J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Power efficient rate monotonic scheduling for multi-core systems

Nasro Min-Allah [a], Hameed Hussain [a], Samee Ullah Khan [b,*], Albert Y. Zomaya [c]

[a] COMSATS Institute of Information Technology, Islamabad 44000, Pakistan
[b] North Dakota State University, Fargo, ND 58108-6050, USA
[c] University of Sydney, Sydney NSW 2006, Australia

ARTICLE INFO

Article history:
Received 9 March 2011
Received in revised form
3 July 2011
Accepted 8 July 2011
Available online xxxx

Keywords:
Real-time systems
Non-preemptive scheduling
Fixed-priority scheduling
Feasibility analysis
Online schedulability tests

ABSTRACT

More computational power is offered by current real-time systems to cope with CPU intensive applications. However, this facility comes at the price of more energy consumption and eventually higher heat dissipation. As a remedy, these issues are being encountered by adjusting the system speed on the fly so that application deadlines are respected and also, the overall system energy consumption is reduced. In addition, the current state of the art of multi-core technology opens further research opportunities for energy reduction through power efficient scheduling. However, the multi-core front is relatively unexplored from the perspective of task scheduling. To the best of our knowledge, very little is known as of yet to integrate power efficiency component into real-time scheduling theory that is tailored for multi-core platforms. In this paper, we first propose a technique to find the lowest core speed to schedule individual tasks. The proposed technique is experimentally evaluated and the results show the supremacy of our test over the existing counterparts. Following that, the lightest task shifting policy is adapted for balancing core utilization, which is utilized to determine the uniform system speed for a given task set. The aforementioned guarantees that: (i) all the tasks fulfill their deadlines and (ii) the overall system energy consumption is reduced.

1. Introduction

The birth of multi-core systems have significantly advanced the existing technologies in the domain of computer architecture. However, this advantage presents the research community with enormous challenges, such as the efficient handling of thermal dissipation and the lack of mature scheduling techniques. Normally, all the cores of a chip operate in the same clock domain, clock frequency, and operational voltage [35]. However, there exist systems in which the cores do not operate at the same frequency. Therefore, maintaining performance symmetry among asymmetrically operating cores is one of the most critical issues that the researchers are dealing with today [15,33]. There are two possible solutions for the aforementioned issues: (i) add dynamic voltage circuitry per core (a hardware solution), or (ii) schedule tasks among cores judicially to enable all the cores to operate on the same clock frequency (a software solution). The former compensation strategy exhibits power leakage at higher frequencies and undermine the thermal throttling [15]. Being a promising alternative, the latter solution is relatively unexplored from the point of view of scheduling. Considering this gap, we partition a given workload among the cores with the intention that all the cores operate on the same clock frequency for maximum energy savings.

The newer processors provide an interface to dynamically adjust the voltage (or speed) for optimized power consumption. This voltage (speed) adjustment on run time is termed as dynamic voltage scaling (DVS), which is an effective methodology for the reduction of core power consumption. The dynamic clock and voltage adjustments represent the cutting edge of power reduction capabilities in CMOS circuitry. The relation between frequency and voltage/power provides foundation for dynamic voltage scaling in modern processors [10,7,14,15]. Theoretically, an ideal processor would be the one that supplies continuous voltage levels. However, using continuous variable voltages is infeasible because of the switching overhead to support several operational levels. Therefore, the latest processors are capable of supporting a fixed number of discrete-level speeds between a predefined minimum and maximum levels. It has been reported in [16] that the energy–speed curve is convex in nature. Therefore, according to Jensen's inequality [17,31,20], as long as the deadline constraints are fulfilled, it is more energy efficient to execute tasks at a constant speed than at a variable speed for each of the individual tasks. We further extend this result by exploring the

* Corresponding author.
E-mail addresses: nasar@comsats.edu.pk (N. Min-Allah), ham.hamdard@gmail.com (H. Hussain), samee.khan@ndsu.edu (S.U. Khan), albert.zomaya@sydney.edu.au (A.Y. Zomaya).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.07.005

possibility of determining a uniform system speed for all the cores by considering a processor that supports a large number of discrete energy–voltage levels.

In real-time systems, tasks are scheduled based on some predefined criteria, such as activation rates, deadlines,and priorities [24,27,12,18,8]. The higher the priority of a task, the more is the attention devoted to the task when a scheduling decision is to be made. Real time systems are usually not fully utilized up to the maximum extent. Therefore, the systems are a promising venue to apply DVS methodologies and DVS enabled scheduling techniques. Applying DVS techniques requires careful consideration of task scheduling and a number of results are available (primarily for the uniprocessor systems) [31,35,19,32,1,3,20,2,30].

Continuous speed levels are normally assumed to obtain optimality. However, the aforementioned is inapplicable to practical systems that have processors with discrete voltage regulators [25,36]. Manufacturers are introducing processors that will operate on more discrete levels than what we see today. For instance, the new Foxon technology is expected to enable the Intel servers to operate on as many as 64 speed grades [25]. Therefore, an accurate model for reducing the energy consumption of the latest systems must capture the discrete rather than the continuous nature of the available speed scaling [25,36,16,6]. However, the work we present here can easily be extending to systems that may operate on a continuous speed spectrum.

The most commonly used policy to schedule real-time tasks is the "priority driven", that can be classified into the following two types: (i) fixed priority and (ii) dynamic priority [23]. A fixed-priority algorithm assigns a fixed value to priority to all jobs in each task, which should be different from the priorities assigned to jobs generated by other tasks within the system. In contrast, dynamic-priority scheduling algorithms place no restrictions on the manner in which priorities are assigned to individual jobs. Although, dynamic algorithms are better considered theoretically, they become unpredictable when transient overload occurs [13]. Therefore, in this paper, we only consider fixed-priority scheduling due to its applicability, reliability, and simplicity [24,18,8,28,5,26].

The problem of scheduling periodic tasks under a fixed-priority scheme was first addressed by Liu and Layland [24] in 1973 with simplified assumptions. They derived the optimal static priority scheduling algorithm for implicit-deadline model (when deadlines coincide with respective periods), termed the rate monotonic (RM) algorithm. The RM algorithm assigns static priorities on the task activation rates (periods) such that for any two tasks τ_i and τ_j, $\text{priority}(\tau_i) > \text{priority}(\tau_j) \Rightarrow \text{period}(\tau_i) < \text{period}(\tau_j)$, wherein ties are broken arbitrarily. For a constrained deadline system, where deadlines are not greater than periods, an optimal priority ordering has been reported in [22], termed the deadline monotonic (DM) scheduling, wherein, the assigned priorities are inversely proportional to the relative deadlines. The RM and DM methodologies are identical when the relative deadlines of tasks are proportional to their periods. In the remainder of this paper, a task model refers to a constrained deadline system, and both RM and DM will be used interchangeably to align with the terminologies used in the literature.

Scheduling policies developed for symmetric multiprocessors may also be applicable to the multi-core counterpart. Recently, the fixed-priority scheduling theory for multi-core environment was studied in [19]. We extend the aforementioned work to further explore the necessary and sufficient condition [21] of the RM paradigm pertaining to multi-core systems.

In particular, more interesting results are revealed for multi-core systems where all the cores operate at the same clock frequency [34]. Once the speed for a generic core Δ_i is determined, the average system speed suitable for all the the cores is calculated. However, this average speed might potentially make the task set unschedulable on some cores. In this work, we address this anomaly to maintain system feasibility by shifting tasks from a heavily utilized core to an underutilized core such that all the cores process the same workload and the task set remains feasible at uniform system speed.

Contribution Synopsis. This work advances the current state of the art of scheduling theory as follows.

