Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 101529, 14 pages
http://dx.doi.org/10.1155/2014/101529

Research Article

A High Performance Load Balance Strategy for Real-Time Multicore Systems

Keng-Mao Cho,[1] Chun-Wei Tsai,[2] Yi-Shiuan Chiu,[1] and Chu-Sing Yang[1]

[1] *Institute of Computer and Communication Engineering, Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan*
[2] *Department of Applied Informatics and Multimedia, Chia Nan University of Pharmacy & Science, Tainan 71710, Taiwan*

Correspondence should be addressed to Chun-Wei Tsai; cwtsai0807@gmail.com

Received 20 February 2014; Accepted 9 March 2014; Published 14 April 2014

Academic Editors: N. Barsoum, V. N. Dieu, P. Vasant, and G.-W. Weber

Finding ways to distribute workloads to each processor core and efficiently reduce power consumption is of vital importance, especially for real-time systems. In this paper, a novel scheduling algorithm is proposed for real-time multicore systems to balance the computation loads and save power. The developed algorithm simultaneously considers multiple criteria, a novel factor, and task deadline, and is called power and deadline-aware multicore scheduling (PDAMS). Experiment results show that the proposed algorithm can greatly reduce energy consumption by up to 54.2% and the deadline times missed, as compared to the other scheduling algorithms outlined in this paper.

1. Introduction

To promote convenience in people's lives, "smart" has become a new requirement for various products [1], which in turn has led embedded systems to being developed. Embedded systems are now widely used in our daily life, such as digital appliances, network devices, portable devices, and diversified information products [2–8]. Various applications are employed in these devices, and multimedia applications are especially prevalent [9–11]. In order to support the plethora of applications, particularly multimedia-related signal processing, superior performance of embedded systems is required. Along with the increasing demand, the system energy consumption is also increasing. As a matter of fact, the advancement in battery technologies has been slower than the advancement of computing speed and the consequent processor energy consumption.

Due to these reasons and to enhance the performance of modern embedded systems [12–14], the system needs to (1) provide more computation power and (2) reduce power consumption while maintaining performance.

To enhance the *performance* of the embedded system, a multicore architecture is one of the possible solutions which allow the system to process numerous jobs simultaneously by parallel computation. Keeping every processor core of the system in high utilization is an important issue to achieve high performance. In order to maximize the parallel computing of a multicore system, load balance becomes an issue that needs to be considered when scheduling. Round robin is one of the simple methods to dispatch tasks in a multicore system [15], where tasks are dispatched to processor cores in a rotated order. Shortest queue first [16] is another method that is often used. In this method, tasks are assigned to the processor core with the shortest waiting queue. To find the shortest queue, the number of tasks or the total computation time of tasks on the processor core can be used to represent the queue length. The latter is also called shortest response time first, and requires a priori knowledge about task service times. Additionally, utilization of a processor is usually considered as the criterion in load balance. To generate the maximum balanced load, tasks should be assigned to the processor core with the lowest utilization [17].

In addition to the performance of the embedded system, energy consumption is also an important issue. Over the last decade, manufacturers have been competing to improve the performance of processors by raising the clock frequency.

Under the same technology level and manufacturing processes, the higher operating frequencies of a CMOS-based processor require higher supply voltage. The dynamic power consumption (P_{dynamic}) of a CMOS-based processor is related to the operating frequency (f) and supply voltage (V_{dd}) as $P_{\text{dynamic}} \propto V_{\text{dd}}^2 \times f$. Thus, higher operating frequency results not only in higher performance but also higher power consumption. Due to the fact that devices which use batteries carry limited energy, research on power saving has received increasing attention, where dynamic voltage and frequency scaling (DVFS) techniques are often applied to extend the battery life of portable devices. DVFS reduces the supply voltage and operating frequency of processors simultaneously to save energy when performance demand is low. Just as the human brain consumes a lot of energy, the processors of a system consume the majority of the energy too. Consequently, multicore architectures can benefit greatly from DVFS technology. In early multicore systems, all processor cores shared the same clock [18]. Under this architecture, DVFS can still be applied to save energy, but there are more limitations. The tradeoff between performance and energy consumption becomes more difficult. To support more flexible power management for multicore systems, the voltage and frequency island (VFI) technique [19, 20] has been developed, where processor cores are partitioned into groups, with processor cores belonging to the same group sharing one supply voltage and having the same processing frequency [21].

1.1. Motivation. In the past, most studies regarding scheduling on a multicore system [22–25] have not been designed for real-time systems. For some urgent tasks, raising the priority level of these tasks cannot satisfy the urgency completely. In this case, not only a priority but also a deadline will be used to express the character of this task. Tasks with deadlines are called real-time tasks. Nowadays, some studies have focused on scheduling for real-time multicore systems [26, 27]. However, these kinds of algorithms usually view guaranteeing the hard deadline as their main purpose, and therefore limitations arise. Also, these algorithms need more a priori knowledge of tasks. When implementing them into a real system, they must satisfy some requirements, such as fixed application, training, and specific information about the application. However, most portable devices execute nonspecific applications, which are usually not hard-real-time tasks. For example, users may download numerous applications onto their smart phones, where most are soft-real-time tasks and normal tasks. Unfortunately, it is difficult to ensure which applications will be executed on devices before users actually use them. This is why the design of algorithms for specific applications is not suitable, and the requirement of additional priori knowledge is difficult to implement efficiently. Thus, to solve these problems and to consider the tradeoff between performance and energy consumption, this paper applies a solution to the problems of scheduling and power saving in a real-time system for a multicore platform. The proposed algorithms decrease the times of deadline missed and simultaneously consider dynamic power, static power, load balance, and practicability.

1.2. Contribution. The contributions of this paper are as follows.

(1) A power and deadline-aware multicore scheduling algorithm is proposed. It is composed of two parts: a power-aware scheduling algorithm and a deadline-aware load dispatch algorithm. The proposed algorithm is simple and easy to implement and overcomes the problems related to many existing power-saving algorithms that are difficult to implement and not suitable for diverse applications.

(2) In the frequency-scaling part of the power-aware scheduling algorithm, we propose a DVFS-based algorithm called ED^{3}VFS. This algorithm uses task deadlines to determine when to scale the operating frequency and is able to adjust parameters dynamically to suit different task sets. Experimental results show that ED^{3}VFS is very effective and flexible.

(3) This paper also proposes a deadline-aware load dispatch algorithm, called the two-level deadline-aware hybrid load balancer. The proposed load dispatch algorithm includes two levels: the concept of load imbalance in the first level and a novel load balance strategy, distribution of task deadline, in the second level. We also combined the other load balance strategies in the second level and let the proposed load dispatch algorithm deal with real-time tasks and normal tasks simultaneously.

(4) We implemented the proposed load dispatch algorithm in Linux and ported the MicroC/OS-II real-time operating system kernel to a PACDSP on a PAC Duo platform and implemented the proposed power-aware scheduling algorithm in the real-time kernel. Experimental results show that the proposed algorithms work well in a real environment.

1.3. Organization. The remainder of this paper is organized as follows. Section 2 gives a brief introduction to work related to scheduling in a multicore system. Section 3 discusses and defines the problems we aim to solve in this paper, as well as limitations and assumptions. Section 4 describes the proposed power and deadline-aware multicore scheduling algorithm. A performance evaluation of the proposed algorithm is presented in Section 5, with conclusion offered in Section 6.

