GLOBAL SCHEDULING ON TEMPERATURE-CONSTRAINED MULTIPROCESSOR REAL-TIME SYSTEMS

A Thesis

by

JA RYEONG KOO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2008

Major Subject: Computer Science

GLOBAL SCHEDULING ON TEMPERATURE-CONSTRAINED MULTIPROCESSOR REAL-TIME SYSTEMS

A Thesis

by

JA RYEONG KOO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2008

Major Subject: Computer Science

ABSTRACT

Global Scheduling on Temperature-Constrained Multiprocessor Real-Time Systems.

(May 2008)

Ja Ryeong Koo, B.S., Kyunghee University

Chair of Advisory Committee: Dr. Riccardo Bettati

In this thesis, we study temperature-constrained multiprocessor real-time systems, where real-time guarantees must be met without exceeding safe temperature levels within the processors. We focus on Pfair scheduling algorithms, especially ERfair scheduling scheme (a work-conserving extension to Pfair scheduling) as our main multiprocessor real-time scheduling methodology. Then, we study the benefits of simple reactive speed scaling as described in the real-time multiprocessor systems. In this thesis, in support of the temperature-awareness, we extend the applicability of the reactive speed scaling to global scheduling schemes for multiprocessors. We propose temperature-aware scheduling and processor selection schemes motivated by existing (thermally non-optimal) ERfair scheduling in order to reduce thermal stress and therefore increase the processor utilization. Then, we show that the proposed algorithm and reactive scheme can enhance the processor utilization compared with any constant speed scheme on real-time multiprocessor systems. Additionally, we show how the *maximum schedulable utilization* (MSU) for partitioning heuristics can be determined on the temperature-constrained multiprocessor real-time systems.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This chapter introduces an overview of my research and contains *Thermal Issues and Speed Scaling*, *Overview of Real-Time Scheduling on Multiprocessors*, and *Organization.*

A. Thermal Issues and Speed Scaling

In recent years, power density in processors has increased exponentially [1]. Very high power density leads to power and temperature dissipation. Processors tend to easily overheat given the large energy consumption. It is generally assumed that most electronic failures are temperature-related because the circuit reliability is exponentially dependent on the operating temperature [2].

Power dissipation is often a particularly important issue in high-performance embedded systems, where packaging requirement may render active dissipation, such as fans, not practicable. A solution in such systems is to resort to speed scaling, where the power consumption is (at least transiently) reduced by decreasing the processing speed. Recently, a number of approaches have been developed for real-time systems that are based on dynamic speed scaling in a variety of settings. Wang *et al.* [3,4], for example, have stated that temperature-constrained hard real-time scheduling must guarantee real-time constraints within safe temperature levels of the processors. They proceed to show in [3] that the processor utilization can be improved by simple reactive speed scaling, compared with any constant speed scheme. Ferreira *et al.* [5] have developed models for dynamic thermal management in server farms and have shown

The journal model is *IEEE Transactions on Automatic Control.*

how these efforts address dynamic thermal management through speed scaling either for single processor systems [3,4] or for potentially large collections of single processor systems [5].

In this thesis, we plan to study the benefits of reactive speed scaling as it is applied in real-time multiprocessor systems. We will be studying how to optimally allocate and schedule real-time workload in a type of multiprocessor system: systems where processors are largely thermally independent from each other (i.e., typically, systems with discrete, single core processors). We assume that our scheme will be able to be applied to another type of multiprocessor system by simply changing the way to allocate the workload: multiprocessors with inter-processor thermal effects (i.e., typically system-on-chip or multi-core systems). This type of multiprocessor system has been studied in [6].

B. Overview of Real-Time Scheduling on Multiprocessors

In general, there have been two main approaches for scheduling on multiprocessors depending on whether task migration is allowed or not: *partitioning* and *global scheduling*. Between them, we focus on global scheduling schemes in that they are more efficient and flexible than partitioning heuristics. We will treat these more specifically in Chapter II.

Recently, there have been many global scheduling schemes (including Pfair scheduling algorithms [7–11]) and several variants for real-time scheduling on multiprocessors. Among them, the Pfair scheduling algorithms have proved to be optimal for periodic tasks in a large number of multiprocessor settings. In this thesis, we focus on *ERfair scheduling algorithm* [11] which is a work-conserving extension to Pfair scheduling. Main reason for this is discussed in Chapter II.

The Pfair and other multiprocessor algorithms do not take thermal management into consideration. For example, current Pfair scheduling algorithms only consider task urgency in their scheduling decisions and processor availability in the task allocation. Consequently, situations where some processors are suffering from intensive thermal stress and the system unduly has to resort to speed scaling in order to maintain temperature constraint can occur.

The objective of this thesis is to make thermally non-optimal Pfair scheduling algorithms *temperature-aware ones.* As mentioned above, existing Pfair scheduling schemes do not consider thermal issues. Specifically, the Pfair scheduling algorithms assume the schedule of periodic tasks with relative deadlines equal to their periods. Through [3], when $D_i = \delta \cdot P_i$ (where $\delta = 1, D_i$ is relative deadline, δ is deadline ratio, and P_i is a period.), we see that for **MSU** (Maximum Schedulable Utilization), reactive speed scaling has no benefit in comparison with constant speed scaling. That is because there cannot be idle time during each period if the processor is fully-utilized. We specifically deal with this in Chapter IV.

For the temperature-awareness, first of all, this thesis will extend the work of [3] and study the applicability of reactive speed scaling to multiprocessors. Accordingly, as our task model, we consider two different types of sets of identical periodic tasks (one is $D_i = \delta \cdot P$ and the other is $D_i = \delta_i \cdot P$. (here, for all δ, $0 < \delta \leq 1$)).

As a result, this thesis shows how **MSU** (Maximum Schedulable Utilization) can be improved on *multiprocessor real-time systems using classical ERfair scheduling scheme* and *temperature-constrained multiprocessor real-time systems using our temperature-aware ERfair scheduling algorithm.*

C. Organization

In Chapter II, we summarize the previous works related to our research and background concepts. Then, in Chapter III, we study multiprocessor schedules of tasks with early-deadlines. We describe our thermally aware multiprocessor scheduling and perform schedulability tests on the temperature-constrained multiprocessor real-time systems in Chapter IV. In Chapter V, we compare partitioning heuristics with our thermally-aware ERfair scheduling scheme through **MSU**. Finally, in Chapter VI, we conclude our work with final remarks.

CHAPTER II

BACKGROUND AND RELATED WORK

In the following, we give a short overview of the state of the art in real-time multiprocessor scheduling with a focus on Pfair scheduling. I will then proceed to give an overview of thermal models for embedded systems and describe our approach to dynamic thermal management. We conclude this chapter with our task model.

A. Real-Time Multiprocessor Scheduling

Real-time scheduling on multiprocessors makes use of two major approaches depending on whether task migration is permitted: *Partitioning* and *Pfair scheduling* [7].

1. Partitioning

In partitioning schemes [12,13], each processor has its local ready queue and schedules tasks independently. When a new task arrives, the task is assigned to one of the available processors through one of several heuristics (non-optimal) prior to run-time. Then, at run-time the assigned task is scheduled and executed such as EDF [14].

With the methodology above, even though partitioning tends to perform well in practice in that it causes relatively less overhead than global scheduling scheme and it reduces a multiprocessor scheduling problem to a set of uniprocessor problems, it has two critical disadvantages. First, assigning tasks optimally to processors is a bin-packing problem, which has been known to be NP-hard [15]. Thus, tasks are generally partitioned by non-optimal heuristics. Second, task sets exist that are schedulable only if tasks are not partitioned. For example, a task set consisting of three tasks which are each with an execution requirement of 2 and a period of 3 cannot be scheduled on two processors without task migration.

Consequently, in this thesis, we focus on Pfair scheduling in that it provides enhanced schedulability and flexibility [8].

2. Proportionate-Fair (Pfair) Multiprocessor Scheduling

Here, we briefly introduce Pfair scheduling algorithms [7] and summarize related previous works.

Assume that there is a set Γ of N Tasks ($\{\Gamma_i : i = 1, 2, ..., N\}$) to be scheduled on M identical processors. Each task Γ_i has an integer period P_i, an integer execution unit C_i and a weight w_i (which is C_i/P_i) in range $[0, 1)$. Specifically, w_i is the rate at which Γ_i must be executed. Pfair scheduling algorithms allocate processor time in *quanta* which are discrete time unit. We say that *slot t* indicates the quantum-length time interval $[t, t+1)$, where $t \geq 0$. Time t is the beginning of *slot t*. For each slot, each task can be assigned to a different processor (i.e., task migration is permitted), but not to more than one processor in the same slot (i.e., task parallelism is not allowed).

In an ideal schedule, a Task Γ_i in Γ should receive a share of $w_i \cdot t$ time units over the interval $[0, t)$. Unfortunately, in practice, the ideal schedule (i.e., perfectly fair schedule) is impractical. Pfair scheduling uses the notion of *lag* as the formalized concept of tracking the ideal schedule. Formally, the *lag* of Task Γ_i at time t is defined as the following:

$$lag(\Gamma_i, t) = w_i \cdot t - allocated(\Gamma_i, t) \tag{2.1}$$

where $w_i \cdot t$ means the allocation that Task Γ_i should receive in an ideal schedule and $allocated(\Gamma_i, t)$ is the allocation of Task Γ_i in the Pfair schedule. A schedule is defined to be *Pfair* if and only if the size of all task lags is strictly less than one, i.e.,

$$-1 < lag(\Gamma_i, t) < 1 \tag{2.2}$$

where $i = 1, 2, ..., N$.

