2011
Volume 8, Number 4

# ACM Transactions on Architecture and Code Optimization

ACM Transactions on Architecture and Code Optimization Vol. 8 • No. 4 • 2011 Articles 18-55

## SPECIAL ISSUE ON HIGH-PERFORMANCE AND EMBEDDED ARCHITECTURES AND COMPILERS

**continued on back cover**

Association for Computing Machinery

*Advancing Computing as a Science & Profession*

ACM Transactions on
# Architecture and Code Optimization

ACM
2 Penn Plaza, Suite 701
New York, NY 10121-0701
Tel.: 212-869-7440
Fax: 212-869-0481
http://www.acm.org

**Home Page:** http://taco.acm.org/

*The ACM Transactions on Architecture and Code Optimization* (ISSN 1544-3566) is published quarterly in Spring, Summer, Fall, and Winter by the Association for Computing Machinery (ACM), 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Fall 2011. Periodicals class postage paid at New York, NY 10001, and at additional mailing offices. Printed in the U.S.A. POSTMASTER: Send address changes to *ACM Transactions on Architecture and Code Optimization*, ACM, 2 Penn Plaza, Suite 701, New York, NY 10121-0701.

**For manuscript submissions, subscription, and change of address information, see inside backcover.**

# TL-Plane-Based Multi-Core Energy-Efficient Real-Time Scheduling Algorithm for Sporadic Tasks

DONGSONG ZHANG, National Laboratory of Parallel and Distributed Processing, National University of Defense Technology
DEKE GUO and FANGYUAN CHEN, National University of Defense Technology
FEI WU, Shanghai University of Engineering Science
TONG WU, National University of Defense Technology
TING CAO, Australian National University
SHIYAO JIN, National Laboratory of Parallel and Distributed Processing, National University of Defense Technology

As the energy consumption of multi-core systems becomes increasingly prominent, it's a challenge to design an energy-efficient real-time scheduling algorithm in multi-core systems for reducing the system energy consumption while guaranteeing the feasibility of real-time tasks. In this paper, we focus on multi-core processors, with the global Dynamic Voltage Frequency Scaling (DVFS) and Dynamic Power Management (DPM) technologies. In this setting, we propose an energy-efficient real-time scheduling algorithm, the Time Local remaining execution plane based Dynamic Voltage Frequency Scaling (TL-DVFS). TL-DVFS utilizes the concept of Time Local remaining execution (TL) plane to dynamically scale the voltage and frequency of a processor at the initial time of each TL plane as well as at the release time of a sporadic task in each TL plane. Consequently, TL-DVFS can obtain a reasonable tradeoff between the real-time constraint and the energy-saving while realizing the optimal feasibility of sporadic tasks. Mathematical analysis and extensive simulations demonstrate that TL-DVFS always saves more energy than existing algorithms, especially in the case of high workloads, and guarantees the optimal feasibility of sporadic tasks at the same time.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Real-time system, energy-efficient scheduling, multi-core, sporadic task

---

This work was supported in part by the National Science Foundation of China under Grant Nos. 60903206, 61170284, 61173045, 61103084, and 61100075; the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20104307110005; the China Postdoctoral Science Foundation under Grant Nos. 20100480898 and 201104439; the Hunan Provincial Innovation Foundation for Postgraduate of China under Grant No. CX2010B026; and the Research Foundation of National University of Defense Technology under Grant No. JC10-05-01, and the Innovation Program of Shanghai Municipal Education Commission under Grant No. 12ZZ182.
Authors' addresses: D. S. Zhang, F. Y. Chen, and S. Y. Jin, PDL, School of Computer, National University of Defense Technology, China; email: dszhang@nudt.edu.cn; D. K. Guo, School of Information System and Management, National University of Defense Technology, China; F. Wu, College of Electronic and Electrical Engineering, Shanghai University of Engineering Science, China; T. Wu, Center for National Security and Strategic Studies, National University of Defense Technology, China; T. Cao, Research School of Computer Science, Australian National University, Australia.

DOI 10.1145/2086696.2086726 http://doi.acm.org/10.1145/2086696.2086726

**ACM Reference Format:**
Zhang, D., Guo, D., Chen, F., Wu, F., Wu, T., Cao, T., and Jin, S. 2012. TL-plane-based multi-core energy-efficient real-time scheduling algorithm for sporadic tasks. ACM Trans. Architec. Code Optim. 8, 4, Article 47 (January 2012), 20 pages.
DOI = 10.1145/2086696.2086726 http://doi.acm.org/10.1145/2086696.2086726

## 1. INTRODUCTION

With the rapid development of semiconductor technology, multi-core processors (On-chip Multiprocessors or CMP) have been increasingly adopted by chip designers. Most chip manufacturers (Intel, AMD, etc) have embedded two, four and eight-core processors into one single chip [Dorsey 2007; Naveh 2006; Kumar and Hinton 2009]. Furthermore, many efforts are moving toward integrating hundreds or thousands of cores on a single chip (called many-core processors [Mosley 2008]). Multi-core processors are increasingly being used in high-end real-time systems for achieving high performance/throughput, such as the robot control, image processing, and automatic target recognition. Energy consumption, however, is one of the primary design objectives in many embedded real-time fields, especially in the field of portable computing. With the increasing requirements of green computing, energy-efficient real-time scheduling is becoming increasingly important in multi-core systems.

To reduce the energy consumption, modern multi-core processor systems utilize a wide variety of hardware energy-efficient technologies, such as DPM [Rele et al. 2002] and DVFS [Chandrakasan et al. 1992]. The energy-efficient real-time scheduling technologies can take advantages of DPM and DVFS to dynamically adjust the operating mode and decrease the supply voltage and frequency [Dorsey 2007; Naveh 2006] of each processor. Those energy-efficient scheduling technologies implemented by hardware-based approaches, however, do not take account of the real-time constraint in real-time systems. The decreased frequency is likely to incur the increase in the execution time of a task, thus violating the constraint of real-time [Pillai and Shin 2001]. Therefore, the goal of an energy-efficient real-time scheduling scheme is to achieve energy-saving, as many as possible, and satisfy all real-time constraints simultaneously.

Recently, researchers have presented many energy-efficient real-time scheduling algorithms [AlEnawy and Aydin 2005; Aydin and Yang 2003; Chen et al. 2006; Yang et al. 2009] for real-time embedded systems running on traditional multiprocessor platforms. The methods adopted by these algorithms usually first assign each task to one designated processor and then use the EDF algorithm to implement an energy-efficient scheduling on a single processor. Each processor in such a multiprocessor system has an individual voltage level. Many modern multi-core processor platforms have a remarkable feature compared to traditional multiprocessor platforms, that is, all cores share the same voltage and frequency [Herbert and Marculescu 2007; Kim et al. 2008]. For example, Intel's Itanium [McGowen et al. 2006], Core Duo [Naveh 2006], i7 [Intel 2011], and IBM's Power 7 Series [McCreary et al. 2007] support the global DVFS on-chip technology [Kim et al. 2008].

More and more researchers have focused on the energy-efficient real-time scheduling on multi-core platforms, with the global DVFS on-chip technology. With the constraints of global voltage and frequency, a core with the maximum workload at the scheduling time plays a dominant factor [Seo et al. 2008; Yang et al. 2005]. The current study on multi-core platforms with the global DVFS technology focuses on the load balancing across cores, so as to minimize the energy consumption. Existing research efforts [Seo et al. 2008; Yang et al. 2005; Devadas and Aydin 2010; Huang et al. 2009], however, still face three challenging issues: (1) most of energy-efficient scheduling algorithms are based on the partitioning scheduling method or the non-optimal global scheduling method; hence, the optimal feasibility of a set of real-time tasks could not be guaranteed. Aydin and Yang [2003] have proved that the task-to-processor assignment with the optimal load balancing and feasibility is an NP-hard problem. (2) The existing algorithms often utilize the method of turning off one or more cores for reducing energy consumption. Although such method can obtain more energy-saving for low workloads, it is not suitable for high workloads. (3) Recent research efforts are based on the frame task model or periodic task model. However, they paid little attention to the sporadic task model which is more general compared to the frame and periodic task models in reality. In the case of the sporadic task model, the deadline and release time of a real-time task are uncertain. Moreover, most recent algorithms are static and did not consider the dynamic releases of sporadic tasks. Thus, it is not efficient and practical to directly apply existing algorithms to solve the energy-efficient real-time scheduling problem of sporadic tasks.

In this paper, we propose TL-DVFS, an energy-efficient real-time scheduling algorithm with regard to sporadic tasks running on multi-core processors with the DVFS and DPM technologies. We also prove the optimal schedulability of the TL-DVFS for ensuring its advantages in theory. Additionally, the TL-DVFS can not only guarantee the optimal feasibility of sporadic tasks, but also achieve more energy-saving, even in the case of high workloads.

The rest of this paper is organized as follows: Section 2 discusses the related work. Section 3 provides the system model. In Section 4, we propose the idea of energy-efficient real-time scheduling in a TL plane, which is the basis of the TL-DVFS algorithm presented in Section 5. The evaluation results are presented in Section 6. In Section 7, we conclude this paper.

## 2. RELATED WORK

In past decades, many efforts have been done on the energy-efficient real-time scheduling on a single processor [Pillai and Shin 2001; Lee and Shin 2004]. Along with such research efforts, many researchers began studying the same problem on multiprocessor systems [AlEnawy and Aydin 2005; Aydin and Yang 2003; Chen et al. 2006; Yang et al. 2009].

