國立交通大學

網路工程研究所

碩士論文

異質雙核心嵌入式即時系統之

有效低能源消耗排程演算法

An Efficient Low Power Scheduling for Heterogeneous Dual-Core

Embedded Real-Time Systems

研 究 生：林柏君

指導教授：王國禎　教授

中 華 民 國　九 十 七　年 六 月

異質雙核心嵌入式即時系統之有效低能源消耗排程演算法

An Efficient Low Power Scheduling for
Heterogeneous Dual-Core Embedded Real-Time Systems

研 究 生：林柏君　　Student：Pochun Lin

指導教授：王國禎　　Advisor：Kuochen Wang

國 立 交 通 大 學

資 訊 學 院

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

異質雙核心嵌入式即時系統之
有效低能源消耗排程演算法

學生：林柏君　　指導教授：王國禎 博士

國立交通大學 資訊學院 網路工程研究所

摘 要

近年來，異質雙核心嵌入式即時系統，如個人數位助理及手機越來越受到歡迎。為了達到即時以及低能源的耗損，低能源消耗排程成為一個值得重視的研究課題。大多數現有低能源消耗排程的動態電壓調整演算法，其目標多在單一 CPU 或是在同質性的多核心系統上。本論文中，在異質性的雙核心嵌入式即時系統上，我們提出一個較長共同執行時間(LCET)演算法，以適用在具動態電壓調整能力的異質雙核心。本演算法包含兩個階段：第一，運用所提出的排班演算法來縮短在異質雙核心嵌入式即時系統上的工作總執行時間；第二，我們更進一步探討利用縮短過後的總執行時間來調整電壓及頻率以達到低能量的耗損。模擬結果顯示，相較於 Kim et al. 方法，在使用(不使用)動態電壓調整之下，我們所提出的方

法分別在可插隊的 LCET (P-LCET)及不可插隊的 LCET (NP-LCET)，各節省了 8%及 16% ~ 25% (13%及 33% ~ 38%)的總能量耗損。

關鍵詞：動態電壓調整，嵌入式即時系統，異質雙核心，低能源消耗排程，總執行時間。

An Efficient Low Power Scheduling for Heterogeneous Dual-Core Embedded Real-Time Systems

Student: Pochun Lin **Advisor: Dr. Kuochen Wang**

Department of Computer Science

National Chiao Tung University

Abstract

In recent years, heterogeneous dual-core embedded real-time systems, such as personal digital assistants (PDAs) and cellular phones, have become more and more popular. In order to achieve real time performance and low energy consumption, low power scheduling becomes a critical issue. Most researches on low power scheduling with dynamic voltage scaling (DVS) were targeted at only one CPU or homogeneous multi-core systems. In this thesis, we propose a low power scheduling algorithm called *Longer Common Execution Time* (LCET) for DVS enabled heterogeneous dual-core embedded real-time systems, which includes two steps. First, we reduce total execution time of tasks by using LCET in heterogeneous dual-core embedded real-time systems. Second, we further exploit the reduced total execution time to adjust voltage and frequency levels in order to reduce the total energy consumption. Simulation results show that the proposed P-LCET (a preemptive version) and NP-LCET (a non-preemptive version) can effectively reduce the total energy consumption by 8% and 16% ~ 25% (13% and 33% ~ 38%) compared with the work by Kim et al. with (without) dynamic voltage scaling.

Keywords: Dynamic voltage scaling, embedded real-time system, heterogeneous dual-core,

low power scheduling, total execution time.

Acknowledgements

Many people have helped me with this thesis. I deeply appreciate my thesis advisor, Dr. Kuochen Wang, for his intensive advice and instruction. I would like to thank all the classmates in *Mobile Computing and Broadband Networking Laboratory* (MBL) for their invaluable assistance and suggestions. The support by the NCTU EECS-MediaTek Research Center under Grant Q583 and the National Science Council under Grants NSC96-2628-E-009-140-MY3 and NSC96-2628-E-002-138-MY3 is also grateful acknowledged.

