Energy-Efficient Scheduling of Real-Time Periodic Tasks in Multicore Systems

Xiaodong Wu, Yuan Lin, Jian-Jun Han, Jean-Luc Gaudiot

Energy-Efficient Scheduling of Real-Time Periodic Tasks in Multicore Systems

Xiaodong Wu[1], Yuan Lin[1], Jian-Jun Han[1], Jean-Luc Gaudiot[2]

[1]College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei province, China
[2]Department of Electrical Engineering and Computer Science, University of California, Irvine, U.S.A.
cdxiaodongwu@gmail.com, gamespiay123@163.com, jasonhan@mail.hust.edu.cn, gaudiot@uci.edu

Abstract. Nowadays, energy savings have become one of the most critical issues. In this paper, we propose an energy-efficient approach to scheduling periodic real-time tasks in the multicore context. Within a voltage/frequency domain (VFD), a simple static voltage/frequency scaling schedule (SimpleVS) is first introduced to select the utilization of the heaviest-loaded core as the shared operating frequency of this VFD. Next, the slack reallocation policy is proposed to further reclaim slack times while satisfying timeliness requirements. The slack reallocation strives to redistribute the slack times uniformly to the cores on the same VFD by appropriate job migrations. Experimental results show that compared with the static frequency schedule, our proposed policy can achieve energy savings up to 22 percent when the system is under-utilized.

Keywords: energy-efficient, multicore systems, real-time scheduling

1 Introduction

Nowadays power has become the most critical design constraint. The increasing on-chip power dissipation and the demand of portability make low power consumption one of the primary circuit and system design goals. This not only applies to battery powered devices, but also to desktop computers and high-performance systems.

Many solutions have been proposed to mitigate the energy pressure. The processor industry is now shifting towards Chip Multi-Processors (CMP) architectures [1] by integrating several cores onto a single chip to reduce the power consumption. Multicore systems usually can deliver a higher throughput at the cost of lower power consumption than uniprocessor systems. As the demand for energy efficiency grows, multicore processors have been widely used in real-time systems. It is well known that the dynamic voltage/ frequency scaling (DVFS) [2- 6] is one of the most promising solutions to optimizing the energy consumption, which scales the operating voltage and clock frequency of processors together to meet dynamic performance demands. Pillai and Shin [2] present a class of real-time DVS (RT-DVS) algorithms,

including the static voltage scaling, cycle-conserving RT-DVS and Look-ahead RT-DVS, which modify the OS's real-time scheduler and task management service to provide energy savings while maintaining real-time deadline guarantees. However, in the context of multicore systems, all active cores need to work at the same operating voltage/frequency [7,8]. This characteristic of the multicore system is referred to as *frequency synchronization* in this work. This implies that when the operation frequency is scaled on one core, the performances of the other cores are intuitively influenced. For instance, if a low speed is selected for a processor core to aggressively reduce the energy consumption, the jobs running on other cores may miss their deadlines if there are heavy workloads mapped onto those processor cores.

Recently, Globally Asynchronous, Locally Synchronous (GALS) [9- 12] design is being widely investigated as an alternative solution to the totally synchronous voltage/frequency design, because it inherits the advantages of both synchronous and asynchronous design. The GALS technology makes the clock distribution and timing closure more manageable. On the other hand, the GALS can be well adapted to the voltage-frequency island (VFI) technology [12,13]. Multicore systems implemented with the GALS are partitioned into several voltage/frequency domains (VFD). In such systems, each core belongs to a specific VFD, while the active cores within the same VFD must share the same supply voltage and operating frequency. The operating frequency/voltage of each domain can be adjusted independently of other domains so as to meet the dynamic performance demands. Moreover, considering the fact that the processor cores in a VFD-based GALS system can be easily put into power saving state with the clock gating technique, an idle processor core or even a complete domain can be placed into the sleep mode to further reduce both the dynamic and the leakage power [12].

Some energy-aware approaches using DVFS have been presented to schedule real-time tasks on multicore architectures. Yang et al. [8] proved that the energy-efficient scheduling of periodic real-time task set in the multicore context is an NP-hard problem. Seo et al. [14] presented a dynamic repartition algorithm for periodic tasks to balance the system load as well as to reduce the leakage power at run-time. Pepijn et al. [15] tried to reduce the leakage current, supply voltage and clock frequency in an integrated way in order to gain the maximum system energy savings. Dinna et al. [16] proposed a power-aware real-time scheduler for the multicore multithreaded processor, which implements dynamic voltage scaling techniques. But they aimed at the soft real-time applications and supposed that all cores in the system shared a global frequency. Wan [17] addressed an energy-saving scheduling scheme of periodic real-time tasks with the DVFS on the lightly loaded multicore platform. However [17] assumed that the tasks can be divided into multiple independent subtasks and concurrently executed in parallel on multiple cores. It is also presumed there are more processing cores than running tasks and each core could adjust its operating frequency independently.