Energy management of real-time tasks on CMP platforms under the global DVFS constraint has started to attract the attentions of research communities. Yang et al. [2005] proved that the energy management of a frame-based system is NP-Hard and provided a static energy-efficient real-time scheduling algorithm. This algorithm is only applied to a simple task model with the same deadline and release time; hence, it is not effective for the periodic and sporadic task models. Bautista et al. [2008] proposed a power-aware scheduler for soft real-time tasks on multi-core systems, which cannot guarantee hard real-time constraints. With regard to the energy-efficient scheduling of periodic hard real-time tasks on CMP systems, Seo et al. [2008] proposed two dynamic schemes: the Dynamic Repartition (DR) and Dynamic Core Scaling (DCS). The DR scheme repartitions tasks at runtime by resorting to task migrations, so as to create a more balanced schedule that can adapt to the dynamic workload. The DCS adjusts the number of active cores at runtime to reduce static power consumptions under the assumption that transitions between off and active states can be done instantaneously without any additional overheads. They, however, ignored the utilization of a critical frequency due to the impractical consumption that the leakage power consumption depends on the frequency. Huang et al. [2009] presented a Global EDF-based energy-efficient real-time scheduling algorithm for periodic tasks in multi-core systems. The algorithm can decrease the leakage power consumption and achieve more energy-saving in the case of low workloads. However, the algorithm is not well suitable to high workloads. Devadas and Aydin [2010] also proposed an online energy-efficient real-time scheduling algorithm, based on a variety of heuristic strategies for periodic tasks in multi-core systems.

To meet real-time constraints, research efforts usually utilize the real-time scheduling theory. Most of them adopt the partitioning schedule or non-optimal schedule [Baker 2005] to implement a static or dynamic energy-efficient real-time scheduling for frame or periodic tasks. Based on the optimal real-time scheduling algorithm LLREF [Cho et al. 2006] proposed for periodic tasks, Funaoka et al. [2008] presented an optimal static energy-efficient real-time scheduling algorithm for multi-core platforms with the global DVFS technology. However, static methods do not take account of the uncertain release time of a sporadic task; hence, energy-saving can be further improved. For a sporadic task model, it is difficult to design a real-time scheduling algorithm to guarantee the deadlines of all tasks. Fisher et al. [2010] proved that there is no optimal online scheduling algorithm for sporadic tasks, each with an arbitrary release time. For sporadic tasks whose deadlines are equal to their periods, Funk and Nadadur [2009] proposed an optimal online real-time scheduling algorithm, called a LRE-TL. In this paper, we design an energy-efficient real-time scheduling algorithm, based on the LRE-TL, to obtain energy-saving while guaranteeing the optimal feasibility.

## 3. SYSTEM MODEL

### 3.1. Task Model

We explore the energy-efficient real-time scheduling for a set of sporadic tasks on an $m$-core processor with the DVFS technology. Let the task set $\Gamma = \{T_1, T_2, \ldots, T_n\}$ denote a set of $n$ sporadic tasks, each of which only has a single control thread. Each task $T_i$ is an infinite sequence of jobs, denoted as $T_{i1}, T_{i2}, \ldots. T_i$ is described using the following 4-tuple $(\varphi_i, a_{ij}, P_i, C_i)$, where $\varphi_i$ is the offset of $T_{i1}$, $a_{ij}$ is the release time of the $j$-th invocation, $P_i$ is its period or the minimum release interval, and $C_i$ is the amount of the worst-case execution time at the highest profiling frequency $f_m$. $T_i$ invokes its first job at the time of $\varphi_i$ and its remaining jobs are processed at an interval of no less than $P_i$ time units, i.e., $a_{i1} = \varphi_i \geq 0$ and $a_{ij} \geq a_{i(j-1)} + P_i$. The relative deadline of the job $T_{ij}$ is equal to its period $P_i$. If the job $T_{ij}$ is released at the time of $a_{ij}$, its absolute deadline is $d_{ij} = a_{ij} + P_{ij}$ during $[a_{ij}, a_{ij} + P_i]$. All jobs must be completed before their respective deadlines. That is, if a job $T_{ij}$ is executed on a processor core $core_k$ at a frequency $\alpha_k (0 \leq \alpha_k \leq 1)$, it requires $C_i/\alpha_k$ time units at every $P_i$ interval. The ratio $C_i/P_i$, denoted as $u_i$, is the task utilization of $T_i$. Let $U = \Sigma_{T_i \in \Gamma} u_i$ denote the utilization of a task set. The maximum task utilization is defined as $u_{max} = \max\{u_i | T_i \in \Gamma\}$. We assume that all independent tasks may be preempted and migrated among cores at any time and all tasks will not be given any precedence.

### 3.2. Power Model

This paper considers the energy-efficient real-time scheduling problem on a homogeneous multi-core system. We assume that there are $m$ homogeneous cores, denoted as $\{core_1, core_2, \ldots, core_m\}$. The voltages and frequencies of all cores are scaled together. The design of electronic circuitry is usually done such that different supply voltages would result in different execution frequencies.

The total power consumption ($P_{tot}$) of a CMOS-based processor consists of dynamic power consumption ($P_d$), static power consumption ($P_l$), and an inherent power cost ($P_{on}$) [ACPI 2011]. That is, $P_{tot} = P_d + P_l + P_{on}$. The dynamic power consumption resulting from charging and discharging gates on a CMOS DVFS processor can be expressed in terms of the operating voltage $V_{dd}$, the clock frequency $f$, and the switching capacity $C_{eff}$ as follows [Jejurikar et al. 2004]:

$$P_d = C_{eff} \cdot V_{dd}^2 \cdot f. \tag{1}$$

Table I. The 70 nm Technology Constants for Transmeta Crusoe Processor

| Variable | Value | Variable | Value | Variable | Value |
|---|---|---|---|---|---|
| $K_1$ | 0.063 | $K_6$ | $5.26 \times 10^{-12}$ | $L_d$ | 37 |
| $K_2$ | 0.153 | $V_{bs}$ | $-0.7$ | $L_g$ | $4 \times 10^6$ |
| $K_3$ | $5.38 \times 10^{-7}$ | $V_{th1}$ | 0.244 | $\epsilon$ | 1.5 |
| $K_4$ | 1.83 | $I_j$ | $4.8 \times 10^{-10}$ | $V_1$ | $0.5 \times 10^9$ |
| $K_5$ | 4.19 | $C_{eff}$ | $4.3 \times 10^{-10}$ | $V_m$ | $1.0 \times 10^9$ |

According to Formula (2), we can see that the clock frequency $f$ is related to several factors. The threshold voltage $V_{th}$ is a function of the body bias voltage $V_{bs}$, as shown in Formula (3). Here, $\epsilon$, $V_{th1}$, $L_d$, $K_1$, $K_2$ and $K_6$ are constants depending on the processor fabrication technology.

$$f = \frac{(V_{dd} - V_{th})^{\epsilon}}{L_d \cdot K_6}. \tag{2}$$

$$V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs}. \tag{3}$$

The static power dissipation due to the subthreshold leakage ($I_{subn}$) and the reverse bias junction current ($I_j$) is given by Formula (4), where $L_g$ is the number of components in the circuit, $K_3$, $K_4$, and $K_5$ are constants determined by the processor fabrication technology.

$$P_l = L_g \cdot (V_{dd} \cdot I_{subn} + |V_{bs}| \cdot I_j). \tag{4}$$

$$I_{subn} = K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}}. \tag{5}$$

The realistic constants based on the 70nm technology for Transmeta Crusoe processor are presented in Table I, as given in Jejurikar et al. [2004] and Martin et al. [2002]. To reduce the leakage power substantially, we assume that $V_{bs} = -0.7V$. We also assume a conservative value of $P_{on} = 0.1W$ as an existing literature [Jejurikar et al. 2004] did.

This type of power consumption model has been adopted by many related works [Jejurikar et al. 2004; Seo et al. 2008; Huang et al. 2009]. Note that the proposed scheduling algorithm would be applied to multi-core systems with other power consumption models after minimal modifications.

When a core is not executing any instruction, modern multi-core processors usually adopt the DPM technique to put the core into a dormant mode (or shutdown) [ACPI 2011]. Additionally, each core would be switched from the dormant mode to the active mode (or wakeup). In this paper, we assume that the overhead of shutdown/wakeup is neglected because this factor can be treated easily in practical implementations [Seo et al. 2008]. When a core becomes idle, the DPM can shutdown it and assume that its power consumption should be zero. Note that those results proposed in this paper still hold when the power consumption of an idle core is not zero.

## 3.3. Energy Model

According to Jejurikar et al. [2004], the sum of the static and dynamic energy consumptions per cycle, denoted as $f^{-1} \cdot P_{tot}$, decreases as $V_{dd}$ scales up to $0.7V$, beyond which the static energy consumption dominates the value of $f^{-1} \cdot P_{tot}$. In other words, it is more energy-efficient to execute at $V_{dd} = 0.7V$ and shut down the system than execute at lower voltage levels. This implies that there is a critical frequency $f_{crit}$ that minimizes the energy consumption per cycle. Note that the critical frequency can be computed by evaluating the gradient of the energy function $f^{-1} \cdot P_{tot}$ with respect to

$V_{dd}$ [Jejurikar et al. 2004]. From the relationship between the voltage and frequency in Formula (2), we can see that $V_{dd} = 0.7V$ incurs a critical frequency of $1.26GHz$.

We assume that the processor voltage can be a discrete variable ranging from $V_1$ to $V_m$, and the corresponding discrete frequency variable ranging from $f_1$ to $f_m$. The possible values of such two variables are set to be $\{(V_1, f_1), \ldots, (V_m, f_m) | V_1 < \cdots < V_m, f_1 < \cdots < f_m\}$, where $V_1$ and $V_m$ represent the lowest and highest voltages, respectively, while $f_1$ and $f_m$ represent the lowest and highest frequencies, respectively. According to the voltage frequency relation described in Formula (2), $V_{dd} = 0.5V$ results in the minimum frequency $f_1 = 394MHz$ and the maximum frequency is $f_m = 3.1GHz$ when $V_{dd} = 1.0V$.