Finally, I thank my father, my mother and my friends for their endless love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With more and more multimedia applications, low energy consumption is extremely important for heterogeneous dual-core embedded real-time systems, like the PDA and smart-phone. Most mobile handhelds are dual-core systems [6] . A dual-core system is mainly composed of an ARM processor and a DSP. Fig. 1 shows a heterogeneous dual-core architecture. Like the OMAP processor, which is manufactured by TI (Texas Instruments) for mobile applications, includes two cores, ARM926 (ARM9 core) and TMS320C55X (DSP coprocessor) [12] . The Freescale i.300-30 processor also includes two cores: ARM11 and StarCore SC140 (DSP processor) [13].

In order to conserve energy for battery-powered real-time systems, several low power techniques were proposed. Dynamic voltage scaling (DVS) and dynamic power management (DPM) have been employed as available techniques to reduce the energy consumption of CMOS microprocessor systems [1]. The DVS is a design technique to adjust the CPU's supply voltage and frequency. The primary design goal is to exploit the slack time. Since in battery-powered systems, the battery lifetime impacts the utility and duration of the system directly, reducing the energy consumption and extending the battery lifetime should be a primary design metric [16].

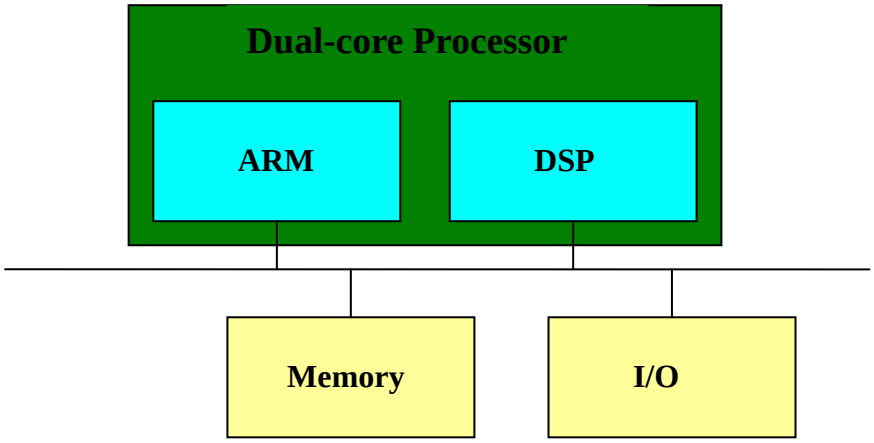


Fig. 1 Heterogeneous dual-core architecture [6].

We know that the energy consumption E of a CMOS circuit is dominated by its supply voltage and is proportional to the square of its supply voltage, which is expressed as $E = C_{eff} \cdot V_{dd}^2 \cdot C$ [2], where C_{eff} is the effective switched capacitance, V_{dd} is the supply voltage, and C is the number of execution cycles. Reducing the supply voltage also drops the maximum operating frequency proportionally ($V_{dd} \propto f$). Thus, E could be approximated as being proportional to the operating frequency squared ($E \propto f^2$). Therefore, lowering operating frequency and according supply voltage is an effective technique for reducing energy consumption [14].

In real-time systems with periodic tasks, no deadline miss is an important requirement of the systems. For example, embedded real-time systems must complete the tasks before their deadlines to maintain the system stability. Energy-efficient scheduling for hard real-time tasks on DVS processors is to minimize the energy consumption, while all the real-time tasks are done in time.

In this paper, we focus on low power scheduling for heterogeneous dual-core embedded real-time systems. In contrast to most of existing low power scheduling approaches that were targeted at only one CPU or homogeneous multi-core systems, we consider low power scheduling for heterogeneous dual-core embedded real-time systems.

The rest of the thesis is organized as follows. Chapter 2 includes DVS and scheduling preliminaries. Chapter 3 reviews related work. The system model and the proposed design approach are described in Chapter 4. Simulation results are discussed in Chapter 5 and conclusions and future work are given in Chapter 6.

Chapter 2
Preliminaries

DVS exploits the slack time to adjust the CPU frequency and voltage levels in order to reduce the energy consumption and guarantee all tasks completed before the deadlines. Therefore, a good slack time estimation method is very important to reduce energy consumption.