The previous studies usually concentrated on two power management mechanisms. They either assumed that each processor core can adjust its operating frequency independently of the other cores, or presumed the global frequency synchronization, namely, all active cores in the system share the same operating voltage/frequency.

In this paper, we will address the problem of the energy-efficient scheduling for periodic real-time tasks in the VFD-based multicore system. The proposed scheduling

algorithm strives to balance the slack times on the cores in VFDs so as to scale down the operating frequency and reduce energy consumptions, while meeting timing constraints.

Given a set of periodic real-time tasks, we define the hyperperiod as the least common multiple (LCM) of the periods of these tasks. Intuitively, each task releases its invocations at the same relative phases of different hyperperiods. For this reason, based upon the worst-case execution time of the tasks, we off-line simulate the dynamic executions of the tasks in the interval of one hyperperiod, then off-line calculate and record the start times as well as the executing speeds of the invocations of each task. At run-time, for each invocation of a task, the scheduler only needs to look up its scheduling information from a pre-determined table. As a result, the on-line scheduling overhead can be significantly reduced.

For uniprocessor systems, the static voltage scaling mechanism is proposed in [2] to select the lowest possible operating frequency that will allow the scheduler to meet all the deadlines for the given task set. This static policy set frequency statically, and the frequency will not be changed unless the task set is changed. We will extend the static voltage scaling policy to the multicore platform (called SimpleVS). The SimpleVS algorithm selects the maximum utilization among all cores in a VFD as the shared operating frequency, such that the frequency synchronization of the VFD, the deadline constraints of all tasks and the energy savings can be met simultaneously.

Based on the SimpleVS schedule, we propose a slack reallocation algorithm to further reclaim the slack times to conserve energy. This policy adjusts the slack times on each core and divides the schedule into segments. Then, for each segment, it seeks to balance the slack times on cores within a VFD via appropriate job migrations. Hence, almost the same amount of slack time will be allocated to each core in a VFD, thereby a lower operating frequency can be selected for this VFD so as to save energy.

We evaluated the performance of our proposed algorithm through extensive simulation experiment under different situations. The results show that compared with the static voltage scaling schedule (SimpleVS), the proposed slack reallocation policy can gain the energy savings by about 22% under low-loaded conditions.

The rest of this paper is organized as follows: Section 2 describes the system models, including the energy consumption and the task set. Section 3 introduces the static SimpleVS and gives motivational examples to illustrate the target problem. The proposed energy-efficient scheduling algorithm and the experimental results are discussed in detail in Section 4 and Section 5, respectively. Section 6 concludes this paper and offers our future work.

2 Research Model

2.1 System Model

In this research, we consider the multicore system with voltage-scalable processor elements for our study. There are cn identical cores in the system: $C = \{c_1, c_2 ..., c_{cn}\}$.

Furthermore, the GALS and VFD are implemented in our target platform. All those cores are partitioned into dn VFDs: $D = \{d_1, d_2 ..., d_{dn}\}$. The i^{th} domain d_i contains dcn_i processor cores, and then we have: $cn = \sum_{i=1}^{dn} dcn_i$.

In the target system, each core has two states: sleep and active. If all tasks on a core finish their executions, this core can be placed into the sleep state for the sake of energy reduction. Further, a VFD is considered to be put in the sleep mode if all cores on this VFI are in the sleep state. A VFD can control its supply voltage and operating frequency independently of other VFDs, while all active cores in the same VFD must share the same operating frequency at any given time. Here, the maximum operating frequency of a core is normalized to 1.

2.2 Energy and Power Model

As for the CMOS-based processor, the total power consumption consists of dynamic power $P_{dynamic}$ and leakage power $P_{leakage}$. The detailed energy and power model used in this paper can be found in [14]. For some very large scale integrated systems, $P_{leakage}$ is also a critical issue because the off-state current increases about five times per generation [18]. However, the dynamic power still dominates the overall energy consumed by a processor core [19]. Therefore, we only target reducing the dynamic energy consumptions in this paper.

For simplicity, we assume that the overheads of voltage/frequency transition and job migration are negligible in this paper. However, these overheads can be easily incorporated into the processing time of a task, if necessary.

2.3 Task Model

The task set to be scheduled is composed of n periodic tasks: $T = \{t_1, t_2, ..., t_n\}$. Each task t_i is associated with a tuple $\{p_i, w_i\}$, where p_i denotes the period of task t_i and w_i the worst-case execution time (WCET) of t_i at the maximum processing speed. Here, the period of any task t_i is assumed to be the same as the relative deadline of this task. Then, the load of task t_i , tl_i , is defined as w_i / p_i . Each invocation of the task is called a job and the k^{th} invocation of task t_i is denoted by j_{ik} . Furthermore, every task starts to release jobs at time 0. If the number of tasks mapped onto c_i is ctn_i , then we can obtain: $n = \sum_{i=1}^{cn} ctn_i$.