The execution of each task Γ_i corresponds to the execution of an infinite sequence of quantum-length *subtasks* (subjobs), $\Gamma_{i,1}$, $\Gamma_{i,2}$, ..., $\Gamma_{i,j}$. Each subtask has its own *pseudo-release* $r(\Gamma_{i,j})$ and a *pseudo-deadline* $d(\Gamma_{i,j})$, which can be formally defined as the follows:

$$r(\Gamma_{i,j}) = \left\lfloor \frac{j-1}{w_i} \right\rfloor \quad \wedge \quad d(\Gamma_{i,j}) = \left\lceil \frac{j}{w_i} \right\rceil \tag{2.3}$$

To satisfy (2.2), Subtask $\Gamma_{i,j}$ has to be scheduled in the interval $W(\Gamma_{i,j}) = [r(\Gamma_{i,j}), d(\Gamma_{i,j}))$, called its *window*. Figure 1 describes the windows of the first job of a periodic task Γ_i with weight 6/14. In this figure, the first job is composed of six subtasks $\Gamma_{i,1}, ..., \Gamma_{i,6}$, each of which must be scheduled within its window. Otherwise, a lag-bound violation occurs by (2.2). For instance, Subtask $\Gamma_{i,1}$ must be scheduled at one of the three slots (slot 0, slot 1, and slot 2). The pattern of these windows is repeated for every job.

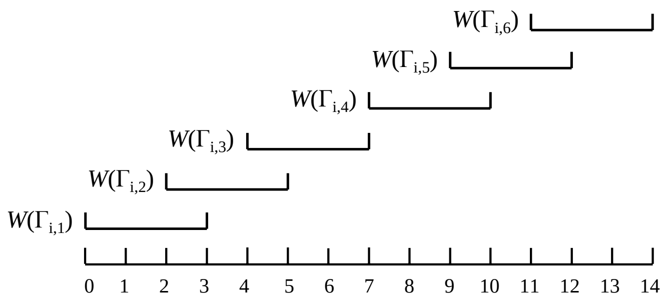


Fig. 1. Windows of the first job of Γ_i with weight 6/14

Baruah *et al.* [9] shows that a periodic task set $\{\Gamma_i : i = 1, 2, ..., N\}$ has a *Pfair* *schedule* on M processors if and only if

$$\sum_{i=1}^{N} \frac{C_i}{P_i} \leq M \tag{2.4}$$

So far, there have been PF [7], PD [9], PD^2 [10] as the only known optimal Pfair scheduling algorithms. These algorithms prioritize subtasks on an *Earliest Pseudo-Deadline First* basis (EPDF) [7] (i.e., a subtask with an earlier pseudo-deadline has a higher priority than one with a later deadline.), but differ only in their tie-breaking rules. For both synchronous and asynchronous periodic task systems, PD^2 is the most efficient Pfair scheduling algorithm of the three and its two tie-breaking rules incorporate those of the other two algorithms in a more comprehensive category [8,11]. For this reason, we assume use of the priority definition of PD^2 as a way to prioritize given tasks.

As one of extensions to Pfair scheduling, for increased flexibility of Pfair scheduling, Anderson *et al.* [11] introduced a work-conserving variant of Pfair scheduling named *Early-release* fair (*ERfair*) scheduling, and proved that the early-release version of PD^2 is also optimal for scheduling periodic tasks on multiprocessors. Note that under the ERfair scheduling, no processor is left idle as long as M unfinished jobs (not subtasks) exist. This makes PD^2 scheduling algorithm a work-conserving version, and thereby ERfair scheduling scheme can be considered a variant of Pfair scheduling algorithm which guarantees the *shortest response time* for all jobs among all other Pfair variants. Under ERfair scheduling, if two subtasks are parts of the same job, then the second subtask becomes eligible for execution immediately after execution of the first subtask completes. Consequently, since in this thesis, we analyze critical instance and busy interval which can be generated by a work-conserving scheduler, we mainly assume use of this ERfair version [11] of PD^2 Pfair scheduling algorithm defined in [10] as our *task selection scheme.* The following Subsection shows this variant [11] using PD^2 priority definitions [10] more specifically.

Early-Release Fair Scheduling (ERfair Scheduling)

In this section, we introduce Early-Release fair scheduling algorithm (ERfair scheduling algorithm) [11].

ERfair scheduling can be obtained by simply dropping the -1 lag constraint from Equation (2.2). Formally, a schedule is *ERfair* if and only if

$$lag(\Gamma_i, t) < 1, \tag{2.5}$$

where there is a set Γ of N tasks ($\Gamma_i : i = 1, 2, ..., N$).

Although ERfair scheduling version is a variant of PD^2 [10,11], it can easily adapt other Pfair scheduling schemes such as PF [7] and PD [9] as well. As mentioned in the previous section, we only consider ERfair scheduling scheme which uses the priority definition of PD^2 as underlying *task selection algorithm.* Under PD^2, Subtask $\Gamma_{i,j}$'s priority is higher than that of Subtask $\Gamma_{p,q}$, if one of the following three rules is satisfied [10]:

$$\begin{cases} \bullet\ d(\Gamma_{i,j}) < d(\Gamma_{p,g}), \\ \bullet\ d(\Gamma_{i,j}) = d(\Gamma_{p,q}) \ \text{ and } \ b(\Gamma_{i,j}) > b(\Gamma_{p,q}), \\ \bullet\ d(\Gamma_{i,j}) = d(\Gamma_{p,q}),\ \ b(\Gamma_{i,j}) = b(\Gamma_{p,q}) = 1, \\ \quad \text{and } \ D(\Gamma_{i,j}) \geq D(\Gamma_{p,q}). \end{cases} \tag{2.6}$$

where $d(\Gamma_{i,j})$, $b(\Gamma_{i,j})$, and $D(\Gamma_{i,j})$ are a *pseudo-deadline*, a *successor bit*, and a *group-deadline* of Subtask $\Gamma_{i,j}$, respectively.

The *successor bit* is a parameter for tie-breaking to be used when multiple subtasks have same pseudo-deadlines. Formally, the successor bit of Subtask $\Gamma_{i,j}$ is defined as follows.

$$b(\Gamma_{i,j}) = \left\lceil \frac{j}{w_i} \right\rceil - \left\lfloor \frac{j}{w_i} \right\rfloor \tag{2.7}$$

Therefore, $b(\Gamma_{i,j})$ is either 0 or 1. Informally, $b(\Gamma_{i,j})$ indicates the number of slots by which $\Gamma_{i,j+1}$'s window overlaps $\Gamma_{i,j}$'s window.

The *group-deadline* is another parameter for the tie-breaking. To motivate the definition of the group deadline, let me consider a sequence $\Gamma_{i,1}, ..., \Gamma_{i,j}$ of subtasks such that $b(\Gamma_{i,k}) = 1$ and the size of windows of $\Gamma_{i,k}$ is 2 for all $1 \leq k < j$. In this case, scheduling $\Gamma_{i,k}$ in the last slot on its window will force the other subtasks in this sequence to be scheduled in their last slots. The group deadline of Subtask $\Gamma_{i,k}$ is the earliest time by which such a *cascade* must end. More detailed description for these parameters can be found in [10].

With regard to the above rules, any ties not resolved by these three rules can be broken arbitrarily. Thus, for example, according to the definition (2.6), $\Gamma_{i,j}$'s priority is higher than that of $\Gamma_{p,q}$ if it has an earlier pseudo-deadline. If $\Gamma_{i,j}$ and $\Gamma_{p,q}$ have equal pseudo-deadlines, but $b(\Gamma_{i,j}) = 1$ and $b(\Gamma_{p,q}) = 0$, then the tie is broken in favor of $\Gamma_{i,j}$. Finally, if $\Gamma_{i,j}$ and $\Gamma_{p,q}$ have equal pseudo-deadlines and successor bits of one, then their group deadlines will be inspected to break the tie.

ERfair version [11] of the above priority definition contains additional policies to make early-release fair scheduling feasible. Briefly speaking, under the ERfair version of the above rules, a job that has not completed execution is always eligible for scheduling during its period under ERfair version [11]. Again, specifically, in ERfair scheduled system, if Subtask $\Gamma_{i,j-1}$ and $\Gamma_{i,j}$ are part of the same job, then $\Gamma_{i,j}$ becomes eligible for execution immediately after $\Gamma_{i,j-1}$ executes, which may be before $\Gamma_{i,j}$ Pfair window. If $\Gamma_{i,j-1}$ and $\Gamma_{i,j}$ are part of different jobs, then $\Gamma_{i,j}$ is inserted into an appropriate release queue for a future time slot.

The following Figure 2 shows how the original PD^2 (above) is different with ERfair version of the PD^2 (below). It is easy to see that ERfair scheduling is a work-

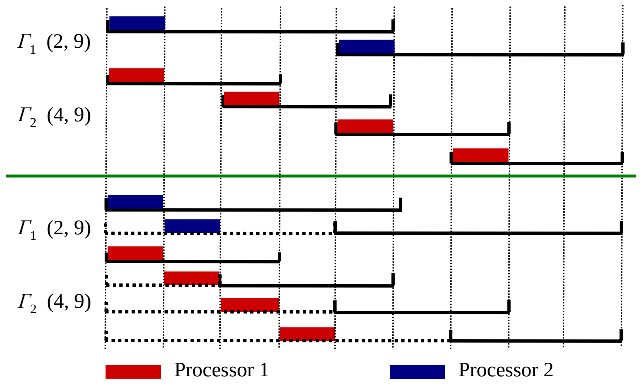


Fig. 2. Comparison between PD^2 schedule (above) and ERfair version of the PD^2 (below) of two tasks (Γ_1, Γ_2) on two processors

conserving extension to Pfair scheduling. That is because as long as at least M (here, M is 2) unfinished jobs exist, no processor is left idle under ERfair scheduling.

B. Thermal Model

In the last few years, several kinds of thermal models for processor have been proposed. HotSpot [16] is an example describing the thermal behavior at architecture-level, identifying the effects of hotspots more precisely.

In this paper, we will be using a basic chip-wide thermal model [17–20]. Although this model does not show fine-grained thermal behaviors, [16] states that it is likely to be suitable for study of chip-level techniques like speed-scaling. Moreover, current processors, such as the Intel Core Duo processor, have sophisticated hotspots, and the requirement of the fine-grained temperature modeling is relieved through accurate arrangement of sensors. This simple model can be used to derive more accurate models by more precisely modeling the power dissipation or by increasing the input power by a stochastic component, etc.