2.1 Categories of inter-task DVS strategies

There are two categories of DVS algorithms [3]: *inter-task DVS* and *intra-task DVS*. The inter-task DVS algorithm adjusts the CPU frequency task-by-task, which allocates the slack time between the current task and the following tasks. And the intra-task DVS algorithm adjusts the CPU frequency within a task, which uses the slack time when a task is predicted to complete before its worst-case execution time (WCET).

In this thesis, we consider inter-task DVS algorithms for periodic tasks, which usually exploit one or more of the following four strategies to estimate the slack time.

(1) Minimum constant speed [3] [7]

This strategy is defined as the lowest possible clock speed that guarantees the feasible scheduling of the task set.

(2) Stretching to NAT [3] [7]

This strategy is based on that the scheduling already knows the next task arrival time (NAT) of periodic tasks.

(3) Priority-based slack stealing [3] [7]

Not all the execution times of tasks are in the worst cases. If high priority tasks complete earlier than their WCETs, the next lower priority task can use the remaining slack time to adjust the frequency.

(4) Utilization updating [3] [7]

The utilization updating technique estimates the required processor performance at the current scheduling point by recalculation the expected worst case processor utilization using the actual execution times of completed task instances.

2.2 Priority scheduling

Existing real-time scheduling policies can be classified into rate-monotonic (RM) scheduler and earliest-deadline-first (EDF) scheduler. Both of them are dynamic scheduling.

(1) *Rate-Monotonic scheduling* (RM)

The RM scheduling is the fixed-priority scheduling. It always gives the highest priority to the task which has the shortest period in the ready queue.

(2) *Earliest-Deadline-First scheduling* (EDF)

The EDF scheduling is the dynamic-priority scheduling. It always gives the highest priority to the task which has the latest deadline in the ready queue.

Chapter 3
Related Work

Recently, to achieve high computation performance and lower energy consumption, the researches on multi-core embedded systems have become more and more popular [4] [5] [6] [10] [11] [15]. There are two categories of multi-core architecture. The cores in a given chip package are symmetric, called a homogeneous multi-core; otherwise, it is called a heterogeneous multi-core for asymmetric processors in a chip package.

In homogeneous multi-core systems, Alon et al. [4] shows that the total energy consumption of applications with single-thread (ST) is different from that of multi-thread (MT). It shows that the energy consumption by using an MT code is twice less than that by an ST code in corresponding performance-states and reduces the half of total execution time in Intel Core Duo systems. In the real-time loop task scheduling problem, Chen et al. [15] proposed the retiming and rotation algorithms. By reducing the task schedule length and using slack time, it can reduce more energy consumption and solve this problem.

In the periodic hard real-time tasks scheduling problem on heterogeneous dual-core systems composed by ARM core and DSP, Gai et al. [5] proposed a mechanism that divides the tasks into two groups: regular and DSP, where regular is the tasks without DSP workload and the one with DSP workload is called DSP, and each group has its corresponding queue: regular and DSP queues. It can increase the schedulability bound in the considered architecture and allow a more efficient use of the computational resources without still maintaining some kind of real-time guarantee. However, there are two problems in [5]. One is that the high priority DSP tasks will be blocked by the low priority DSP tasks and the other is that a regular task with low priority can be executed earlier than a DSP task with high priority. Due to these two problems, Kim et al. [6] proposed a new scheduling model using only a

queue combined with regular tasks and DSP tasks ordered by priority. It has better schedulability and fewer deadline misses.

In task critical problems with the mixed workload composed of periodic and aperiodic real-time jobs on heterogeneous distributed real-time embedded system, Marcus et al. [10] proposed an energy-efficient genetic list scheduling algorithm (EE-GLSA) and an energy-efficient genetic list mapping approach (EE-GMA) algorithm to get each aperiodic job's task mapping and find the shortest tasks scheduling length. It has higher energy reductions compared to previous DVS scheduling approaches based on constructive techniques and total energy savings for mapping and scheduling optimized DVS systems.

In the valid power-efficient scheduling based on task critical path analysis, Luo et al. [11] shows that the static and dynamic variable voltage scheduling algorithms in hard and soft real-time systems have hard task deadlines miss in heterogeneous distributed real-time embedded systems.