In this paper, the overhead of voltage and frequency switching is assumed to be neglected. The execution frequency of a processor $core_k$ is assumed to be no less than the critical frequency. Then we update the computed frequency scaling factor $\alpha_k = f/f_m$ of $core_k$ based on the critical frequency. For sporadic tasks running on $core_k$, the energy consumption $E_k(\alpha_k)$ within the interval $L$ can be defined as follows:

$$E_k(\alpha_k) = \begin{cases} 0, & \text{if } \quad \alpha_k = 0 \\ L \cdot {f_j}^{-1} \cdot P_{tot}(f_j) \cdot \alpha_k \cdot f_m, & \text{if } 0 < \alpha_k \leq f_{crit}/f_m, \exists j, f_{j-1} < f_{crit} \leq f_j \\ L \cdot {f_{i+1}}^{-1} \cdot P_{tot}(f_{i+1}) \cdot \alpha_k \cdot f_m, & \text{if } f_{crit}/f_m < \alpha_k \leq 1, f_i < \alpha_k \cdot f_m \leq f_{i+1} \end{cases} \tag{6}$$

Where $f_j$ is the lowest discrete frequency among frequencies that are higher than $f_{crit}$, $f_i$ is the highest discrete frequency among frequencies that are lower than $\alpha_k \cdot f_m$, and $f_{i+1}$ is the lowest discrete frequency among frequencies that are higher than $\alpha_k \cdot f_m$. In other words, we choose the lowest frequency $f_i \in \{f_1, \ldots, f_m | f_1 < \cdots < f_m\}$ such that $\alpha_k \leq f_i/f_m$ in actual discrete environments.

## 4. TL-PLANE-BASED ENERGY-EFFICIENT REAL-TIME SCHEDULING

The TL plane is an abstraction technique of real-time scheduling. It is proposed in the LLREF algorithm [Cho et al. 2006] for periodic tasks and is applied to the LRE-TL algorithm [Funk and Nadadur 2009] for sporadic tasks. The TL plane is a two dimensional plane where the horizontal axis represents the time (T) and the vertical axis represents the local remaining execution (L). Absolute deadlines divide time as the vertical dotted lines and a TL plane is placed between every two adjacent deadlines.

To implement an energy-efficient real-time scheduling, the dynamic voltage and frequency scaling technologies are introduced into the optimal real-time scheduling algorithm LRE-TL which is based on the TL plane. We use the fluid scheduling mode [Holman and Anderson 2005] to perform an energy-efficient real-time scheduling of tasks within the same TL plane. The aim is to achieve energy-saving by using the dynamic voltage and frequency scaling while guaranteeing the optimal feasibility of tasks in each TL plane, as shown in Figure 1. The identifier token refers to the status of each task in a TL plane. Its location is marked by the horizontal axis (time) and the vertical axis (local remaining execution time). The $\alpha$ oblique side is defined as the slope line from an initial time $t_0$ to the end time $t_f$ with the slope of $-\alpha$, which implies that the frequency of a processor core is $\alpha \cdot f_m$. It can be seen from Figure 1 that the local remaining execution time of the task $T_1$ is $l_1 = \alpha \cdot t_f$ and the effective local laxity time of the task $T_n$ is defined as $(t_f - t_0 - l_n/\alpha)$.

Figure 1 shows that the $\alpha$ oblique side is not fixed since the execution speed of each task can be dynamically changed within $[f_1, f_m]$. We perform the frequency scaling at the time when three kinds of events occur: (1) the first event occurs at the time when a task's local remaining execution time is 0. In this case the marked token arrives at the horizontal axis, for example, $T_m$ moves to the time $t_b$, as shown in Figure 1. We name this kind of event as *Event B*. Once an event B occurs, a task $T_m$ is finished and

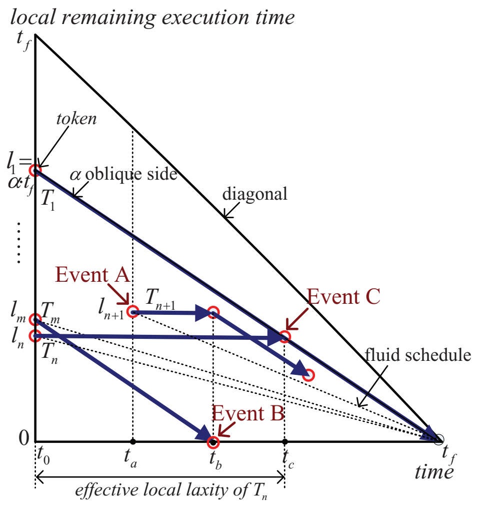


Fig. 1. Energy-efficient real-time scheduling in a TL plane.

the system can schedule other tasks. (2) The second event occurs at the time when a sporadic task is released in a TL plane. In this case a new marked token is added into the TL plane. For example, a task $T_{n+1}$ is released at the time of $t_a$. This event is named as *Event A*. Once an event A occurs, the task set is changed and the system should reset the frequency. (3) The third event occurs at the time when the effective local laxity time is 0. In this case the token hits the $\alpha$ oblique side. For example, $T_n$ moves to the time $t_c$. We name this type of event as *Event C*. Once an event C occurs, a task $T_n$ must be selected to run at the frequency of $\alpha \cdot f_m$; otherwise, its local feasibility will not be guaranteed.

When the schedule and the frequency are determined, the marked token of each task moves within a TL plane. Each task is allowed to move towards two directions: (1) If the task is selected to run, the marked token will move along the $\alpha$ oblique side, such as $T_1$ and $T_m$. (2) Otherwise, the marked token moves along the horizontal direction, such as $T_n$ moving in Figure 1. At any time, up to $m$ tasks move along the $\alpha$ oblique side within each TL plane, where $m$ denotes the number of cores.

At any time $t$, the local utilization $u_{i,t}$ of task $T_i$ is defined as the proportion of the local remaining execution time to the remaining time within the current TL plane, that is $u_{i,t} = l_{i,t}/(t_{f_t} - t)$ [Cho et al. 2006]. If the local remaining execution time of $T_i$ is larger than 0, that is $l_{i,t} > 0$, $T_i$ is active. To distinguish the active tasks and inactive tasks, we let $Active(t)$ be the set of active tasks at the time of $t$ [Funk and Nadadur 2009]. Furthermore, the task $T_i$ has an effective local remaining execution time at the time of $t$, namely $l'_{i,t} = l_{i,t}/\alpha_i$. Task $T_i$'s effective local utilization is the proportion of time that $T_i$ must be executed at a frequency factor $\alpha_i$ during the remainder of the current TL-plane, that is $r'_{i,t} = l'_{i,t}/(t_{f_t} - t)$. The total utilization of $Active(t)$, denoted as $U_t$, is defined to be the sum of each active task's utilization at the time of $t$, that is $U_t = \sum_{\forall T_i \in Active(t)} u_i$. The maximum utilization $u^t_{max}$ at the time of $t$ is $u^t_{max} = \max\{u_i | \forall T_i \in Active(t)\}$.

## 5. ALGORITHM: TL-DVFS

We consider the problem of scheduling sporadic tasks on homogeneous multi-core processors. We propose TL-DVFS, an energy-efficient real-time scheduling algorithm based on LRE-TL, to achieve as many energy-savings as possible while guaranteeing the optimal feasibility.

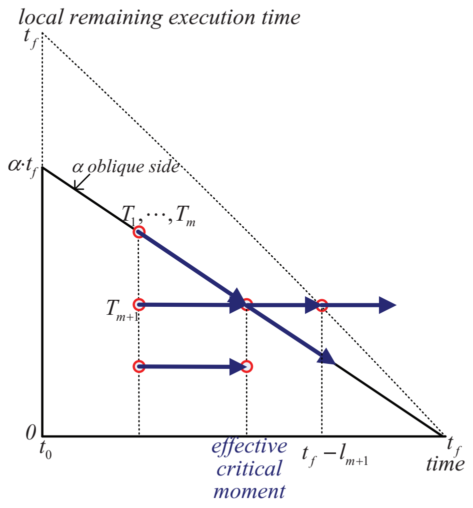


Fig. 2. Effective critical moment.

## 5.1. Effective Critical Moment

We assume that $U \leq m$ and $u_{max} \leq 1$ for all sporadic tasks. The following theorems demonstrate the properties that must be guaranteed at the initial time of a TL plane for all the active jobs.

THEOREM 5.1. *With a frequency scaling factor $\alpha$, if all jobs within each TL plane can reach their end times, the initial effective local utilization of each active job $T_i \in Active(t_0)$ is computed as: $r'_{i,0} = u_{i,0}/\alpha$.*

PROOF. The remaining execution time of any job $T_i$ is $l_i = 0$ when the job reaches the end time of the last TL plane. Therefore, at the initial time $t_0$ of the current TL plane, these jobs must be restarted from the start time of the fluid scheduling in an ideal case. In this case, the job's local remaining execution time is $l_{i,0} = u_{i,0}(t_f - t_0)$ at the initial time. With the frequency scaling factor $\alpha$, each active job $T_i \in Active(t_0)$ moves along the $\alpha$ oblique side. The actual local remaining execution time of job $T_i$ is the effective local remaining execution time, that is $l'_{i,0} = l_{i,0}/\alpha$. Therefore, we have $r'_{i,0} = l'_{i,0}/(t_f - t_0) = u_{i,0}/\alpha$ and Theorem 5.1 holds. □

All jobs are locally feasible if the local remaining execution time of each job is 0 at the end time $t_f$ of a TL plane [Cho et al. 2006]. In order to analyze the local feasibility, we propose the concept of effective critical moment to describe the sufficient and necessary condition of a local unfeasibility within a TL plane.

*Definition* 5.2. The effective critical moment is the first time that is the time more than $m$ jobs simultaneously hit the $\alpha$ oblique side of a TL plane with a frequency scaling factor $\alpha$. Here $m$ is the number of cores.