We use $\Re(t_i)$ to indicate the core onto which task t_i is mapped. It is noteworthy that all jobs invoked by any task t_i must execute on core $\Re(t_i)$ except for job migrations. Then, the utilization of core c_i , cu_i , can be defined as: $cu_i = \sum_{\Re(t_k)=c_i} tl_k$.

Thus, the average utilization acu_i of a VFD d_i is: $acu_i = \sum_{c_k \in d_i} cu_k \Big/ dcn_i$, where dcn_i is the number of cores in the VFD d_i.

It is assumed that there are no interdependencies among the tasks, that is, no common resources are shared by these tasks and they have no precedence relationships. The tasks are scheduled by the EDF in a preemptive way on any processor core. Due to the repeatable job release of a task, we only need to concentrate on the schedule within the interval of a hyperperiod. In addition, the schedule is calculated off-line once and for all such that the on-line scheduling cost can be reduced.

Since data transfer across different VFD will result in more energy consumption as well as a high overhead, the job migration is only permitted within the same VFD by our approach.

3 Motivational Examples

In this section we give the motivational examples and introduce the SimpleVS that extends the static voltage scaling to the multicore platform.

For the target multicore system, each VFD can control its supply voltage and operating frequency independently of other VFDs. Further, as long as the schedule of each VFD is carried out, the global schedule result can be easily obtained. Hence, for simplicity, we will focus on the energy-efficient schedule within one VFD.

Consider scheduling six real-time periodic tasks in a VFD with three cores. The parameters of the tasks are listed in Table 1. It is easy to see that the hyperperiod of this task set is 12. Since we mainly concern with the schedule during a hyperperiod, the tasks are scheduled within the interval [0, 12].

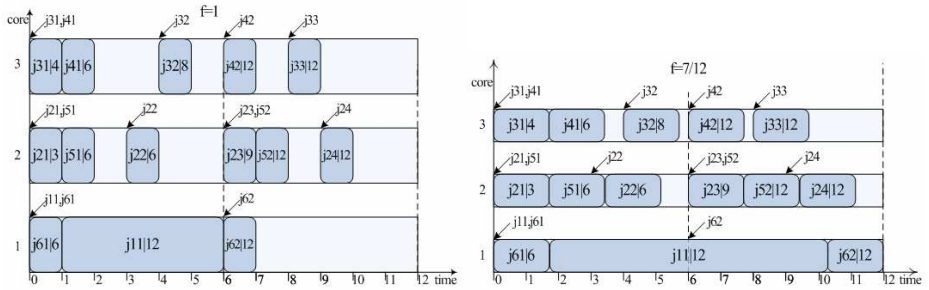
Table 1. Parameters of the task set

Task ID	WCET	Period	Utilization
1	5	12	5/12
2	1	3	1/ 3
3	1	4	1/ 4
4	1	6	1/ 6
5	1	6	1/ 6
6	1	6	1/ 6

Table 2. Task-core mapping

Core ID	Task ID	Utilization
1	1, 6	7/12
2	2, 5	1/ 2
3	3, 4	5/12

The task-core mapping listed in Table 2 is determined by the Worst-Fit Decreasing (WFD) policy (see Subsection 4.2). As can be seen, the heaviest- loaded core among all cores is core 1 and the corresponding utilization of core 1 is 7/12.



(a) (b)

Fig. 1. (a) Schedule the tasks by EDF algorithm without the voltage scaling. (b) Schedule the tasks with the static SimpleVS algorithm.

In Fig. 1, each job is denoted by a block labeled with the job ID and its absolute deadline. For instance, the block with label "$j11|12$" denotes the job j_{11} with its absolute deadline 12. Fig. 1 (a) shows the EDF schedule of the task set without energy-saving policy, where all cores work at the maximum operating frequency. In contrast, as shown in Fig. 1 (b), SimpleVS algorithm chooses the maximum utilization among all cores as the shared operating frequency of this VFD so as to conserve energy, while ensuring the schedulability of the task set. Consequently, due to the frequency synchronization, the operating frequencies of all cores in the VFD are scaled down to 7/12. Then, the slack on core 1 is fully utilized. In Fig. 1 (b), 73.6% dynamic energy can be saved by the SimpleVS scheduling versus the NonVS scheduling as shown in Fig. 1 (a).

However, we can observe from Fig. 1(b) that, there are still some slack times unused on the cores with light utilizations (e.g., core 2 and 3) because of the frequency synchronization. As a matter of fact, these slack times can be utilized to further reduce the energy consumption. To this end, we propose the slack reclamation approach.

4 Hyperperiod-Based Multicore Voltage Scaling Schedule

This section describes in detail our proposed hyperperiod-based multicore energy-efficient voltage scaling scheduling algorithm (called as HMVS). We first present the high-level description of our approach.