- *Identification of the lowest possible core speed*. This work identifies the implicit disadvantage associated with the first feasible speed approach that is often used in the literature. We further investigate this issue and identify properties and bounds that enable us to identify a procedure, which can further reduce the core speed to the minimum possible level and also ensure that the task set remains RM schedulable.
- *Practical power savings with adjustable core speeds*. Because of the practical limitations of the available DVS-enabled processors, the tasks are mapped using a finite number of discrete voltage levels. However, our work can also be equally applicable to future generation processors that may support continuous voltage levels.
- *Present a simple but practical core load balancing procedure*. We propose the lightest task shift procedure to load balance the system cores. The motivation behind this mechanism is based on the observation that the lightest task (with lowest utilization among all the tasks assigned to a core) is the only task that decreases the core utilization by the minimum possible load by shifting (or migrating) the task from an over utilized core to the underutilized cores.
- *Achieving uniform system power consumption and utilization*. The focus of our work was kept as general as possible to include heterogeneous system cores. Our approach can fine-tune the system so that all the cores operate on the same clock rate and have equally proportionate core utilization. The aforementioned results in a uniform system performance with predictable power consumption. Our approach can be useful for designing applications that demand homogeneous performance over a heterogeneous system.

The remainder of this paper is organized as follows. First, we discuss the system model and the necessary background information pertaining to the RM scheduling theory in Section 2. The main contributions of this work are divided into two grouped sections: Sections 3 and 4, which detail the derivation of the lowest possible speed for a give task set, the average system speed, and the experimental results pertaining to the derived speeds, respectively. Sections 5 and 6, which details the derivation of the uniform system core speeds and the performance evaluations of the proposed methodology, respectively. Finally, we conclude the paper in Section 7.

2. System model and background

In the periodic model of hard real-time systems, a task τ_i is described by: (i) a task period P_i that is the time between any two consecutive instances of τ_i, (ii) a worst-case execution time C_i that is scalable with core speed, and (iii) a relative deadline D_i. A task τ_i must have C_i units of CPU shares by D_i. However, C_i varies considerably at run time. All the aforementioned parameters are integers. The task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ consists of n tasks and can be divided into subsets such that $\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_n\}$, where $\Omega_1 = \{\tau_1, \tau_2, \ldots, \tau_k\}$ and $\Omega_2 = \{\tau_{k+1}, \tau_{k+2}, \ldots, \tau_i\}$, and so on. Moreover, a set of cores $\Delta = \{\Delta_1, \Delta_2, \ldots, \Delta_m\}$, $(m \leq n)$ is available. The system speed f_i is within a predefined range [0.1, 1.0], with a step size of 0.01. The core utilization of an individual task τ_i is given as $U_i(\Delta_i) = C_i/P_i$ and the cumulative core utilization is denoted by $U(\Delta_i) = \sum_{i=1}^{k} C_i/P_i$. The problem that we are

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005

addressing here is to map Ω over Δ under the fixed priority scheduling paradigm.

The first feasibility test for an RM scheduling on a uniprocessor (also to be understood as a uni-core) system was reported in [24], which was termed as the Liu and Layland bound (LL-bound). The LL-bound states that a periodic task system where $d = p$ is static-priority feasible if and only if

$$U(\Delta_i) \leq n(2^{1/n} - 1), \tag{1}$$

where n denotes the number of tasks in Γ. The term $n(2^{1/n} - 1)$ decreases monotonically from 0.83 (when $n = 2$) to ln(2) as $n \to \infty$. This result mandates that any periodic task set of any size is static priority feasible on a preemptive uniprocessor if and only if the RM scheduling is used and $U(\Delta_i)$ is not greater than 0.693. This result gives a simple $O(n)$ procedure to test the task feasibility when tasks arrive at run time. However, the aforementioned is only a sufficient condition. Therefore, it is quite possible that an implicit-deadline synchronous periodic task system that exceeds the LL-bound to be static-priority feasible. The LL-bound for the RM paradigm is quite pessimistic. Therefore, it has been proven that for the average case [21]:

$$U(\Delta_i) \simeq 0.88. \tag{2}$$

A better utilization based test, termed the hyperbolic bound (HB) was detailed in [4]. Using the HB test, a periodic task set is deemed schedulable if and only if

$$\prod_{h=1}^{n} (U_i(\Delta_i) + 1) \leq 2. \tag{3}$$

The classic work reported in [24] was later extended by modifying the task parameters in [23]. However, all the aforementioned tests cover only the sufficient conditions (SC) and trade utilization for performance.

One possible solution to the aforementioned problem is to first equally distribute a given workload among all the cores and then to find the feasibility of using the RM bounds, such as the LL-bound [24] or the H-bound [4]. However, these bounds provide only the sufficient conditions and a thick share of the core utilization is compromised for schedulability. To the best of our knowledge, this work is the first to: (i) derive the exact RM scheduling conditions for a multi-core system and (ii) determine a uniform lowest possible system speed for a given workload that maintains system feasibility. Symmetric performance among cores is only possible when all the cores operate at the same low speed. The disadvantage associated with higher core frequency is that of leakage power, i.e. higher clock frequency increases the system power leakage. Therefore, cores must operate on the same minimum possible frequency for the following two reasons to: (i) avoid power leakage and (ii) conserve energy by allowing cores to execute tasks at a constant speed.

In our proposed model, we assume that a processor has 10 major operational levels as detailed in the Table 1. Let f_i denote a speed level and the corresponding range is given by Δf_i, as per our processor specifications. If for a particular speed f_i^0 that is unavailable within the range Δf_i (minor levels), then the next (higher) nearest value is assigned to f_i^0 from the range Δf_i. We must note that f_i is the highest possible speed within a level. Therefore, any task τ_i that is schedulable with any speed in Δf_i, is also schedulable with f_i (for any i, $f_i = \max(\Delta f_i)$). However, the converse may not hold.

Initially, we assume Γ to be scheduled on a single core. For our model to be as close as possible to the real-world scenarios, we opt for a constrained task model ($D_i \leq P_i$). Let time $t = 0$ be the critical instant and the cumulative work load of a task τ_i at any instance of time t running at speed f_i ($f_i, f_j \leq 1$) is to be represented by

Table 1
Operational levels and the respective speed ranges.

Level (i)	f_i	Δf_i (respective subranges/minor-levels for speed f_i)
0	0.1	0.01, 0.02, …, 0.09, 0.10
1	0.2	0.11, 0.12, …, 0.19, 0.20
2	0.3	0.21, 0.22, …, 0.29, 0.30
3	0.4	0.31, 0.32, …, 0.39, 0.40
4	0.5	0.41, 0.42, …, 0.49, 0.50
5	0.6	0.51, 0.52, …, 0.59, 0.60
6	0.7	0.61, 0.62, …, 0.69, 0.70
7	0.8	0.71, 0.72, …, 0.79, 0.80
8	0.9	0.81, 0.82, …, 0.89, 0.90
9	1.0	0.91, 0.92, …, 0.99, 1.00

$$Ł_i(t) = \frac{C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil (C_j)}{f_i}. \tag{4}$$

The classic work reported in [21] details a solution that a task τ_i is always feasible on a generic core Δ_i at any instance of time t if and only if

$$\min_{t \in S_i} L_i(t) \leq t, \tag{5}$$

where t is a scheduling point and S_i denotes a set of all the scheduling points constituted by $S_i = lP_j$ ($j = 1, \ldots, i; l = 1, \ldots, \lfloor P_i/P_j \rfloor$). The whole of the task set Γ becomes RM feasible when

$$\max_{i=1,\ldots,k} \left\{ \min_{t \in S_i} L_i(t)/t \right\} \leq 1. \tag{6}$$

3. Lowest speed calculations

In this section and in Section 5, we address the problem of scheduling hard-deadline periodic tasks on a multi-core environment. Section 4 details the simulation results for uniprocessor systems, which are extended to encompass the multi-core counterpart in Section 5.

Ref. [15] reports that the performance of the cores is asymmetric. Therefore, tasks cannot be assigned to the cores with the implicit assumption that all the cores are operating at the maximum clock frequency. Moreover, heat dissipation increases when processors operate at higher clock rates. Because of the aforementioned issues pertaining to higher clock rates, we must first determine the appropriate core performance and once that is known, uniform system speed can be calculated by distributing the workload among the cores based on some schedulability tests. It has been reported in [19] that the bin-packing technique allows only half of the core utilization and the technique trades utilization at the cost of performance. To overcome the aforementioned gap of 50%, we derive and utilize the necessary and sufficient condition. For our analysis, and for simplicity, we assume that initially the system is a single core entity. Once the average core speed is determined, we relax the abovementioned assumption to accommodate multiple cores.