2. Related Work

2.1. DVFS-Based Power Saving Technologies. There are two strategies for using DVFS techniques to reduce energy consumption. The first strategy is scaling voltage and frequency at task slack time. When a processor serves a task, the operating frequency is multiplied by the rate between the worst-case execution time (WCET) and the deadline of the task [28] to reduce power consumption, as shown in Figure 1. Shin et al. [29] combined offline and online components to satisfy the time constraints and reduce energy consumption. The offline component finds the lowest possible processor speed that

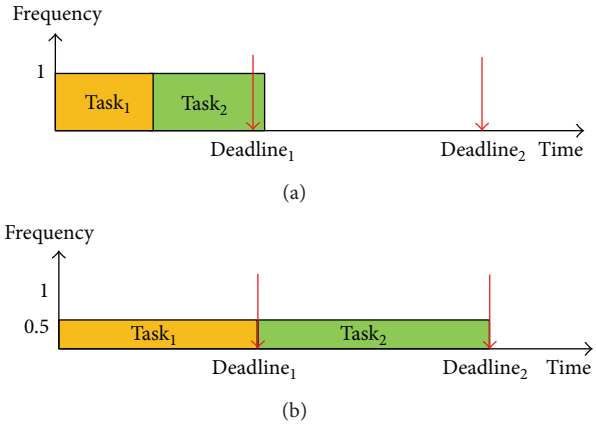

FIGURE 1: Scaling voltage and frequency at task slack time. (a) Without DVFS and (b) with DVFS.

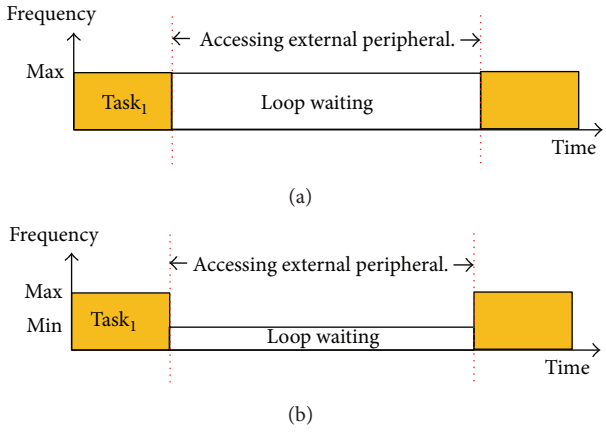

FIGURE 2: Scaling voltage and frequency when access external peripheral. (a) Without DVFS and (b) with DVFS.

satisfies all the time constraints, while the online component varies the operating speed dynamically to save more power. Since the task execution time may be changed slightly when executed, Salehi et al. [30] used an adaptive frequency update interval to follow sudden workload changes. The history data is used to predict the next workload and then, according to the prediction error, adjust the frequency update interval.

The second strategy is scaling voltage and frequency when accessing external peripherals. References [31, 32] pointed out that the operating speed of memory and peripherals is much lower than that of the processors. For tasks that are memory-bounded or I/O-bounded, the operating frequency of a processor can be decreased to save power while waiting for the external peripherals to finish their jobs, as shown in Figure 2. Liang et al. [33] proposed an approximation equation called the memory access rate-critical speed equation (MAR-CSE) and then defined and used the memory access rate (MAR) to predict its critical speed.

2.2. Scheduling on Real-Time Multicore Systems. Because the classical approaches need a priori knowledge of the application to achieve the target, especially when real-time guarantees are provided, Lombardi et al. [34] developed a precedence constraint posting-based offline scheduler for uncertain task durations. This method uses the average duration of a task to replace the probability distribution and calculates an approximate completion time by this cheaper-to-obtain information. Kim et al. [35] presented two pipeline time balancing schemes, namely, workload-aware task scheduling (WATS) and applied database size control (ADSC). Because the execution time of each pipeline stage will change along with the input data, different execution times in each pipeline stage reduce the performance of a system. To achieve higher performance, the pipeline time of each pipeline stage must be in a balanced state. The basic idea of the pipeline time balance schemes is monitoring and modifying the parameter value of the function in each pipeline stage, thereby allowing the execution time of each pipeline stage to be close to the same average value. Jia et al. [36] presented a novel static mapping technique that maps a real-time application onto a multiprocessor system, which optimizes processor usage efficiency. The proposed mapping approach is composed of two algorithms: task scheduling and cluster assignment. In task scheduling, the tasks are scheduled into a set of virtual processors. Tasks that are assigned to the same virtual processors share the maximized data, while data shared among virtual processors is minimized. The goal of cluster assignment is to assign virtual processors to real processors so that the overall memory access cost is minimized.

In addition to balancing the utilization of each processor core, how to tackle the communications among tasks with performance requirements and precedence constraints is another challenge in the scheduling on real-time multicore systems. Hsiu et al. [37] considered the problem of scheduling real-time tasks with precedence constraints in multilayer bus systems and minimized the communication cost. They solved this problem via a dynamic-programming approach. First, they proposed a polynomial-time optimal algorithm for a restricted case, where one multilayer bus and the unit execution time and communication time are considered. The result was then extended as a pseudopolynomial-time optimal algorithm to consider multiple multilayer buses. To consider transition overhead and design for applications with loops, Shao et al. [38] proposed a real-time loop scheduling-algorithm called dynamic voltage loop scheduling (DVLS). In DVLS, the authors succeeded in repeatedly regrouping a loop based on rotation scheduling and decreased the energy consumed by DVS within a timing constraint.

In addition to the abovementioned studies, there are many research directions and issues regarding real-time multicore systems. For real-time applications, it is common to estimate the worst case performance early in the design process without actual hardware implementation. It is a challenge to obtain the upper bound on the worst case response time considering practical issues such as multitask applications with different task periods, precedence relations, and variable execution times. Yet, Yang et al. [39] proposed an analysis technique based on mixed integer linear programming to estimate the worst case performance of each task in a non-preemptive multitask application on a multiprocessor system.

Seo et al. [26] tackled the problem of reducing power consumption in a periodic real-time system using DVS on a multicore processor. The processor was assumed to have the limitation that all cores must run at the same performance level. And so to reduce the dynamic power, they proposed a dynamic repartitioning algorithm. The algorithm dynamically balances the task loads of multiple cores to optimize power consumption during execution. Further, they proposed a dynamic core scaling algorithm, which adjusts the number of active cores to reduce leakage power consumption under low load conditions.

Cui and Maskell [40] proposed a look-up table-based event-driven thermal estimation method. Fast event driven thermal estimation is based upon a thermal map, which is updated only when a high level event occurs. They developed a predictive future thermal map and proposed several predictive task allocation policies based on it. Differing from the utilization-based policy, they used the thermal-aware policies to reduce the peak temperature and average temperature of a system. Han et al. [27] presented synchronization-aware energy management schemes for a set of periodic real-time tasks that accesses shared resources. The mapping approach allocates tasks accessing the same resources to the same core to effectively reduce synchronization overhead. They also proposed a set of synchronization-aware slack management policies that can appropriately reclaim, preserve, release, and steal slack at runtime to slow down the execution of tasks and save more energy. Chen et al. [41] explored the online real-time task scheduling problem in heterogeneous multicore systems and considered tasks with precedence constraints and nonpreemptive task execution. In their assumption, the processor and the coprocessor have a master-slave relationship. Each task will first be executed on the processor and then dispatched to the coprocessor. During online operation, each task is tested by admission control, which ensures the schedulability. Since the coprocessor is nonpreemptive, to deal with the problem of a task having too large a blocking time, the authors inserted the preemptive points to configure the task blocking time and context switch overhead in the coprocessor.

2.3. Summary. To extend the system lifetime for energy-limited devices, one of the possible ways is to use DVFS-based technology [28, 29, 31, 32] to save energy. Because the requirements will change when the real-time system is being used, other studies [27, 42, 43] have combined DVFS technology and a real-time scheduler to meet the time constraint while reducing energy consumption. Now, under the environment of multicore architectures, researchers have proposed multicore schedulers, which can meet real-time constraints and consume lower energy. Along with the development of technology, the algorithms have become much more complex and have increasing restrictions when multiple issues need to be considered simultaneously. As a consequence, these algorithms become difficult to implement and work in real environments. Thus, this paper relaxed some limitations to allow the proposed algorithm to be easier to implement and work well in real environments with simultaneous consideration to the real-time, power, and load balance issues.