First of all, we define the *processor speed* (*frequency*), $s(t)$, at time t. Then we can represent the input power $P(t)$ at time t as follow:

$$P(t) = as^{\alpha}(t), \tag{2.8}$$

for some constant a and $\alpha = 1$. Usually, it is assumed that $\alpha = 3$ [17,18]. Most works in the literature assume the above relation between power and processor speed [17].

We define the temperature, $T(t)$, at time t. We apply the same approach in [17, 18] and adopt Fourier's Law of heat conduction which states that the cooling rate is proportionate to the difference in temperature between the environment and the object. Here, we assume that the environment has a defined temperature, and scale the temperature in order for the ambient temperature to be zero. Then we can represent the Fourier's Law as the follow [17,18]:

$$T'(t) = P(t) - bT(t), \tag{2.9}$$

where b is a positive constant that indicates the power dissipation rate. By (2.8) and (2.9), we can have the following formula:

$$T'(t) = as^{\alpha}(t) - bT(t), \tag{2.10}$$

where a and b are positive constants which we have addressed above, (2.8) and (2.9). Equation (2.10) is a classic linear differential equation. With an assumption that the temperature at time t_0 is T_0, i.e., $T(t_0) = T_0$, we can solve Equation (2.10) as the follow:

$$T(t) = \int_{t_0}^{t} as^{\alpha}(\tau)e^{-b(t-\tau)}d\tau + T_0e^{-b(t-t_0)}. \tag{2.11}$$

With (2.10) and (2.11), to appropriately scale the speed to control the temperature, we consider the observations from [3]. In the observations:

- If we want to keep the temperature constant at a value T_C during a time interval $[t_0, t_1]$, then for any $t \in [t_0, t_1]$, the speed at time t, $s(t)$, is set as the follow:

$$s(t) = (\frac{bT_C}{a})^{\frac{1}{\alpha}}. \tag{2.12}$$

- On the other hand, if we want to keep the speed constant, i.e., $s(t) = s_C$, during the same interval, then the temperature at time t, $T(t)$, can be developed as the follow:

$$T(t) = \frac{as_0^\alpha}{b} + (T(t_0) - \frac{as_C^\alpha}{b})e^{-b(t-t_0)}. \tag{2.13}$$

We will be using this relation between processor speed and temperature as the basis of our speed scaling scheme.

C. Reactive Speed Scaling

The purpose of temperature control is to keep the processor temperature within safe temperature level, and thereby not exceed the *highest-temperature threshold* T_H. This temperature threshold should be at a safe margin from the maximum temperature causing DTM events such as clock throttling. Temperature control must make sure that

$$T(t) \leq T_H. \tag{2.14}$$

The processor speed can be also set within some maximum speed, s_H, as the follow:

$$0 \leq s(t) \leq s_H. \tag{2.15}$$

However, without use of dynamic speed scaling, we must run the processor at some constant speed so that the temperature never exceeds T_H, i.e., $s(t) = s_C$, where s_C is constant. With Fourier's Law in (2.10), after the temperature first increases at some rate, eventually it will reach a steady-state, which is $T'(t) = as_C^\alpha - bT(t) = 0$ (i.e., from then, the temperature will remain constant). In this paper, we consider the *equilibrium speed* s_E defined in S. Wang *et al.* [3] as the maximum speed of s_C for constant-speed scaling, assuming that the initial temperature is less than T_H:

$$s_E = (\frac{b}{a} T_H)^{\frac{1}{\alpha}}. \tag{2.16}$$

Equation (2.16) means that if we use a constant processor speed s_E, the processor temperature does not exceed the threshold temperature T_H. Again, the speed s_E is the maximum speed for constant-speed scaling that keeps the processor within the safe temperature level.

A *dynamic speed scaling scheme* exploits the power dissipation during idle times. It uses periods where the processor is "cool" after idle periods to increase the speed temporarily and execute tasks at speeds higher than s_E. In this paper, we adopt a fully *reactive speed-scaling* technique defined in [3] to enhance the overall processor utilization. S. Wang *et al.* [3] defines the reactive speed-scaling scheme using the following formula:

$$s(t) = \begin{cases} s_H, & (W(t) > 0) \wedge (T(t) < T_H) \\ s_E, & (W(t) > 0) \wedge (T(t) = T_H) \\ 0, & W(t) = 0 \end{cases} \tag{2.17}$$

where $W(t)$ means the backlogged workload at time t.

This scheme makes sure that the temperature does not exceed the threshold temperature T_H in any case. By using speeds sometime that are significantly higher than s_E, we achieve processor utilizations than those for constant-speed scaling. As described in the formula above, the reactive speed scaling is simple: The processor runs at full speed s_H whenever backlogged workload exists and the temperature is below T_H. If the temperature hits T_H and the backlogged workload still exists, the processor run at the equilibrium speed s_E defined in (2.16). Whenever there is no backlogged workload, the processor will be idle (i.e., zero speed). Figure 3 describes an example of a single periodic task under reactive speed scaling. Note that the temperature at the beginning of each period increases as time goes by. With this effect, it is easy to see that for the same amount of workload in each period, the length of the execution at speed s_H is decreasing.

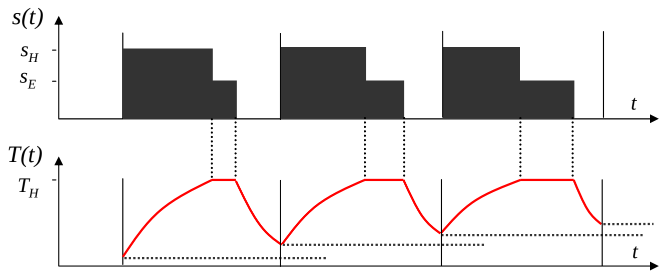


Fig. 3. Description of reactive speed scaling

In Chapter IV, we take advantage of reactive speed scaling and apply it to temperature-aware global scheduling on temperature-constrained multiprocessor real-time systems.

D. Task Model

We consider a set of identical-periodic tasks $\{\Gamma_i : i = 1, 2, ..., n\}$, where a Task Γ_i is characterized by two integer parameters - an *execution requirement* C_i defined for each task and an identical period P (i.e., $\Gamma_i = (P, C_i)$). We assume that each task has its own *deadline ratio* δ_i ($0 < \delta_i \leq 1$ for all i). Note that when $\delta_i = 1$ for all i ($i = 1, 2, ..., n$), reactive speed scaling has no benefit, as constant-speed scaling would be sufficient to generate a feasible schedule. Therefore, we will be focusing on tasks with relative deadlines less than their periods.

In our task model, a task generates a job at each integer multiple of the period P, and each such job has an execution requirement of C_i execution cycles, which must be completed by its relative deadline equal to or less than its period P.

Under ERfair scheduling, each job is broken into a series of quantum-length *subtasks*, and then each period P of a task splits up a sequence of *windows* of approximately equal lengths (potentially overlapping). In this paper, we assume that each job (likewise, each subtask) is independent in the sense that it does not affect other jobs of another task.

Additionally, we assume that both **job preemption and job migration are permitted** with *no penalty* [21]. Job preemption means that a job executing on a processor is preempted before completing its execution, and then is resumed later. In case of job migration, a job that has been preempted on a specified processor continues execution on a different processor. We also assume that there is **no task parallelism** [21], which means that each task (i.e., each subtask in the task) should be executed on at most one processor at any given instant in time. Note that under ERfair scheduling, a job consists of at least one subtask, and the job must execute without preemption or migration for at least duration of a subtask execution.

A given task set is said to be *feasible* on a given multiprocessor system if there exists a schedule for the task set in which all jobs of tasks in the task set meet their deadlines. We consider feasibility analysis of a task set on M multiprocessors in the absence of the *Integer Boundary Constraint* (IBC) defined in [21] as a particular kind of restriction on the preemptability of jobs. In [21], Baruah defined the *Integer*

Boundary Constraint (IBC), a limitation on the preemptability of jobs. For example, in the presence of the IBC, a job executing on a processor may be preempted if and only if it has finished an integer number of execution units on that processor. In this thesis, we consider real-time scheduling in *the absence of IBC* because the amount of cycles executed in a slot may not be integer under reactive-speed scaling. However, in the absence of IBC, a job must execute without preemption or migration for at least duration of a subtask execution.

Finally, in order to determine the worst-case delay for jobs in tasks, we first need to think through the worst-case arrival pattern, called *critical instance*. Since we must consider the additional temperature constraints, note that the analysis of the critical instance is different from the traditional one. In Chapter IV, we study how the critical instance can be determined for reactive speed scaling on temperature-constrained multiprocessor real-time systems when we use ERfair scheduling scheme.

CHAPTER III

MULTIPROCESSOR SCHEDULING OF TASKS WITH EARLY-DEADLINES

In this chapter, we consider multiprocessor real-time schedules of a set of *identical-periodic tasks* $\{\Gamma_i : i = 1, 2, ..., n\}$, where $\Gamma_i = (P, C_i, \delta_i)$. The deadline D_i of Task Γ_i is defined by the deadline ratio δ_i as follows: $D_i = \delta_i P$. In this thesis, we explore early deadlines, that is, $0 < \delta < 1$. We define the ERfair scheduling algorithm with early deadlines which we call *ED-ERfair* in the following. Prior to study thermally-aware schedules of tasks with early-deadlines in the next chapter, we first consider how to schedule tasks with early-deadlines using the ED-ERfair scheduling scheme on *non thermally-constrained* multiprocessors. Then, we identify the *critical instance*, which is the worst-case arrival pattern that gives rise to the longest busy interval. As a result, the response times are maximized when a task invocation arrives at the critical instant. In this chapter, we also study whether a given task set can satisfy ERfairness with the ED-ERfair scheduling scheme or not. We conclude this chapter with an analysis of Maximum Schedulable Utilization for the ED-ERfair scheduling algorithm on M processors.

A. Early-Deadline Early-Release Fair Scheduling (ED-ERfair)

In this section, we propose *ED-ERfair*, a new algorithm for tasks with early-deadlines. ED-ERfair is based on ERfair (described in Section (A.2) of Chapter II) with special consideration for early deadlines when recalculating and reconstructing *lag* and *pseudo-deadlines* [11].