A task τ_i is schedulable on a generic core Δ_i if and only if Eq. (5) holds true. However, it is possible that the task may also be schedulable at a lower core speed. Therefore, we add the speed component into the schedulability analysis to determine the required task speed, which can be represented by

$$\min_{t \in S_i} \left(\frac{C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil C_j}{t} \right) \leq f_i. \tag{7}$$

Any value of $t \in S_i$ that satisfies Eq. (7) ensures that τ_i is also schedulable with speed f_i. However, for different values of t, there

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005

could be a set of respective speed levels, guarantying the schedulability of τ_i.

Ref. [5] reports a methodology that for a given workload returns the speed determined at the first feasible point in the scheduling point set, termed first feasible speed (FFS). Therefore, as soon as the schedulability is confirmed at the first true scheduling (the time where τ_i is schedulable), the value of f_i is also determined. From the aforementioned discussion, an interesting observation can be made that we state below.

Observation 1. *The set of scheduling points S_i for task τ_i is always in a non-decreasing order and the first value of $t \in S_i$ that satisfies Eq. (7) does not guarantee the lowest system speed required.*

To further elaborate on Observation 1, we highlight the point with the help of an example task set given below.

Example 1. Given three tasks $\tau_1(1.1, 3)$, $\tau_2(1, 5)$, $\tau_3(1, 10)$, where each task τ_i is represented by its parameters C_i and P_i, as an ordered pair $\tau_i(C_i, P_i)$. Determine the lowest core speed to schedule the lowest priority task τ_3, in addition to higher priority tasks τ_1 and τ_2.

According to the RM scheduling theory, task τ_3 is schedulable if and only if it satisfies Eq. (7). Task τ_3 has a set of scheduling points $S_3 = \{3, 5, 6, 9, 10\}$.

List 1. Task τ_3 is RM-schedulable if and only if

$$C_1 + C_2 + C_3 \leq 3$$
$$2C_1 + C_2 + C_3 \leq 5$$
$$2C_1 + 2C_2 + C_3 \leq 6$$
$$3C_1 + 2C_2 + C_3 \leq 9$$
$$4C_1 + 2C_2 + C_3 \leq 10.$$

It can be observed that, in the presence of the workload due to τ_1 and τ_2, task τ_3 is also schedulable at points 5, 6, 9, and 10. The speed required at the respective points becomes 0.84, 0.86, 0.7, and 0.74. The lowest speed is 0.7 that is achieved at the scheduling point 9, which is the fourth element in set S_3. Therefore, the first element does not always guarantee the lowest system speed. From the aforementioned discussion, we can conclude that all the values of $t \in S_i$ need to be tested for finding the lowest core speed for task τ_i. That is, f_i may be obtained by the following

$$\min\left(\max_{t\in S_i}\frac{C_i + \sum_{j=1}^{i-1}\left\lceil \frac{t}{p_j}\right\rceil C_j}{t} \leq f_i\right). \tag{8}$$

Figs. 1 and 2 depict the Gantt charts for the task set given in Example 1. The charts are drawn for the task set at the speeds of 0.84 and 0.7, respectively. The values for C_i are rounded off to two decimals points to avoid cumbersome Gantt charts. We must note that, irrespective of the representation of decimal fractions, Eq. (8) always results in the exact same analysis and always respects the timing constraints of the task set. In both cases, the entire task set is schedulable with lower speeds. The task set when executed at the speed of 0.84 becomes $\tau_1(1.30, 3)$, $\tau_2(1.19, 5)$, $\tau_3(1.19, 10)$ and the same is reflected in Fig. 1. It can be observed from Fig. 1 that, after scheduling all the jobs of the tasks, there still are 1.53 : $[4.98, 5] + [7.49, 9]$ time slots unused and these slots can further be utilized for lowering the system speed. Similarly, Fig. 2 reflects the Gantt chart for the modified task set when executed at a lower speed of 0.7 and the original task set (give in Example 1) is transformed into $\tau_1(1.57, 3)$, $\tau_2(1.42, 5)$, $\tau_3(1.42, 10)$. In contrast to Fig. 1, there are only 0.03: [8.97, 9] unused slots in Fig. 2, which is a clear advantage and results in maximum system utilization.

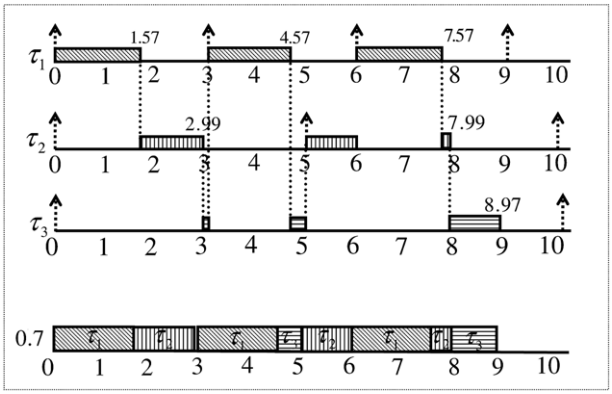


Fig. 1. Gantt chart for $\tau_1(1.30, 3)$, $\tau_2(1.19, 5)$, and $\tau_3(1.19, 10)$.

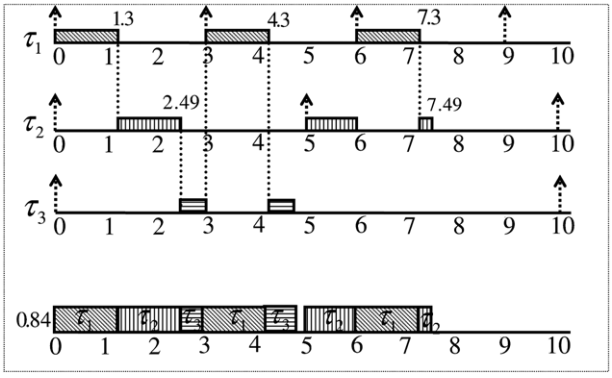


Fig. 2. Gantt chart for $\tau_1(1.57, 3)$, $\tau_2(1.42, 5)$, and $\tau_3(1.42, 10)$.

4. Experimental analysis

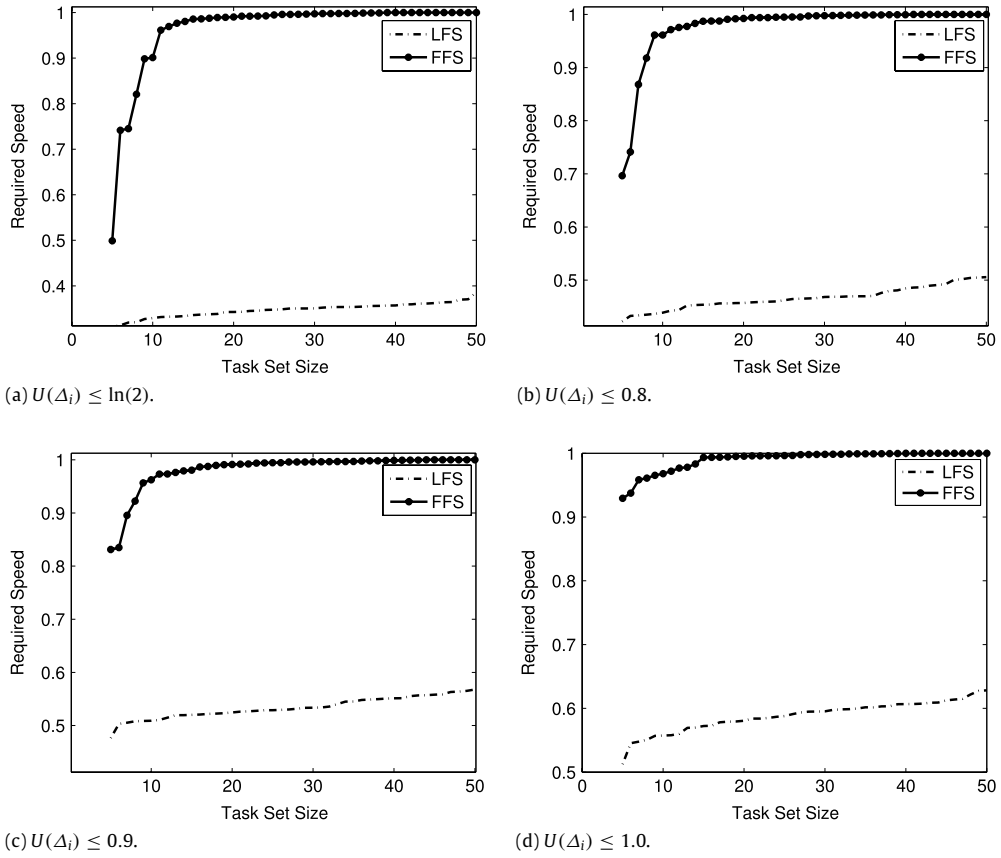
4.1. Determining the lowest speed

In this section, we evaluate the performance of our proposed technique, LFS, by comparing it with the previously mentioned FFS methodology. Both the aforementioned methodologies are compared from the perspective of system speed. The lower the speed, the better is the technique.