3. Problem Definition

The DVFS-based power-aware scheduling problem is defined as finding a schedule that can satisfy all the constraints of a system while consuming less energy to execute tasks. Differing from the aforementioned traditional scheduling on a single core system, scheduling on a multi-core system needs to decide not only the execution order of tasks, but also which tasks should be executed on which processor core. A good load dispatch can improve the performance and reduce energy consumption, so load dispatch is a very important issue in multicore scheduling. In this paper, we divided the power-aware multicore scheduling problem into load dispatch and power-aware scheduling and proposed different algorithms to solve them individually. Additionally, the key points of using the DVFS technique to reduce the energy consumption of a system include deciding when the operating voltage and frequency should be adjusted and selecting the operating state. To relax the limitations and make the proposed algorithm easier to implement and more light-weight, missing deadline is allowed in this research. The problem we consider in this paper can be defined as follows and is illustrated in Figure 3.

System Model. There is one master processor unit and n slave processor cores in the system, and each processor core has its own operating system and can scale the operating voltage and frequency independently. When tasks are released, the master processor unit exchanges the status information with each slave processor core by IPC and dispatches them to a suitable slave processor core individually; then, the slave processor cores schedule tasks that are dispatched to them individually. Under this architecture, the proposed algorithm can apply to either homogeneous multicore systems or heterogeneous multicore systems and each processor core can manage itself. In this work, the platform that is used contains an ARM core as master processor unit and two DSPs.

Input. A task set $T = \{T_{\text{real}}, T_{\text{normal}}\}$, where T_{real} is the set of real-time tasks and T_{normal} is the set of normal tasks. Each real-time task can be represented as (*Release_time*, *Priority*, *Related_Deadline*) and it can be a periodic task or an aperiodic task. In a dynamic environment, it is difficult to get all the information about tasks and the overhead of using optimization method is too heavy when every task is released. Therefore, we tried to schedule real-time tasks without considering the execution time of tasks. Although hard deadline is not guaranteed, methods are proposed to decrease the missing-deadline probability. For normal tasks, we only use (*Release_time*, *Priority*) to represent them. Generally, the execution order of real-time tasks is based on the absolute deadline of tasks. *Priority* of real-time tasks is used only when two or more absolute deadlines are identical. The details of scheduling algorithm will be described in Section 4.

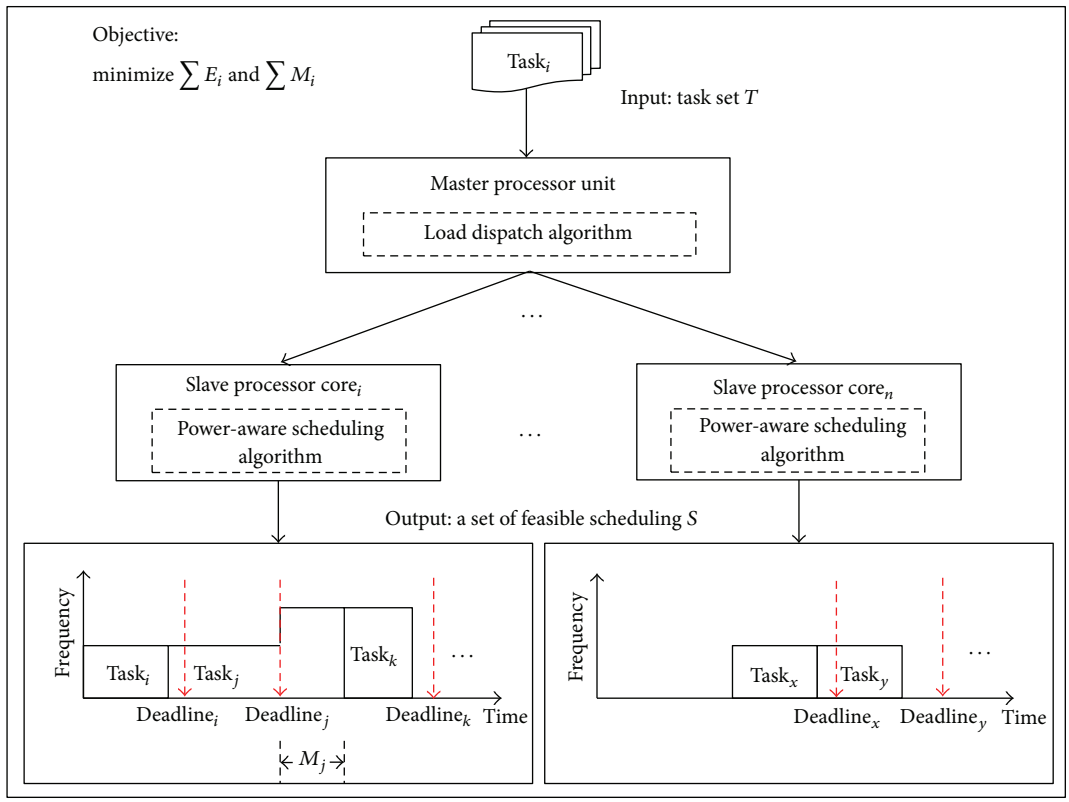

FIGURE 3: System model.

Output. A set of feasible scheduling, $S = \{s_1, s_2, \ldots, s_n\}$, where s_i is the scheduling result of the ith slave processor core and scaling operating voltage and frequency produced by the proposed scaling algorithm.

Objective. To minimize total energy consumption E_{total}, the objective function is expressed in

$$\text{Minimize}\left(E_{\text{total}}\right), \tag{1}$$

$$E_{\text{total}} = \sum E_i, \tag{2}$$

$$E_i = \sum E_{ij}, \tag{3}$$

where E_i is the energy consumption of the ith slave processor core and E_{ij} is the energy consumption of the jth task in the ith slave processor core, as expressed in

$$E_{ij} = \sum \left(P_{ijk} \times t_{ijk}\right), \tag{4}$$

$$P_{ijk} = c \times V_{ijk}^2 \times f_{ijk}, \tag{5}$$

where P_{ijk} is the power of the jth task at the kth time slice in the ith slave processor core, expressed as (5), and t_{ijk} is the duration of the kth time slice for the jth task in the ith slave processor core. Since the operating mode will not always be the same for a given task under the proposed algorithm, the energy in each time slice must be individually calculated and then summarized. In (5), c is the load capacity, V_{ijk} is the operating voltage, and f_{ijk} is the operating frequency of the jth task at the kth time slice in ith slave processor core.

It is allowed that tasks are finished after their deadlines in this paper. The processing speed will be increased when a deadline is missed so that the task can be finished faster. Thus, the performance constraint can be expressed as

$$\text{Minimize}\left(\sum M_i\right), \tag{6}$$

where M_i represents if the deadline of ith task is missed and is defined as

$$M_i = \begin{cases} 1, & \text{if } ft_i > d_i \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where ft_i is the finish time and d_i is the absolute deadline of the ith task.

4. Power and Deadline-Aware Multicore Scheduling

In this section, an efficient multicore scheduling algorithm is presented, called power and deadline-aware multicore scheduling, which integrates three different parts (modules): (1) mixed-earliest deadline first (MEDF) [42], (2) enhanced deadline-driven dynamic voltage and frequency scaling (ED^{3}VFS), and (3) two-level deadline-aware hybrid load balancer (TLDHLB). Among them, MEDF is used to schedule

the tasks that have been dispatched to a processor core. ED^{3}VFS is an enhanced version of D^{3}VFS [42] and is used to scale the operating mode on each slave processor core. And finally, TLDHLB is used for task dispatch and is composed of two levels: the first level is the load imbalance strategy, while the second level is load balance. For example, when a new task that needs to be served by DSP arrived, TLDHB will dispatch it to a DSP with consideration of load balance. After the DSP receives the task, MEDF will schedule the new task. At the same time, ED^{3}VFS will be executed on DSP periodically to reduce the energy consumption.