4.1 The High Level Description

The proposed approach is based upon an EDF schedule. Then, the slack times are reclaimed and utilized by two strategies, i.e., SimpleVS and slack reallocation, to conserve energy.

The high level description of the proposed HMVS is shown in Algorithm 1. First, all tasks are mapped onto the cores by the heuristic of WFD in line 1 (see Subsection

4.2). Then, we use two policies to reclaim the slack times for the energy savings on a per-VFD basis. Finally, the execution speed is calculated and assigned to each job.

Algorithm 1: High Level Description of HMVS

```
1: Map tasks onto processor cores by WFD (see Subsection 4.2);
2: foreach d_i ∈ D do
3:   Schedule the tasks by EDF without voltage scaling;
4:   Use SimpleVS to reclaim slack times;
5:   Adjust slack times on each core (see Subsection 4.3);
6:   Reclaim slack times by slack reallocation (see Subsection 4.4);
7: endfor;
8: Calculate the execution speed for each job;
```

In line 3, the tasks are scheduled by the NonVS (the EDF without DVFS) algorithm on each core. Next, the SimpleVS policy tries to utilize the slack times by selecting the maximum utilization among all cores as the shared operating frequency of this VFD. However, the majority of the slack times are usually generated closely to the end of a hyperperiod, which prevents the slack reallocation from effectively exploring these slack times. For this reason, line 5 tries to move the slack times forward. At the same time, the start times of some jobs are postponed. Fig. 2 (a) (see Subsection 4.4) shows the scenario after the slack shifts have been applied. Then, the slack reallocation in line 6 strives to redistribute the slack times that are not reclaimed by the SimpleVS uniformly to the cores by appropriate job migrations.

4.2 Task-Core Mapping Heuristic

Many heuristics have been introduced to map the tasks onto processors, such as Best-Fit Decreasing (BFD), First-Fit Decreasing (FFD), Next-Fit Decreasing (NFD), Worst-Fit Decreasing (WFD), etc. Aydin and Yang [20] pointed out that balancing the utilizations of the processing elements facilitates maximizing the energy savings. They further claimed that the WFD enables to generate a better balanced partition than other well-known heuristics. Therefore, we choose the WFD as the policy of task-core assignment for our study.

Before the tasks are assigned to the processor cores, they are sorted in a non-increasing order of their loads (i.e., tl). Next, these ordered tasks are partitioned into cn subsets by the WFD. Then, we allocate these subsets of tasks to the cn processor cores respectively. Consequently, a quasi-balanced task-core mapping can be obtained.

4.3 Moving Slack Times Forward

Observe that the slack times are usually distributed closely to the end of the hyperperiod, which prevents efficient slack utilizing. Hence, we adjust the slack times in the original schedule, while maintaining the schedulability of these tasks.

As mentioned before, we off-line simulate the dynamic executions of the tasks, and record the necessary information of jobs on each core. Assume that the schedule

determined by the SimpleVS on a core c_i is represented by $tsn[i]$ continuous time slices, $c_i.Sched = \{ts_1, ts_2, ..., ts_{tsn[i]}\}$. These time slices are sorted in an increasing order of their start times. The time interval of a slice ts is denoted by $[ts.starttime, ts.endtime]$. Since the preemptive scheduling is applied, a job may execute in several time slices because of the job preemptions. A time slice ts has a predecessor $ts.previous$ and a successor $ts.next$ except that it is the first or the last time slice of the schedule. Then we have: $ts.starttime = ts.previous.endtime$, $ts.endtime = ts.next.starttime$.

Each time slice has two states: *active* or *idle*. If there exists one job executing during the time slice ts, then we say this time slice is active: $ts.state = active$. Further, this job and its absolute deadline are represented by $ts.job$ and $ts.job.deadline$, respectively. Otherwise, this time slice is in the idle status in the sense that it is a slack: $ts.state = idle$. Algorithm 2 describes how our proposed slack-shifting policy adjusts the schedule on a core.

The slack adjustment algorithm tries to postpone the start times of job executions during the active time slices, while the idle time slices (i.e., slack times) are moved "forward" at the same time. The shifting operation is performed from the last time slice to the first one on each core. From line 4 to line 8, the algorithm shifts the current time slice ts backward when the following three conditions are satisfied simultaneously: 1) the current time slice is active, 2) the next one is idle, and 3) the absolute deadline of the current-running job is greater than the completion time of the current time slice. The algorithm delays the active time slice ts while ensuring that it will complete execution before $ts.job.deadline$ (line 5). Thus, it can be inferred that this adjustment does not violate the schedulability of any task. Fig. 2 (a) illustrates the scenario after the slack adjustment has been applied.