To compare both techniques, we generated random task sets of sizes within the range of [5, 50], with a step size of 1. The plots reported in this paper are the average values of 300 runs of all the task sets 5 through 50. The task periods were randomly generated from a uniformly distributed range of [100, 10,000]. To obtain the corresponding task execution demands C_i for τ_i, random values were taken from within the range of $[1, p_i]$, also with uniform distribution. The priorities were assigned to the tasks as per RM scheduling rules. That is, the smaller the task period, the higher is the task priority. To have a feasible RM schedulable task set, initially, we keep the system utilization at $\ln(2)$, which is quite low. This low system utilization ensures that all the tasks within a given task set are RM feasible. Otherwise, it is very likely that some of the tasks may not be RM feasible when the system utilization is kept high. Moreover, this also will pertain to an unfair comparison.

Fig. 3 depicts the advantage of the LFS methodology over the FFS technique. We can observe that the LFS approach continues until it finds the minimum possible speed for a given task set, while maintaining task set schedulability. In contrast, the FFS procedure stops searching for the scheduling points immediately as soon as it finds the first feasible point. The difference between both techniques is quite large. For instance, for the task set having only 5 tasks, the system speed required by the LFS methodology is much lower than compared to the FFS procedure.

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005



(a) $U(\Delta_i) \leq \ln(2)$. (b) $U(\Delta_i) \leq 0.8$.

(c) $U(\Delta_i) \leq 0.9$. (d) $U(\Delta_i) \leq 1.0$.

Fig. 3. Effect of utilization on system speed.

To further illustrate the effectiveness of the proposed methodology, we report several simulation results with different system utilization. Fig. 3(a) reports that, with an RM schedulable task set, the LFS procedure is always able to execute all the tasks with lesser system speed. This includes the task set with 50 tasks. As shown in Fig. 3(d), higher system speed is required for larger task sets. This is an understandable phenomenon, because when the workload increases, more computational cycles are needed to complete all the tasks by their respective deadlines. From the plots, we can also see that the aforementioned behavior is exhibited by both the techniques as expected. Although the FFS technique is based on the necessary and sufficient conditions of the RM scheduling theory, the system feasibility is a must and it is maintained with the FFS approach. However, when the task set size increases, the FFS procedure allows the system speed to grow very rapidly to accommodate the resource requirements to maintain the deadline constraints. On the other hand, our proposed methodology gradually increases the system speed in accordance with the principal objective, which is to conserve energy as much as possible by allowing the system to operate at a clock rate that is the slowest and keeping the task deadline constraints intact. Fig. 3(a) through Fig. 3(d) reveals the performance of both techniques with the system utilization kept at 70%, 80%, 90%, and 100%. It can be observed that when the system utilization increases, the task computational demands also increase and both techniques need more system speed to accommodate the workload presented. Therefore, the system operates at a higher clock rate.

4.2. Energy savings

As indicated in the introductory passage, the DVS is a promising technique for lowering the power consumption of a CMOS circuitry. Before presenting our experimental analysis using the DVS technique, we establish the necessary formulations for the DVS methodology from the previous literature [10,7,14,32].

The average power dissipation P_{avg} of modern processors is composed of four parts.

$$P_{avg} = P_{leak} + P_{cap} + P_{std\text{-}by} + P_{short}, \tag{9}$$

where P_{leak}, P_{cap}, $P_{std\text{-}by}$, and P_{short} denotes the power leakage, capacitive, standby, and short-circuit power, respectively. The most critical component of Eq. (9) is the term P_{cap}. Therefore, we can ignore the rest of the terms as in Ref. [14]. Being the dominating term, P_{cap} can be expressed as:

$$P_{cap} = \Upsilon \times V_{dd}^2 \times f, \tag{10}$$

where Υ represents a transition activity dependent parameter and the switched capacitance, and V_{dd} is the supply voltage. Eq. (10) indicates the quadratic dependence of V_{dd} and f. It can be concluded that lowering the supply voltage is the most effective factor in lowering the dynamic power consumption. However, the lowering of V_{dd} increases the circuit delay, which may be represented by the following

$$T_{delay} = k\frac{V_{dd}}{(V_{dd} - V_{th})^{\eta}}, \tag{11}$$

where k is a constant specific to a given technology and depends on the gate size and capacitance, V_{th} is threshold voltage that is the minimum required voltage, and η is the velocity saturation index of a CMOS circuit within the range of $1 \leq \eta \leq 2$. Because f and T_{delay} are inversely related, we can say that

$$f = k\frac{(V_{dd} - V_{th})^{\eta}}{V_{dd}}. \tag{12}$$

Eq. (12) reflects that f is linearly related to the supply voltage. That is, the processor speed is a direct consequence of the supplied

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005

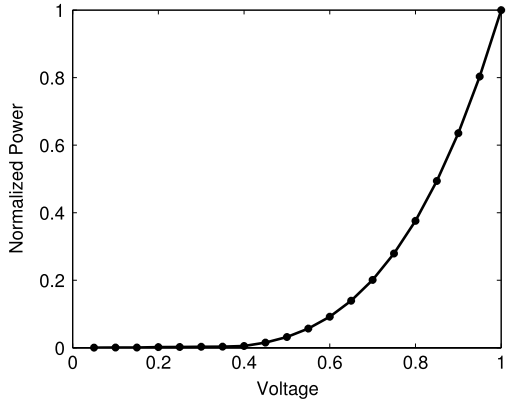


Fig. 4. Power consumption of Crusoe processor at respective voltage levels.

voltage [32]. Therefore, by assuming that $P_{avg} = P_{cap}$, Eq. (10) can be rewritten as

$$P_{avg} = \Upsilon \times V_{dd}^2 \times f. \tag{13}$$

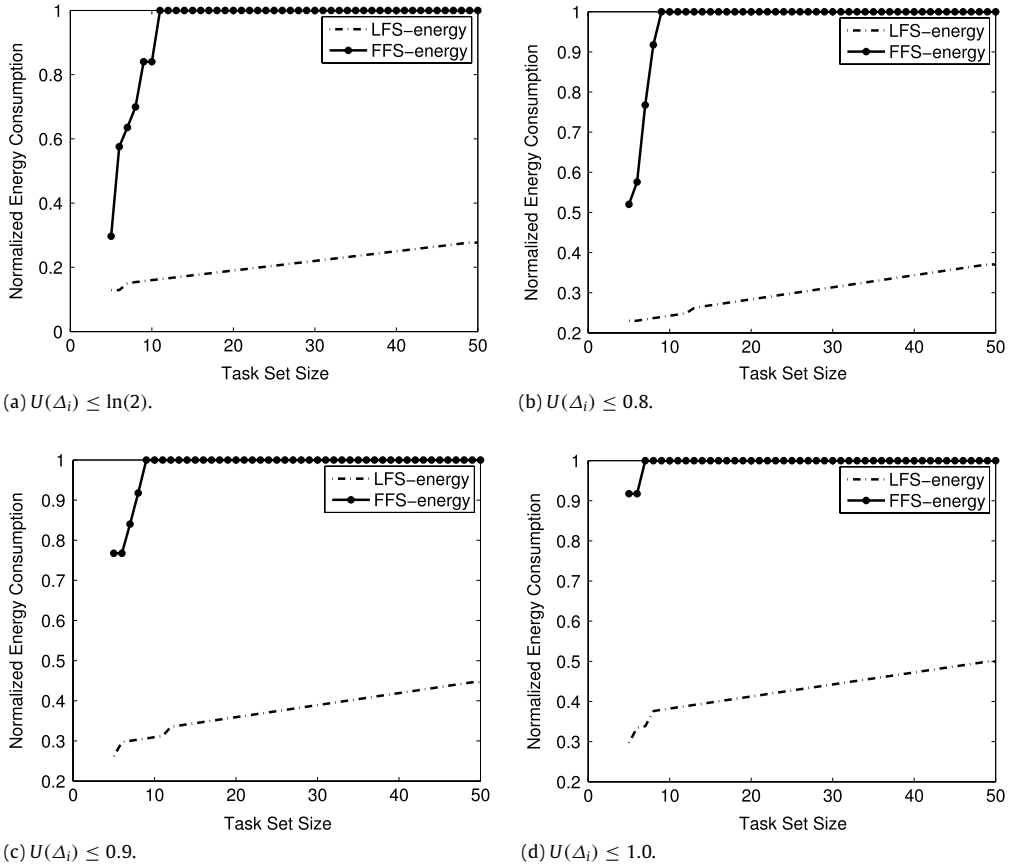
It can also be observed that P_{avg} is an increasing function of f. Let E be the energy consumed while running a task with an average power P_{avg} at the processor speed of f for T time units. The aforementioned relationship can be represented mathematically by the following