As shown in Figure 2, on the right side of the effective critical moment, no more than $m$ jobs are selected to run along the $\alpha$ oblique side. Those unselected jobs will be off the $\alpha$ oblique side and leave the TL plane at the time of $(t_f - l_{m+1})$. In this case those unselected jobs are unable to reach the end time $t_f$. Note that all jobs within the TL plane are only allowed to move horizontally or along the $\alpha$ oblique side.

THEOREM 5.3. *With the frequency scaling factor $\alpha$, the task set is unfeasible in a TL plane if and only if at least one effective critical moment occurs.*

Proof.

*Necessary*. We prove it by using the contradiction method. We assume that an effective critical moment does not occur when the task set is unfeasible. In this case, according to Definition 5.2 we can conclude that less than $m$ jobs simultaneously hit the $\alpha$ oblique side of the TL plane at any time. Therefore all jobs on the $\alpha$ oblique side will be selected to execute until the end time of the TL plane. Thus, by contradictory an effective critical moment must occur when a task set is unfeasible.

*Sufficient*. We assume that an effective critical moment occurs. In this case, those unselected jobs must leave the TL plane. With a frequency scaling factor $\alpha$, all jobs move horizontally or along the $\alpha$ oblique side. The slope of their path is 0 or $-\alpha$. Therefore the jobs leaving the TL plane can not reach the end time of the TL plane. □

The total effective local utilization of a task set is defined as $R'_t = \sum_{i=1}^{n} r'_{i,t}$. We have the following corollary.

Corollary 5.4. *With a frequency scaling factor $\alpha$, $R'_t > m$ must hold when the effective critical moment $t$ occurs.*

Proof. Since all jobs move horizontally or along the $\alpha$ oblique side with the frequency scaling factor $\alpha$, the local remaining execution time $l_{i,t}$ of the $m$ jobs on the $\alpha$ oblique side must be equal to $\alpha(t_{f_t} - t)$ at the effective critical moment $t$. In this case, $R'_t = \sum_{i=1}^{N} r'_{i,t} = \sum_{i=1}^{m} \frac{l_{i,t}/\alpha}{t_{f_t}-t} + \sum_{i=m+1}^{N} \frac{l_{i,t}/\alpha}{t_{f_t}-t} = \sum_{i=1}^{m} \frac{t_{f_t}-t}{t_{f_t}-t} + \sum_{i=m+1}^{N} \frac{l_{i,t}/\alpha}{t_{f_t}-t} > m$, where $N = |Active(t)|$ and $N \leq n$. □

### 5.2. TL Plane Initialization

In order to achieve more energy-saving while guaranteeing the local feasibility of jobs in each TL plane, we need to choose an initial frequency scaling factor for all active jobs. At the initial time of each TL plane, not all sporadic jobs are active and only the feasibility of current active jobs needs to be satisfied; hence, it is not necessary to choose the highest frequency.

The voltage and frequency of $core_k$ can be determined by a frequency scaling factor $\alpha_k$. Thus, the dynamic scaling of the processor voltage and frequency is equivalent to the dynamic choice of frequency scaling factor. In this paper the target platform is the multi-core processors with the on-chip global DVFS and DPM techniques. The frequency scaling factors of all the $m'$ running cores must be the same, that is $\alpha = \alpha_1 = \cdots = \alpha_{m'}$. Here, $m' = \min\{m, |Active(t)|\}$, and $m$ is the number of cores. According to the idea of the fluid scheduling, the job frequency is not less than its task utilization and the maximum utilization of all active jobs at the initial time $t_0$ of each TL plane (as shown in lines 3 to 7 in Algorithm 1). Otherwise, the deadline is missed. Meanwhile, in order to guarantee the local feasibility of all active jobs with the frequency scaling factor $\alpha$, we have $\alpha \geq U_0/m'$ according to the EDF algorithm. Therefore $\alpha = \alpha_1 = \cdots = \alpha_{m'} = \max\{u_{max}, U_0/m'\}$ (as shown in lines 8 to 9 in Algorithm 1). We prove this by using the following corollary.

Corollary 5.5. *At the initial time $t_0$ of a TL plane, with the frequency scaling factor $\alpha$ computed by Algorithm 1, the effective local utilization $r'_{i,0}$ does not exceed one for any job $T_i \in Active(t_0)$. Meanwhile the total effective local utilization is $R'_{t_0} \leq m$.*

Proof. At the initial time $t_0$ of a TL plane, $\forall T_i \in Active(t_0)$, $u_{i,0} \leq u^0_{max}$. According to Algorithm 1 we have $u^0_{max} \leq \alpha \leq 1$. We can derive from Theorem 5.1 that $r'_{i,0} \leq 1$. Meanwhile, according to Algorithm 1 we have $U_0/m' \leq \alpha \leq 1$, where $U_0 = \sum_{T_i \in Active(t_0)} u_{i,0}$

**ALGORITHM 1:** TL-Initialize-Frequency-Selection

```
Input: All jobs in $Active(t)$ task set at the current time $t$.
Output: The frequency scaling factor $\alpha$ for all the $m'$ processor cores.
1: $U = 0$; /*$U$ is a global variable*/
2: $u_{max} = 0$; /*$u_{max}$ is a global variable*/
3: for each job $T_i \in Active(t)$ do
4:     $u_i = C_i/P_i$;
5:     $U = U + u_i$;
6:     if $u_{max} < u_i$ then
7:         $u_{max} = u_i$;
8: $m' = \min\{m, |Active(t)|\}$;
9: return $\alpha = \max\{u_{max}, U/m'\}$;
```

and $m' = \min\{m, |Active(t_0)|\}$. Without lose of generality, we assume that $N = |Active(t_0)|$, $N \leq n$. Therefore, $R'_{t_0} = \sum_{i=1}^{N} r'_{i,0} = \sum_{i=1}^{N} u_{i,0}/\alpha = U_0/\alpha \leq m' \leq m$. □

In order to guarantee the optimal feasibility, all active jobs must be initialized after scaling the frequency. We adopt two heaps, $H_B$ and $H_C$, to initialize all active jobs. $H_B$ and $H_C$ consist of running active jobs and suspended active jobs, respectively.

During the process of initialization, all jobs are inserted into $H_B$ or $H_C$. We then set the *key* term as the amount of time triggering a certain event B or C. When the frequency is scaled, the local remaining execution time $l_{i,t}$ is changed to be the effective local remaining execution time $l'_{i,t}$. Therefore, if a job $T_i$ is inserted into $H_B$, the *key* term is set to be $(t + l'_{i,t})$ where $t$ denotes the initial time of the TL plane (as shown in lines 14 to 20 in Algorithm 2). If a job $T_i$ is inserted into $H_C$, the *key* term is set to be $(t_f - l'_{i,t})$ where $t_f$ denotes the end time of the TL plane (as shown in lines 21 to 23 in Algorithm 2). Due to the uncertain release time of each sporadic job, the TL-DVFS algorithm adopts the same method as the LRE-TL algorithm to determine the end time of the TL plane (as shown in lines 1 to 8 in Algorithm 2).

### 5.3. Event B or C

An event B refers to the normal completion of a job within the current TL plane. In other words, the local remaining execution time is 0. An event C refers to the time when the effective local utilization of a job is 1. When a job triggers an event C, if the job is not scheduled to be run immediately, it will leave the $\alpha$ oblique side and its effective local utilization exceeds 1. In this case, the job cannot reach the end time of the TL plane with the frequency scaling factor $\alpha$. We adopt the same method as the LRE-TL algorithm to handle the event B and event C. Due to space limitations, please refer to Funk and Nadadur [2009] for more information about this issue.

In a TL plane, there is no effective local laxity time when an event B and an event C occur. Therefore we do not perform frequency scaling. However, the triggering time of an event B or an event C is changed; hence, the scheduling time is different from that under the LRE-TL algorithm. We need to prove that the operations of an event B and an event C will not affect the local feasibility of all jobs within the TL plane.

THEOREM 5.6. *$\Gamma$ is assumed to be the task set running on an m-core processor. In each TL plane $[t_0, t_f]$, the effective local remaining execution time of each active job is computed in terms of the job utilization and frequency scaling factor $\alpha$. At the initial time of each TL plane, we have $U_0 \leq m$ and $u^0_{max} \leq 1$. Let $s \in [t_0, t_f]$ denote any time in the TL plane, $\forall T_i \in Active(t)$, $R'_s \leq m$, and $r'_{r,s} \leq 1$. Let X be the set of any $x_s$ jobs scheduled in Active(s) at the time of s. If t is the next time when Event B or C occurs,*

**ALGORITHM 2:** TL-DVFS-Initialize

```
$t = 0; t_f = 0; P_{min} = \infty; z = 1;$
for any job $T_i$ arrived at the time of $t$ do
    if $H_D.find\_key(t + P_i) == NULL$ then
        $H_D.insert(t + P_i);$
        $P_{min} = \min\{P_{min}, t + P_i\};$
$t_f = t + P_{min};$
if $H_D.min\_key() \le t_f$ then
    $t_f = H_D.extract\_min();$
$\alpha$ =TL-Initialize-Frequency-Selection();
for each job $T_i$ in $Active(t)$ do
    $T_i.speed = \alpha;$
    $l = u_i \cdot (t_f - t);$
    $l' = l / T_i.speed;$
    if $z \le m$ then
        $T_i.key = t + l';$
        $T_i.proc\_id = z;$
        $z.task\_id = T_i;$
        $z.speed = T_i.speed;$
        $H_B.insert(T_i);$
        $z = z + 1;$
    else
        $T_i.key = t_f - l';$ /*all cores are busy*/
        $H_C.insert(T_i);$
for all cores $z'$ s.t. $m \ge z' > z$ do
    $z'.task\_id = NULL;$
```

*then* $\forall\Delta$, $0 \le \Delta \le t - s$, $R'_t \le R'_{s+\Delta} \le R'_s$. *Furthermore, if* $R'_s < m$, *then* $R'_t < R'_{s+\Delta} < R'_s$. *In addition, if* $R'_s = m$, *then* $R'_t = R'_{s+\Delta} = R'_s$.

