebeck, "The Synergy between Power aware Memory systems and Processor Voltage", in Workshop on Power-Aware Computing Systems, December 2003.
27. R. Jejurikar and R. Gupta, "Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems", ISLPED 2004.
28. J. Zhuo and C. Chakrabarti, "System level energy efficient dynamic task scheduling", DAC, 2005.
29. S. Agrawal, R. S. Yadav, Ranvijay, "A Preemption Control Technique for System Energy Minimization of Weakly Hard Real-time Systems", SNPD 2008.
30. M. Kim and S. Ha, "Hybrid run-time power management technique for real-time embedded system with voltage scalable processor", OM'01, pages 11–19, 2001.
31. [31] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems", ISLPED, 2004.
32. [32] H. Cheng and S. Goddard., "Online energy-aware i/o device scheduling for hard real-time systems", DATE, 2006.
33. P. Rong and M. Pedram. Hierarchical power management with application to scheduling. ISLPED, 2005.
34. V. Swaminathan and K. Chakrabarty, "Pruning-based, energy optimal, deterministic i/o device scheduling for hard real-time systems". Trans. on Embedded Computing Sys., ACM Transactions on Embedded Computing Systems, Vol. 4, No. 1, February 2005, Pages 141–167.
35. L. Niu and G. Quan, "System-wide dynamic power management for multimedia portable devices", accepted by IEEE International Symposium on Multimedia (ISM'06), 2006.
36. L. Niu and G. Quan, "Peripheral-Conscious Scheduling on Energy Minimization for Weakly Hard Real-time Systems", DATE07.
37. Liliana Cucu and Jo¨el Goossens, "Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary deadline Periodic Systems", DATE07.
38. Y. H. Lu and G. D. Micheli, "Comparing system-level power management", IEEE Design and Test of Computers, March-April 2001.
39. G. Quan and X. Hu. "Enhanced fixed-priority scheduling with (m, k)-firm guarantee". In RTSS, pages 79–88, 2000.

40. G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," in Proc. RTSS, 1995, p. 110.
41. P. Ramanathan, "Overload management in real-time control applications using (m; k)-firm guarantee," IEEE Trans. Parallel. Distrib. Syst., vol. 10, no. 6, pp. 549–559, Jun. 1999.
42. http://www.purplemath.com/modules/drofsign.htm
43. R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems", in Proc. of the Design Automation Conf, pp. 275–280, 2004

This page is intentionally left blank