$$E = P_{avg}(f) \times T. \tag{14}$$

From the aforementioned discussion, we can deduce that an ideal processor would be the one that can operate on continuous voltage levels. However, due to the switching overhead, a continuous voltage spectrum is not provided for a CMOS circuit [9]. Therefore, only a discrete number of supply voltage levels are provided that can be controlled with a DVS technique [11]. In our work, we assume a processor that can support multiple discrete frequency levels within the range of [0.1, 1.0], with a step of 0.01, where 0.1 is the minimum speed needed to make the peripherals and interrupts remain powered and active. For our study, we operate within the bounds reported in [35] for a 70 nm Crusoe processor. The bounds and the discrete voltage levels are plotted for the readers' convenience in Fig. 4, which illustrates the relationship between the power consumption and the supplied voltage per cycle.

To evaluate the two methodologies, namely the LFS-energy and FFS-energy, from the point of view of energy savings, we simulate the system under the same arrangement as previously discussed in Section 4.1. For this set of simulations, the speed of the system was based on Eq. (7) for the FFS-energy technique and Eq. (8) for the LFS-energy methodology. The workload of the system was the task set within the range of [5, 50], with an increase of a single task after every 1000th iteration.

The voltage and the clock rate required for a successful completion of all the tasks within the task set is determined by the FFS-energy and LFS-energy techniques. The total energy consumed within the interval of $[T^0, T^1]$ is measured by $\int_{T^0}^{T^1} P_{avg}(t, f_i)\mathrm{d}t$, where $P_{avg}(t, f_i)$ is the power consumption of the core when executing a task τ_i at the speed of f_i for t time units. The system utilization is again kept within the range of [ln(2), 1.0], with a step size of 0.1. That is, the energy values are measured for the task set after a 10% increase in the system utilization. It can be observed from Fig. 5 that when all the tasks are schedulable, the savings in energy consumption of both techniques are very encouraging, up to a certain level. The reason behind this low energy consumption is the likelihood of the RM feasibility of all the tasks due to low



(a) $U(\Delta_i) \leq \ln(2)$. (b) $U(\Delta_i) \leq 0.8$.

(c) $U(\Delta_i) \leq 0.9$. (d) $U(\Delta_i) \leq 1.0$.

Fig. 5. Normalized energy consumptions for the task set under varying utilizations.

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005

system utilization. Therefore, the computational demands of the individual tasks are much lower than their respective periods.

The only difference is that with the FFS-energy approach, the plot trend remains higher when the task set increases. This is due to the fact that there is a possibility that some of the task sets may contain some tasks that must be run at a higher speed. Therefore, the system energy consumption increases as the power function is quadratically proportional to the system speed. Our proposed LFS-energy technique projects a lower system speed compared to the FFS-energy approach. This is due to the fact that its implicit characteristic of continuously searching for the lowest possible feasible speed out of all the possible speeds, which tends to be never higher than that obtained through Eq. (7). With increased system utilization, the computational demands of individual tasks increase. Therefore, the workload also increases in the allowable time window. It can also be observed from Fig. 5 that the LFS-energy approach also increases the system speed with the increase in utilization. This is due to the fact that the scheduler must respect all the deadlines of the tasks. However, the FFS-energy approach is a very reactive procedure as the first feasible scheduling point might demand high system speed. Therefore, the cores run at a higher clock rate. In contrast, the LFS-energy approach determines the point where the minimum task speed is calculated. Therefore, the speed required for the same task set is much lower. However, the lower system speed identified by the Eq. (8) also means that the computation demands of the task have now prolonged.

5. Task partitioning in multi-core systems

To avoid testing the schedulability of a task at reduced number of scheduling points, authors in [29] introduced the concept of false point.

Definition 1. Under a fixed priority scheduling, a point t is termed a false point for a generic task τ_i, if and only if it satisfies the inequality constraint of $L_i(t) > t$.

The concept of false point is plausible; however, it is inapplicable to DVS-enabled cores for the following reason.

Theorem 1. *Under fixed priority scheduling and multiple system speed levels, a false point t for τ_i is not necessarily a false point for the lower priority tasks $\tau_{i+1}, \ldots, \tau_n$.*

Proof. The proof is presented for τ_{i+1} that can easily be extended to the case of $\tau_{i+2}, \ldots, \tau_n$. As mentioned in Section 2, a scheduling point t for τ_i is also the scheduling point for τ_{i+1}. Let t' be such a point that is present within the set S_i and all the subsequent sets $S_{i+1}, \ldots, S_n$. If t' is a false point for τ_i that is executing at the speed of f_i, then,

$$Ł_i(t') = \frac{C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t'}{P_j} \right\rceil C_j}{f_i} > t'. \tag{15}$$

Similarly, the workload for τ_{i+1} at the same point t' that is running at the speed of f_i' can be expressed as

$$Ł_{i+1}(t') = \frac{C_{i+1} + \sum_{j=1}^{i} \left\lceil \frac{t'}{P_j} \right\rceil C_j}{f_i'}. \tag{16}$$

When $f_i \geq f_i'$, the false point for τ_i also remains the false point for τ_{i+1}. However, if $f_i < f_i'$, then

$$\frac{C_{i+1} + \sum_{j=1}^{i} \left\lceil \frac{t'}{P_j} \right\rceil C_j}{f_i'} < \frac{C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t'}{P_j} \right\rceil C_j}{f_i}, \tag{17}$$

which is a contradiction because: (a) $f_i' > f_i$ and (b) the workload due to τ_{i+1} at t' is lowered due to the higher value of f_i'. Therefore,

$$\frac{C_{i+1} + \sum_{j=1}^{i} \left\lceil \frac{t'}{P_j} \right\rceil C_j}{f_i'} \not> t', \tag{18}$$

which contradicts that t' is a false point for τ_{i+1}.

First, determine the speed for the execution of task τ_i. Then calculate the core's specific minimum required speed so that all the tasks remain schedulable on the core Δ_i with speed f_{Δ_i}. This relationship can be captured by the following expression

$$f_{\Delta_i} = \max\left(\min_{0<i\leq n} f_i\right). \tag{19}$$

Eq. (19) ensures that the core operates on the appropriate speed to execute all the tasks $\tau_1, \ldots, \tau_k \in \Omega_i$ successfully. Next, we must find the average system speed (uniform speed for all the cores) to execute Ω tasks on Δ cores. As previously mentioned in Section 2, the aforementioned result (Eq. (19)) is applicable only to a single core system. Therefore, we must relax some of the assumptions for the multi-core model. The same technique as discussed in Section 2 can also be applied to the whole task set Ω and the tasks can be mapped on Δ cores, which we detail in the subsequent text.