Proof. With the frequency scaling factor $\alpha$, for any $\Delta$ such that $0 \le \Delta \le t - s$, the load completed by all $x_s$ jobs during a time interval $[s, s + \Delta]$ is equal to $x_s \times \Delta \times \alpha$. So we have $\sum_{i=1}^{N} l_{i,s+\Delta} = \sum_{i=1}^{N} l_{i,s} - x_s \times \Delta \times \alpha$ and $\sum_{i=1}^{N} l'_{i,s+\Delta} = \sum_{i=1}^{N} l'_{i,s} - x_s \times \Delta$, where $N$ is the number of jobs from $Active(s)$, namely $N = |Active(s)|$. Thus, we have $R'_{s+\Delta} = \sum_{i=1}^{N} r'_{i,s+\Delta} = \sum_{i=1}^{N} \frac{l'_{i,s+\Delta}}{t_f-s-\Delta} = \frac{\sum_{i=1}^{N} l'_{i,s} - x_s \times \Delta}{t_f-s-\Delta} = \sum_{i=1}^{N} \frac{l'_{i,s}}{t_f-s} \times \frac{t_f-s}{t_f-s-\Delta} - \frac{x_s \times \Delta}{t_f-s-\Delta} = R'_s + \frac{\Delta(R'_s - x_s)}{t_f-s-\Delta}$. To prove $R'_{s+\Delta} \le R'_s$, we further discuss the following two cases:

(1) If $Active(s) \ge m$, then $x_s = m$. According to Corollary 5.5 and its induction, we can derive that $R'_s \le m$; hence, $R'_s - x_s = R'_s - m \le 0$ is available.
(2) If $Active(s) < m$, then $x_s = |Active(s)|$. According to Corollary 5.5 and its induction, we can derive that $r'_s \le 1$; hence, $R'_s - x_s \le 0$ is available.

Thus, $R'_t \le R'_{s+\Delta} \le R'_s$ is available. If $R'_s < m$, there must be $r'_{i,s} < 1$ and $R'_s - x_s < 0$; hence, we have $R'_t < R'_{s+\Delta} < R'_s$. If $R'_s = m$, then $r'_{i,s} = 1$ and $R'_s - x_s = 0$ because of $r'_{i,s} \le 1$. In summary, $R'_t = R'_{s+\Delta} = R'_s$. □

According to Theorem 5.6 and Algorithm 1, if $U_0 \le m$ and only an event B or an event C occurs, the total utilization of the task set will remain unchanged at any time $t$ in the current TL plane. That is, $U_t = U_0$.

## 5.4. Event A

We further introduce a new event, namely Event A (arrival event). An event A is triggered when a sporadic task invokes a new job. When an event A occurs, the system's

**ALGORITHM 3:** TL-A-Event-Frequency-Selection

```
Input: A sporadic job not in $Active(t)$ at time $t$, triggers an event A, whose utilization is $u_s$.
Output: The frequency scaling factor $\alpha$ for all the $m'$ processor cores.
1: $U = U + u_s$; /*$U$ is a global variable*/
2: $u_{max} = \max\{u_s, u_{max}\}$; /*$u_{max}$ is a global variable*/
3: $m' = \min\{m, |Active(t)|\}$;
4: return $\alpha = \max\{u_{max}, U/m'\}$;
```

workload increases and the number and properties of active jobs are changed. We try to reset the frequency scaling factor to achieve more energy-saving while guaranteeing the local optimal feasibility of sporadic jobs, as shown in Algorithm 3. When a sporadic task $T_s$ invokes a new job at the time of $t_s$, it triggers an event A. According to Theorem 5.6, the total effective local utilization of the active task set is at most $(m{-}u_s)$ before the release of $T_s$. Moreover, the local remaining execution time of $T_s$ is proportional to its utilization, namely, $l_{s,t_s} = u_s(t_{f_{t_s}} - t_s)$. To guarantee the local feasibility of all active jobs in the current TL plane, we need to add the utilization of $T_s$ into the total utilization of all jobs, that is $U_{t_s} = U_{t_s} + u_s$. Meanwhile, the maximum utilization of the current TL plane is $u_{max} = \max\{u_s, u_{max}\}$. So we can determine a new frequency scaling factor $\alpha_{t_s} = \max\{u_{max}, U/m'\}$ when an event A occurs, where $m' = \min\{m, |Active(t_s)|\}$ (as shown in lines 1 to 4 in Algorithm 3).

The event A handler is illustrated in Algorithm 4. When a new frequency scaling factor $\alpha_{t_s}$ is computed, we also need to update all properties of active jobs, such as the effective local remaining execution time, execution frequency and the *key* value in the heap $H_B$ or $H_C$ (as shown in lines 3 to 18 in Algorithm 4). If the number of active jobs is less than that of the processor cores, $T_s$ will be added into the heap $H_B$. If $u_s < T_s.speed$, $T_s$ is added into the heap $H_C$. If $u_s \geq T_s.speed$, $T_s$ is added into the heap $H_B$, and preempts the job $T_b$ with the minimum *key* value for executing. Meanwhile, the job $T_b$ will be added into the heap $H_C$.

In this paper, Theorem 5.7 proves that adding $T_s$ into the list of active jobs at the time of $t_s$ to handle an event A does not make other jobs, within the same TL plane, miss their deadlines. Furthermore, $T_s$'s deadline can also be guaranteed.

THEOREM 5.7. *$\Gamma$ is assumed to be the task set running on a m-core processor. For each TL plane in $[t_0, t_f]$, the effective local remaining execution times of all active jobs are computed in terms of the job utilization and frequency scaling factor $\alpha$. At the initial time of each TL plane, we have $U_0 \leq m$ and $u^0_{max} \leq 1$. Suppose a task $T_s$ is not active at the initial time $t_0$, but becomes active at the time of $t_s \in (t_0, t_f)$. Let $\alpha$ be the original frequency scaling factor when the job $T_s$ is released. If $l_{s,t_s}$ is computed according to Algorithm 4, and the frequency of re-scaling factor is determined as $\alpha_{t_s}$ by Algorithm 3, the job $T_s$ will not make any other tasks miss their deadlines. Furthermore, the $T_s$ will finish before the deadline $(t_s + P_s)$.*

PROOF. We prove by using the induction method.

Initial situation. Assume that $T_s$ denotes the first job released in the time interval $(t_0, t_f)$. We prove that $T_s$ cannot make any other jobs or itself miss a deadline.

Since the job $T_s$ is not active at the initial time $t_0$, the total utilization of the active task set is at most $(m - u_s)$ before the release of $T_s$. We have $\sum_{i=1}^{N} \frac{l_{i,t_0}}{t_f - t_0} = U_0 \leq m - u_s$, where $N$ is the number of active jobs at the time of $t_0$, that is $N = |Active(t_0)|$. Before the time $t_s$, only an event B or C occurs. $\forall \triangle, 0 < \triangle \leq t_s - t_0, R'_{t_s-\triangle} \leq R'_{t_0} \leq m$ and $U_{t_s-\triangle} = U_0$ are available according to Corollary 5.5 and Theorem 5.6 and Algorithm 1. Therefore, when the job $T_s$ is released at the time of $t_s$, $l_{s,t_s} = u_s(t_{f_{t_s}} - t_s)$ is available. According to Algorithm 3,

**ALGORITHM 4:** TL-DVFS-A-Event

```
 1: $l = u_s \cdot (t_f - t)$;
 2: $\alpha$ =TL-A-Event-Frequency-Selection();
 3: for each job $T_i$ in $Active(t) = H_B \bigcup H_C \bigcup \{T_s\}$ do
 4:    if $T_i \in H_B$ then
 5:       $l' = T_i.key - t$;
 6:       $l' = l' \times T_i.speed/\alpha$;
 7:       $T_i.speed = \alpha$;
 8:       $T_i.key = t + l'$;
 9:       $(T_i.proc\_id).speed = T_i.speed$;
10:    if $T_i \in H_C$ then
11:       $l' = t_f - T_i.key$;
12:       $l' = l' \times T_i.speed/\alpha$;
13:       $T_i.speed = \alpha$;
14:       $T_i.key = t_f - l'$;
15:    if $T_i = T_s$ then
16:       $l' = l/\alpha$;
17:       $T_s.speed = \alpha$;
18:       $T_s.key = l'$;
19: if $H_B.size() < m$ then
20:    $T_s.key = t + T_s.key$;
21:    $T_s.proc\_id = z$;
22:    $z.task\_id = T_s$;
23:    $z.speed = T_s.speed$;
24:    $H_B.insert(T_s)$;
25: else
26:    if $u_s < T_s.speed$ then
27:       $T_s.key = t_f - T_s.key$;
28:       $H_C.insert(T_s)$;
29:    else
30:       $T_b = H_B.extract\_min()$;
31:       $T_s.key = t + T_s.key$;
32:       $T_b.key = t_f - T_b.key + t$;
33:       $z = T_b.proc\_id$;
34:       $T_s.proc\_id = z$;
35:       $z.speed = T_s.speed$;
36:       $H_B.insert(T_s)$;
37:       $H_C.insert(T_b)$;
38: if $H_D.find\_key(t + P_s) == NULL$ then
39:    $H_D.insert(t + P_s)$;
```