4.1. Mixed-Earliest Deadline First. The original scheduling algorithm used in the MicroC/OS-II kernel is a fundamental priority-based scheduling algorithm. To support real-time tasks and normal tasks in the same time, mixed-earliest deadline first is selected to replace the original scheduling algorithm. MEDF combined EDF and fixed-priority scheduling. For real-time tasks, it uses EDF to schedule the tasks. When there are two or more deadlines of real-time tasks that are identical or for normal tasks, MEDF uses fixed-priority scheduling to decide the execution order. Moreover, MEDF will always select real-time tasks first when there are real-time tasks and normal tasks in the ready queue simultaneously.

To cooperate with TLDHLB to save static power, we modified MEDF to let it turn off the processor cores while the ready task, which has the highest priority, is idle while there is no real-time task or other normal tasks. This means that when a processor core finishes all tasks, it will turn itself off to save power.

4.2. Enhanced Deadline-Driven Dynamic Voltage and Frequency Scaling. The D^{3}VFS scales operating mode dynamically by the active status of a system. α and β are two parameters used in D^{3}VFS and are set as 10. D^{3}VFS will scale operating modes while the system continues to be busy for α time units or when no deadlines are missed for β time units. Inspired by observations of D^{3}VFS, we present a better strategy to set parameters α and β to enhance the performance of the scheduling system. First, in D^{3}VFS, the related deadlines of tasks are not needed to be longer than 10 or a fixed threshold. When the shortest related deadline is shorter than this value of 10 (threshold), α and β become negative, which should be amended. Second, the purports of α and β are not exactly the same, and so their impacts are different. The bigger value we set to α, which is the longer time that the processor stays in lower speed, because the system will increase the operating frequency in a slower rate. This leads to more power savings with worse computing capacity. On the other hand, β is used for decreasing the operating frequency. A bigger β will allow the system to stay in high speed for longer, so that the performance will be better, but more power is consumed. Third, there is more than one task working in a system simultaneously. And in real environment, these tasks may not always be the same. The best setting for each task is different, and so giving different settings for different task sets is more flexible.

Table 1: Overall settings of α and β.

Settings	Group 1 ($\alpha = \beta$)		Group 2 ($\alpha = D_{sr} - \beta$)	
	α	β	α	β
1	$0.2D_{sr}$	$0.2D_{sr}$	$0.2D_{sr}$	$0.8D_{sr}$
2	$0.4D_{sr}$	$0.4D_{sr}$	$0.4D_{sr}$	$0.6D_{sr}$
3	$0.6D_{sr}$	$0.6D_{sr}$	$0.6D_{sr}$	$0.4D_{sr}$
4	$0.8D_{sr}$	$0.8D_{sr}$	$0.8D_{sr}$	$0.2D_{sr}$

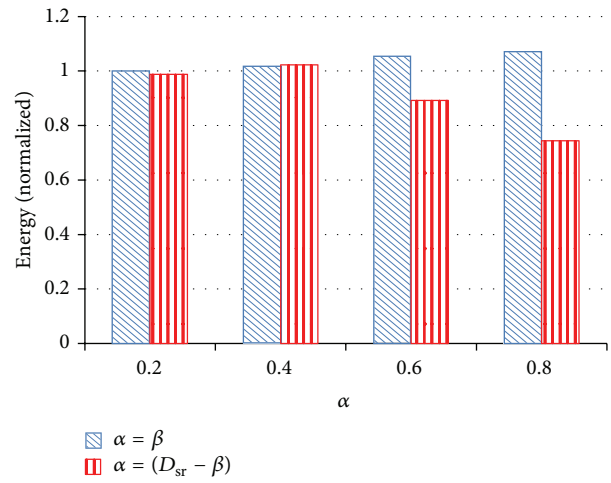

Figure 4: Energy comparison between original D^{3}VFS and ED^{3}VFS.

To solve these aforementioned problems, this paper proposes a different concept to improve the performance of power saving algorithms and conforms to real situations while the system is working. Pseudocode 1 shows the pseudocode of ED^{3}VFS. The basic idea of ED^{3}VFS is that the settings of α and β are free to change along with different task sets and these two parameters will be set to two different values. According to the basic idea of ED^{3}VFS, we ran a series of experiments to find a better setting and to verify that the new setting is superior to the original D^{3}VFS setting. The experimental settings include two main groups and are described as follow.

(i) The first group is just like the original D^{3}VFS, and the settings of α and β are the same. There are four settings in this group, and the four settings are $\alpha = \beta = D_{sr} \times 0.2, 0.4, 0.6$, and 0.8, where D_{sr} is the shortest related deadline.

(ii) The second group used in ED^{3}VFS also features four settings in. In this group, α and β will be set into different values. $\alpha = D_{sr} - \beta$ and $\beta = D_{sr} \times 0.8, 0.6$, 0.4, and 0.2. Just as in the above, the effects of α and β are opposite, so we gave them opposite values. Table 1 lists the overall settings of α and β in these experiment series.

Figure 4 shows the energy comparison between the two different settings, namely, the original D^{3}VFS and ED^{3}VFS.

```
(1)  Initially, setting the power mode of DSP to default level f_d.
(2)  Every timer interrupt occur
(3)    If (There is any real-time task.)
(4)      Setting α = D_sr × 0.2.
(5)      Setting β = D_sr − α.
(6)      if (Deadline missed.)
(7)        Extending deadline of the task that missed deadline for d_e ticks.
(8)        Rising power mode.
(9)      else if (Utilization of DSP > S_r for α ticks.)
(10)       Rising power mode.
(11)     else if (There is no deadline missed for β ticks.)
(12)       Falling power mode.
(13)     else
(14)       Setting the power mode of DSP to default level f_d.
(15)   End If
```

Pseudocode 1: Pseudocode of enhanced deadline-driven dynamic voltage and frequency scaling (ED^{3}VFS).

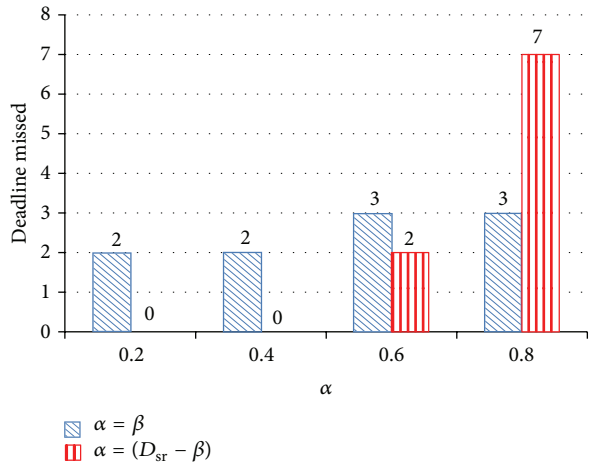

Figure 5: Performance comparison between original D^{3}VFS and ED^{3}VFS.

The vertical axis shows energy consumption while the horizontal axis shows the setting of α, where $\alpha = D_{sr} \times 0.2$, 0.4, 0.6, and 0.8. In this research, the first experimental result of energy is used to normalize all of the other results and make them easy to see the differences in comparison. The results show that the energy consumption of ED^{3}VFS is lower than the original D^{3}VFS in most cases. Figure 5 shows the performance comparison between original D^{3}VFS and ED^{3}VFS. We used the number of deadlines missed as the criterion to compare their performance. For a real-time system, a lower number of missed deadlines are better. The results show that ED^{3}VFS is better than the original D^{3}VFS in performance, except for the case when $\alpha = D_{sr} \times 0.8$. Although the proposed power-saving algorithm does not guarantee the hard deadline, we still try to stop missed deadlines from happening. According to the experimental results shown in Figure 5, there are two settings we can choose, $\alpha = D_{sr} \times 0.2$ and 0.4 when $\beta = D_{sr} - \alpha$. There was no deadline missed in these two cases. Since the energy consumption of $\alpha = D_{sr} \times 0.2$ is less than $\alpha = D_{sr} \times 0.4$, we chose $\alpha = D_{sr} \times 0.2$ and $\beta = D_{sr} - \alpha$ as the final setting of ED^{3}VFS. In this setting, both energy consumption and performance of ED^{3}VFS are superior to the original D^{3}VFS. Actually, if energy consumption is more important than performance in a given system, setting $\alpha = D_{sr} \times 0.6$ and $\beta = D_{sr} - \alpha$ is also a good choice. In that case, more energy can be saved and a certain level of performance maintained.

