JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 1599-1615 (2018)
DOI: 10.6688/JISE.201811_34(6).0013

Energy Efficient Online Scheduling of Real Time Tasks on Large Multi-threaded Multiprocessor Systems

MANOJIT GHOSE, ARYABARTTA SAHU AND SUSHANTA KARMAKAR
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Assam, 781039 India
E-mail: {g.manojit; asahu; sushantak}@iitg.ernet.in

Today's large computing systems are empowered with high processing capabilities and they are often used to run real time applications. But these systems consume huge amount of energy while executing these applications. In this paper, we have exploited the actual power consumption pattern of a few recent commercial multi-threaded processors and derived a simple power model which considers the power consumption at a coarser granularity instead of finer granularity like DVFS. We have then proposed an online energy efficient task scheduling policy namely, smart allocation policy for scheduling aperiodic independent real time tasks onto such large systems having multi-threaded feature in the processors. We have further added three variations of our proposed policy to efficiently address different situations which can occur at execution time and to further reduce energy for some kinds of applications. We have analyzed the instantaneous power consumption and the overall energy consumption of four proposed task allocation policies along with other five baseline policies for a wide variety of synthetic data sets and real trace data considering different computation time models and deadline schemes. Experimental results show that our proposed policies achieve average energy reduction of 45% (maximum up to 92%) for synthetic data set and 30% (maximum up to 47%) for real data sets as compared to baseline policies. All the proposed policies ensure that no task misses its deadline.

Keywords: real time tasks, online scheduling, energy efficient, multi-processor systems, multi-threaded systems

1. INTRODUCTION

Now a day, the number of processors and number of threads per processor have increased to a significant number in computing systems. Thus the processing capabilities of these systems are sufficient enough to handle most of the recent applications. But the major concern in these computing systems is the growing energy consumption; which has sought the attention of the research community to a great extent. These compute systems include battery operated mobile devices, desktops and servers. Though the processing capabilities of most of the data centers are sufficient enough, they remain under utilized most of the times. This is same with the mobile devices and desktop computers. These underutilized systems unnecessarily consume a significant amount of power (or energy). The power consumption in servers and workstations increases then heat dissipation also increases. Thus reducing the power consumption in servers and workstations not only reduces the operating electricity cost but also lowers the cooling cost. In many computing systems, we require to deal with real time tasks where executing the tasks

Received March 9, 2017; revised June 11, 2017; accepted October 11, 2017.
Communicated by Ce-Kuen Shieh.

before their deadline is essential. Scheduling real time tasks in multiprocessor domain is a promising research area in recent time. But the traditional multiprocessor real-time scheduling algorithms aim to (a) improve utilization bound, (b) reduce approximation ratio, (c) reduce resource augmentation and (d) improve some empirical factors, like total schedule length, total laxity, number of deadline misses, *etc.* [1, 2]. But the current research trend is to associate the power consumption of the processors with scheduling and the goal is to design scheduling algorithms which reduce the power (or energy) consumption of the processing elements. These algorithms are commonly termed as power aware scheduling or energy efficient scheduling [3, 4].

Traditional power aware (or energy efficient) scheduling techniques use the dynamic voltage and frequency scaling (DVFS) to design their algorithms [5]. They consider the power model as: $P \propto f^3$ where P is the power consumption of a processor and f is the operating frequency of that processor. DVFS focuses on power consumption behavior of the processor at a finer granularity. On the other hand, in large scale computing systems, we need to consider the power consumption model at a coarser granularity. In the era of dark silicon, power consumption behavior is handled at the higher granularity by completely switching off or on a computing region based on the requirements to minimize the power consumption [6, 7]. Similar concepts can be seen profoundly in cloud systems which manage the virtual machines [8, 9]. In this case, the common idea is to run as less physical nodes as possible.

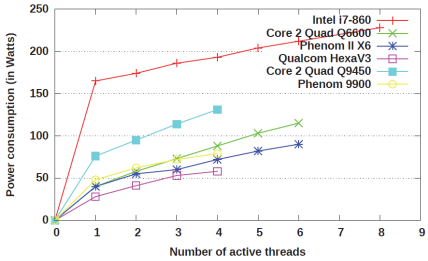


Fig. 1. Power consumption of various commercial processors with threads [10, 11].

Fig. 1 shows a plot of the power consumption of various commercial processors with different number of active hardware threads. We can easily observe a relationship between the power consumption of a processor and the number of active hardware threads of that processor. Processor consumes a significant amount of power when it gets started and runs 1 thread; after that the power value increases almost linearly with increase in number of active threads till the thread number reaches processor's maximum capacity. The power consumption of Qualcom Hexa V3 with 1 active thread is 40 watts. When it runs two threads, the power consumption value becomes 55 watts. This 15 watts (= 55 − 40) is the extra power consumption for the 2nd thread. Similarly, subsequent threads consume additional power values as 5 watts (= 60 − 55), 12 watts (= 72 − 60), 10 watts (= 82 − 72) and 8 watts (= 90 − 82) respectively. Observing this power consumption pattern of the recent processors, we have derived a simple power model for a multi-threaded single processor as described in Section 3. To the best of our knowledge, no scheduling algorithm on real time tasks were proposed considering the thread based

power consumption pattern of commercial processors as explained in Fig. 1. We believe that our derived power consumption model is well suited for Large Multi-threaded Multiprocessor Systems (LMTMPS).