Algorithm 2: Moving Slack Times Forward

```
1: Input: the schedule on core c_i , c_i.Sched ;
2: Output: the schedule with slack times shifted forward;
3: foreach ts ∈ c_i.Sched from tail to head do
4:   if ( ts.state = active ∧ ts.next.state = idle ∧ ts.job.deadline > ts.endtime )
5:     newendtime = min{ts.job.deadline, ts.next.endtime} ;
6:     Move ts backward such that: ts.endtime = newendtime ;
7:     Update the starttmie , endtime and state of the two adjacent time slices:
         ts.previous and ts.next ;
8:   endif;
9: endfor;
```

4.4 Slack Reallocation

The slack reallocation is adopted after the slack adjustment. First, the whole hyperperiod is divided into segments for each VFD prior to the slack reallocation. Then the slack reallocation is performed on a per-segment basis. This policy attempts

to redistribute the available slack times to the cores by job migrations so that a lower operating frequency of a VFD can be obtained.

We divide the hyperperiod into *sn* consecutive and non-overlapped segments in a VFD: $SG = \{sg_1, sg_2, ...sg_{sn}\}$. The segment sg_i starts from $sg_i.starttime$ and ends at $sg_i.endtime$. Suppose that SG is determined by $sn+1$ boundary points: $B = \{b_0, b_1, ..., b_{sn}\}$. The start and end time of the segment sg_i are denoted by b_{i-1} and b_i ($1 \le i \le sn$), respectively. The boundary points are defined as following:

$$b_0 = 0, b_{sn} = hyperperiod, \tag{1}$$

$$b_i = \min\{ts.starttime \mid ts \in \bigcup_{i=1}^{dcn} c_i.Sched \wedge ts.starttime > b_{i-1} \wedge ts.stae = idle\}, 0 < i < sn, \tag{2}$$

where *dcn* is the number of cores contained in the VFD, and $c_i.Sched$ is defined in Section 4.3. Thus, *SG* can be determined by (1) and (2).

The reason the segments are defined in such a way is that when a job executes, only the available slack prior to its execution is allowed to be used by this job to lower its executing speed. Therefore, the schedulability of the task set can be guaranteed.

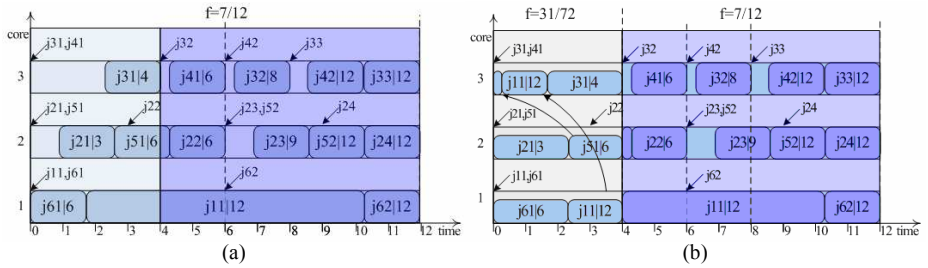


Fig. 2. (a) Adjust the slack times by shifting them forward. (b) Reallocate the slack times and scale down the operation frequency in [0, 4].

After the segments have been determined, some time slices may be separated by these segments as well (e.g., j_{51} on core 2). We re-partition these time slices such that each time slice only belongs to one segment. As shown in Fig. 2 (b), the interval of the hyperperiod [0, 12] is divided into 4 segments by the dashed lines.

In multicore systems, all cores usually share the last-level cache and main memory. This implies the cost of context switching caused by the migration of a job from a core to another is relative low. However, we constrain frequent job migrations to reduce the on-line overhead. Hence, the degree of un- balancedness *ub* for a segment *sg* in a VFD is introduced to help improve the trade-off between the overhead and the energy-saving for each segment:

$$ub = \frac{\max\{slack_i \mid 1 \le i \le dcn\} - \min\{slack_i \mid 1 \le i \le dcn\}}{\max\{slack_i \mid 1 \le i \le dcn\}} \times 100\%, \tag{3}$$

where $slack_i$ is the available amount of slack times on c_i during sg, and dcn is the number of cores contained in the VFD. The job migration is applied only when the degree of unbalancedness is greater than a given threshold value. Further, the number of migrations for each job is constrained by an upper bound mb during its execution in order to reduce the overhead caused by the job migrations. Hence, the trade-off between the energy-saving and the overhead can be flexibly controlled.

In this work, the threshold of the degree of unbalancedness ub and the upper bound of migrations for a job mb are set to 10% and 3, respectively.

Algorithm 3 illustrates the slack reallocation within a VFD d_i. The slack reallocation is performed on a per-segment basis (line 1). The amount of slack on each core in VFD d_i and the ub within the current segment sg are calculated in line 2 and line 3, respectively. Then, the total amount of slack times in VFD d_i is summed and the amount of slack that can be ideally distributed to each core is calculated in line 4. From line 5 to line 11, the algorithm strives to balance the slack times on the cores in d_i within the current segment sg until ub is less than the given threshold 10%. If the current core c_j has more slack than the ideal slack, then line 7 selects the core $c_{\min}$ with the minimum slack. Next, line 8 tries to perform the job migrations from $c_{\min}$ to c_j so as to reduce the slack on c_j to the ideal slack. Thus, the slack times distributed on all cores can be more evenly.