$(U_{t_s} + u_s) \geq U_{t_s} = U_0$, $\max\{u_s, u^{t_s}_{max}\} \geq u^{t_s}_{max}$, $\alpha_{t_s} = \max\{\max\{u_s, u^{t_s}_{max}\}, (U_{t_s} + u_s)/m'\}$ and $(U_{t_s} + u_s)/m' \leq \alpha_{t_s} \leq 1$, where $m' = \min\{m, |Active(t_s)|\}$. The occurrence of an event B will reduce the number of active jobs, that is, $|Active(t_s)| \leq |Active(t_0)|$. We have $\sum_{i=1}^{|Active(t_s)|} u_{i,t_s} \leq \sum_{i=1}^{|Active(t_0)|} u_{i,0} = U_0$ because of $R'_{t_s-\Delta} \leq R'_{t_0}$. Thus, $R'_{t_s} = \sum_{i=1}^{|Active(t_s)|} \frac{l'_{i,t_s} \cdot \alpha}{(t_f - t_s) \cdot \alpha_{t_s}} + \frac{l_{s,t_s}}{(t_f - t_s) \cdot \alpha_{t_s}} = \frac{\sum_{i=1}^{|Active(t_s)|} u_{i,t_s}}{\alpha_{t_s}} + \frac{u_s}{\alpha_{t_s}} \leq \frac{U_0 + u_s}{\alpha_{t_s}} = \frac{U_{t_s} + u_s}{\alpha_{t_s}} \leq m' \leq m$. Once the job $T_s$ is added into the *Active* task set, Algorithm 3 will compute a new frequency scaling factor that still meets the local execution time demand of all active jobs. At the initial time of subsequent TL planes, the initial remaining execution time of the job $T_s$ is set to be proportional to the job's utilization. Since $U \leq m$ and $u_{max} \leq 1$, the total utilization of any TL plane is at most $m$, and at any time $t$ the total effective local utilization is also at most $m$. Therefore, $T_s$ does not cause any other job to miss its deadline.

The TL-DVFS algorithm not only adopts the same method as the LRE-TL algorithm to guarantee that $t_s + P_s \geq t_f$ (as shown in lines 38 to 39 in Algorithm 4) but also forms a number of consecutive and non-overlapping TL planes, which can ensure the existence of a TL plane that ends at the time of $(t_s + P_s)$. Therefore, $T_s$ will meet its deadline at the time of $(t_s + P_s)$.

Inductive proof. Assume that a job $T_{s+1}$ is released at the time of $t_{s+1}$ in a time interval $(t_0, t_f)$. Through the induction, conditions of this theorem and consequence are met before $T_{s+1}$. We will prove that the consequence of this theorem is also met at the time of $t_{s+1}$.

Suppose that the job $T_{s+1}$ is not active at the initial time $t_0$, but becomes active at the time of $t_{s+1} \in (t_0, t_f)$. Suppose that the job $T_s$ is released at the time of $t_s$ and is released before the job $T_{s+1}$, where the original frequency scaling factor is $\alpha_{t_s}$. From the induction hypothesis and Theorem 5.6, $\forall \varepsilon$, $0 < \varepsilon \leq t_{s+1} - t_s$, $R'_{t_{s+1}-\varepsilon} \leq R'_{t_s} \leq m$. Since there are no other jobs released at the time intervals of $t_s$ and $t_{s+1}$, only an event B or event C occurs. So $\forall \varepsilon$, $0 < \varepsilon \leq t_{s+1} - t_s$, $U_{t_{s+1}-\varepsilon} = U_{t_s} \leq m$. If $l_{s+1,t_{s+1}} = u_{s+1} \cdot (t_{f_{t_{s+1}}} - t_{s+1})$, and the frequency scaling factor determined by Algorithm 3 is $\alpha_{t_{s+1}}$, then $(U_{t_{s+1}} + u_{s+1}) \geq U_{t_{s+1}} = U_{t_s}$, $\max\{u_{s+1}, u_{max}^{t_{s+1}}\} \geq u_{max}^{t_{s+1}}$, $\alpha_{t_{s+1}} = \max\{\max\{u_{s+1}, u_{max}^{t_{s+1}}\}, (U_{t_{s+1}} + u_{s+1})/m'\}$ and $(U_{t_{s+1}} + u_{s+1})/m' \leq \alpha_{t_{s+1}} \leq 1$ are available, where $m' = \min\{m, |Active(t_{s+1})|\}$. Since $|Active(t_{s+1})| \leq |Active(t_s)|$ and $R'_{t_{s+1}-\varepsilon} \leq R'_{t_s}$, $\sum_{i=1}^{|Active(t_{s+1})|} u_{i,t_{s+1}} \leq \sum_{i=1}^{|Active(t_s)|} u_{i,t_s} = U_{t_s}$, then $R'_{t_{s+1}} = \sum_{i=1}^{|Active(t_{s+1})|} \frac{l'_{i,t_{s+1}} \cdot \alpha_{t_s}}{(t_f - t_{s+1}) \cdot \alpha_{t_{s+1}}} + \frac{l_{s+1,t_{s+1}}}{(t_f - t_{s+1}) \cdot \alpha_{t_{s+1}}} = \sum_{i=1}^{|Active(t_{s+1})|} \frac{l'_{i,t_{s+1}} \cdot \alpha_{t_s}}{(t_f - t_{s+1}) \cdot \alpha_{t_{s+1}}} + \frac{u_{s+1}}{\alpha_{t_{s+1}}} = \frac{\sum_{i=1}^{|Active(t_{s+1})|} u_{i,t_{s+1}}}{\alpha_{t_{s+1}}} + \frac{u_{s+1}}{\alpha_{t_{s+1}}} \leq \frac{U_{t_s} + u_{s+1}}{\alpha_{t_{s+1}}} = \frac{U_{t_{s+1}} + u_{s+1}}{\alpha_{t_{s+1}}} \leq m' \leq m$. Once the job $T_{s+1}$ is added to the *Active* task set, after the frequency scaling factor is calculated by Algorithm 3, this job will not cause other jobs to miss their deadlines before the time of $t_f$.

From the induction hypothesis, we can derive that $t_{s+1} + P_{s+1} \geq t_f$; hence, the task $T_{s+1}$ will not have an earlier deadline than the $t_f$. Meanwhile, the TL-DVFS algorithm can guarantee the existence of a TL plane that ends at the time of $(t_{s+1} + P_{s+1})$. Therefore, the job $T_{s+1}$ will always meet its deadline at the time of $(t_{s+1} + P_{s+1})$. $\square$

## 5.5. Algorithm Description

The main function of the TL-DVFS is similar to that of the Algorithm 1 in Funk and Nadadur [2009]. At the initial time of each TL plane, the TL-DVFS initializes the TL plane (Algorithm 2) and computes the frequency scaling factor of a multi-core processor (Algorithm 1). When a sporadic job is released in a TL plane, the TL-DVFS can handle the event A to satisfy the requirement of workload augment, so as to obtain energy-saving under the condition of meeting the feasibility of sporadic jobs (Algorithm 3 and 4). The TL-DVFS adopts the same method as Funk and Nadadur [2009] to handle the event B or C. Once the initialization or an event handler is finished, the TL-DVFS will determine the frequency of each core. Furthermore, cores execute the assigned jobs at the selected frequency.

This paper considers the sporadic jobs whose deadlines equal to the period, and the absolute deadlines equal to the end time of a TL plane. Therefore, given any TL plane and any sporadic task $T_i$, at most one job of $T_i$ overlaps the TL plane. In other words, because all energy-efficient scheduling strategies are performed within a TL plane, the scheduled job always overlaps the current TL plane, which does not conflict with which job of each task scheduled for execution.

## 5.6. Algorithm Analysis

We formally prove that the TL-DVFS is optimally feasible through the following theorems.

THEOREM 5.8. *When $n \le m$, a sporadic task set is always locally feasible within each TL plane by the TL-DVFS.*

PROOF. Proof by contradiction. Assume that when $n \le m$, a sporadic task set is local unfeasible within a certain TL plane by the TL-DVFS. According to Theorem 5.3, at least one effective critical moment occurs in this TL plane. As a result, at least one task is not selected to run, which contradicts our assumptions since all tasks are selected by the TL-DVFS to run. □

When the number of tasks is less than that of a processor's cores, we assume that each job will be completed at the cost of the worst-case execution time. In this setting, the TL-DVFS selects all tasks and executes them until their local remaining execution times become zero. Meanwhile, whenever an event B occurs, the number of active tasks is decreasing and the frequency scaling factor remains unchanged. However, when an event A occurs, the frequency scaling factor will be adjusted according to the additional active tasks. In this case, the number of events of type B or A is at most $n$, since it cannot exceed the number of tasks. Note that an event C never happens when $n \le m$ since all tasks are selected to run and there are no idle tasks.

Now, we discuss the local feasibility when $n > m$.

THEOREM 5.9. *When $n > m$, a sporadic task set is always locally feasible within each TL plane by the TL-DVFS.*

PROOF. Proof by induction. It is known that the sporadic task set is scheduled by the TL-DVFS only when a TL plane is initialized or at the time when an event A, B, or C occurs. This basically shows that if the current time $t$ is $t_0$, we can derive from Corollary 5.5 that $R'_{t_0} \le m$, where $R'_{t_0}$ is the total effective local utilization of $Active(t_0)$ task set. If $t$ is the time when an event B or C occurs, then $\forall\Delta$, $0 \le \Delta \le t - t_0$, $R'_t \le R'_{t_0+\Delta} \le R'_{t_0}$ by Theorem 5.6. If $t$ is the time when an event A occurs, according to Theorem 5.7, we have $R'_t \le m$ if $R'_{t_0} \le m$.