Our contributions in this paper are (i) we have exploited the power consumption pattern of the commercial processors and derived a simple power consumption model, (ii) we have designed four online energy efficient scheduling policies namely (a) *smart allocation policy*; (b) *smart Early Dispatch allocation policy*; (c) *smart − Reserve allocation policy* and (d) *smart − Handling Immediate Urgency allocation policy* which use the derived power model to reduce the overall energy consumption while meeting the deadline constraints of all the tasks, and (iii) we have compared and analyzed the performance of the proposed policies with five baseline policies and found that the proposed policies perform better than others.

Rest of the paper is organized as follows: Section 2 provides a brief description of previous work in the area of power aware or energy efficient scheduling techniques. Section 3 describes the problem formulation and system model. Sections 4 and 5 explain different existing and proposed scheduling policies. Section 6 contains the experiment details and the paper is concluded in Section 7.

2. PREVIOUS WORK

In this section, we present a brief overview of the research works done in the context of energy efficient or power-aware scheduling. The work can be classified into mainly two categories (a) fine-grained approach; (b) coarse-grained approach. In fine-grained approach, research mainly targets to reduce the power consumption of the cores by exploiting the DVFS techniques. This approach is driven by the fact that the actual computation time of a task is often less than the worst case computation and the algorithms make use of this slack time in different ways. On the other hand, coarse-grained approach works at the processor level for the small systems and at the host level or server level for the large systems (host or server have many processors). Coarse-grained approach mainly focuses on reducing the number of active hosts and putting the passive hosts in low power state or in sleep mode. Weiser *et al.* [12] was pioneer to start the research in this direction by associating power consumption with scheduling and used DVFS technique to study the power consumption of some scheduling techniques. They took advantage of CPU idle time and reduced the operating frequency of CPU so that the tasks were finished without violating any deadline. Lee and Zomaya [13, 14] have proposed makespan-conservative energy reduction along with simple energy conscious scheduling to find a trade-off between the makespan time and energy consumption. Recently, Li and Wu [15-17] have considered execution of various task models by further exploiting the DVFS technique for both homogeneous and heterogeneous processor environments. All of these studies consider the DVFS properties for the processors and do not consider the multi-threaded feature of the processors.

Chase *et al.* [18] proposed a coarse-grained power management technique for Internet server clusters by dynamically adjusting the list of active servers. Srikantaiah *et al.* [19] studied the relationship between the energy consumption and the performance of the system which was determined by the CPU (or processor) utilization and disk utilization.

The problem was viewed as a bin packing problem where applications are mapped into servers optimally. Verma *et al.* [20] studied the power consumption pattern of various HPC applications and the typical workload behavior of a virtualized server and in [21] they developed a framework called pMapper where the applications are placed onto different servers for execution based on the utilization values of the servers.

Our work is similar to the above mentioned research but it differs from others in the sense that we have considered the execution of real time tasks in large systems where we try to minimize the overall energy consumption with a guarantee of real-time constraints in a multi-threaded multiprocessor environment.

3. PROBLEM FORMULATION AND SYSTEM MODEL

3.1 Problem Formulation

We wish to design scheduling policies for online aperiodic independent tasks onto large multi-threaded multiprocessor systems such that no task misses its deadline and the total energy consumption of the system is minimized. In this work, we assume that the compute capability of the system is high *i.e.* the number of processors is taken as sufficiently large. Power consumption in the multi-threaded multiprocessor systems under the considered model is not proportional to the utilization of the system (or number of active hardware threads on the system) as there is static component in the power. In this work, we have utilized this power consumption behavior to minimize the total energy consumption of the system.

3.2 System Model

Fig. 2 shows the block diagram of our considered system which schedules and executes a set of online aperiodic real time tasks on large multi-threaded multiprocessor system. Tasks arrive to the system dynamically (online) and the scheduler schedules the tasks on to the processing system. Our considered processing system consists of M number of multi-threaded processors (P_0, P_1, ..., P_M) where M is sufficiently large. It is justified to assume a system with such large number of processors, as the modern days computing systems like cloud system, promise virtually infinite resources to all the applications.

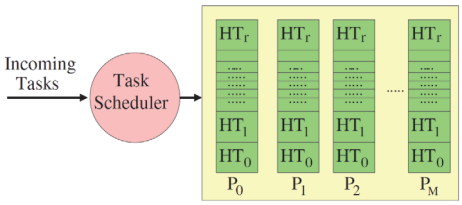


Fig. 2. Scheduling of online aperiodic tasks on LMTPS.

All the processors are homogeneous, non-DVFS capable and each processor is multi-threaded having r hardware threads. In this paper, hardware thread of a processor is

also termed as virtual processor. We have also assumed that one task gets allocated to exactly one hardware thread of a processor and one thread can execute only one task at a time.

As mentioned in Section 1 of the paper, we have exploited the power consumption behavior of a few commercial multithreaded processors and derived a simple power model where each processor consumes a significant amount of power as it gets started. This significant amount is termed as the base power consumption of the processor. Each active thread contributes some amount to power called the thread power consumption. The processor power consumption of such large multi-threaded multiprocessor systems (LMTMPS) can be modeled as described in Eqs. (1) and (2).

For a single multi-threaded processor, power consumption can be modeled as:

$$P_s = C + id \tag{1}$$

where, P_s = power consumption of a single processor, C = base power consumption of the processor, i = number of active hardware threads and δ = thread power consumption. In general, the value of C is greater than δ. Typical values of C is 5 to 10 times of δ [12, 13].

Similarly, the power consumption for the whole LMTMPS can be modeled as:

$$P_{LMTMPS} = L(C + r\delta) + (C + i\delta). \tag{2}$$

Here, L = number of switched-on processors which are utilized fully and r is the maximum number of hardware threads a processor can run.