We note that *pseudo-release* does not matter for early-release tasks. It is therefore sufficient to only treat *lag* and *pseudo-deadlines* for tasks under ERfair scheduling. Equations (3.1) and (3.2) show the lag and pseudo-deadline definitions, respectively, for the traditional ERfair scheduling algorithm when the relative deadlines of tasks are equal to their periods

$$lag(\Gamma_i, t+1) = \begin{cases} lag(\Gamma_i, t) + w_i \\ lag(\Gamma_i, t) + w_i - 1 \end{cases} \text{, and} \tag{3.1}$$

$$d(\Gamma_{i,j}) = \left\lceil \frac{j}{w_i} \right\rceil . \tag{3.2}$$

Equation (3.1) shows that $lag(\Gamma_i, t+1)$ is equal to $lag(\Gamma_i, t) + w_i$ if Γ_i is not scheduled in slot t. Likewise, it indicates that $lag(\Gamma_i, t+1)$ is equal to $lag(\Gamma_i, t) + w_i - 1$ if Γ_i is scheduled in slot t.

For identical-periodic tasks with relative deadlines *less than* or equal to the period P, we use the *density* concept defined in [22]. The *density* of the task, which we denote by Δ_i, is defined as the ratio of the execution requirement C_i of Task Γ_i to the minimum of its relative deadline D_i and period P, that is, $\Delta_i = \frac{C_i}{min(D_i, P)}$, in our case $\Delta_i = \frac{C_i}{\delta_i P}$. With this definition, we replace w_i with Δ_i in the equations for *lag* and *pseudo-deadlines*:

$$lag(\Gamma_i, t+1) = \begin{cases} lag(\Gamma_i, t) + \Delta_i \\ lag(\Gamma_i, t) + \Delta_i - 1 \end{cases} \text{, and} \tag{3.3}$$

$$d(\Gamma_{i,j}) = \left\lceil \frac{j}{\Delta_i} \right\rceil . \tag{3.4}$$

In the following, we describe *ED-ERfair scheduling algorithm* with the recalculated *lag* and *pseudo-deadlines*.

Now, we describe the *ED-ERfair scheduling algorithm* for a set of identical-period tasks with relative deadlines less than or equal to the period. The following description shows how tasks are scheduled using ED-ERfair. We consider how to assign intermediate deadlines to subjobs (i.e., subtasks) and how to schedule the prioritized subjobs in the ED-ERfair scheduling algorithm.

Algorithm A: ED-ERfair scheduling algorithm

- Upon arrival of the k^{th} invocation, $J_{i,k}$, of a Job J_i in Task Γ_i, assign intermediate deadlines as follows:

1. Partition J_i into subjobs $J_{i,k}$: *All task parameters are expressed as integer multiples of the slot size,* s. *The jobs of a Task* Γ_i *with execution time* C_i *are partitioned into* $\lceil \frac{C_i}{s} \rceil$ *subjobs of length* s *each.*

2. Assign priorities to the subjobs: *For tasks with relative deadlines less than or equal to a period* P*, calculate both lag and pseudo-deadline* d *for each subjob as follows:*

$$lag(\Gamma_i, t+1) = \begin{cases} lag(\Gamma_i, t) + \Delta_i \\ lag(\Gamma_i, t) + \Delta_i - 1 \end{cases} \text{, and} \tag{3.5}$$

$$d(J_{i,k}) = \left\lceil \frac{k}{\Delta_i} \right\rceil . \tag{3.6}$$

With the modified pseudo-deadline information for each subjob, we use the PD^2 *ordering (2.6) to assign priorities to subjobs.*

- Schedule prioritized subjobs by ERfair scheduling scheme.

1. *Given the* PD^2 *priority assignment from the previous step 2, at the start of each slot, the* M *highest priority subjobs are selected to execute in the current slot. (*M *is the number of processors)*

2. *If Subjob* $J_{i,k-1}$ *and* $J_{i,k}$ *are part of the same job, then* $J_{i,k}$ *becomes eligible for execution immediately after* $J_{i,k-1}$ *finishes execution, which may be before* $J'_{i,k}s$ *Pfair window.*

In order to determine the *maximum schedulable utilization* for ED-ERfair scheduling, we must identify the worst-case arrival pattern, also named *critical instance* for each task. In the next section, we study the *critical instance* and the *busy intervals* that occur as a result of the critical instance on M processors.

B. Critical Instance and Busy Interval

In a single-processor static-priority real-time system without blocking, the critical instance of a Task Γ_i occurs when one of its job $J_{i,c}$ is released at the same time with a job in every higher-priority task [22]. Under Pfair scheduling, the critical instance for a Task Γ_i also occurs when all higher-priority tasks are released during the same slot as task Γ_i [23]. Figure 4 shows an example of the critical instance for Task Γ_3 in a three-task system with ERfair scheduling. Each upward arrow indicates the relative deadline of a task. Therefore, in Figure 4, it is easy to see that Task Γ_3 has the longest relative deadline (i.e., the lowest priority), and its response time forms the maximum busy interval for all three tasks.

According to [23], in a multiprocessor system with M processors, a Task Γ_i is released at the critical instance when all higher-priority tasks are released in the same slot as Γ_i.

We assume that whenever the number of eligible tasks are less than M, the tasks are first scheduled on smaller-indexed processors. For example, there are three processors and only two tasks that are eligible at time t. In this case the two tasks will be scheduled at time t on the first and second processor, respectively. With this

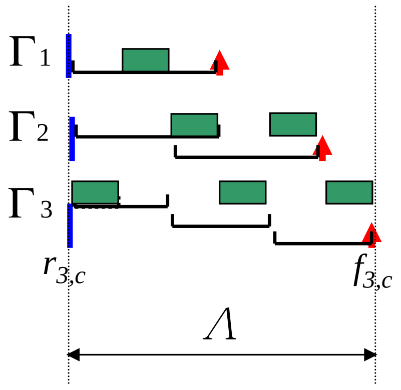


Fig. 4. Critical instance for task 3 under ED-ERfair scheduling

assumption, it is easy to see that the maximum busy interval will be always formed on the first processor when we schedule a set of *synchronous* tasks (In a synchronous task system, each task releases its first job at time 0 [8]). Figure 5 describes the maximum busy intervals beginning with the critical instance of the Task Γ_3 on two processors. From Figure 5 it is easy to see that the relative deadlines of Task Γ_3 and Task Γ_2 form two busy intervals on two processors when the processors are *fully utilized*. *Fully-processor utilized* case means the case where total utilization of given tasks is equal to the maximum schedulable utilization, which will be determined by Theorem 1 in later part of this section.

Before showing the schedulable utilization bound for the ED-ERfair scheduling algorithm, in the next section we study whether ED-ERfair schedules satisfy *ERfairness*.

C. ERfairness Test

In Section (A.2) of Chapter II, we have introduced *ERfairness*. In this section, we perform the schedulability test to check if a given task set is schedulable with the

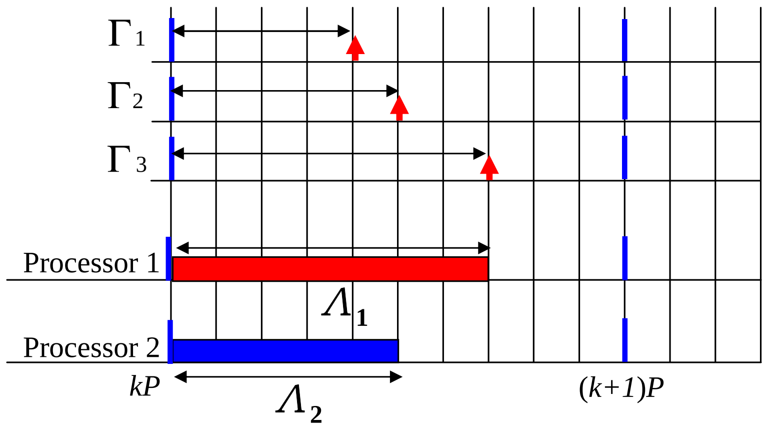


Fig. 5. Description of the maximum busy intervals on two processors

ED-ERfair scheduling algorithm. For this lemma, we assume that each task Γ_i has its own deadline ratio δ_i, and the number of tasks is larger than that of processors.

Lemma 1 *A system of* n *independent, preemptable, identical-period, and synchronous tasks with relative deadlines less than their periods satisfies ERfairness when scheduled according to the ED-ERfair algorithm on* M *processors if and only if the following inequality is satisfied:*

$$\forall i : \frac{dn_i}{\delta_i \cdot P} \leq M, \tag{3.7}$$

where $i = 1, 2, ...n$, and dn_i denotes the number of pseudo-deadlines before the relative deadline D_i.

Proof. First, we show that Inequality (3.7) is a necessary condition for ERfairness. With ERfair scheduling, a subjob may be released "early", i.e., before the beginning of its Pfair window. Therefore, more than one subjob can be scheduled within a "window" of time slots because the subjobs can be early-released. In order to satisfy the lag bound for ERfairness, at least one subjob must execute before its pseudo-deadline. If not, the subjob will miss its pseudo-deadline, and therefore a task set including the subjob will not satisfy *ERfairness*. Therefore, we conclude that *the number of pseudo-deadlines which are before time t* is equal to the minimal number of subjobs which should be scheduled by the time t. With the assumption that each task has its own deadline ratio (i.e., $D_i = \delta_i \cdot P$, where $0 < \delta_i \leq 1$, $i = 1, 2, ..., n$), we check if ERfairness is satisfied or not at each relative deadline of each task. If the ratio, $\frac{dn_i}{\delta_i \cdot P}$, exceeds M at the relative deadline of Task Γ_i, the minimally-required total utilization before the relative deadline must exceed M in order to satisfy ERfairness on M processors. However, by (2.4), the given task set is not schedulable on M processors. Therefore, some subjobs will not be scheduled before their pseudo-deadlines as well. Consequently, Inequality (3.7) is a necessary condition for ERfairness.