Therefore, at any time during a TL plane the total effective local utilization is no more than $m$ by the TL-DVFS. When $R'_t$ is less than $m$ for any time $t$, there should be no effective critical moment in the TL plane, according to the contraposition of Corollary 5.4. By Theorem 5.3, no effective critical moment implies that all sporadic tasks are locally feasible. □

When $n > m$ and $R'_{t_0}$ is less than $m$ at the initial time $t_0$, no any effective critical moment happens at an event B, C, and A , according to Theorem 5.9. Whenever an event B happens, the number of non-selected tasks decreases until $m$ tasks remain. As such, according to Theorem 5.8, all tasks are selected such that they finish their local remaining executions at the end time of the TL plane with consecutive event B's.

Recall that we consider the TL-DVFS's local feasibility in each TL plane, and the objective of the TL-DVFS is to complete all tasks before their deadlines.

THEOREM 5.10. *Any sporadic task set with $U \le m$ and $u_{max} \le 1$ will be scheduled to meet all deadlines by the TL-DVFS.*

PROOF. We prove this theorem using the induction method. With adjacent TL planes, if $U \le m$ and $u_{max} \le 1$ is available to any sporadic task set, then all tasks are locally feasible in the first TL plane based on Theorems 5.8 and 5.9. The initial $R'_{t_0}$ for the next adjacent TL plane is less than $m$ according to Theorem 5.1 and Corollary 5.5; hence, according to the inductive hypothesis, the TL-DVFS guarantees the local feasibility for each TL plane, which makes all tasks satisfy their deadlines. □

## 6. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our algorithm on a multi-core processor with the global DVFS technology. We use the C language to implement a discrete event simulator in Linux 2.6. A range of energy-efficient real-time algorithms are evaluated, including the LRE-TL [Funk and Nadadur 2009], the Uniform RT-SVFS [Funaoka et al. 2008], the TL-DVFS, and the DVS-EDF [Huang et al. 2009]. The experimental results are normalized against those of the LRE-TL.

The LRE-TL, Uniform RT-SVFS, and TL-DVFS are all based on the optimal real-time scheduling approaches; thus, their Feasibility Performance (FP) is the same. The FP refers to the ratio of feasible tasks to all tasks. In this case we compare their Normalized Energy Consumption (NEC), which refers to the normalized average energy consumption of schedulable tasks. However, the DVS-EDF is based on the EDF which is a non-optimal global scheduling algorithm. In this case, the NEC is not a fair metric; hence, we perform experiments from three aspects. Firstly, their energy consumption is compared. Secondly, we compare their FPs. Finally, the ratio of FP to NEC is used as a metric to make a fair comparison among these algorithms. Clearly, the algorithm with a higher FP and lower NEC will be the best algorithm.

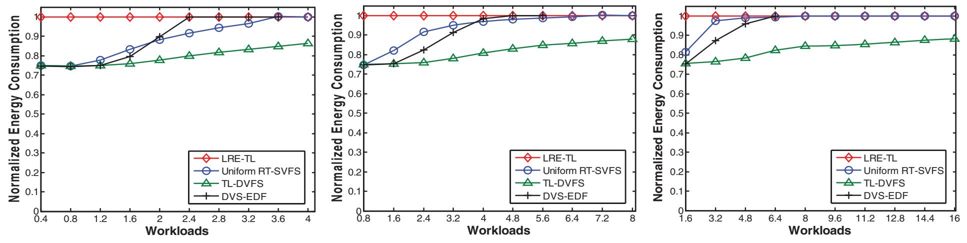
### 6.1. Parameters and Performance Metrics

We use the processor power model, described in Section 3, to evaluate the power consumption of the proposed algorithm, which is the same as the literature [Huang et al. 2009] did. We assume that the processor supports discrete voltage levels in steps of 0.05V in the range from 0.5V to 1.0V [Jejurikar et al. 2004]. These voltage levels correspond to discrete frequency scaling factors and each of which is mapped to the smallest discrete level greater than or equal to it.

To evaluate the performance of all those algorithms, all experiments adopt the parameters as in the following: $u_{sys}$, $m$, and $n$. Here, $u_{sys}$ denotes the system utilization, $m$ indicates the number of cores, while $n$ denotes the number of tasks.

We construct a set of 16 tasks [Qu 2007] which consists of both real benchmark applications and randomly generated ones. The former ones are extracted from the FFT1 and FFT2 (Fast Fourier Transform), Laplace (Laplace transform) and Karp10 (Karplus-Strong music synthesis algorithm with 10 voices). The latter ones are obtained from the TGFF suite [Dick et al. 1998]. For each task, there is a makespan that indicates the lifetime of a task and is in general equivalent to the task's minimum release interval time. The makespan for each task used in our simulation is obtained from Hua et al. [2006]. Due to space limitations we refer the reader to Table 1 in Qu [2007] for more details.

For each parameter setting, we test 100,000 task sets each of which is randomly selected from the set of 16 tasks. Assume that the deadline of each task is equal to the minimum release interval time. The system utilization $u_{sys}$ increases from 10% to 100% with 10% per step. The value of $m$ is set to 4, 8, and 16, respectively. Meanwhile, the corresponding value of $n$ is set to 10, 20, and 40, respectively. The total load of a task set is $Workloads = u_{sys} \times m$. The utilizations of all $n$ tasks are generated using the UUnifast algorithm [Bini and Buttazzo 2005; Davis and Burns 2009], discarding any task set with a task with $u_i > 1$. The UUnifast can generate an unbiased distribution for the $n$ task utilization values in the range $[0, Workloads]$. The minimum release interval time $P_i$ of each task is equal to its makespan. For each task $T_i$, the worst-case execution time is $C_i = u_i \times P_i$. We assume that the actual execution time of each task is equal to its worst-case execution time. For each sporadic task set, the simulation time is set to be 6,000,000ms (100minutes).



(a) $m = 4, n = 10$. (b) $m = 8, n = 20$. (c) $m = 16, n = 40$.

Fig. 3. Normalized energy consumption.

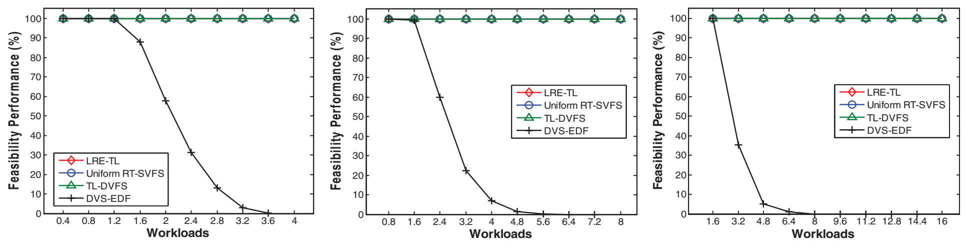
### 6.2. Evaluation Results

Figure 3 shows the normalized energy consumptions for sporadic tasks scheduled by the LRE-TL, Uniform RT-SVFS, TL-DVFS, and DVS-EDF algorithms under the same configurations. We can find from Figure 3(a) that for a quad-core processor and 10-task sets, the TL-DVFS always outperforms other algorithms, irrespective of how the total utilization changes.

On the one hand, when the system utilization is less than 0.3, the energy-saving of the TL-DVFS is as much as that of the DVS-EDF. Both of such algorithms achieve more energy-saving than the Uniform RT-SVFS. The root cause is that the Uniform RT-SVFS is a static method to implement the voltage and frequency scaling for all tasks. During the runtime the voltage and frequency will not be changed, consequently, the dynamic workload resulting from the release of sporadic tasks cannot be well balanced. The TL-DVFS and DVS-EDF allow task migrations and can achieve a well-balanced workload among cores. On the other hand, when the system utilization is more than 0.3, the energy-saving of the Uniform RT-SVFS and DVS-EDF decreases continuously while the energy-saving of the TL-DVFS reduces smoothly from about 25 percent and steadily up to about 15 percent. This is because when the workload increases, all cores will start running to guarantee the feasibility of all tasks. However, the Uniform RT-SVFS is a static method and the DVFS-EDF is a non-optimal global EDF scheduling algorithm that allows some task migrations. The two algorithms can not achieve a well-balanced workload among cores and are not suitable to the varied workloads. The TL-DVFS not only implements the voltage and frequency scaling for all active tasks at the initial time of each TL plane, but also takes the arrival event of sporadic tasks into account; hence, it can realize a well-balanced workload among cores through job migrations. Consequently, although the change of workloads has a little effect on the TL-DVFS algorithm, the TL-DVFS can still obtain more energy-saving compared to other algorithms.

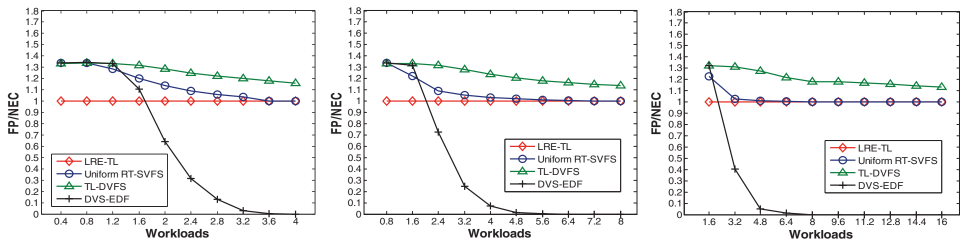
Figures 3(b) and 3(c) illustrate the NECs for a 8-core processor with a set of 20 tasks and a 16-core processor with a set of 40 tasks, separately. The experimental results are similar to that as shown in Figure 3(a). We can conclude that the TL-DVFS always outperforms other algorithms. Additionally, the energy-saving of the TL-DVFS decreases tardily from about 25 percent, steadily up to about 15 percent in the case of high workloads.

Note that in Figures 3(a), 3(b) and 3(c), the NEC of the DVS-EDF is nonexistent when the system utilization is more than 0.5, for example, when the sum of all task workloads is 4, 6.4 and 8 for 4-core, 8-core and 16-core processors, respectively. This is because the DVS-EDF implements the global scheduling based on the EDF which takes account of the average energy consumption of schedulable tasks. Thus, in the case of non-feasibility, the NEC cannot be determined, as shown in Figure 4.