The first part of Eq. (2) represents the total power consumption of the fully utilized processors and the second part ($C+i\delta$) indicates the total power consumption of the partially filled processor. If there is no partially filled processor, then the second part of the equation evaluates to zero. Fully utilized (filled) processor means all the hardware threads of the processor are active and executing some tasks, otherwise the state of the processor is partially filled or partially utilized. Here we assumed that our task scheduler regularly (or frequently) runs through the system and consolidates the running tasks into fewer numbers of processors such that (a) only 1 processor or no processor will be partially filled, and (b) other processors will be either completely filled or switched off. Fig. 3 shows the power consumption behavior of LMTMPS. We can see that there is a sharp jump in power consumption value when the number of virtual processors (or active hardware threads) is 1, 9 and 17.

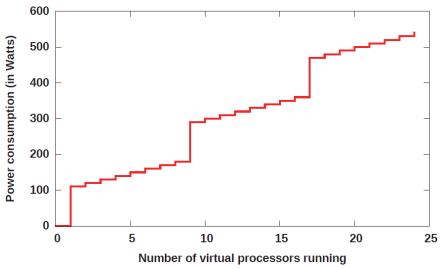


Fig. 3. Power consumption of LMTMPS C = 100, δ = 10 and r = 8.

3.3 Task Model

In this paper, we have considered scheduling a set of online independent aperiodic real time tasks, $T = \{T_1, T_2, \ldots, T_N\}$ onto a large multi-threaded multiprocessor system. Each task T_i is characterized by its arrival time (a_i), computation time (c_i) and deadline (d_i) where $d_i \geq (a_i + c_i)$. We assume that all the tasks are sequential and it can be run by only one hardware thread (virtual processor) at any time instant. In this paper, we have considered scheduling of both synthetic real time tasks and real trace data which are described in following subsections.

3.3.1 Synthetic tasks

We assume that a set of independent aperiodic real time tasks are arriving to the system in such a way that the inter-arrival time between any two consecutive tasks follow discretized Gaussian distribution. As the execution (or compute) time of tasks is a key factor in scheduling and it has a significant impact on the overall performance of the system [22, 23], we have considered a wide variety of task computation time with different distributions in order to establish the effectiveness of our work. These distributions are: **Random**, **Gaussian**, **Poisson** and **Gamma**. We have further considered computation time as a function of time or task sequence number: ***INC*(*i*)** and ***DEC*(*i*)**. In *INC*(*i*), computation time of tasks increases with task number. *INC*(*i*) is taken as $k.i$, $i \in \{1, 2, ..., N\}$ and $c_i \geq c_i - 1$. Similarly, in *DEC*(*i*), computation time of tasks decreases with task number. *DEC*(*i*) is taken as $k.(N + 1 - i)$, $i \in \{1, 2, ..., N\}$ and $c_i \leq c_i - 1$. In addition to these variations of computation time, we have considered five different variations in deadline of tasks. For a task T_i (a_i, c_i, d_i), $d_i - (a_i + c_i)$ is called the slack time (*SLK*) of the task. Variation in deadline can be achieved by considering variation in *SLK*. Similar to the former case, we considered **Random** and **Gaussian** distribution, ***INC*(*i*)** and ***DEC*(*i*)** function over *SLK*. We have also considered a special kind of deadline schemes where deadline for all the tasks is same. We name it as **common deadline** scheme. This can be written as $d_i = D$; where D is the common deadline for all the tasks satisfying the condition $D \geq max\{a_i + e_i\}$. This is also termed as common due date problems in literature [24].

3.3.2 Real workload traces

In addition to synthetic real time tasks with many variations, we have also considered four different real-life workload traces (or logs) [26] in our work. These workload traces are generated from TORQUE and PBSpro traces in different times which contain the job descriptions of different clusters in Standard Workload Format (SWF). These are (i) CERITSC: 17,900 jobs, collected during Jan-Mar 2013, (ii) MetaCentrum-1: 495,299 jobs, collected during Jan-June 2013, (iii) Zewura: 17,256 jobs, Jan-May 2012, and (iv) MetaCentrum-2: 103,656 jobs, collected during Jan-May 2009. These logs and data sets are provided by the Czech National Grid Infrastructure MetaCentrum [25].

4. STANDARD TASK ALLOCATION POLICIES

This section contains description of a couple of existing task allocation policies.

Utilization based workload consolidation is one of the state of art policy.

• **Utilization Based Allocation Policy (UBA)**

In this policy, tasks are allocated to the processors based on their processor utilization value and all the tasks finish exactly at their deadline. This policy provides the minimum processor share (*i.e.* utilization) required for a task for all the time instants starting from its arrival time till its deadline. Thus the execution of a task T_i spreads for the whole duration with minimum processor requirement u_i at every time instant between a_i and d_i. This policy is mainly of theoretical interest and the assumption made in Section 3 that one thread can execute exactly one task at a time does not hold here.

• **Front Workload Consolidation (FWC)**

In this policy the execution of tasks are consolidated towards beginning of the time axis. As soon as a task arrives, it is allocated to a thread of a processor. As all the tasks start their execution immediately after they enter the system, the policy can also be termed as *Immediate Allocation Policy* or as soon as possible (ASAP) policy. This policy turns out to be a non-preemptive on and there is no *migration* of tasks. In this policy, start time (s_i) and finish time (f_i) of a task $T_i(a_i,c_i,d_i)$ become $s_i = a_i$ and $f_i = a_i + c_i$ respectively.