Algorithm 3: Slack Reallocation for VFD d_i

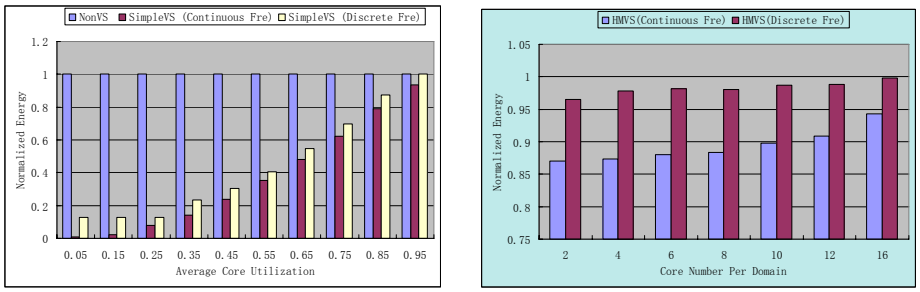
```
1: foreach segment sg in SG do
2:   Calculate the slack within sg on each core c_j ∈ d_i as slack_j ;
3:   Calculate the degree of unbalancedness ub ;
4:   Sum up the slack times to sumslack and compute the ideal slack
       as: idealslack = sumslack / dcn_i ;
5:   foreach c_j ∈ d_i do
6:     if ( slack_j > idealslack ∧ ub ≥ 10% )
7:       Choose the core with minimum slack: c_min ;
8:       Migrate jobs from c_min to c_j , try to reduce the slack on
           c_j to idealslack ;
9:       Re-calculate ub ;
10:    endif;
11:  endfor;
12: endfor;
```

It is noteworthy that the job migrations are on the basis of maintaining the schedulability. Further, unlike [17], we assume that one job is disallowed to execute on different cores at the same time (i.e., *serial-run-rule*). As a result, after the slack reallocation is processed, the slack times are not necessarily distributed absolutely uniformly on the cores in the system.

5 Experimental Results

In this section, the simulation experiments are used to evaluate the performance of the proposed scheduling approach with respect to energy consumption. In this experiment, the simulator scheduled the randomly generated tasks and recorded the energy consumptions within a hyperperiod under different scheduling policies, i.e., NonVS, SimpleVS, and the proposed scheduling with slack reallocation (HMVS). Usually, the current CMOS- based systems provide discrete operating voltage/frequency. However, we still evaluate the performance of our algorithm under continuous voltage/ frequency levels as well as discrete voltage/frequency levels in this experiment. The parameters *ub* and *mb* are set to 10% and 3, respectively. Namely, the slack reallocation and job migrations are performed only when the degree of unbalancedness (defined in Subsection 4.4) is greater than 10% and one job cannot be migrated more than 3 times. Recall that the maximum frequency of any processor core is normalized to 1. In the case of discrete frequency levels, seven different frequency levels were assumed to be supported by the target system: $\{0.36,0.55,0.64,0.73,\ 0.82,0.91,1.0\}$.

In Fig. 3(a), the normalized energy consumed by the SimpleVS and the NonVS are compared. It can be seen that the SimpleVS achieved significant energy savings against the NonVS. Furthermore, more energy-saving was achieved by the scheduling with the continuous operating frequency selection, while the performance improvement decreases with increasing the average core utilization *acu* . In the case of the continuous frequency scaling, more than 99% dynamic energy was saved by the SimpleVS when *acu* is between 0 and 0.1. Whereas only less than 7% energy could be saved when *acu* was in the range of 0.9-1.0. In contrast, the energy consumptions saved by the SimpleVS with the discrete frequency scaling are 88% and 0, respectively.



(a) (b)

Fig. 3. (a) Normalized energy consumption of NonVS and SimpleVS schedule. (b) Normalized energy consumption of HMVS with continuous and discrete frequency selection, $acu \in (0.2,0.4)$.

Fig. 3(b) shows the energy consumption of the proposed scheduling algorithm with the continuous and the discrete frequency selection. The average utilization *acu* was in the range $[0.2,0.4]$, and the energy consumptions of their corresponding SimpleVS

schedules are normalized to 1. In contrast to the discrete frequency selection, the HMVS algorithm gains about 5% to 11% more energy saving with the continuous frequency selection. This is because that the later is more flexible in selecting the operating frequency and can utilize the slack times more efficiently to save energy.

As most of the state-of-the-art realistic multicore systems can adjust their operating voltages and executing speeds discretely, we will focus on the experiment based on the discrete frequency selection hereafter. In this case, when our algorithm assigns an operating frequency to a processor core, the nearest discrete frequency no less than the operating frequency which is calculated under the continuous voltage/frequency levels is selected.