Let the tuple $(\Omega_i, \Delta_i, f_{\Delta_i})$ represent the task set Ω_i assigned to core Δ_i running at the speed of f_{Δ_i}. Because it is preferred to execute all the cores at a uniform speed, the average system speed must be calculated. In other words,

$$f_\Delta = \frac{\sum_{j=1}^{m} f_{\Delta_i}}{m}. \tag{20}$$

To achieve load balancing among the cores, we adopt a task shifting strategy that migrates a task from an unschedulable core to a core with the smallest workload. (This is completely different from the traditional task splitting technique.) □

Theorem 2. *If a task τ_i is shifted from an unschedulable core Δ_i to another core Δ_j (wherein both the cores run at the same speed), then the schedulability of Ω_i on Δ_i increases by a factor of C_i.*

Proof. If τ_i is unschedulable on Δ_i at $t \in S_i = lP_j$ ($j = 1, \ldots, i$; $l = 1, \ldots, \lfloor P_i/P_j \rfloor$), then

$$\left(C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{P_j} \right\rceil C_j\right) > f_i \times t,$$

$$\sum_{j=1}^{i-1} \left\lceil \frac{t}{P_j} \right\rceil (C_j) > f_i \times t - C_i.$$

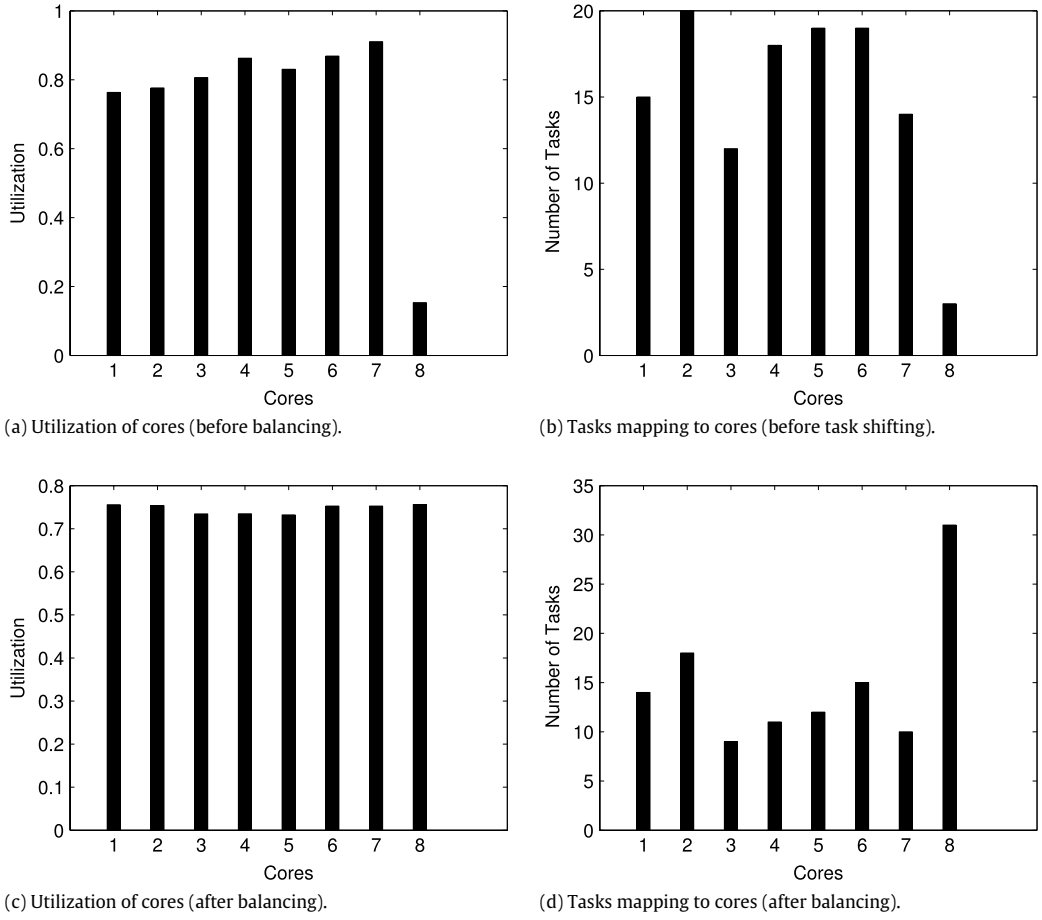
The aforementioned is true because $f_i \times t - C_i < f_i \times t$, C_i time units are reduced from core Δ_i by assuming that both cores run at the same speed f_i. □

Theorem 3. *If all the cores run at the same speed, then adding a task τ_i to Δ_i weakens the schedulability of Δ_i by C_i.*

Proof. Follows from Theorem 2. □

Theorem 4. *If all the cores run at the same speed, then no task can be added to the barely schedulable core.*

Proof. Let τ_i be a task such that in addition to the already schedulable $i-1$ tasks, is schedulable on a barely schedulable core Δ_i. The term barely schedulable refers to a system in which only δ_i ($0 < \delta_i < 1$) slots are available on a core Δ_i, i.e.



(a) Utilization of cores (before balancing).
(b) Tasks mapping to cores (before task shifting).
(c) Utilization of cores (after balancing).
(d) Tasks mapping to cores (after balancing).

Fig. 6. Load distribution on system with 8 cores.

$$\left(\sum_{j=1}^{i-1}\left\lceil\frac{t}{P_j}\right\rceil C_j\right)+\delta\le f_i\times t.$$

By adding τ_i to the aforementioned, we obtain

$$\left(C_i+\sum_{j=1}^{i-1}\left\lceil\frac{t}{P_j}\right\rceil C_j\right)+\delta\le f_i\times t,$$

$$\left(C_i+\sum_{j=1}^{i-1}\left\lceil\frac{t}{P_j}\right\rceil C_j\right)\le f_i\times t-\delta.$$

Because $(f_i\times t-\delta)<C_i$, we get

$$\left(C_i+\sum_{j=1}^{i-1}\left\lceil\frac{t}{P_j}\right\rceil C_j\right)>f_i\times t-\delta, \tag{21}$$

which shows that the available slot is small enough to accommodate the task τ_i. However, the aforementioned claim contradicts the assumption that τ_i is barely schedulable on Δ_i.

There are two possible cases to balance the load among all the cores of the underlying system, which we detail below.

Case 1. ($f_\Delta \le f_{\Delta_i}$): Because all the computation times are proportionate to the core speed, a lower speed core would prolong the task computations. Therefore, the task set Ω_i that was previously feasible at the speed of f_{Δ_i} on core Δ_i, now becomes infeasible at the speed of f_Δ. Care must be taken when migrating tasks from Ω_i to another core Δ_j. We must find the most underutilized core among all the cores that are operating at the speed f_Δ, which also offer space to accommodate more tasks. Once the particular core is identified, the task shifting (or migration) process begins. Let core Δ_i be the task donor and core Δ_j be the task acceptor, i.e. $(\Omega_i,\Delta_i,f_{\Delta_i})-\tau_l\in\Omega_i\,\forall l: U_l(\Delta_i)\le U_q(\Delta_i)\mid q=1,\ldots,k; q\ne l$. In our system, if a task $\tau_i\in\Omega_i$ is to be shifted to Ω_j, then the task with the lowest utilization on Δ_i is chosen as the candidate for shifting. The process continues until utilization of all the cores is leveled. This arrangement guarantees to meet all the task deadlines and to run all the cores at the same speed.