• **Rear Workload Consolidation (RWC)**

This policy is similar to the front work consolidation policy, but here the tasks are consolidated towards the end of the time axis. Every task execution gets consolidated towards the deadline of the task. This policy is also non-preemptive and does not require migration for system with sufficiently large number of single threaded processor. As the tasks enter the system, they are accumulated and remain in the waiting queue as long as their deadlines permit them to wait. This policy ensures that all the tasks can finish their execution just in time (*i.e.* on their deadlines). As the execution of all the tasks are delayed till their urgent points, the policy can also be termed as *Delayed Allocation Policy.*

• **Utilization Based Workload Consolidation (UBWC)**

In this policy, the scheduler tries to schedule the currently arrived task $T_i(a_i, c_i, d_i)$ at any time instants (slots) in such a way that the task does not miss its deadline and there is minimum number of increase in the active processor count. For the task T_i, the UBWC schedules c_i units of unit time compute slot between the current time a_i and deadline of the task d_i, so that the increase in number of active processors count will be minimum in all the time slots between a_i and d_i [19, 26, 27]. Processor count in a time slot increases when the total utilization of that time slot crosses a whole number. This scheduling policy requires preemption and migration of the tasks; but the number of preemptions and number of migrations for a task T_i is bounded by $(c_i - 1)$ if we assume the time axis is discretized in unit time slot.

• **Earliest Deadline First Allocation Policy (EDF)**

Earliest deadline first (EDF) is a well-known scheduling policy where tasks are considered based on their deadline values. Task with the earliest deadline value is executed first [2]. At any point of time t, we consider all the arrived tasks which have not

started their execution. Out of these waiting tasks, task with the minimum deadline value is chosen for execution. We need to use minimum number of processors at any instant of time such that no task misses its deadline.

5. PROPOSED TASK ALLOCATION POLICIES

We have proposed four energy efficient online task scheduling policies for executing a set of aperiodic independent real time tasks onto LMTMPS, where instantaneous system power consumption (IPC) is not proportional to instantaneous utilization of the system. The power consumption model of LMTPS is described in Section 3.2. Our proposed policies take advantage of this non-proportionality of IPC to the instantaneous utilization. In our designed policies, execution of a task is almost continuous. Ignoring the migration time, the time difference between the finishing time (f_i) and start time (s_i) of any task T_i is same as the execution (or compute) time of the task, that is $(f_i - s_i) = c_i$. But in the utilization based work consolidation (UBWC), this difference is more than the given computation time for all the tasks, *i.e.* $(f_i - s_i) \geq c_i$.

5.1 Smart Allocation Policy (Smart)

We have designed an allocation policy called smart allocation policy which benefits of the non-proportionality of IPC to instantaneous utilization of the system. The scheduling policy works on two basic ideas: (a) number of active processors should be as less as possible (active processors mean the processors which are switched on) and (b) whenever a processor is required to switch on, it should be utilized to its maximum capacity by assigning tasks to all the hardware threads of that processor if sufficient tasks are available in the system. When there is no free hardware thread in any active processor, execution of the tasks should be delayed as much as possible without missing their deadlines.

Pseudo-code of smart allocation policy is shown in Algorithm 1. At every time instant, the policy finds all the urgent tasks and allocates to free hardware threads of active processor if exists. Otherwise new processor is switched on and new hardware threads are initiated on it to allocate such tasks. On the other hand, if there is no urgent task present in the system but there is a partially filled processor, then all the hardware threads of active processors are fully utilized by allocating tasks from the waiting queue based on some policy (for simplicity we have used FCFS). The scheduler performs a consolidation operation in periodic basis. All the active hosts are sorted based their utilization (that is number of active threads). Then tasks from the least utilized hosts are migrated to the most utilized host which has a free thread. Consolidation operation is repeated such that at most one processor remains partially filled. Then all idle processors are switched off and removed from the active list. Line number 18 to 22 in the pseudo code reflects the same. The parameter consolidation Interval determines the frequency of consolidation operation in the scheduling process.

Algorithm 1 Smart allocation policy

1: timestamp $t \leftarrow 0$;

2: **while** true **do**
3: **if** any task arrives, then Add it to Waiting Queue **WQ**.
4: **while WQ** is NOT Empty **do**
5: **while** T_i = *findUrgentTask()* is not NULL **do**
6: **if** free hardware thread exists in any active processor **then**
7: Allocate task T_i to a free hardware thread.
8: **else**
9: Switch on a new processor and initiate a hardware thread.
10: Allocate the task T_i to the hardware thread.
11: $s_i \leftarrow t; f_j \leftarrow t + c_i$; **WQ.remove**(T_i); ◅ t is current time
12: **while** there is free hardware threads in any processor **do**
13: Choose a task T_j from waiting queue based on the some policy.
14: Assign the task T_j to a free hardware thread of that processor.
15: $s_j \leftarrow t; f_j \leftarrow t + c_j$ **WQ.remove**(T_i);
16: **if** any task finishes its execution, **then** update system information; $t \leftarrow t + 1$;
17: **if** (t % *consolidationInterval* == 0) **then**
18: Sort the active processors in ascending order of their utilization.
19: Migrate tasks from least utilized processor to most utilized processor.
20: Repeat step 22 until there is at most one partially filled processor.
21: Turn off the idle processors.

procedure *findUrgentTask()*
1: **for** each task T_i in Waiting Queue **do**
2: **if** $d_i == c_i + t$ **then** **return** T_i ◅ t is current time
3: **end for**
4: **return** NULL

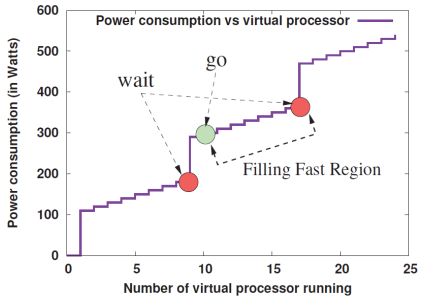


Fig. 4. Extra annotation to Fig. 3 to describe the smart allocation policy.