The impact of various VFD granularities on the energy consumption is depicted in Fig. 4. In general, the performance improvement brought by the HMVS versus the SimpleVS begins to decrease with a coarser granularity of a VFD. The reason is that, it is usually easier for HMVS to reallocate the slack times evenly to the cores in a fine-grained VFD than in a coarse-grained VFD. Hence, multicore systems with fine-grained VFDs can help improve the effectiveness of the slack utilization and can lead to more energy savings. We can find from Fig. 4 (a) that, when *acu* falls into the range of $[0.3,0.4]$, 22% energy was saved by HMVS in a system with 4 cores in a VFD, while only 1% energy could be saved when a VFD contains 16 cores in the target system.

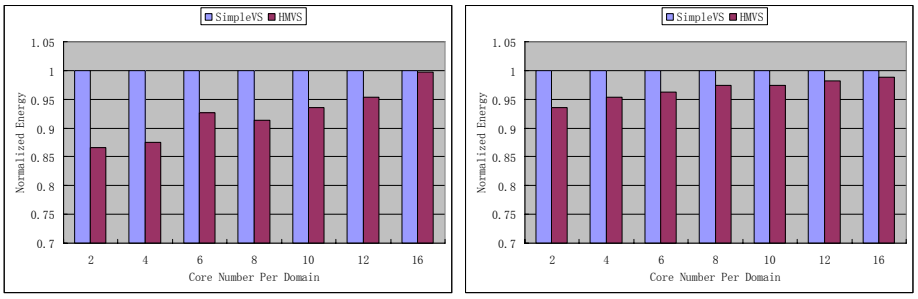


Fig. 4. Normalized energy consumption under different VFD granularity (a) The average core utilization $acu \in [0.3,0.4]$. (b) The average core utilization $acu \in [0.6,0.7]$.

Moreover, by the comparison between Fig. 4(a) and (b) we can observe the influence of *acu* upon the energy saving. In principle, the energy consumption increases with an increasing *acu*. This is because that there are more available slack times on the core with a lower utilization than those on the core with higher utilization, which can be used to reduce the energy consumption. As far as the system in which each VFD contains 4 cores is concerned, the HMVS saved 22% energy compared with the SimpleVS when the *acu* is between 0.3 and 0.4. On the contrary, only 4.7% energy could be saved when the *acu* falls into the range $[0.6,0.7]$.

Fig. 5 presents the energy consumption of the proposed algorithm with various *acu*. Note that the energy consumption of the two scheduling policies are the same when *acu* is smaller than 0.35. The reason is that the minimum discrete executing

frequency of the active core is 0.36 in the experiment. The operating frequency could not be decreased any more even if the utilizations of all cores are quite low (i.e., less than 0.36). Therefore, the discrete frequency levels will affect the energy saving efficiency to a certain extent. As Fig. 5 (a) shows, our algorithm achieved better energy conservations when *acu* is between 0.3 and 0.4.

It can also been observed by comparing Fig. 5(a) and (b) that the system with finer grained VFDs can achieve more energy saving. When the *acu* falls into the range $[0.3,0.4]$, compared with NonVS schedule, the HMVS algorithm can save energy by 22% and 1% if each VFD contains 4 and 16 cores, respectively.

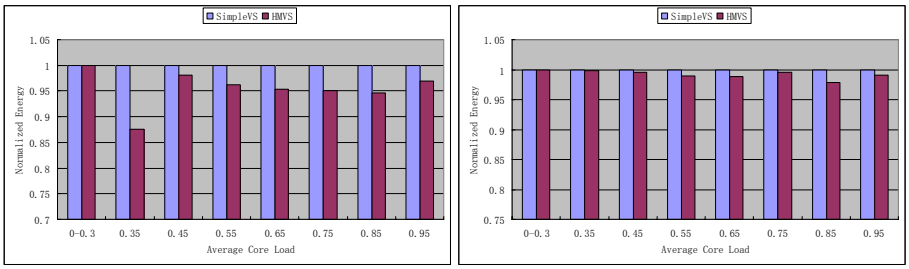


Fig. 5. Normalized energy consumption under different core utilizations. (a) Each voltage/ frequency domain contains 4 cores. (b) Each voltage-frequency domain contains 16 cores.

6 Conclusions and Future Work

In this paper, the problem of energy-efficient scheduling for real-time periodic tasks on multicore systems is studied. The static voltage/frequency scaling of the multicore system (SimpleVS) is firstly introduced. SimpleVS selects the utilization of the heaviest-loaded core in a VFD as the operating frequency of each core in this VFD. Next, based upon the SimpleVS, the slack reallocation is proposed to take use of slack times under the constraint of the frequency synchronization and real-time schedulability. The slack reallocation tries to uniformly redistribute the slack times to the cores on the same VFD so that the synchronous executing frequency can be lowered. Consequently, energy savings and timeliness requirements can be satisfied concurrently.