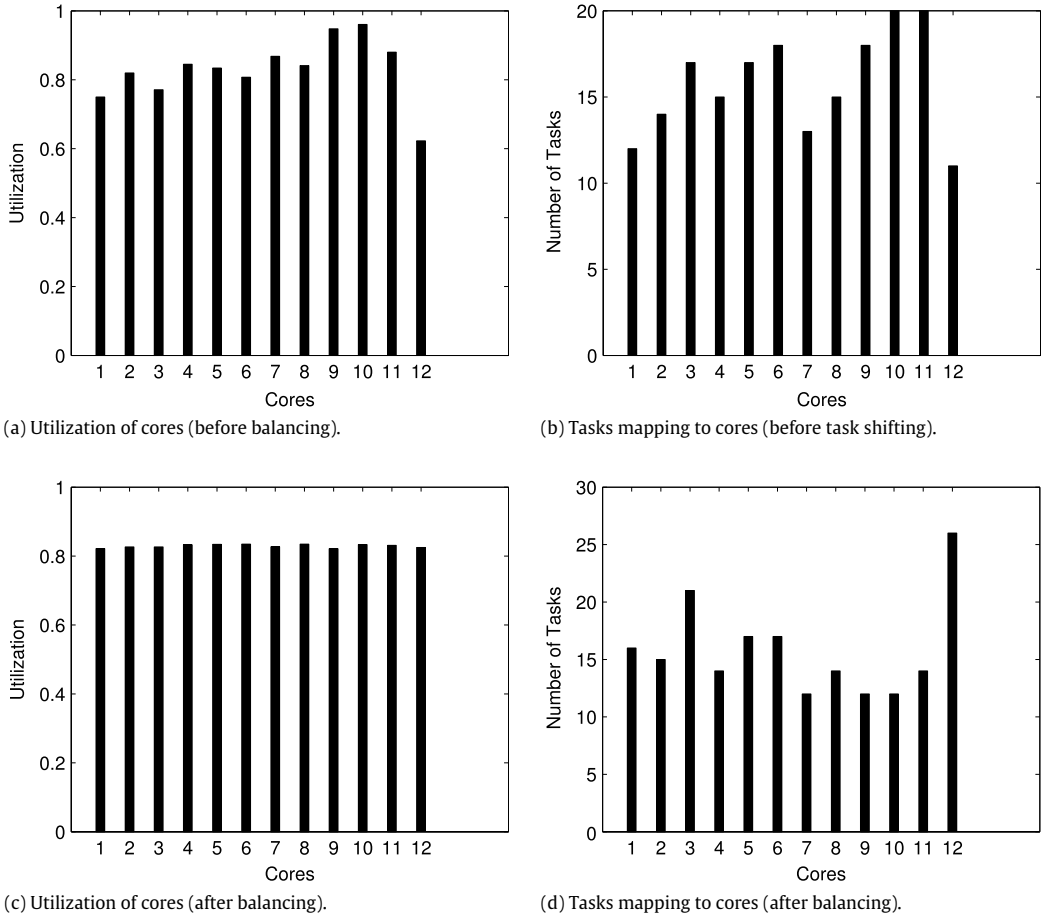
Case 2. ($f_\Delta > f_{\Delta_i}$): In this case, the task subset Ω_i is feasible on Δ_i and the core might be underutilized at the speed of f_Δ. As mentioned above, Δ_i can accommodate more tasks that are assigned to other cores within the system.

Once the cure utilization is balanced among all the cores under RM scheduling, a uniform speed f_Δ is recalculated. This uniform speed mandates that all the tasks are schedulable and allows the system to operate at the lowest possible speed. Therefore, the overall system power consumption is also reduced. In other words, it is the core utilization that decides the system speed and not the number of tasks. This is due to the fact that there may exist a core Δ_i that has a higher number of tasks than those assigned to a core Δ_j, while $U(\Delta_j) > U(\Delta_i)$. Therefore, Δ_j must operate at a higher speed than Δ_i. □

6. Task mapping on cores

In this section, we generate the task set with the same procedure as previously described in Sections 4.1 and 4.2. Initially, we start with a task set of size 120 to observe its mapping on 8 cores. We plot these results in Fig. 6, which are categorized into two domains: (i) Fig. 6(a) and (b) depict the results before applying the proposed strategy and (ii) Fig. 6(c) and (d) show the results after applying the proposed technique.

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005



(a) Utilization of cores (before balancing).

(b) Tasks mapping to cores (before task shifting).

(c) Utilization of cores (after balancing).

(d) Tasks mapping to cores (after balancing).

Fig. 7. Load distribution on system with 12 cores.

Fig. 6(a) reflects the case, where a task set of size 120 is distributed over 8 cores and some cores are heavily utilized than others. For instance, the utilization of core Δ_7 is the highest among all the 8 cores while Δ_8 has the lowest utilization. Similarly, Fig. 6(b) shows the corresponding number of tasks, where core Δ_2 has the highest number of tasks, while core Δ_8 has the lowest number of tasks. The load is balanced on the basis of core utilization; therefore, some of the tasks are shifted from core Δ_7 to the other cores. Because we have used the necessary and sufficient conditions in our work, the tasks are assigned to the cores based on the exact feasibility analysis. That is, when a core, say Δ_1, is assigned a certain number of tasks, the rest of the tasks are mapped onto the next core Δ_2. The same is observed from Fig. 6, where the cores 1 through 7 are fully utilized while core Δ8 remains underutilized. This is due to the fact that fewer tasks are left for core Δ_8. It can also be deduced from Fig. 6(a) and (b) that Δ_3 has a utilization of 80%, while the total number of tasks assigned is only 12 (2nd lowest after core Δ_8 in the system, see Fig. 6(b)). After applying our proposed technique, the results are plotted in Fig. 6(c) and (d). The core utilization is almost the same; however, the number of tasks assigned to the cores is not uniform (see Fig. 6(d)).

We further increase the number of the cores to 12 and distribute the workload of the task set of size 190. The corresponding results are shown in Fig. 7. Although we have applied a heavy system load, it can be seen from Fig. 6(a) and (c), and Fig. 7(a) and (c) that the utilization of a core never reaches 100%, which is due to the implicit characteristics of the RM scheduling algorithm. Fig. 7(a) and (b) report the utilization and the task mapping of cores before load balancing, while Fig. 7(c) and (d) plot the results after applying the task shifting technique. From Fig. 7(a), we can deduce that core Δ_9 and core Δ_{10} are heavily utilized, while Fig. 7(b) reports that core Δ_{10} and core Δ_{11} have the maximum number of tasks assigned. It is worth mentioning that the task shifting is performed in such a way that the lightest task among all the assigned tasks to the maximum utilized core is shifted to the minimum utilized core. This arrangement: (i) results in minimum possible workload shifting from the higher to a lower utilized core and (ii) does not violate the timing constraints of the already assigned tasks to the cores. It can be observed from Fig. 7(b) and (d) that 15 tasks from other cores are shifted to core Δ_{12} under the load balancing mechanism. Interestingly, it can also be observed from Fig. 7(c) that core Δ_7 has the highest number of tasks (26 in total), while core Δ_7, Δ_9, and Δ_{10} have the lowest number of tasks (12 each). However, the utilization of all the cores is almost the same as reported in Fig. 7(c). Moreover, initially, core Δ_{12} was underutilized as can be observed from Fig. 7(a) and had the lowest number of task assignments. Because of the shifting of the lightest task from other cores, the tasks assigned to core Δ_{12} is the highest (26 as can be seen from Fig. 7(d)). However, the utilization is balanced with the remaining cores (see Fig. 7(c)). Therefore, all the cores can now be run on a uniform speed, which was the intention of this work.

As reflected in Figs. 6(c, d) and 7(c, d), any further task shifting is not possible until task splitting techniques are applied. Since we do not consider the task splitting case here, there might be situations of uniform utilization and hence uniform speed will not be possible. In such cases, the speed assigned to the system is the speed of the core that is highly utilized.

7. Concluding remarks

In this paper, we integrated dynamic voltage scaling with the fixed priority scheduling paradigm. A solution was proposed to

Please cite this article in press as: N. Min-Allah, et al., Power efficient rate monotonic scheduling for multi-core systems, J. Parallel Distrib. Comput. (2011), doi:10.1016/j.jpdc.2011.07.005

find the lowest possible core speed for a single task. The proposed technique was then applied to the multi-core system to identify a uniform system speed to conserve energy while maintaining the system timing requirements. The proposed methodology was compared to existing techniques and the simulation results revealed superior performance.

As future work, we expect promising results when the tasks are split among the cores and the task assignment problem is addressed by naturally inspired algorithms.

Acknowledgments

The authors are extremely grateful to Rob Davis (University of York) for his valuable suggestions on the initial draft of this paper.