Fig. 4 shows the plot of total power consumption verses virtual processors similar to Fig. 3 with extra annotation to describe our smart allocation policy. In this case, every processor has r = 8 hardware threads (virtual processors) and each hardware thread consumes δ = 10 amount of power. Whenever a processor is switched on, it consumes 100 units and each additional thread consumes 10 units of power. This continues till number of active hardware threads reaches r = 8. At this point, the processor is utilized to its

maximum capacity and the total power consumption of the system is 180 (=100 + 8 ∗ 10) units. The intention of the policy is to keep the system at this point if the deadline constraints of the tasks allow it to do so. As soon as the number reaches 9, another processor needs to be switched on and the total power consumption of the system increases by 110 units and reaches a total of 290 (= 180 + 110) units.

We define two type of points called *wait* and *go* as appeared in Fig. 5. At point *wait*, the target of the policy is to delay the ready tasks as much as possible without missing any deadline. This is because, whatever processors are there in the system, they are utilized to their maximum capacity. And if another task is to be allocated, we need to switch on a new processor and the power consumption will increase and make a sharp jump to the next higher level. The other point is called the *go* point. This indicates the point when a new processor is already switched on with only one active hardware thread. The region between the *go* point and *wait* point is called the *Filling Fast* region. In this region, the system tries to allocate as much tasks as possible either from the waiting queue or the newly coming tasks till the *wait* point is reached Smart is an online scheduling policy and it takes scheduling decision dynamically at runtime. Even if the actual execution time of the tasks are not known before hand and tasks do not always consume worst case execution time, the smart policy can handle this dynamic situation because mapping of tasks to the hardware threads of a processor is done irrespective to the execution time of the tasks.

5.2 Smart Allocation Policy with Early Dispatch (Smart-ED)

In smart allocation policy, initially, all the arrived tasks wait in the waiting queue till the time instant when further waiting will result a deadline miss. At this time instant, at least one task becomes urgent. We name this time instant as urgent point. If the urgent points for many tasks (>> r) get accumulated to one time instant, then we need to switch on many processors and it will create a high jump in the instantaneous power consumption value. To handle such situation we have modified our proposed smart allocation policy and the modified policy includes early dispatch along with the smart allocation.

The basic idea of this policy is that whenever there are tasks in the system, they need to be executed. The smart policy would make the tasks wait in the system if their deadlines permit them to wait. But this policy does not make the tasks wait till their urgent points. In this smart allocation with early dispatch policy, at any time instant a new processor is switched on if the number waiting tasks ≥ r. After switching on the processor, r number of tasks from waiting queue is selected and are allocated to all the hardware threads of the newly switched on processor. The selection can be done using any policy.

Fig. 5 shows an example of task scheduling system, where 10 tasks (with $c_i = 4$) arrived to the system on or before time 3 and earliest deadline of the tasks is at time 9. So the scheduler switches on a new processor (assuming each processor has $r = 8$ hardware threads) and schedules execution of 8 tasks (based on EDF) on to the system at time 3. Based on EDF, the selected and scheduled tasks at time 3 are T_1, T_2, T_3, T_4, T_5, T_6, T_9 and T_{10} of hardware threads *hwt0* to *hwt7* of processor P_0. In this policy, at all the time instants, the scheduler selects the earliest deadline task and schedules onto an already switched-on processor if any hardware thread is free.

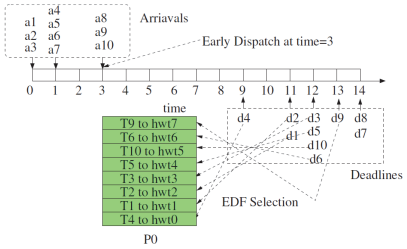
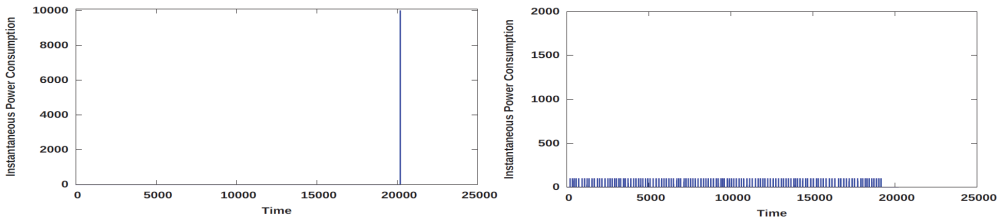


Fig. 5. Smart allocation policy with Early Dispatch (Smart-ED).

This policy efficiently handles the common deadline scheme where the sharp jump in the instantaneous power value near the urgent point is avoided. The policy tries to equally distribute the power consumption value across time line. Fig. 6 shows the instantaneous power consumption for both Smart and Smart-ED policy for the common deadline scheme. This figure clearly depicts the difference between these two policies. Smart policy shows a huge jump in the power consumption value from 0 to 12570 units at time 20155. This policy is preferred over the smart policy when the system is not capable of handling such high value (or a sudden jump) of power consumption. Another benefit of this policy is that it reduces the waiting time of the tasks but this is not within the scope of the paper and is not discussed here.



(a) Smart policy. (b) Smart-ED policy.

Fig. 6. IPC for common deadline scheme with $\mu = 20$, $\sigma = 10$.