(a) $m = 4, n = 10$. (b) $m = 8, n = 20$. (c) $m = 16, n = 40$.

Fig. 4. Feasibility performance.



(a) $m = 4, n = 10$. (b) $m = 8, n = 20$. (c) $m = 16, n = 40$.

Fig. 5. Ratio of feasibility performance to normalized energy consumption (FP/NEC).

With the same configuration used in Figure 3, Figure 4 plots the feasibility of all algorithms. We can see that both the TL-DVFS and the Uniform RT-SVFS guarantee the optimal feasibility while the feasibility of DVS-EDF decreases with the increasing total workloads. This is because that the TL-DVFS and the Uniform RT-SVFS implement the task scheduling based on the LRE-TL that is an optimal real-time scheduling algorithm for sporadic tasks, while the DVS-EDF is based on the EDF which is non-optimal [Baker 2005]. Furthermore, we find that the TL-DVFS can not only achieve more energy-saving, but also guarantee the optimal feasibility of sporadic tasks, as shown in Figures 4(a), 4(b) and 4(c).

In Figures 3 and 4, the DVS-EDF takes account of the average energy consumption of schedulable tasks but ignores that of unfeasible tasks. The TL-DVFS and Uniform RT-SVFS, however, consider both the two types of tasks. Therefore, it is not fair to measure which is better simply using the metric NEC or FP; hence, we need to design a new metric that takes both the NEC and FP into consideration.

Consequently, the ratio of feasibility to normalized energy consumption is defined as the performance metric which results in a fair measure. In this case, the experiments are performed in the same settings of Figure 4, and the experimental results are depicted in Figure 5. As shown in Figure 5(a), the TL-DVFS always outperforms other algorithms in terms of FP/NEC, irrespective the total utilization of tasks. Furthermore, the FP/NEC of the TL-DVFS decreases tardily from about 1.3 and is stably maintained at about 1.2 as the increase of the total workloads. This is because that the TL-DVFS not only guarantees the optimal feasibility but also achieves more energy-saving. For the Uniform RT-SVFS, its FP/NEC is close to that of the LRE-TL when the total workload increases. This is in keeping with the features of the corresponding static energy-efficient real-time scheduling. For the DVS-EDF, its FP/NEC is close to that of the TL-DVFS when the system utilization is less than 0.3, while the value of FP/NEC decreases rapidly when the system utilization is more than 0.3. This is because that the feasibility of the DVS-EDF decreases along with its energy-saving when the workload is high. Figures 5(b) and 5(c) illustrate the similar trend as shown in Figure 5(a), which implies that the TL-DVFS always achieves higher FP/NEC than other algorithms, even in the case of high workloads.

## 7. CONCLUSIONS

For multi-core processors with the global DVFS and DPM technologies, this paper proposes TL-DVFS, which is an energy-efficient real-time scheduling algorithm based on LRE-TL. TL-DVFS utilizes the concept of the TL plane to dynamically scale the voltage and frequency of the multi-core processor at the initial time of each TL plane or at the release time of a sporadic task in each TL plane. Consequently, TL-DVFS can obtain a reasonable tradeoff between real-time constraint and energy-saving while meeting the constraints for the optimal feasibility of sporadic tasks. TL-DVFS is also adaptive to variable workloads due to the dynamic release of sporadic tasks; hence, it can obtain greater energy-saving. Experimental results show that compared to existing algorithms, TL-DVFS not only guarantees the optimal feasibility of sporadic tasks, but also achieves more energy-saving.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers and Dr Cong Liu for their constructive comments.

## REFERENCES

ACPI 2011. ACPI. http://www.acpi.info.

AlEnawy, T. and Aydin, H. 2005. Energy-aware task allocation for rate monotonic scheduling. In *Proceedings of the IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05)*. IEEE, Los Alamitos, CA, 213–223.

Aydin, H. and Yang, Q. 2003. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the 17th IEEE International Parallel and Distributed Processing Symposium*. IEEE, Los Alamitos, CA, 22–26.

Baker, T. P. 2005. An analysis of EDF schedulability on a multiprocessor. *IEEE Trans. Paral. Distrib. Syst. 16,* 8, 760–768.

Bautista, D., Sahuquillo, J., Hassan, H., Petit, S., and Duato, J. 2008. A simple power-aware scheduling for multicore systems when running real-time applications. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'08)*. IEEE, Los Alamitos, CA, 1–7.

Bini, E. and Buttazzo, G. 2005. Measuring the performance of schedulability tests. *Real-Time Syst. 30,* 1-2, 129–154.

Chandrakasan, A., Sheng, S., and Brodenson, R. W. 1992. Low-power cmos digital design. *IEEE J. Solid-State Circuit 27,* 4, 473–484.

Chen, J., Hsu, H., and Kuo, T. 2006. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the IEEE Real-time and Embedded Technology and Applications Symposium*. IEEE, Los Alamitos, CA, 408–417.

Cho, H., Ravindran, B., and Jensen, E. 2006. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE Real-Time System Symposium (RTSS'06)*. IEEE, Los Alamitos, CA, 101–110.

Davis, R. and Burns, A. 2009. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Proceedings of Real-Time Systems Symposium*. IEEE, Los Alamitos, CA, 398–409.

Devadas, V. and Aydin, H. 2010. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Proceedings of the International Conference on Green Computing (GREENCOMP'10)*. IEEE, Los Alamitos, CA, 61–72.

Dick, R., Rhodes, D., and Wolf, W. 1998. Tgff: Task graphs for free. In *Proceedings of the IEEE International Workshop Hardware / Software Codesign*. IEEE, Los Alamitos, CA, 97–101.

Dorsey, J. 2007. An integrated quad-core opteron processor. In *Proceedings of the IEEE International Solid State Circuits Conference (ISSCC'07)*. IEEE, Los Alamitos, CA, 102–103.

Fisher, N., Goossens, J., and Baruah, S. 2010. Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real-Time Syst. 45,* 1-2, 26–71.

Funaoka, K., Kato, S., and Yamasaki, N. 2008. Energy-efficient optimal real-time scheduling on multiprocessors. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC'08)*. IEEE, Los Alamitos, CA, 23–30.

Funk, S. and Nadadur, V. 2009. Lre-tl: An optimal multiprocessor algorithm for sporadic task sets. In *Proceedings of the Symposium on Real-Time and Network Systems (RTNS'09)*. 159–168.

Herbert, S. and Marculescu, D. 2007. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, Los Alamitos, CA, 38–43.

Holman, P. and Anderson, J. 2005. Adapting pfair scheduling for symmetric multiprocessors. *J. Embed. Comput. 1,* 4, 543–564.

Hua, S., Qu, G., and Bhattacharyya, S. 2006. Energy-efficient embedded software implementation on multiprocessor system-on-chip with multiple voltages. *ACM Trans. Embed. Comput. Syst. 5,* 2, 321–341.

Huang, X., Li, K., and Li, R. 2009. A energy efficient scheduling base on dynamic voltage and frequency scaling for multi-core embedded real-time system. In *Proceedings of ICA3PP*. Lecture Notes in Computer Science, vol. 5574, Springer, Gemany, 137–145.

Intel 2011. Intel i7 processor specifications. http://www.intel.com/products/processor/corei7/specifications.htm.

Jejurikar, R., Pereira, C., and Gupta, R. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the Design Automation Conference*. IEEE, Los Alamitos, CA, 275–280.

Kim, W., Gupta, M., Wei, G., and Brooks, D. 2008. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture (HPCA'08)*. IEEE, Los Alamitos, CA, 123–134.

Kumar, R. and Hinton, G. 2009. A family of 45nm ia processors. In *Proceedings of the IEEE International Solid State Circuits Conference (ISSCC'09)*. IEEE, Los Alamitos, CA, 58–59.

Lee, C. and Shin, K. 2004. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS'04)*. IEEE, Los Alamitos, CA, 319–335.

Martin, S., Flautner, K., Mudge, T., and Blaauw, D. 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the Conference on Computer Aided Design*. IEEE, Los Alamitos, CA, 721–725.

McCreary, H., Broyles, M., Floyd, M., et al. 2007. Energyscale for ibm power6 microprocessor based systems. *IBM J. Res. Devel. 21,* 6, 775–786.

McGowen, R., Poirier, C., Bostak, C., et al. 2006. Power and temperature control on a 90-nm itanium family processor. *IEEE J. Solid-State Circ. 37,* 8, 229–237.

Mosley, L. 2008. Power delivery challenges for multicore processors. In *Proceedings of CARTS*.

Naveh, A. 2006. Power and thermal management in the intel core duo processor. *Intel Techn. J. 10,* 2.

Pillai, P. and Shin, K. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems (SOSP'01)*. ACM, New York, NY, 89–102.

Qu, G. 2007. Power management of multicore multiple voltage embedded systems by task scheduling. In *Proceedings of the IEEE International Conference on Parallel Processing Workshops (ICPPW'07)*. IEEE, Los Alamitos, CA.

Rele, S., Pande, S., Onder, S., et al. 2002. Optimizing static power dissipation by functional units in superscalar processors. In Lecture Notes in Computer Science, vol. 2304. Springer, 85–100.

Seo, E., Jeong, J., Park, S., and Lee, J. 2008. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parall. Distrib. Syst. 19,* 11, 1540–1552.

Yang, C., Chen, J., and Kuo, T. 2005. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *Proceedings of the ACM/IEEE Conference of Design, Automation, and Test in Europe (DATE'05)*. ACM/IEEE, 468–473.

Yang, C., Chen, J., Kuo, T., and Thiele, L. 2009. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In *Proceedings of the ACM/IEEE Conference of Design, Automation, and Test in Europe (DATE)*. ACM/IEEE, 694–699.

Received July 2011; revised November 2011; accepted January 2012