References

[1] N. AbouGhazaleh, B. Childers, D. Mosse, R. Melhem, M. Craven, Energy management for real-time embedded applications with compiler support, in: ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems, 2003, pp. 284–293.
[2] J. Anderson, S. Baruah, Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms, in: Proc. Distributed Computing Systems, 24th International Conference, 2004, pp. 428–435.
[3] H. Aydin, R. Melhem, D. Mosse, P. Alvarez, Dynamic and aggressive scheduling techniques for power-aware real-time systems, in: Proc. IEEE Real-Time Syst. Symp., 2001, p. 95.
[4] E. Bini, G.C. Buttazzo, G. Buttazzo, Rate monotonic analysis: the hyperbolic bound, IEEE Trans. Comput. 7 (52) (2003) 933–942.
[5] E. Bini, G.C. Buttazzo, G. lipari, Minimizing CPU energy in real-time systems with discrete speed management, ACM Trans. Embedded Comput. Syst. 8 (4) (2009).
[6] J. Brateman, C. Xian, Y. Lu, Frequency and speed setting for energy conservation in autonomous mobile robots, in: Proceedings of the IFIP International Federation for Information Processing, vol. 249/2008, 2008, pp. 197–216.
[7] T.D. Burd, T.A. Pering, A.J. Stratakos, R.W. Brodersen, A dynamic voltage scaled microprocessor system, IEEE J. Solid State Circuits 35 (11) (2000) 1571–1580.
[8] A. Burns, A.J. Wellings, Real-Time Systems and Programming Languages, 4th ed., Addison Wesley, 2009, 602 pages.
[9] A.P. Chandrakasan, R.W. Brodersen, Low Power Design, Kluwer Academic Publishers, Dordrecht, 1995.
[10] A.P. Chandrakasan, S. Sheng, R.W. Brodersen, Low power CMOS digital design, IEEE J. Solid State Circuits (1992) 472–484.
[11] Crusoe Processor Model TM5800 Specifications, http://www.charmed.com/PDF/TM5800.pdf,2011.
[12] R.I. Davis, T. Rothvo, S.K. Baruah, A. Burns, Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling, Real Time Syst. 43 (3) (2009) 211–258.
[13] L. George, N. Riverre N, M. Spuri, Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling, Research Report 2966, INRIA, France, 1996.
[14] T. Gloker, H. Meyr, Design of Energy-Efficient Application-Specific Instruction Set Processors, Kluwer Academic Publisher, Dordrecht, 2004.
[15] E. Humenay, D. Tarjan, K. Skadron, Impact of process variations on multicore performance symmetry, in: Proceedings of the Conference on Design, Automation and Test in Europe, 2007, pp. 1653–1658.
[16] T. Ishihara, H. Yashura, Voltage scheduling problem for dynamically variable voltage processors, in: International Symposium on Low Power Electronics and Design, 1998, pp. 197–202.
[17] J.L.W.V. Jensen, Sur les fonctions convexes et les inegalites entreles valeurs moyennes, Acta Math. 30 (1) (1906) 175–193.
[18] C.M. Krishna, Kang G. Shin, Real-time Systems, Tsinghua University Press, McGraw-Hill, 2001.
[19] K. Lakshmanan, R. Rajkumar, J.P. Lehoczky, Partitioned fixed-priority preemptive scheduling for multi-core processors, in: Proceedings of the 21st Euromicro Conference on Real-Time Systems, 2009, pp. 239–148.
[20] W. Lee, H. Kim, H. Lee, Maximum-utility scheduling of operation modes with probabilistic task execution times under energy constraints, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 28 (10) (2009) 1531.
[21] J.P. Lehoczky, Fixed priority scheduling of periodic task sets with arbitrary deadline, in: Proceedings of the 11-th IEEE Real-Time System Symposium, 1990, pp. 201–209.
[22] J.Y.T. Leung, J. Whitehead J., On the complexity of fixed-priority scheduling of periodic, Real-time tasks performance evaluation 2 (1982) 237–250.
[23] J.W.S. Liu, Real Time Systems, Prentice Hall, 2000.
[24] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 40–61.
[25] F. Li, F.F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM J. Comput. 35 (2005) 658–671.
[26] N. Min-Allah, I. Ali, J. Xing, Y. Wang, Utilization bound for periodic task set with composite deadline, J. Comput. Electr. Eng. 36 (6) (2010) 1101–1109.
[27] N. Min-Allah, S.U. Khan, A hybrid test for faster feasibility analysis of periodic tasks, IJICIC 7 (10) (2011) 5689–5698.
[28] N. Min-Allah, SU Khan, Y. Wang, Optimal task execution times for periodic tasks using nonlinear constrained optimization, J. Supercomput. (2010) doi:10.1007/s11227-010-0506-z.
[29] N. Min-Allah, Y. Wang, X. Jian-Sheng, J. Liu, Revisiting fixed priority techniques, in: Proceedings of Embedded and Ubiquitous Computing, EUC07, in: LNCS, vol. 4808, 2007, pp. 134–145.
[30] P. Pillai, K.G. Shin, Real-time dynamic voltage scaling for lowpower embedded operating systems, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles, 2001, pp. 21–24.
[31] V. Raghunathan, C. Pereira, M. Srivastava, R. Gupta, Energy aware wireless systems with adaptive power-fidelity tradeoffs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 13 (2) (2005).
[32] S. Saewong, R. Rajkumar, Practical voltage-scaling for fixedpriority rt-systems, in: Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS03, 2003, pp. 106–115.
[33] J. Sartori, A. Pant, R. Kumar, P. Gupta, Variation aware speed binning of multi-core processors, in: Proceedings of the 11-th IEEE International Symposium on Quality Electronic Design, 2010, pp. 307–314.
[34] E. Seo, Y. Koo, J. Lee, Dynamic repartitioning of real-time schedule on a multicore processor for energy efficiency, in: LNCS, vol. 4096/2006, 2006, pp. 69–78.
[35] H. Xin, L. KenLi, L. RenFa, A energy efficient scheduling base on dynamic voltage and frequency scaling for multi-core embedded real-time system, in: Algorithms and Architectures for Parallel Processing, in: LNCS, vol. 5574, 2009, pp. 137–145 (Chapter).
[36] F. Zhang, S. Chanson, Processor voltage scheduling for realtime tasks with non-preemptible sections, in: Real-Time System Symposium, Austin, TX, Dec. 2002.

Nasro Min-Allah received his Undergraduate and Master's degrees in Electronics and Information Technology in 1998 and 2001, respectively. He obtained a Ph.D. in real-time systems from the graduate university of the Chinese Academy of Sciences, PR China in 2008. Currently, he is the Head of the Department of Computer Science, CIIT, Pakistan. He is the author of enormous research articles and book chapters. His main research is focused on scheduling theory, green computing, and fault tolerant real-time systems. He is a member of the editorial boards of several international journals. He is the recipient of the two most prestigious awards, (i) CIIT Golden Medallion for Innovation (CIMI-2009), and Best Mobile Innovation in Pakistan (BMIP-2011).

Hameed Hussain received his Undergraduate degree in Information Technology in 2007 from the University of Peshawar, Pakistan. He did his M.S. Degree in Computer Science in 2009 from COMSATS Institute of Information Technology (CIIT), Pakistan. He is the author of several international publications. He is currently pursuing his Ph.D. Degree at CIIT, Pakistan. He is an active researcher and his research interest includes real-time systems, resource allocation and load balancing in high performance computing systems.

Samee Ullah Khan is an Assistant Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan has extensively worked on the general topic of resource allocation in autonomous heterogeneous distributed computing systems. As of late, he has been actively conducting cutting-edge research on energy-efficient computations and communications. A total of 107 (journal: 37, conference: 50, book chapter: 12, editorial: 5, technical report: 3) publications are attributed to his name. For more information, please visit: http://sameekhan.org/.

Albert Y. Zomaya is currently the Chair Professor of High Performance Computing & Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing which was established in late 2009. Professor Zomaya is the author/co-author of seven books, more than 400 papers, and the editor of nine books and 11 conference proceedings. He is the Editor in Chief of the IEEE Transactions on Computers and serves as an associate editor for 19 leading journals. Professor Zomaya is the recipient of the Meritorious Service Award (in 2000) and the Golden Core Recognition (in 2006), both from the IEEE Computer Society. Also, he received the IEEE TCPP Outstanding Service Award and the IEEE TCSC Medal for Excellence in Scalable Computing, both in 2011. Professor Zomaya is a Chartered Engineer, a Fellow of AAAS, IEEE, IET (UK), and a Distinguished Engineer of the ACM.