5.3 Smart Allocation Policy with Reserve Slots (Smart-R)

This is a variation of the smart allocation policy where a fraction of the processor capacity is reserved for future urgent use. In the filling fast region (as described in Fig. 4), all the free hardware threads of the processor was filled with waiting tasks in basic smart policy. But in this policy, a few threads (called *Reserve Factor*) are kept free such that they can execute suddenly occurring urgent tasks (for these tasks slack time $(d_i - (c_i + a_i))$ is almost zero). This reduces the number of processors to be switched on (by compulsion) for servicing the suddenly occurring urgent tasks. This in turn might reduce the power consumption of the system. This policy will be highly beneficial for the applications where some critical tasks (having tight deadline) arrive in between regular tasks.

5.4 Smart Allocation Policy with Handling Immediate Urgency (Smart-HIU)

In the baseline smart allocation policy, when there is free hardware threads (repre-

sented by fast filling region in Fig. 4), the policy selects some tasks from the waiting queue using FCFS in order to utilize the free hardware threads. But it might so happen that FCFS method selects tasks whose deadlines are relatively far. The near deadline tasks will eventually become urgent in near future. This urgency may force the system to start a new processor. Again it is already discussed earlier that switching on a processor by compulsion generally increases energy consumption and it is always beneficial to avoid the compulsion scenario.

So in this modified smart allocation policy (Smart-handling immediate urgency allocation policy), the scheduler selects and executes the tasks whose deadlines are comparatively near. That is tasks with earliest deadline (immediate urgent) from waiting queue is selected to utilize the free hardware threads in fast filling region. This in turn results in forming a long time gap between the current time and time of occurrence of next urgent point. This long time gap allows the scheduler to avoid switching on a new processor and this eventually reduces the instantaneous power consumption of the system.

6. EXPERIMENTS AND RESULTS

6.1 Experimental Setup, Machine and Task Parameters and Migration Overhead

We have created a simulation environment to simulate large multi-threaded multiprocessor system for carrying out our experiments where number of processors, number of threads per processor, base power consumption of a processor, power consumption per thread can be varied. Our simulation environment generates a wide range of output statistics, including instantaneous power consumption for all time instants/slots and the overall power consumption for different input parameters and task allocation policies. Since existing energy efficient scheduling techniques for large systems are not directly comparable to our work (to the best our knowledge), the comparison is carried out with the standard task allocation policies.

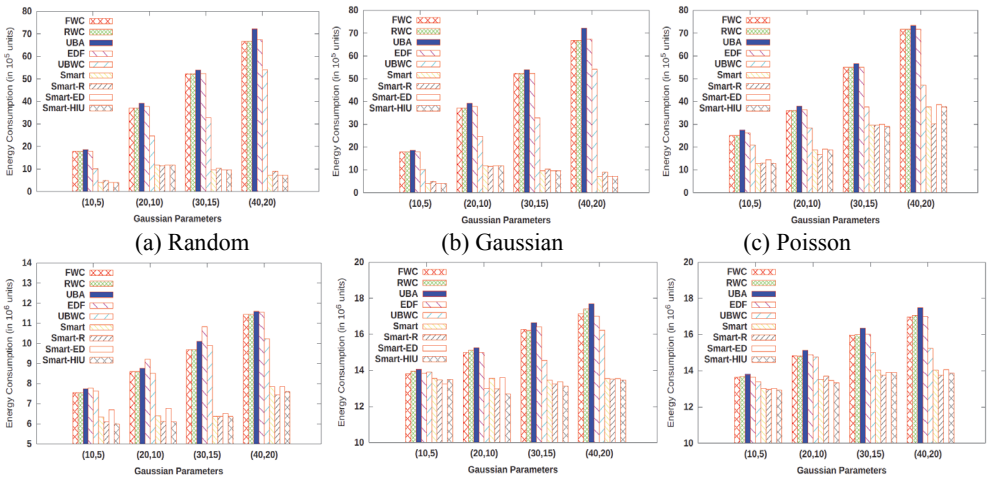
We have considered the total power consumption of a processor is 100 units and each processor can run up to 8 hardware threads, the base power consumption of a processor is taken as 70 units (*i.e.* 70% of 100 as reported in [28]) and per thread power consumption is taken as 3.75 units (*i.e.* (100 − 70)/8). We have performed experiments in our simulation environment using both real workload trace data and synthetic data sets as described in Section 3. For synthetic data set, we used several pairs of (μ, σ) values *e.g.* (10,5), (20,10), (30,15) and (40,20) for generating the arrival pattern of the tasks. The number of tasks generated for each experiment is 1000 and we used 10 such sets for all the cases. The parameters for computation time distributions are: for Random, C_{max}=100, for Gaussian μ = 100, σ = 20, for Poisson λ = 100, for Gamma α = 50, β = 10, for Increasing k = 2 and for Decreasing k = 2. We have also considered five different types of deadline schemes. In Random, Z_{max}=1000; in Gaussian, μ = 10, σ = 5; in Increasing, k = 3; in Decreasing, k = 3; and in Common, D = 10205.

Migration overhead in a system typically depends on various factors like total number of migrations, frequency of migrations, migration path, working set size. The overhead is expressed in terms of performance degradation of the system (decrease in Instruction per cycle), increase in execution time, *etc.* and the literature reported migra-

tion overhead as an average of 2 to 3% and context switch overhead as less than 0.1% [21, 29]. As the thread executing a task need to run for some additional amount of time, we assume an increase of 2.5% in thread power in case of migration and 0.1% in case of preemption as the overhead.