4.3. Two-Level Deadline-Aware Hybrid Load Balancer. For systems limited by battery power, letting all processor cores work continuously in active mode is not a good idea, as much energy will be consumed. In a multicore system, balancing the workload between each core can reduce the completion times of all tasks. Processor cores thus can turn to sleep mode for longer times and save more energy.

In this paper, a novel task dispatch algorithm, called two-level deadline-aware hybrid load balancer (TLDHLB), is presented. The first level is the load imbalance strategy used for saving static power, and it was inspired by [44]. The basic idea of the first level of task dispatch is dispatching tasks to the processor cores working in active mode and turning off the processor cores when all tasks are finished. For example, suppose there are one MPU and two DSPs in the system. Initially, the system will turn off two DSPs until there are tasks needing to be processed by DSP, as shown in Figure 6(a). When task$_1$ was released, MPU will check the state of the DSPs. If there is no DSP working in active mode, then turn on DSP$_1$ and dispatch task$_1$ to DSP$_1$. According to ED^{3}VFS, DSP$_1$ will work at the default speed, normally, and at the lowest speed in the beginning, as shown in Figure 6(b). Figure 6(c) shows that if task$_n$ is released at time$_n$ while DSP$_1$ works at full speed, then turn on DSP$_2$ and dispatch task$_n$ to DSP$_2$. Figure 6(d) shows that, after DSP$_1$ finished all tasks assigned to it, it will turn itself off by MEDF.

The second level is used for load balance. When there are two or more DSPs working in active mode, the load balance strategies in the second level will be used to dispatch tasks that are newly released. Unlike traditional systems that do

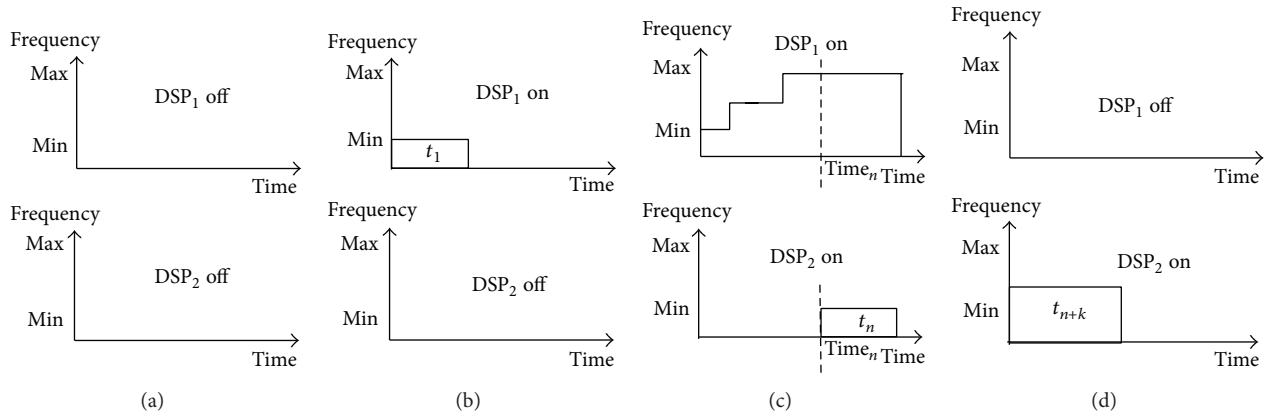

FIGURE 6: Example of load unbalance. (a) Initial state, (b) Task_1 released and turning on DSP_1. (c) Task_n released while DSP_1 worked in full speed and turning on DSP_2. (d) DSP_1 finished all of tasks that dispatched to it and turning off DSP_1.

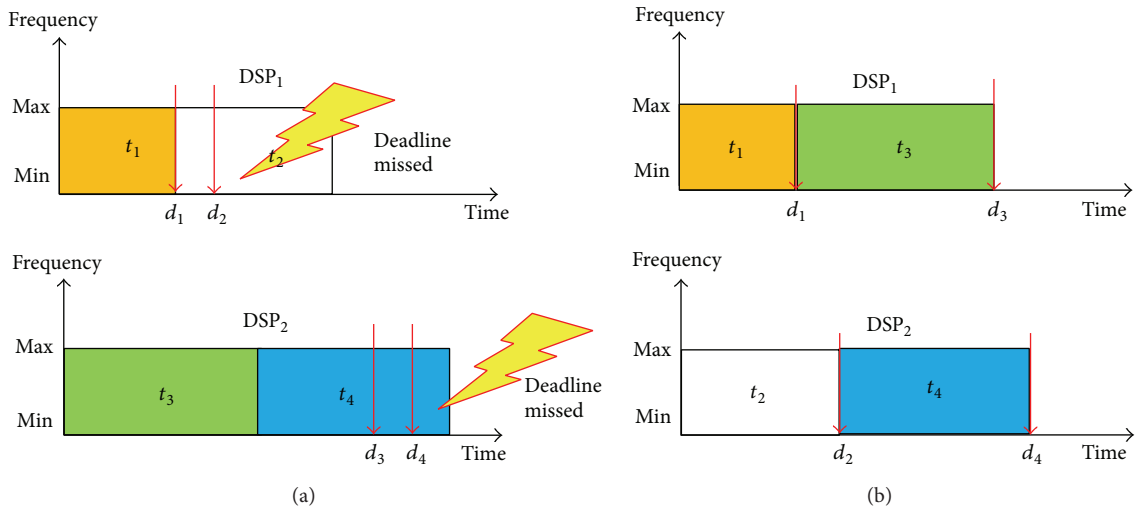

FIGURE 7: Example of real-time tasks dispatch. (a) Distribution of task deadlines is uneven. (b) Distribution of task deadlines is uniform.

not contain real-time tasks, there are simultaneously real-time tasks and normal tasks in a system in our assumption. Traditional load balance strategies were not designed for real-time systems, so we propose a new dispatch criterion to process the problem of dispatch in real-time tasks. We also combined other criteria to allow our load balance algorithm to process real-time tasks and normal tasks simultaneously and improve robustness.

The novel strategy uses the distribution of task deadlines as the criterion for load balance. According to our observation, the more uniform the distribution of task deadlines is, the lower the missing-deadline probability will be. Figure 7 is a simple example that supports our observation. There are two DSPs and four tasks; Figure 7(a) shows that one of the dispatch results' distribution of task deadlines is uneven. In this example, task_1 is finished at its deadline, d_1, so that there is not enough time to execute task_2 and a deadline missed occurs. A similar situation occurred in DSP_2 for task_4. Figure 7(b) shows that a different dispatch result features a more uniform distribution of task deadlines. In this situation, the time slot between two deadlines is longer, which means that there is more time to execute the next task when a task is finished, leading to the probability of a deadline missed being lower. Now, the problem is how to express the distribution of task deadlines.