Next, we prove that Inequality (3.7) is a sufficient condition for ERfairness. On M processors, if ERfairness is satisfied for a given task set, then the task set must have a ERfair schedule at any time instant, and total utilization of the task set is equal to or less than M. If a schedule is not ERfair, at least one subjob is not scheduled before its pseudo-deadline. In such a schedule, the total utilization by subjobs needing to be scheduled before the same pseudo-deadline exceeds M, which means that $\frac{dn_i}{t}$ is more than M. In this situation, it is easy to see that the left side of Inequality (3.7) must exceed M. Therefore, Inequality (3.7) is a sufficient condition of ERfairness. ■

In the next section, we study how the maximum schedulable utilization (MSU) for the ED-ERfair scheduling algorithm can be calculated.

D. Maximum Schedulable Utilization for ED-ERfair Scheduling

The following theorem gives us the schedulable utilization of our ED-ERfair scheduling algorithm. Note that we only focus on tasks with relative deadlines less than or equal to an identical period P. We also assume that each task has its own deadline ratio, δ_i, and $\delta_{i+1} \geq \delta_i$ for all i, where $i = 1, 2, ..., n$. In this theorem, we use the definition for busy intervals on M processors, which has been described in the previous section.

Theorem 1 *A system of* n *independent, preemptable tasks with identical period* P *and relative deadlines less than or equal to* P *can be feasibly scheduled on* M *processors using ED-ERfair scheduling algorithm if its total utilization* U *is less than or equal to*

$$U_{ED-ERfair}(n, M, \delta_i) = \sum_{i=n-(M-1)}^{n} \delta_i \ , \tag{3.8}$$

where $\delta_{i+1} \geq \delta_i$ $(i = 1, 2, ..., n)$.

Proof. Figure 6 shows the case where there are M processors and n tasks with relative deadlines less than or equal to an identical period P. Specifically, in this case, tasks are ordered in increasing deadline ratios, that is, $\delta_{i+1} \geq \delta_i$, $(i = 1, 2, ..., n)$.

From Figure 6, first we calculate the schedulable utilization for the area A. Note that the ED-ERfair algorithm uses ERfair scheduling with PD^2 priority assignment. Since the schedulable utilization for ERfair scheduling scheme on M processors is M when $D_i = P$ [11], it is easy to see that the schedulable utilization for the area A is $\delta_1 M$. Likewise, the schedulable utilization for the area B is $(\delta_2 - \delta_1)M$. Consequently, we repeat these calculations until the number of the remaining tasks (after the calculations) become smaller than that of processors (i.e., $n - M + 1$ times). This

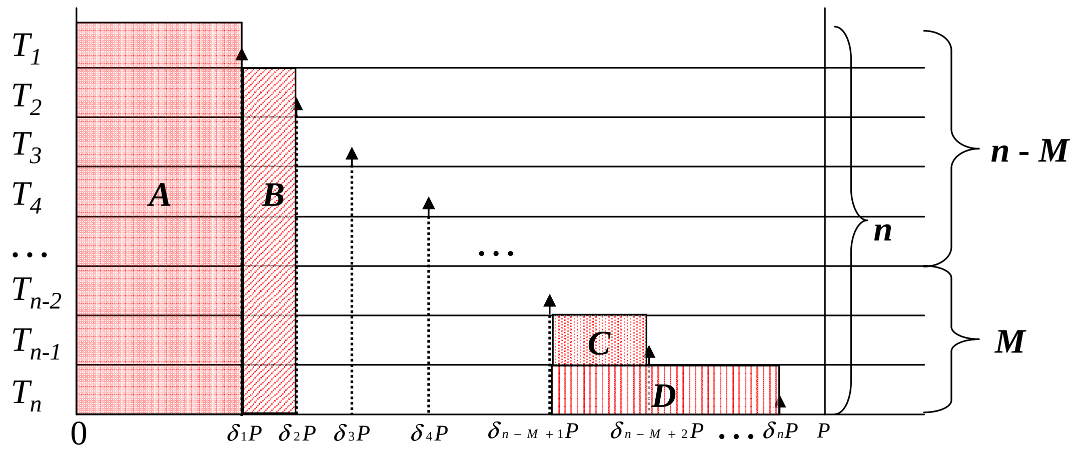


Fig. 6. Description for MSU analysis of ED-ERfair scheduling

calculation can be expressed as follows:

$$\delta_1 M + (\delta_2 - \delta_1)M + (\delta_3 - \delta_2)M + ... + (\delta_{n-M+1} - \delta_{n-M})M \ . \tag{3.9}$$

Note that tasks indexed from $(n - M + 2)^{th}$ to n^{th} cannot be fully scheduled on M processors because the number of remaining tasks is smaller than the number of processors. Therefore, we calculate the schedulable utilization for the area C and D as follows:

$$(\delta_{n-M+2} - \delta_{n-M+1}) + (\delta_{n-M+3} - \delta_{n-M+1}) + ... + (\delta_n - \delta_{n-M+1}) \ . \tag{3.10}$$

Finally, by summation of (4.11) and (3.10), we have the following *maximum schedulable utilization* of the ED-ERfair scheduling algorithm on M processors.

$$\delta_{n-M+1} \cdot M + \sum_{i=n-M+2}^{n} (\delta_i - \delta_{n-M+1}) \tag{3.11}$$

$$= \delta_{n-M+1} \cdot M + (\sum_{i=n-M+2}^{n} \delta_i) - (M-1) \cdot \delta_{n-M+1} \tag{3.12}$$

$$= \delta_{n-M+1} \cdot \{M - (M-1)\} + \sum_{i=n-M+2}^{n} \delta_i \tag{3.13}$$

$$= \delta_{n-M+1} + \sum_{i=n-M+2}^{n} \delta_i \tag{3.14}$$

$$= \sum_{i=n-M+1}^{n} \delta_i \ . \tag{3.15}$$

■

We use the following example to illustrate Theorem 1: When we use the ED-ERfair scheduling algorithm, if there are two processors and three tasks, the two largest relative deadlines of the three tasks determine the maximum schedulable utilization.

So far, we have studied schedulability for ED-ERfair scheduling on *non thermally-constrained* multiprocessor real-time systems. In the next chapter, we proceed to study the schedulability for global scheduling on *temperature-constrained* multiprocessor real-time systems.

CHAPTER IV

THERMALLY-AWARE MULTIPROCESSOR REAL-TIME SCHEDULING

In this chapter, we study ED-ERfair scheduling on *Temperature-Constrained Multiprocessor Real-Time Systems* (TCMRTS). In the first section, we introduce previous works from Wang *et al.* [3, 4, 24]. Then, based on these results, we proceed to show how the critical instance and busy intervals on TCMRTS can be identified with ED-ERfair scheduling scheme. We conclude this chapter with a schedulability analysis for ED-ERfair scheduling on TCMRTS.

A. Background: Real-Time Scheduling on Temperature-Constrained Real-Time Systems

In this section, we briefly describe a model for temperature-aware computation in real-time systems based on reactive speed scaling, which has been defined in [3, 4, 24].

Using the thermal model and reactive speed scaling previously introduced in Chapter II, we make two observations [3, 4, 24]. The observations show how the change of temperature, job arrival, and job execution will affect the temperature at a later time (*Observation A*) or the delay of a later job (*Observation B*). These observations will be used when we describe the critical instance and busy interval in the later part of this section. The proofs of these observations are found in [24].

Observation A. (stated in [4]) *In a system under reactive speed scaling, given a time instance t, a job with release time t_r and completion time t_f can be considered such that $t_r < t$ and $t_f < t$. It is assumed that the processor is idle during $[t_f, t]$. If either of the following actions takes, as shown in Figure 7, then $T_t \leq T_t^*$, where T_r and T_t^* are the temperatures at time t in the original and the modified scenarios* *respectively.*

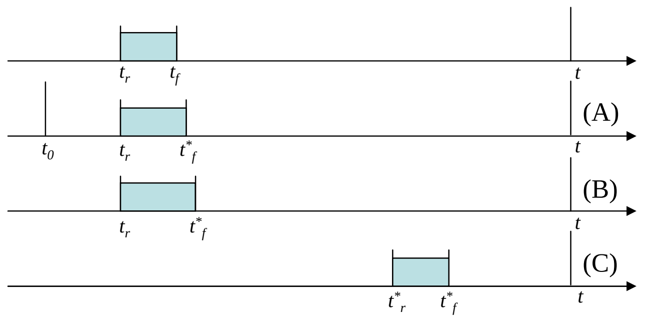


Fig. 7. Temperature effect

- *Action A*: *Increasing the temperature at time t_0 ($t_0 \leq t_r$ such that the job has the same release time t_r but a new completion time t^*_f satisfying $t^*_f < t$.*

- *Action B*: *Increasing the processor cycles for this job such that the job has the same release time t_r but a new completion time t^*_f satisfying $t^*_f < t$.*

- *Action C*: *Shifting the job such that the job has a new release time t^*_r and a new completion time t^*_f satisfying $t_r < t^*_r < t$ and $t_f < t^*_f < t$.*

Observation B. (stated in [4]) *In a system under the reactive speed scaling, two jobs J_k's ($k = 1, 2$), each of which has a release time $t_{k,r}$ and the completion time $t_{k,f}$ are considered. $t_{1,f} < t_{2,f}$ is assumed. If either of the following actions as shown in Figure 8, then $t_{2,f} \leq t^*_{2,f}$. If d_2 and d^*_2 as the delay of Job J_2 in the original and the modified scenariods respectively, then $d_2 \leq d^*_2$.*

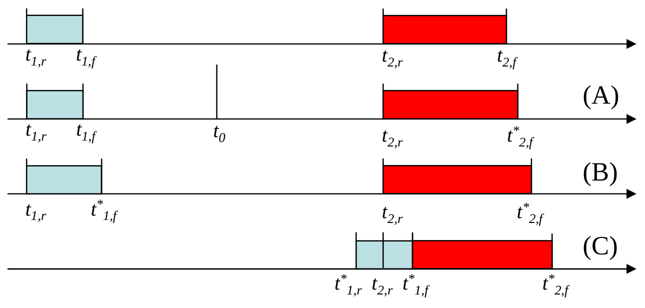


Fig. 8. Delay effect

- *Action A*: *Increasing the temperature at t_0 ($t_0 \leq t_{2,r}$) such that Job J_2 has the same release time $t_{2,r}$ but a new completion time $t^*_{2,f}$.*

- *Action B*: *Increasing the processor cycles of Job J_1 such that Job J_k ($k = 1, 2$) has the same release time $t_{k,r}$ but a new completion time $t^*_{k,f}$.*

- *Action C*: *Shifting Job J_1 has a new release time $t^*_{1,r}$ and a new completion time $t^*_{1,f}$, and Job J_2 has the same release time $t_{2,r}$ and a new completion time $t^*_{2,f}$ satisfying $t_{1,r} \leq t^*_{1,r}$ and $t^*_{1,f} \leq t^*_{2,f}$*

  .