6.2 Result and Analysis

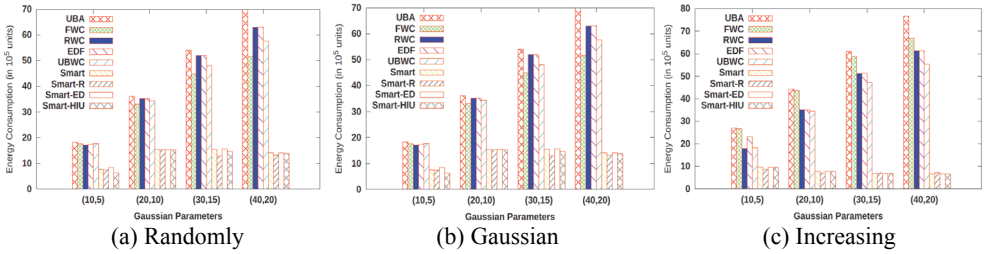
Figs. 7 (a)-(f) show the energy consumption of 1000 aperiodic tasks on considered large multi-threaded multiprocessor system using different allocation policies for synthetic task set under (a) random distribution; (b) Gaussian distribution; (c) Poisson distribution; (d) Gamma distribution; (e) increasing; and (f) decreasing computation time schemes respectively (random deadline scheme). We observe that the proposed policies perform better than all other baseline policies for all the computation time schemes. Similarly, Figs. 8 (a)-(e) show the total energy consumption of the system under (a) randomly distributed; (b) Gaussianly distributed; (c) increasing; (d) decreasing; and (e) common deadline schemes respectively (random computation time model). We can clearly observe that the energy consumption under our proposed policies is significantly lesser than all other baseline policies for all kinds of task models.



(a) Random (b) Gaussian (c) Poisson

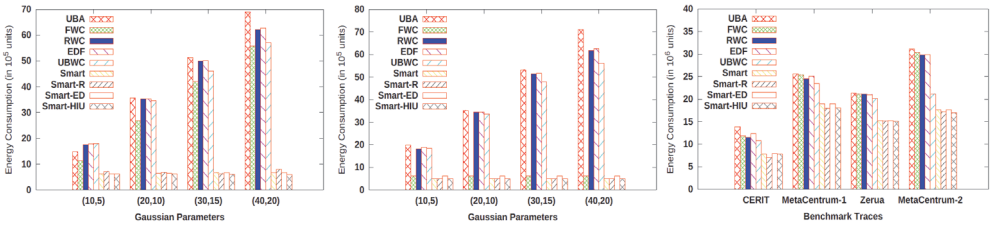
(d) Gamma (e) Increasing (f) Decreasing

Fig. 7. Energy consumption for different computation time schemes.



(a) Randomly (b) Gaussian (c) Increasing

Fig. 8. Energy consumption for different deadline schemes.



(d) Decreasing (e) Common (f) Real data sets

Fig. 8. (Cont'd) Energy consumption for different deadline schemes.

Energy consumption of the system is the summation of IPC over all the time slots. A low value of μ and σ indicates that the inter-arrival time of tasks are less, which signifies the system is overloaded or filled with many tasks. In such cases, opportunity to take the advantage of the smart allocation policy idea to save energy is thin. Hence the benefit is not that significant for low values of μ and σ. As the values of μ and σ increase, the inter-arrival time between tasks increases. Thus the system becomes loosely loaded and smart allocation takes the benefit of it.

Table 1. Noumbers of migrations in different allocation policies.

Data \ Policy		UBWC	Smart	Smart-ED	Smart-R	Smart-HIU
	Random	490	7	2	9	7
	Gaussian	624	17	27	32	15
	Poisson	527	10	16	28	9
	Gamma	37421	142	204	92	97
	Inc	14542	124	102	99	112
	Dec	14221	112	90	87	120
	Random	442	4	0	2	2
	Gaussian	538	9	0	1	1
	Inc	342	3	3	2	8
	Dec	382	12	9	15	17
	Common	702	24	21	19	17
Real data		29908	240	196	29	24

The experimental results also establish our claim that Smart-R and Smart-HIU can further reduce the energy consumption of the system almost in all the cases. But the total energy consumption for Smart-ED is same or little higher than Smart. This is justified as the purpose of the Smart-ED policy is to avoid the sharp jump in the power consumption value and this policy distributes the power value across the time axis. The proposed policy achieves average energy reduction of 37% as compared to others for different computation time schemes and of *55*% as compared to others for different deadline schemes. Based on this experimental result, we can firmly say that our proposed policies reduce energy consumption of the system significantly for all of the cases of synthetic workload.

Table 1 shows the number of migrations with different task allocation policies both for synthetic and real trace data. As already explained earlier, UBA is of theoretical interest and has an unbounded number of migrations. FWC and RWC are of non-preemp-

tive nature and do not require any migration. We have considered non preemptive implementation of EDF and thus no migrations happened in this case also. We observe that total number of migrations in all our proposed policies is within a reasonable range.

Fig. 8 (f) shows the energy consumption of real workload traces (CERIT-SC, MetaCentrum-1, Zewura and MetaCentrum2) on considered large multi-threaded multi-processor system using different allocation policies. Our proposed policies outperform the rests. Our proposed policies achieve maximum energy reduction up to 44% as compared to all the earlier allocation policies UBA, WFC, RWC, EDF and UBWC. The proposed policies achieve average energy reduction of 30% as compared to the baseline policies. Experimental results show that the energy reduction in case of real trace data is comparatively lesser than that of synthetic data. This is because the inter-arrival time of tasks in case of real trace data is less and the smart policy cannot take the benefit of completely switching off the processors for longer time.

Energy consumption in case of UBWC is in general lesser than that of UBA, FWC and RWC but this policy incurs significant number of migrations. It can be lucidly seen from the experimental data that our proposed policies not only reduces overall energy consumption by a significant margin, but also the number of migrations in these policies are within reasonable range. Thus it can be concluded that even in case of high migration overhead system, the proposed policies will achieve a significant energy reduction.