In this paper, the variance of task deadlines was used as the feature of deadline distribution. Since variance expresses how far a set of numbers is spread out, variance can be used to express the density of data distribution, which is what we need. A smaller variance of task deadlines implies that the time slot between two task deadlines is shorter. Equation (8) is the formula of variance, where N is the number of data, x_i expresses ith data, and $\overline{x}$ is the data mean:

$$\text{Var}(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2. \tag{8}$$

Except for the distribution of task deadlines, the other three strategies were combined to dispatch normal tasks. These three strategies can also be used to dispatch real-time

```
(1)  Initially, turning off all of DSPs, DSP_off = DSP_all.
(2)  When a task that needs be processed by DSP is released
(3)    getStatus(DSP_all)
(4)  % First level for load unbalance%
(5)    If (DSP_off == DSP_all)
(6)      DSP_t = minSerial(DSP_off)
(7)      Turn on minSerial(DSP_off).
(8)    Else if (DSP_fs == DSP_on)
(9)        if (DSP_on == DSP_all)
(10)         go to line 21
(11)       else
(12)         DSP_t = minSerial(DSP_off)
(13)         Turn on minSerial(DSP_off).
(14)   Else if (Num(DSP_on − DSP_fs) == 1)
(15)     DSP_t = DSP_on − DSP_fs
(16)   Else
(17)     go to line 21
(18)   Dispatch the task to DSP_t.
(19)   End
(20)  % Second level for load balance%
(21)   If (Num (maxUniformity (DSP_on)) == 1)
(22)     DSP_t = maxUniformity(DSP_on)
(23)   Else if (Num (minOrder (DSP_on)) == 1)
(24)     DSP_t = minOrder(DSP_on)
(25)   Else if (Num (minTaskNum (DSP_on)) == 1)
(26)     DSP_t = minTaskNum(DSP_on)
(27)   Else
(28)     DSP_t = minSerial(DSP_on)
(29)   Dispatch the task to DSP_t.
(30)   End
```

Pseudocode 2: Pseudocode of two levels deadline-aware hybrid load balancer.

tasks when the uniformity of deadline distributions is equal. The first strategy is the execution order of the task. This means that the dispatcher will dispatch a task to the DSP that provides higher priority. For example, assume there are two DSPs and two tasks on each DSP. When a new task is released, if the execution order of the task is second in DSP_1 and third in DSP_2, this task will be dispatched to DSP_1. The second strategy is the number of tasks. The dispatcher will dispatch a task to the DSP that has the fewest number of tasks on it. When the dispatcher cannot make the decision via the three strategies in the second level mentioned above, then the last strategy will be used. The last strategy is very simple, it simply chooses the DSP whose serial number is the minimum and is working in active mode. The pseudocode of two levels deadline-aware hybrid load balancer is shown in Pseudocode 2, where DSP_{all} is the set of all DSPs, DSP_{off} is the set of DSPs that are not staying in active mode, DSP_t is the target that the task will be dispatched to, DSP_{on} is the set of DSPs that are staying in active mode, DSP_{fs} is the set of DSPs that are working at full speed, function *getStatus*() is used to obtain the status information of the DSPs, function *minSerial*() returns the DSP whose serial number is the minimum, function *Num*() returns the number of input data, function *maxUniformity*() returns the DSP whose uniformity of deadline distribution is the maximum, function *minOrder*() returns the DSP that can provide the highest priority to new released task and function, *minTaskNum*() returns the DSP whose number of tasks is the fewest.

What is worth noticing is that when calculating the uniformity of deadline distribution, we should take the newly released task into account because we want to find a DSP whose deadline distribution is still uniform after inserting the newly released task. Furthermore, *maxUniformity*(), *minOrder*(), and *minTaskNum*() may return more than one DSP, when there are two or more DSPs with the same status. In that case, TLDHLB will use another strategy to dispatch the task and is why we combined four criteria to become a hybrid strategy.

5. Experiments

In this section, we describe the experimental environment and the setting of parameters. Experimental results and analyses are then shown.

5.1. Experimental Environment. In this study the PAC Duo platform was used for the experiments and includes an ARM926 processor and two PACDSPs. The operating system kernel running on ARM is Linux 2.6.27. We ported the

Table 2: Operating voltage and frequency of PACDSP.

Power Mode (operating mode)	Voltage (V)	Frequency (MHz)
7	1.0	204
6	1.0	136
5	0.9	102
4	0.9	68
3	0.9	51
2	0.8	34
1	0.8	24

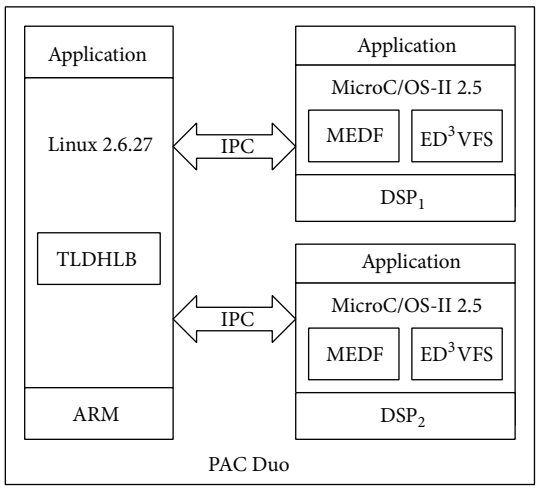

Figure 8: System architecture.

MicroC/OS-II kernel (version 2.5) to the PACDSPs and implemented the proposed power-aware scheduling algorithm on the MicroC/OS-II kernel and the proposed load dispatch algorithm on ARM. Figure 8 shows the experimental system architecture, while Table 2 shows the operating frequencies used in the experiments and the corresponding voltages. In the experiments, we used a digital multimeter (FLUKE 8846A) to measure the voltage and current of the PACDSPs, the data of which was used to calculate energy consumption.

5.2. Experimental Settings. In the experiments, we used matrix multiplication, π calculation, quick sort, jpeg decoder, and histogram equalization as the workload. Other than the proposed algorithms, we also implemented two load balance algorithms and three frequency scaling strategies as the comparisons. Table 3 shows the algorithm usage in the experiments.

(i) Seven sets of settings were used. The first set is the proposed algorithms and combines the proposed load dispatch algorithm and power-aware scheduling algorithms. Worthy of note is that the DSPs on our experimental platform cannot turn off and then turn on, so we scaled the operating frequency to the lowest frequency to represent the DSP as being turned off and assumed the energy consumption was zero until the proposed algorithms turn it on again.

(ii) The second set to the fourth set used the same load balance algorithm. In these three sets, the load balancer used the number of tasks as the criterion to dispatch tasks. The frequency scaling strategies used in the second and third sets were two static settings: one is the highest operating frequency and the other is the lowest operating frequency. The fourth set used Linux-ondemand [45] as the frequency scaling strategy. Linux-ondemand is a dynamic frequency scaling algorithm; it is used in Linux kernel and dynamically scales the operating frequency according to the utilization of the processor.

(iii) The fifth set to seventh set used the utilization of processor as the criterion to dispatch tasks. The frequency scaling strategies used in these three sets were the highest frequency, the lowest frequency, and Linux-ondemand in ordering. Except for the first set, the other settings use original scheduling used in MicroC/OS-II to schedule tasks.

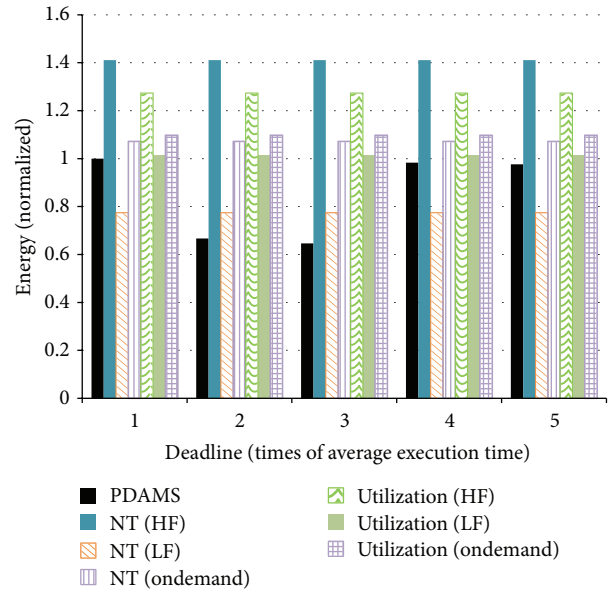

Figure 9: Comparison of energy consumption.

For each task, we used the times of the average execution time of the task as its deadline, from one time to five times. There are five settings of task deadline for each set of settings in the experiments.

5.3. Experimental Results

5.3.1. Comparison of Energy Consumption. Figure 9 shows the comparison of energy consumption. The vertical axis shows the energy consumption while the horizontal axis shows the setting of the task deadline. The results show that the energy consumption of the proposed algorithms is lower than that of other algorithms in almost every case. Compared

Table 3: Usage of algorithms for experiments.