The two observations above are particularly significant for analysis of the critical instance under our algorithm when reactive speed scaling is used on temperature-constrained systems. Consequently, *Observation A* shows that the temperature for the critical instance always raises when the last job before the critical instance is delayed. Then, *Observation B* states that the completion of the job released at the critical instance is delayed with increasing temperature.

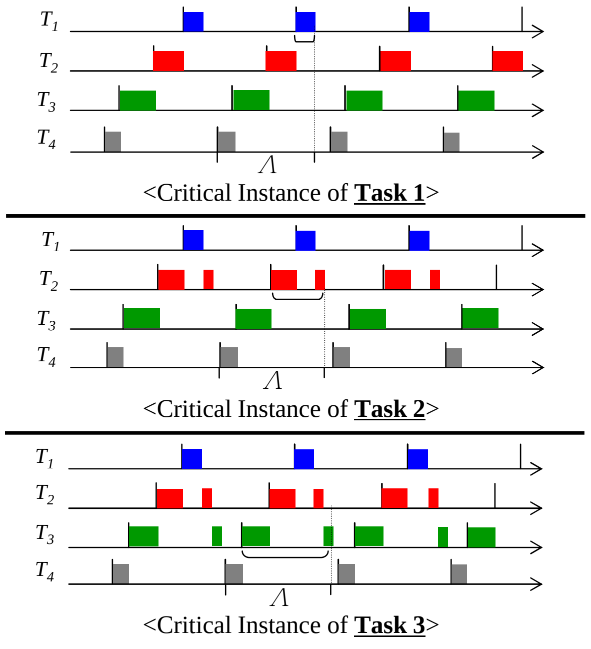
In order to determine the worst-case delay for jobs in tasks, Wang *et al.* [3] have identified the critical instance for fixed-priority scheduling on temperature-constrained real-time systems. Due to the additional temperature constraints, the analysis of the critical instance is different from the traditional one. In a fully preemptable system with reactive speed scaling, Wang *et al.* [3] consider the following two factors that affect the critical instance:

- *How many high-priority jobs will preempt this job? This is same as in the traditional critical instance analysis.*

- *What is the temperature at this time instance? With reactive speed scaling, once the temperature hits the threshold, the speed drops to no higher than equilibrium speed. Therefore, the response time of a job is affected by the temperature at its arrival. The higher the temperature at this time instance, the longer the response time.*

Consequently, Wang *et al.* [3] consider both the worst-case preemption and the worst-case temperature for their critical-instant analysis. Then, they consider the maximum busy interval Λ beginning with the critical instance for each task in a traditional real-time system without blocking. Based on this maximum busy interval Λ, finally, they obtain the critical instance for each task in the temperature-constrained real-time system. Figure 9 illustrates the critical instant for each task in a set of three tasks. The jobs of lower-priority task are scheduled to be aligned back-to-back before the critical instance. Then, by *Observation A and B*, the initial temperature is raised; tasks released at the critical instant will experience the worst-case temperature, which in turn further delays the completion of the job under consideration.

Since the task set is single-period, and the scheduler is work-conserving, the entire task set can be represented as a single task. Figure 10 describes this single



T_1
T_2
T_3
T_4
Λ
<Critical Instance of **Task 1**>
T_1
T_2
T_3
T_4
Λ
<Critical Instance of **Task 2**>
T_1
T_2
T_3
T_4
Λ
<Critical Instance of **Task 3**>

Fig. 9. Description of critical instance on temperature-constrained RTS

job case. For the single task, the following Equations (4.1), (4.2), (4.3), and (4.4) are obtained by using the temperature formulas from Chapter II:

$$\pi_{k,RH} = t_{k,H} - t_{k,R} = \frac{1}{b} \ln \frac{\frac{aS_H^\alpha}{b} - T_{k,R}}{\frac{aS_H^\alpha}{b} - T_H} \tag{4.1}$$

$$\pi_{k,HC} = t_{k,C} - t_{k,H} = \frac{S_H}{S_E}(\frac{C}{S_H} - \pi_{k,RH}) \tag{4.2}$$

$$\pi_{k,RC} = t_{k,C} - t_{k,R} = \frac{C}{S_E} + (1 - \frac{S_H}{S_E}) \cdot \pi_{k,RH} \tag{4.3}$$

$$\pi_{k-1,CR} = t_{k,R} - t_{k-1,C} = (P - \frac{C}{S_E}) + \frac{1}{b}(\frac{S_H}{S_E} - 1) \cdot \ln \frac{\frac{aS_H^\alpha}{b} - T_{k-1,R}}{\frac{aS_H^\alpha}{b} - T_H} \tag{4.4}$$

Recall that T_H, S_H, S_E, b, C, and P denote the threshold temperature, some maximum speed (frequency) and equilibrium speed for a given processor, power dissipation

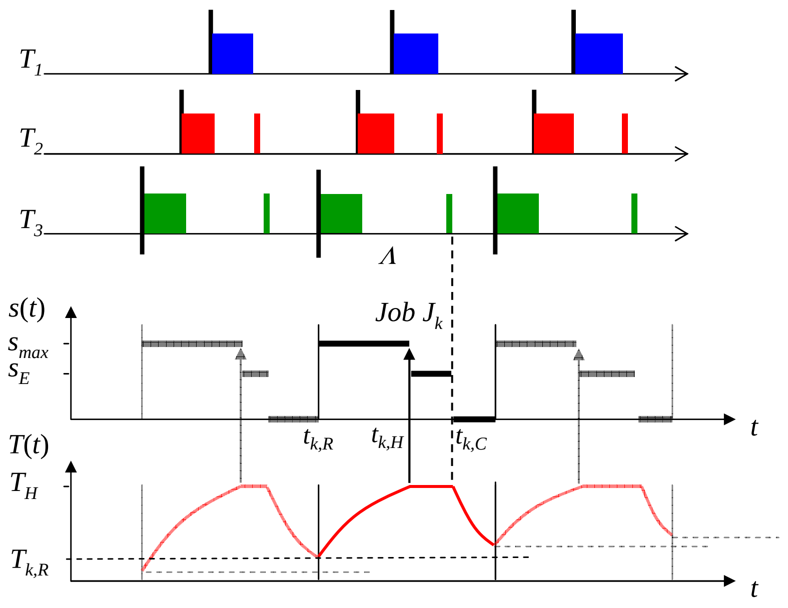


Fig. 10. Single-job case under reactive speed scaling

rate, execution cycles and the identical period of the given tasks, respectively.

Based on these equations, one also obtains Equation (4.5), which calculates the temperature ($T_{k,R}$) at the beginning of the maximum busy interval Λ:

$$\frac{T_{k,R}}{T_H} = (\frac{(\frac{S_H}{S_E})^\alpha - \frac{T_{k-1,R}}{T_H}}{(\frac{S_H}{S_E})^\alpha - 1})^{1-\frac{S_H}{S_E}} \cdot \exp(-b(P - \frac{C}{S_E})) \ . \tag{4.5}$$

The steady-state temperature (a fixed point, T^*) can be calculated (when $\lim_{k\to\infty} T_{k,R}$), and the following Equation (4.6) can be obtained for the steady-state temperature, T^*:

$$\frac{T^*}{T_H} = (\frac{(\frac{S_H}{S_E})^\alpha - \frac{T^*}{T_H}}{(\frac{S_H}{S_E})^\alpha - 1})^{1-\frac{S_H}{S_E}} \cdot \exp(-b(P - \frac{C}{S_E})) \ . \tag{4.6}$$

Based on the thermal behavior in the steady-state, one obtains the worst-case delay bound (4.9) for the single job, and the maximum schedulable utilization (4.10). Note that in the case of early deadlines, i.e., $D_i = \delta P$ ($0 < \delta \leq 1$), the MSU ($U_{RSS}(\delta)$) can be computed by setting $P + \frac{1}{b}\ln\frac{T^*}{T_H} = \delta P$. Figure 11 describes how these Equations (4.6), (4.9), and (4.10) are calculated in the thermal steady-state:

$$\begin{aligned}\lim_{k\to\infty} \pi_{k,RC} &= P - \lim_{k\to\infty}(t_{k,R} - t_{k-1,C}) &(4.7)\\ &= P + \frac{1}{b}\lim_{k\to\infty}\ln\frac{T_{k,R}}{T_H} &(4.8)\\ &= P + \frac{1}{b}\ln\frac{T^*}{T_H}\ , &(4.9)\end{aligned}$$

$$U^{RSS}(\delta) = \frac{S_E}{S_H}\delta + (1 - \frac{S_E}{S_H})\frac{1}{bP}\cdot\ln(\frac{(\frac{S_H}{S_E})^\alpha - e^{-b(1-\delta)P}}{(\frac{S_H}{S_E})^\alpha - 1})\ . \qquad (4.10)$$