7. CONCLUSION AND FUTURE WORK

Energy aware scheduling at a coarser granularity level has become essential for the large multi-threaded multiprocessor systems. In this paper, we have derived a simple power consumption model for such large systems and proposed an online energy efficient task allocation policy, namely, smart allocation policy for executing a set of independent real time tasks. We have then proposed three variations of this policy to further reduce energy consumption (for some applications) and to efficiently handle different situations which might occur at runtime. In near future, we will be considering scheduling tasks with dependencies. Efficient scheduling of multiprocessor tasks (tasks requires more than one thread for execution) under this power model will be a great extension to this work.

REFERENCES

1. R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Survey*, Vol. 43, 2011, Article 35.
2. G. C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*, Springer, Berlin, 2011.
3. W. Y. Lee, "Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 23, 2012, pp. 530-537.
4. S. Zhuravlev *et al.*, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed System*, Vol. 24, 2013, pp. 1447-1464.
5. T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design,"

in *Proceedings of the 28th Hawaii International Conference on System Sciences*, 1995.
6. M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE/ACM MICRO Third Berkeley Symposium on Energy Efficient Electronic Systems*, Vol. 33, 2013, pp. 8-19.
7. J. M. Allred *et al.*, "Dark silicon aware multicore systems: Employing design automation with architectural insight," *IEEE Transactions on VLSI Systems*, Vol. 22, 2014, pp. 1192-1196.
8. X. Zhu, L. T. Yang *et al.*, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Transactions on Cloud Computing*, Vol. 2, 2014, pp. 168-180.
9. J. K. Dong, H. B. Wang, Y. Y. Li, and S. D. Cheng, "Virtual machine scheduling for improving energy efficiency in IaaS cloud," *Communications*, Vol. 2, 2014, pp. 1-12.
10. Power consumption tests, http://www.xbitlabs.com/.
11. Power consumption qualcom hexagon v3, http://www.bdti.com/.
12. M. Weiser *et al.*, "Scheduling for reduced CPU energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, 1994, Article No. 2.
13. Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *Proceedings of the 9th IEEE/ACM International Symposium on CCGrid*, 2009, pp. 92-99.
14. Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, 2011, pp. 1374-1381.
15. D. W. Li and J. Wu, "Energy-aware scheduling for aperiodic tasks on multi-core processors," in *Proceedings of the 43rd International Conference on Parallel Processing*, 2014, pp. 361-370.
16. D. W. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, 2015, pp. 810-823.
17. D. W. Li and J. Wu, "Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms," in *Proceedings of the 41st International Conference on Parallel Processing*, 2012, pp. 430-439.
18. J. S. Chase and R. P. Doyle, "Balance of power: Energy management for server clusters," in *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, 2001, pp. 165.
19. S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of Conference on Power Aware Computing and Systems*, *HotPower*, 2008, pp. 10.
20. A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of HPC applications," in *Proceedings of the 22nd Annual International Conference on Supercomputing*, 2008, pp. 175-184.
21. A. Verma, *et al.*, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proceedings of ACM/IFIP/USENIX Conference on Middleware*, 2008, pp. 243-264.
22. S. Ali, *et al.*, "Task execution time modeling for heterogeneous computing systems,"

in *Proceedings of Heterogeneous Computing Workshop*, 2000, pp. 185-199.
23. J. Kim and K. G. Shin, "Execution time analysis of communicating tasks in distributed systems," *IEEE Transaction on Computers*, Vol. 45, 1996, pp. 572-579.
24. P. Brucker, *Scheduling Algorithms*, 5th ed., Springer, Berlin, 2010.
25. Metacentrum data sets, https://www.fi.muni.cz/ xklusac/index.php?page=meta2009.
26. J. Choi *et al.*, "Power consumption prediction and power-aware packing in consolidated environments," *IEEE Transactions on Computers*, Vol. 59, 2010, pp. 1640-1654.
27. A. Beloglazov *et al.*, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, Vol. 28, 2012, pp. 755-768.
28. Y. Ma, B. Gong, R. Sugihara, and R. Gupta "Energy-efficient deadline scheduling for heterogeneous systems," *Journal of Parallel and Distributed Computing*, Vol. 72, 2012, pp. 1725-1740.
29. S. Holmbacka *et al.*, "Task migration for dynamic power and performance characteristics on many-core distributed operating systems," in *Proceedings of the 21st Conference on Parallel*, *Distributed and Network-Based Processing*, 2013, pp. 310-317.

Manojit Ghose completed his B.E. degree in CSE from Jorhat Engineering College, Assam in 2007. He has received M. Tech. degree in CSE from IIT Guwahati in 2013. Currently, he is pursuing his Ph.D. from IIT Guwahati. His research interests include cloud computing, multiprocessor scheduling, memory hierarchy for multi-processor architecture.

Aryabartta Sahu received B.Sc. Physics and M.Sc. Electronics degree from Sambalpur University, M. Tech CS from Utkal University and Ph.D. CSE from IIT Delhi. He is currently working as an Associate Professor in the department of CSE at IIT Guwahati. His research interests include multiprocessor, scheduling, cloud computing and high performance computing.

Sushanta Karmakar received his B.E. and M.E. degrees in CSE from Jadavpur University in 2001 and 2004 respectively. He did his Ph.D. from IIT Kharagpur in 2009. He is currently working as an Associate Professor in the department of CSE at IIT Guwahati. His research interests include distributed algorithms, fault-tolerant distributed algorithms.