Set	Load dispatch			Frequency scaling			
	TLDHLB	The number of tasks	Utilization	ED^{3}VFS + MEDF	Always in the highest frequency	Always in the lowest frequency	Linux-ondemand
1 PDAMS	√			√			
2 NT (HF)		√			√		
3 NT (LF)		√				√	
4 NT (Ondemand)		√					√
5 Utilization (HF)			√		√		
6 Utilization (LF)			√			√	
7 Utilization (Ondemand)			√				√

with other algorithms, the proposed algorithm can reduce energy consumption by up to 54.2%. Experimental results also show that using the number of tasks as the criterion of load balance and always working in the lowest operating frequency can reduce energy consumption the most. The consequence is obvious and predictable, but the computing capacity under this condition is not satisfying. Although the proposed algorithm considered saving static power, due to hardware constrains, the static power consumption could barely be measured independently. As a result, static power consumption was not taken in count in the experiments. Besides, the overhead of scaling voltage and frequency has not been considered either. These will be added to our next work.

5.3.2. Comparison of Performance. Other than energy consumption, performance is a very important criterion to evaluate the effect of an algorithm. How to deal with the tradeoff between energy and performance is a difficult issue. Differing from traditional systems, the finish time of an overall system cannot represent the performance completely in a real-time system. For a real-time task, there is no difference between a system finishing the task very quickly and finishing the task just at deadline. Before the deadline of a real-time task, no matter what time is required to finish it, the effects of the real-time task are the same. Therefore, the number of missed deadlines is a better criterion to represent the performance of a real-time system. Figure 10 offers the performance comparisons. The vertical axis shows the number of deadlines missed while the horizontal axis shows the setting of task deadlines. The results show that the performance of the proposed algorithms is the best. Except for setting the deadline of each task into one time of an average execution time, there is no deadline missed while using the proposed algorithms. Since there are only two slave processor cores in our experimental platform, when there are more than three real-time tasks and their deadlines are just

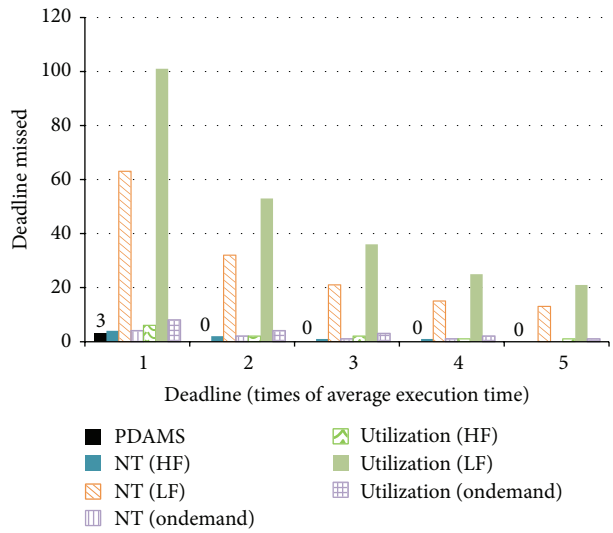

Figure 10: Comparison of performance.

one time of their average execution time, missing deadlines cannot be avoided. Worth to note is that even DSPs always run at the fastest frequency, using less appropriate dispatch method and scheduling algorithm may produce more missed deadlines. The proposed algorithm tries to decrease the probability of missing deadline not only in load dispatch, but also in scheduling. That is why the proposed algorithm has the chance to use less energy and get higher performance.

Although always using the lowest operating frequency can reduce energy consumption the most, the performance is not acceptable. The number of deadlines missed is much more than other algorithms, regardless of which load balance algorithm is used. Experimental results show that the proposed algorithms found a good balance point between energy consumption and performance. By considering the deadline

of tasks, the distribution of task deadlines is more uniform on each slave processor core. This can reduce the probability of deadlines being missed. A lower probability of deadline missed not only represents higher performance, but also reduces more energy consumption because there is a longer time that the system will work in lower speed when using ED^{3}VFS. Moreover, the concept of load imbalance made the proposed algorithms reduce more energy consumption. Differing from dynamic voltage and frequency scaling technology that only reduces the dynamic power, load imbalance also reduces static power.

Both the number of tasks and processor utilization are the most popular strategies to dispatch tasks in real systems, which is why we chose them as the comparisons. The aim of this paper is to develop not only a novel and effective load balance algorithm, but also an algorithm that can be applied in a real environment. Although the performance of some state-of-the-art algorithms may be better than the proposed algorithm, it is very difficult to satisfy their assumptions. Hence, these kinds of algorithms are hard to make work in a real environment. The other reason is the similarity of assumptions that all the algorithms used in this paper do not need the worst case execution time. Nowadays, portable devices that feature the ability to connect to internet are very popular, such as smart phones and tablet PCs. When a user downloads an application from the internet, there is no way for the system to get the worst case execution time of this application immediately. Although not using the worst case execution time of the tasks made the proposed algorithms unable to guarantee the hard-deadline, the proposed algorithm still tries to avoid missing deadline and become more flexible. We implemented the proposed algorithm in a real platform and the experimental results show that the proposed algorithms work well and are superior to others in general performance.

6. Conclusion

This paper applied a solution to the problems of load dispatch and power saving in a real-time system on a multicore platform, called power and deadline-aware multicore scheduling. The proposed algorithm simultaneously considers dynamic power, static power, and load balance. To reduce the dynamic power, we implemented MEDF and fine-tuned the parameters of D^{3}VFS to save more power and improve performance. The concept of load imbalance was introduced in saving static power. Instead of dispatching the workload to every processor core equally, the proposed algorithm turns power on only in parts of processor cores and lets other unnecessary cores turn to sleep mode or turn-off. Finally, deadline is used as a novel strategy for load balance between processor cores in active mode. Combining load imbalance and load balance, this paper proposed a two-level task dispatch algorithm called two-level deadline-aware hybrid load balancer.

To verify that the proposed algorithms are useful, we implemented them on a multicore platform, PACDuo. We also implemented some load balance algorithms and frequency scaling algorithms for comparison. Experimental results show that compared to six combinations of load balance algorithms and frequency scaling algorithms, the proposed algorithms can reduce energy consumption by up to 54.2% and the performance of the proposed algorithms is superior to others. However, much work still needs to be completed in the future. Some areas for future study include (1) adding theoretical analysis to support the proposed algorithm, (2) modelling the energy consumption more detailed, (3) considering the demands of hard-real-time and task migration while keeping the algorithm light-weight, and (4) introducing the concept of heuristic algorithms and improving the proposed algorithms.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the editors and anonymous reviewers for their valuable comments and suggestions on the paper that greatly improve the quality of the paper. This work was supported in part by the National Science Council of Taiwan, under Grants NSC102-2221-E-041-006 and NSC100-2218-E-006-028-MY3 and in part by the Research Center of Energy Technology and Strategy of National Cheng Kung University under Grant D102-23015.

References

[1] W. Elmenreich and D. Egarter, "Design guidelines for smart appliances," in *Proceedings of the 10th Workshop on Intelligent Solutions in Embedded Systems (WISES '12)*, pp. 76–82, Klagenfurt, Austria, July 2012.

[2] Y.-W. Bai, L.-S. Shen, and Z.-H. Li, "Design and implementation of an embedded home surveillance system by use of multiple ultrasonic sensors," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 1, pp. 119–124, 2010.

[3] H. Kim, Y. Won, and S. Kang, "Embedded NAND flash file system for mobile multimedia devices," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 545–552, 2009.

[4] I. Kramberger, M. Grasic, and T. Rotovnik, "Door phone embedded system for voice based user identification and verification platform," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1212–1217, 2011.

[5] H. Jo, H. Kim, J. Jeong, J. Lee, and S. Maeng, "Optimizing the startup time of embedded systems: a case study of digital TV," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 2242–2247, 2009.

[6] N. Barsoum, "Speed control of the induction drive by temperature and light sensors via PIC," *Global Journal of Technology and Optimization*, vol. 1, pp. 53–59, 2010.