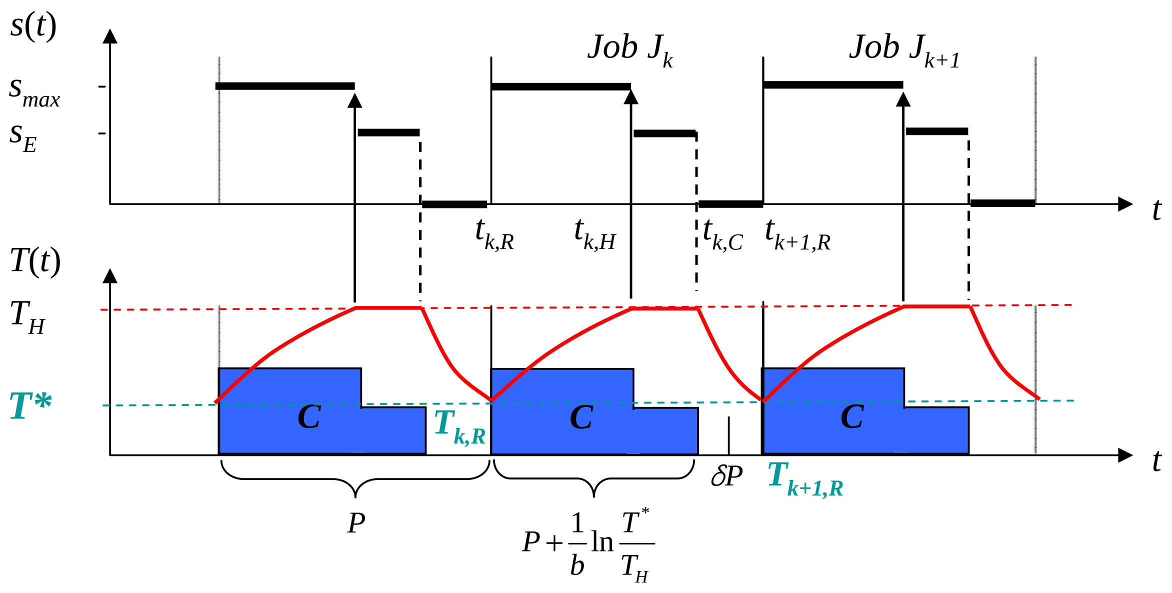


Fig. 11. Description of the worst-case delay under reactive speed scaling

So far, we have studied how thermally-aware computation can be performed in real-time system based on reactive speed scaling. Based on the observations and results in this section, in the next section, we study how the critical instance and busy intervals can be identified on temperature-constrained *multiprocessor* real-time system when we use ED-ERfair scheduling.

B. Critical Instant and Busy Interval on Temperature-Constrained RTS's

In this section, we study how the critical instant and busy interval can be identified on temperature-constrained multiprocessor real-time systems. Before we consider the critical instant for ED-ERfair scheduling on M processors later in this section, we first proceed to study the critical instant for each task under the ED-ERfair scheduling on *a single processor.*

1. Critical Instant for ED-ERfair on Temperature-Constrained *Single Processors*

We showed earlier that the task response time in a single-task system is maximized when the system is in *thermal steady-state.* Therefore, to identify the critical instant and the maximum busy interval, we limit our attention to the *thermal steady-state.*

In order to determine the critical instant for a job, we must consider two causes for delay: *delay due to temperature* and *delay due to preemption.*

We make two observations: (1) The delay due to temperature is maximized whenever each task except T_i is shifted so that its last subjob during the busy interval is completed as late as possible before the completion of the last subjob of T_i. (2) In order to compute the maximum delay due to preemption, we must consider that, under ED-ERfair scheduling, the worst-case preemption can be caused by both *tasks with shorter relative deadlines* and the *intermediate pseudo-deadlines of tasks with* *longer relative deadlines.*

We propose the following *Algorithm B* to identify the critical instant for Task T_i on a temperature-constrained single processor under ED-ERfair scheduling with reactive speed scaling. We start with the maximum busy interval Λ beginning with the critical instant in a traditional real-time system without blocking (See Figure 4). The following example in Figure 12 shows how the critical instance for Task T_2 can be identified by *Algorithm B*.

Algorithm B.

Input: Parameters of the given tasks (execution cycles, period, and deadline ratios), speed ratio, and threshold temperature.

Output: Critical instant for task T_i.

Step 1. Calculate the *steady-state temperature*, T^*, using Equation (4.6) being based on the busy interval Λ.

Step 2. At the thermally steady-state, first, each of tasks except T_i is shifted so that its last subjob during the busy interval Λ is completed as late as possible before the completion of the last subjob of T_i.

Step 3. Then, the subjobs of the task with larger relative-deadline than T_i are continuously shifted by one slot until the release time of the task is same with the release time of T_i.

Step 4. We identify a moment when T_i has the maximum response time for the first time. Then, all tasks are phased like the moment.

In Figure 12, it is easy to see that with *Algorithm B*, all tasks are phased like Figure

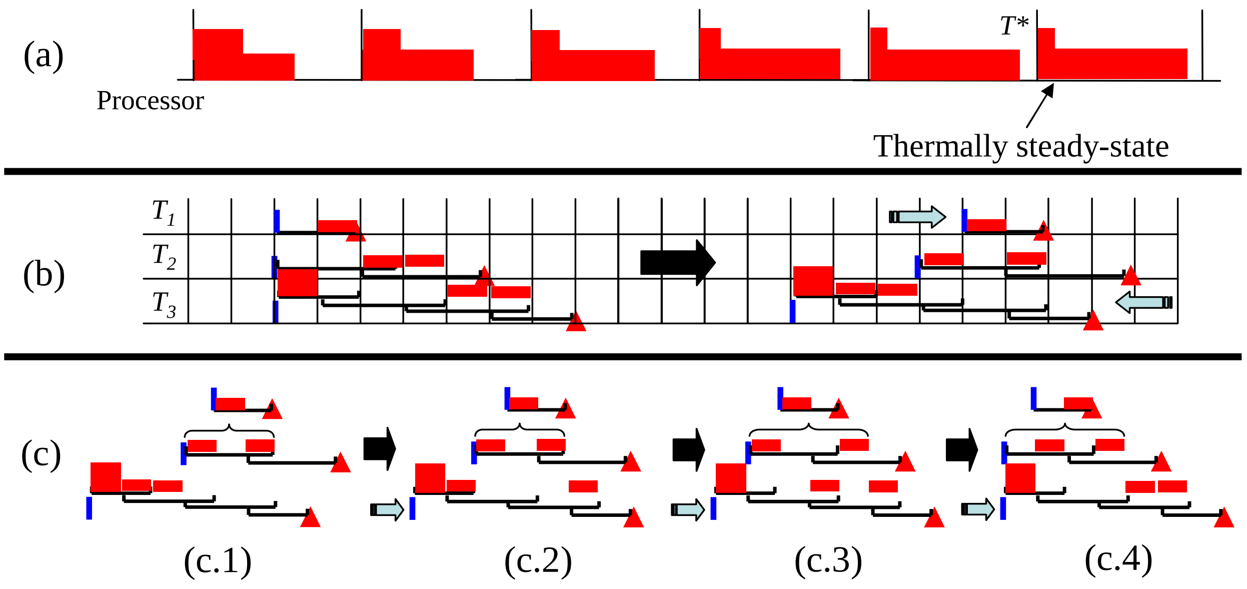


Fig. 12. Description of critical instant for T_2 under ED-ERfair on TCRTS

12(c.3) for the critical instance of Task T_2. Note that the busy interval Λ is maintained in Figure 12 while we identify the critical instance.

Consequently, in Figure 12, with total execution cycles during the busy interval Λ, we can obtain the steady-state temperature (T^*). Then, with the T^*, if $T_2^* < T_H$, we have the worst-case delay $d^{RSS}_{ED-ERfair,2}$ experienced by a job in Task T_2 bounded as $d^{RSS}_{ED-ERfair,2} \leq P + \frac{1}{b}\ln\frac{T^*}{T_H} - \frac{C_{2,r}}{S_H} - \frac{s}{S_E}$, where $C_{2,r}$ and s denote execution cycles of the subjobs of lower-priority task aligned back-to-back before the critical instance of T_2 and the slot size, respectively (This calculation can be done with Equations (4.6) and (4.9)).

Additionally, if the deadline of task $D_i = \delta P$, where $0 < \delta \leq 1$, we have the worst-case delay $d^{RSS}_{ED-ERfair}$ for any job in all n tasks bounded as $d^{RSS}_{ED-ERfair} \leq P + \frac{1}{b}\ln\frac{T^*}{T_H}$.

2. Critical Instance on Temperature-Constrained *Multiprocessor* RTS

Now, we consider the critical instance and busy intervals for tasks scheduled by ED-ERfair on temperature-constrained *multiprocessor* real-time systems. We propose the following *Algorithm C* to identify the critical instance for Task T_i on TCMRTS. Note that based on the busy intervals in traditional (*non thermally-constrained*) multiprocessor real-time systems (see Figure 5), we obtain the critical instance for Task T_i by the following *Algorithm C*.

Algorithm C.

Input: Parameters of the given tasks (execution cycles, period, and deadline ratios), the number of tasks and processors, speed ratio, and threshold temperature.

Output: Critical instant for task T_i.

Step 1. We get *Steady-State Temperature*, T_j^*, based on the busy interval, Λ_j, where $j = 1, 2, ..., M$ (Therefore, T_j^* denotes steady-state temperature on j^{th} processor.).

Step 2. At the thermally steady-state, each of tasks except T_i is shifted so that its last subjob during the busy intervals on M processors is completed as late as possible before the completion of the last subjob of T_i. Here, as the processor for each subjob of T_i, *the hottest processor is* always selected.

Step 3. Then, the subjobs of the task with larger relative-deadline than T_i are continuously shifted by one slot on M processors until the release time of the task is same with the release time of T_i.

Step 4. We find a moment when T_i has the maximum response time for the first time.

All tasks are phased like the moment on M processors.

With the critical instance and busy intervals identified by *Algorithm C*, we can have the worst-case delay for Task T_i assigned on the hottest processor under ED-ERfair scheduling scheme. Let's define T_H^* to be the steady-state temperature on the hottest processor. Then, T_H^* can be obtained by the maximum busy interval formed on the hottest processor. Consequently, with the T_H^*, if the deadline of task $D_i = \delta P$, where $0 < \delta \leq 1$, we have the worst-case delay for any job in all n tasks bounded as $d_{ED-ERfair,TCMRTS}^{RSS} \leq P + \frac{1}{b} \ln \frac{T_H^*}{T_H}$.