The experimental results show that, as far as multicore systems with discrete frequency selection is concerned, the proposed policy can reduce the dynamic energy by up to 22% versus the static policy when the system is under-utilized.

It should be noted that as the proposed scheduling algorithm is based on the worst case executing time (WCET) of tasks and the reclamation of static slack times. However, there are usually dynamic slack times at run-time caused by the difference between the WCET and the actual executing time (AET). We will target studying the on-line schedule dealing with the dynamic slack reclamation.

Acknowledgments

This work is supported in part by the NSF of China under grants No.60503048, by China Post-Doc Foundation (20070410280), by State Key Lab for Novel Software Technology, Nanjing University (kfkt2009b13), and by the US National Science Foundation under grant number CCF-0541403.

References

1. Multi-Core Processors—The Next Evolution in Computing, white paper, Advanced Micro Devices, Inc., http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/IDC_Multi-Core_64-bit_White_Paper.pdf
2. Pillai, P., Shin, K.G.: Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In: Proc. 18th ACM Symp. Operating Systems (SOSP '01), pp. 89-102, (2001)
3. Lee, J., Kim, N.: Optimizing total power of many-core processors considering voltage scaling limit and process variations. In: Proceedings of the 14th ACM/IEEE Int'l Symp. Lower-Power Electronics and Design. (2009)
4. Yang, C.Y., Chen, J.J., Kuo, T.W.: Energy-efficiency for multiframe real-time tasks on a dynamic voltage scaling processor. In: Int'l Conf. on Hardware Software Codesign. (2009)
5. Han, J.J., Li, Q.H.: Dynamic Power-Aware Scheduling Algorithms for Real-Time Task Sets with Fault-Tolerance in Parallel and Distributed Computing Environment. In: 19th IEEE International Parallel and Distributed Processing Symposium, (2005)
6. Zhuo J., Chakrabarti, C.: Energy-efficient Dynamic Task Scheduling Algorithms for DVS Systems. In: ACM Trans. Embedded Computing Systems. 2008, 7(2). (2008)
7. Magklis, G., Semeraro, G., Albonesi, D.H., Dropsho, S.G. et al.: Dynamic Frequency and Voltage/frequency scaling for a Multiple-Clock- Domain Microprocessor. In: IEEE Micro, 2003, 23(6): 62-68. (2006)
8. Yang, C., Chen, J., Luo, T.: An Approximation Algorithm for Energy-Efficient Scheduling on a Chip Multiprocessor. In: Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE 2005), pp. 468–473. (2005)
9. Iyer, A., Marculescu, D.: Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors. In: ISCA, 2002, pp. 652-661. (2002)
10. Semeraro, G., Magklis, G., Balasubramonian, R., et al.: Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In: Proceedings of the 8th International Symposium on High-Performance Computer Architecture (ISHPC), 2002, pp. 29-40. (2002)
11. Semeraro, G. P., Albonesi, D.H., Magklis, G. et al.: Hiding Synchronization Delays in GALS Processor Microarchitecture. In: ASYNC, 2004, pp. 159-169. (2004)

12. Niyogi, K., Marculescu, D.: Speed and voltage selection for GALS systems based on voltage/frequency islands. In: Proc. ASP-Des. Autom. Conf., Jan. 2005, pp. 292–297. (2005)
13. Marculescu, D., Talpes, E.: Variability and energy awareness: A micro-architecture-level perspective. In: Proc. Des. Autom. Conf., Jun. 2005, pp. 11–16. (2005)
14. Seo, E., Jeong, J., Park, S., Lee, J.: Energy efficient Scheduling of Real-Time Tasks on Multicore Processors. In: IEEE Trans. Parallel and Distributed Systems, 2008, 19(11):1540 –1552. (2008)
15. Langen, P.D., Juurlink, B.: Leakage-Aware Multiprocessor Scheduling for Low Power. In: Proc. 19th IEEE International Parallel and Distributed Processing Symposium (2006)
16. Bautista, D., Sahuquillo, J., Hassan, H. et al.: A simple power-aware scheduling for multicore systems when running real-time applications: In: Proc. IEEE International Parallel and Distributed Processing Symposium (2008)
17. Lee, W.Y.: Energy-Saving DVFS Scheduling of Multiple Periodic Real-Time Tasks on Multi-core Processors. In: Proc. IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, 2009, pp. 216-223. (2009)
18. Borkar, S.: Design challenges of technology scaling. In: IEEE Micro, vol. 19, no. 4 1999. pp. 23-29. (1999)
19. Sengupta, D., Saleh, R.A.: Application-Driven Voltage-Island Partitioning for Low-Power System-on-Chip. In: IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 28, no. 3, pp. 316–326. (2009)
20. Aydin, H., Yang, Q.: Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In: Proc. Int'l Parallel and Distributed Processing Symp. (2003)