[7] U. R. Shikoska, D. Davcev, R. Reckoski, G. Petrovska, C. Andreevski, and J. Sikoski, "Space and time in wireless sensor network," *Global Journal of Technology and Optimization*, vol. 2, pp. 73–83, 2011.

[8] C.-F. Lai, M. Chen, M. Qiu, A. V. Vasilakos, and J. H. Park, "A RF4CE-based remote controller with interactive graphical user

interface applied to home automation system," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2, article 30, 2013.

[9] A. Gotchev, G. B. Akar, T. Capin, D. Strohmeier, and A. Boev, "Three-dimensional media for mobile devices," *Proceedings of the IEEE*, vol. 99, no. 4, pp. 708–741, 2011.

[10] G. Correa, P. Assuncao, L. Agostini, and L. A. D. S. Cruz, "Complexity control of high efficiency video encoders for power-constrained devices," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1866–1874, 2011.

[11] C. A. Martinez, J. C. C. San Adrian, and M. V. Cortes, "Dynamic tolerance region computing for multimedia," *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 650–665, 2012.

[12] T. Simunic, L. Beniani, and G. de Micheli, "Event-driven power management of portable systems," in *Proceedings of the 12th International Symposium on System Synthesis*, pp. 18–23, San Jose, Calif, USA, November 1999.

[13] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. de Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 524–529, Las Vegas, Nev, USA, June 2001.

[14] Y.-S. Hwang, S.-K. Ku, and K.-S. Chung, "A predictive dynamic power management technique for embedded mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 713–719, 2010.

[15] M. C. Cera, G. P. Pezzi, E. N. Mathias, N. Maillard, and P. O. A. Navaux, "Improving the dynamic creation of processes in MPI-2," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 4192 of *Lecture Notes in Computer Science*, pp. 247–255, 2006.

[16] S. Dandamudi, "Performance implications of task routing and task scheduling strategies for multiprocessor systems," in *Proceedings of the 1st International Conference on Massively Parallel Computing Systems*, pp. 348–353, Ischia, Italy, May 1994.

[17] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 113, IEEE, April 2003.

[18] G. Magklis, G. Semeraro, D. H. Albonesi, S. G. Dropsho, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage scaling for a multiple-clock-domain microprocessor," *IEEE Micro*, vol. 23, no. 6, pp. 62–68, 2003.

[19] P. Choudhary and D. Marculescu, "Power management of voltage/frequency island-based systems using hardware-based methods," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 427–438, 2009.

[20] W. Jang and D. Z. Pan, "A voltage-frequency island aware energy optimization framework for networks-on-chip," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 3, pp. 420–432, 2011.

[21] L.-F. Leung and C.-Y. Tsui, "Energy-aware synthesis of networks-on-chip implemented with voltage islands," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pp. 128–131, San Diego, Calif, USA, June 2007.

[22] T. Liu, Y. Sun, Z. Zhang, and L. Guo, "Load balancing for flow-based parallel processing systems in CMP architecture," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '09)*, pp. 1–7, Honolulu, Hawaii, USA, December 2009.

[23] Y. Ahn, W.-J. Kim, K.-S. Chung, S.-H. Kim, H.-S. Kim, and T. H. Han, "A novel load balancing method for multi-core with non-uniform memory architecture," in *Proceedings of the International SoC Design Conference (ISOCC '10)*, pp. 412–415, Seoul , Korea, November 2010.

[24] X. Geng, G. Xu, D. Wang, and Y. Shi, "A task scheduling algorithm based on multi-core processors," in *Proceedings of the International Conference on Mechatronic Science, Electric Engineering and Computer (MEC '11)*, pp. 942–945, Jilin, China, August 2011.

[25] X. Kavousianos, K. Chakrabarty, A. Jain, and R. Parekhji, "Test schedule optimization for multicore SoCs: handling dynamic voltage scaling and multiple voltage islands," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 11, pp. 1754–1766, 2012.

[26] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1540–1552, 2008.

[27] J.-J. Han, X. Wu, D. Zhu, H. Jin, L. Yang, and J. L. Gaudiot, "Synchronization-aware energy management for VFI-based multicore real-time systems," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1682–11696, 2012.

[28] C. Tianzhou, H. Jiangwei, X. Liangxiang, and Z. Zhenwei, "A practical dynamic frequency scaling scheduling algorithm for general purpose embedded operating system," in *Proceedings of the 2nd International Conference on Future Generation Communication and Networking (FGCN '08)*, vol. 2, pp. 213–216, Hainan, China, December 2008.

[29] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '00)*, pp. 365–368, San Jose, Calif, USA, November 2000.

[30] M. E. Salehi, M. Samadi, M. Najibi, A. Afzali-Kusha, M. Pedram, and S. M. Fakhraie, "Dynamic voltage and frequency scheduling for embedded processors considering power/performance tradeoffs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1931–1935, 2011.

[31] M. Spiga, A. Alimonda, S. Carta, F. Aymerich, and A. Acquaviva, "Exploiting memory-boundedness in energy-efficient hard real-time scheduling," in *Proceedings of the International Symposium on Industrial Embedded Systems (IES '06)*, pp. 1–10, Antibes, France, October 2006.

[32] W.-Y. Liang, S.-C. Chen, Y.-L. Chang, and J.-P. Fang, "Memory-aware dynamic voltage and frequency prediction for portable devices," in *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '08)*, pp. 229–236, Kaohsiung, Taiwan, August 2008.

[33] W.-Y. Liang, Y.-L. Chen, and M.-F. Chang, "A memory-aware energy saving algorithm with performance consideration for battery-enabled embedded systems," in *Proceedings of the 15th IEEE International Symposium on Consumer Electronics (ISCE '11)*, pp. 547–551, Singapore, June 2011.

[34] M. Lombardi, M. Milano, and L. Benini, "Robust scheduling of task graphs under execution time uncertainty," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 98–111, 2013.

[35] J.-Y. Kim, M. Kim, S. Lee, J. Oh, S. Oh, and H.-J. Yoo, "Real-time object recognition with neuro-fuzzy controlled workload-aware task pipelining," *IEEE Micro*, vol. 29, no. 6, pp. 28–43, 2009.

[36] Z. J. Jia, T. Bautista, and A. Nunez, "Real-time application to multiprocessor-system-on-chip mapping strategy for system-level design tool," *Electronics Letters*, vol. 45, no. 12, pp. 613–615, 2009.

[37] P.-C. Hsiu, C.-K. Hsieh, D. N. Lee, and T.-W. Kuo, "Multilayer bus optimization for real-time embedded systems," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1638–1650, 2012.

[38] Z. Shao, M. Wang, Y. Chen et al., "Real-time dynamic voltage loop scheduling for multi-core embedded systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 445–449, 2007.

[39] H. Yang, S. Kim, and S. Ha, "An MILP-based performance analysis technique for non-preemptive multitasking MPSoC," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1600–1613, 2010.

[40] J. Cui and D. L. Maskell, "A fast high-level event-driven thermal estimator for dynamic thermal aware scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 904–917, 2012.

[41] Y.-S. Chen, H. C. Liao, and T.-H. Tsai, "Online real-time task scheduling in heterogeneous multicore system-on-a-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 118–130, 2013.

[42] K.-M. Cho, C.-H. Liang, J.-Y. Huang, and C.-S. Yang, "Design and implementation of a general purpose power-saving scheduling algorithm for embedded systems," in *Proceedings of the IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC '11)*, pp. 1–5, Xi'an, China, September 2011.

[43] S. Liu, J. Lu, Q. Wu, and Q. Qiu, "Harvesting-aware power management for real-time systems with renewable energy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1473–1486, 2012.

[44] H. Jeon, W. H. Lee, and S. W. Chung, "Load unbalancing strategy for multicore embedded processors," *IEEE Transactions on Computers*, vol. 59, no. 10, pp. 1434–1440, 2010.

[45] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, pp. 215–230, Ottawa, Canada, 2006.