In the next section, with the analysis in this section, we show how MSU can be calculated under ED-ERfair scheduling on temperature-constrained multiprocessor real-time system. Based on the maximum busy intervals formed in traditional multiprocessor real-time systems (see Figure 5), the maximum schedulable utilization on temperature-constrained multiprocessor real-time system is calculated.

C. Maximum Schedulable Utilization for ED-ERfair Scheduling on Temperature-Constrained Multiprocessor Real-Time Systems

In this section, we determine MSU on temperature-constrained multiprocessor real-time systems when we use ED-ERfair scheduling scheme.

Based on the maximum busy intervals formed in traditional multiprocessor systems, we have the following theorem. The proof for this theorem is done by replacing δ_i in Theorem 1 with $U^{RSS}(\delta_i)$ ($U^{RSS}(\delta)$ has been calculated in (4.10).). Recall that MSU is determined when $P + \frac{1}{b} \ln \frac{T_j^*}{T_H} = \delta_i P$, where T_j^* and $\delta_i P$ denote the steady-state temperature on j^{th} processor ($j = M, M-1, ..., 1$) and the relative deadline of Task T_i ($i = n-(M-1), n-(M-1)+1, ..., n$), respectively.

Theorem 2 *A system of* n *independent, preemptable, and identical-periodic tasks* *with relative deadlines less than or equal to a period* P *can be feasibly scheduled on temperature-constrained M processors if scheduled according to the ED-ERfair scheduling algorithm if its total utilization* U *is less than or equal to*

$$U^{RSS}_{ED-ERfair}(n, M, \delta_i) = \sum_{i=n-(M-1)}^{n} U^{RSS}(\delta_i) \tag{4.11}$$

where $\delta_{i+1} \geq \delta_i$ $(i = 1, 2, ..., n)$.

In the next chapter, we study how the maximum schedulable utilization can be determined under existing partitioning heuristics on temperature-constrained multi-processor real-time systems.

CHAPTER V

COMPARISON BETWEEN PARTITIONING AND ED-ERFAIR ON TCMRTS

In this chapter, we compare existing partitioning heuristics with our ED-ERfair scheduling scheme by showing MSU on temperature-constrained multiprocessor real-time system.

A. Partitioning

In this section, we show how MSU can be determined under existing partitioning heuristics on temperature-constrained multiprocessor real-time systems.

As we mentioned in Section (A.1) of Chapter II, in partitioning schemes, each processor schedules tasks independently from a local ready queue. When a new task arrives, it is assigned to one of these ready queues and executes only on the associated processor until the task finishes execution. Therefore, no task migration is permitted in partitioning.

With this consideration, we have the following Theorem 5.1 showing how MSU can be considered for partitioning on temperature-constrained multiprocessor real-time systems. We assume use of a fixed-priority scheduling scheme as the chosen uniprocesor scheduling algorithm.

Theorem 3 *For a system of* n *independent, preemptable, and identical-periodic tasks with relative deadlines less than or equal to the period* P*, the* maximum schedulable utilization $U^{RSS}_{Partitioning}$ *under reactive speed scaling can be expressed if scheduled according to* partitioning heuristics *on temperature-constrained multiprocessor real-time systems as:*

$$U^{RSS}_{Partitioning} = \sum_{i=1}^{M} U^{RSS}(\delta_{\max,i}) \qquad (5.1)$$

where $\delta_{max,i}$ denotes the maximal δ on i^{th} processor.

Proof. When we schedule tasks with partitioning heuristics, each task is assigned to one of M processors. The processor selection scheme depends on which heuristic we uses.

Regardless of the kind of the heuristics, in order to determine MSU for the chosen partitioning scheme on M processors, we simply sum each MSU calculated on each processor. Since each MSU is determined by the largest relative deadline among those of tasks scheduled on the associated processor, MSU on i^{th} processor can be expressed as $U^{RSS}(\delta_{max,i})$. Therefore, total MSU is calculated by summing up all MSU on M processors. ■

CHAPTER VI

CONCLUSION

It is a well-known problem that power density in processors has increased exponentially, and power dissipation is often particularly important in dense packaging environments of high-performance embedded systems.

In this thesis, we describe a methodology for temperature-aware scheduling and computation on thermally-constrained multiprocessor real-time systems. For real-time scheduling on multiprocessors, this thesis is based on ERfair scheduling, which is a work-conserving extension to Pfair global scheduling. Then, we study the benefits of Reactive Speed Scaling, a dynamic thermal management scheme, in real-time multiprocessor systems.

Pointing out that reactive speed scaling has no benefit under existing ERfair scheduling, we propose the *ED-ERfair* (ERfair scheduling for tasks with Early-Deadlines) scheduling scheme for a set of identical-period tasks with relative deadlines less than or equal to the period. We also perform *ERfairness test* to check if the given task set satisfies *ERfairness* under ED-ERfair scheduling. Then, in support of the temperature-awareness, we extend the applicability of reactive speed scaling to our ED-ERfair scheduling algorithm.

We show how the proposed algorithm and reactive speed scaling scheme can enhance the processor utilization compared with any constant-speed scheme on real-time multiprocessor systems. Moreover, we show how the maximum schedulable utilization can be determined for existing partitioning heuristics.

REFERENCES

[1] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, and D. Tarjan, “Temperature-aware computer systems: opportunities and challenges,” in *Proc. IEEE Micro*, Dec. 2003, pp. 52-61.

[2] L. T. Yeh and R. C. Chu, *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods and Design Practices*, New York: ASME Press, 2001.

[3] S. Wang and R. Bettati, “Reactive speed control in temperature-constrained real-time system,” in *Proc. 18th Euromicro Conference on Real-Time Systems (ECRTS ’06)*, Dresden, Germany, Jul. 2006, pp. 161-170.

[4] S. Wang and R. Bettati, “Delay analysis in temperature-constrained hard real-time systems with general task arrivals,” in *Proc. 27th IEEE International Real-Time System Symposium (RTSS ’06)*, Dec. 2006, pp. 323-334.

[5] A. P. Ferreira, Daniel Mosse, and Jae C. Oh, “Thermal faults modeling using a RC model with an application to web farms,” in *Proc. 19th Euromicro Conference on Real-Time Systems (ECRTS ’07)*, Jul. 2007, pp. 113-124.

[6] K. Stavrou and P. Trancoso, “Thermal-aware scheduling: a solution for future chip multiprocessors thermal problems,” in *Proc. 9th Euromicro Conference on Digital System Design (DSD ’06)*, Aug. 2006, pp. 123-126.

[7] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, “Proportionate progress: a notion of fairness in resource allocation,” *Algorithmica*, 1996, pp. 600-625.

[8] J. Anderson and A. Srinivasan, "Mixed Pfair and ERfair scheduling of asynchronous periodic tasks," in *Proc. 13th Euromicro Conference on Real-Time Systems (ECRTS '01)*, Jun. 2001, pp. 76-84.

[9] S. Baruah, J. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proc. 9th International Parallel Processing Symposium*, Apr. 1995, pp. 280-288.

[10] J. Anderson and A. Srinivasan, "A new look at Pfair priorities," Dept. Comput. Sci., Univ. North Carolina, Chapel Hill, NC, Tech. Rep. TR00-023, Sep. 2000.

[11] J. Anderson and A. Srinivasan, "Early-release fair scheduling," in *Proc. 12th Euromicro Conference on Real-Time Systems (ECRTS '00)*, Jun. 2000, pp. 35-43.

[12] S. Davari and S. Dhall, "An on-line algorithm for real-time tasks allocation," in *Proc. 7th IEEE International Real-Time Systems Symposium (RTSS '86)*, 1986, pp. 194-200.

[13] S. Dhall and C. Liu, "On a real-time scheduling problem," *Operations Research*, Jan. 1978, pp. 127-140.

[14] Shelby Funk and Sanjoy Baruah, "Task assignment on uniform heterogeneous multiprocessor," in *Proc. 17th Euromicro Conference on Real-Time Systems (ECRTS '05)*, Jul. 2005, pp. 219-226.

[15] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York: W. H. Freeman and Company, 1979.

[16] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture: extended discussion and re-

sults," Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, Tech. Rep. CS-2003-08, Apr. 2003.

[17] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *Symposium on Foundations of Computer Science (FOCS '04)*, Oct. 2004, pp. 520-529.

[18] N. Bansal and K. Pruhs, "Speed scaling to manage temperature," in *Symposium on Theoretical Aspects of Computer Science (STACS '05)*, Mar. 2005, pp. 1-9.

[19] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch, "Tempest: a thermal enabled multimodel power/performance estimator," in *Workshop on Power-Aware Computer Systems*, 2000, pp. 112-125.

[20] A. Cohen, L. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy, "On estimating optimal performance of cpu dynamic thermal management," *Computer Architecture Letters*, Jan. 2003, pp. 6-15.

[21] S. K. Baruah, "Scheduling periodic tasks on uniform multiprocessors," in *Proc. 12th Euromicro Conference on Real-Time Systems (ECRTS '00)*, Jun. 2000, pp. 7-15.

[22] J. Liu, *Real-Time Systems*, Upper Saddle River, NJ: Prentice Hall, 2000.

[23] S. K. Baruah, "Fairness in periodic real-time scheduling," in *Proc. 16th IEEE International Real-Time Systems Symposium (RTSS '95)*, Dec. 1995, pp. 200-206.

[24] S. Wang and R. Bettati, "Delay analysis in temperature-constrained hard real-time systems with general task arrivals," Dept. Comput. Sci., Texas A&M Univ., College Station, TX, Tech. Rep. tamu-cs-tr-2006-5-3, May 2006.

VITA

Ja Ryeong Koo received his Bachelor of Science degree in computer engineering from Kyunghee University in 2003. He entered the Computer Science program at Texas A&M University in August 2005 and received his Master of Science degree in May 2008. His research interests include real-time operating systems and scheduling.

Mr. Koo may be reached at 301 Harvey R. Bright Building, College Station, TX 77843. His email is wjbkoo@gmail.com.