Table 1 shows the comparison of several existing low power DVS algorithms, and the proposed P-LCET (preemptive longer common execution time) and NP-LCET (non-preemptive longer common execution time) algorithms for heterogeneous dual-core embedded real-time systems. The metric of *multi-core type* describes if the multi-core is homogeneous or heterogeneous. The metric of *total energy consumption* indicates the CPU total energy consumption using each algorithm. The metric of *number of preemptions* indicates the frequency of preemptions. The metric of *average waiting time* indicates the average queuing time of tasks. The metric of *deadline miss* indicates the real-time tasks can not complete before the time constraint. In Chapter 5, we will compare our proposed P-LCET and NP-LCET with the work by Kim et al. [6], since it has no deadline miss.

Table 1 Comparison of related work.

Algorithm	Multi-core type	Total energy consumption	Number of preemption	Average waiting time	Deadline miss
Intel dual-core [4]	Homogeneous	Low	N/A	N/A	N/A
ILOSA [15]	Homogeneous	Low	N/A	N/A	N/A
EE-GLSA [10]	Heterogeneous	Medium	N/A	N/A	N/A
S-and-D [11]	Heterogeneous	Medium	N/A	N/A	N/A
Gai et al. [5]	Heterogeneous	High	Medium	Medium	Yes
Kim et al. [6]	Heterogeneous	High	Medium	Low	No
P-LCET (proposed)	Heterogeneous	Low	Medium	Low	No
NP-LCET (proposed)	Heterogeneous	Lowest	Low	Medium to High	Yes

Chapter 4
Proposed Low Power Scheduling Algorithm

4.1 System model

The target architecture is a heterogeneous dual-core embedded real-time systems, composed by ARM and DSP cores, that can change their supply voltage and operating frequency continuously within its operational ranges, $[V_{\min}, V_{\max}]$ and $[f_{\min}, f_{\max}]$, and all cores need to be executed at the same frequency [8]. A task set T of n periodic tasks is denoted as $T = \{T_1, T_2, T_3, ..., T_n\}$. Each task T_i has its own period p_i and worst-case execution time (WCET) w_i. The deadline d_i of T_i is assumed to be equal to its period p_i. The j^{th} instance of T_i is denoted by $T_{i,j}$. Each task releases its instance periodically and all tasks are assumed to be mutually independent [14].

The task model we used for low power scheduling with DVS is illustrated in Fig. 2. Each task executes for C_i units of time on the master CPU, and may request a DSP activity for C_i^{DSP}. We assume that each task performs at most one DSP request, after C_i^{pre} units of time, and then execution for other C_i^{post} units, such as $C_i = C_i^{pre} + C_i^{post}$ [6].

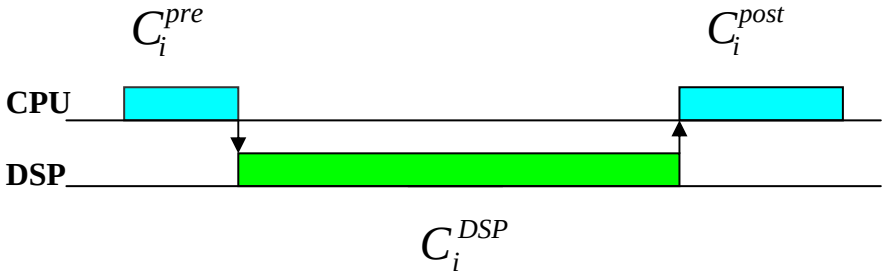


Fig. 2 Heterogeneous dual-core task model.

Besides, in consideration to hardware power saving support, ACPI spec. (Advanced Configuration and Power Interface Specification) [9] is the most used technique for the CPU. The power states transition flow diagram of ACPI is shown in Fig. 3 and detailed descriptions are shown as follows.

4.2 State definitions [9]

G_0 *Working*

In this state, peripheral devices (peripherals) are having their power states changed dynamically.

C_0 *Processor Power State*

While the processor is in this state, it executes instructions.

P_0 *Performance State*

While a device or processor is in this state, it uses its maximum performance capability and may consume maximum power.

P_1 *Performance State*

In this performance state, the performance capability of a device or processor is limited below its maximum and consumes less than maximum power.

P_n *Performance State*

In this performance state, the performance capability of a device or processor is at its minimum level and consumes minimal power while remaining in an active state.

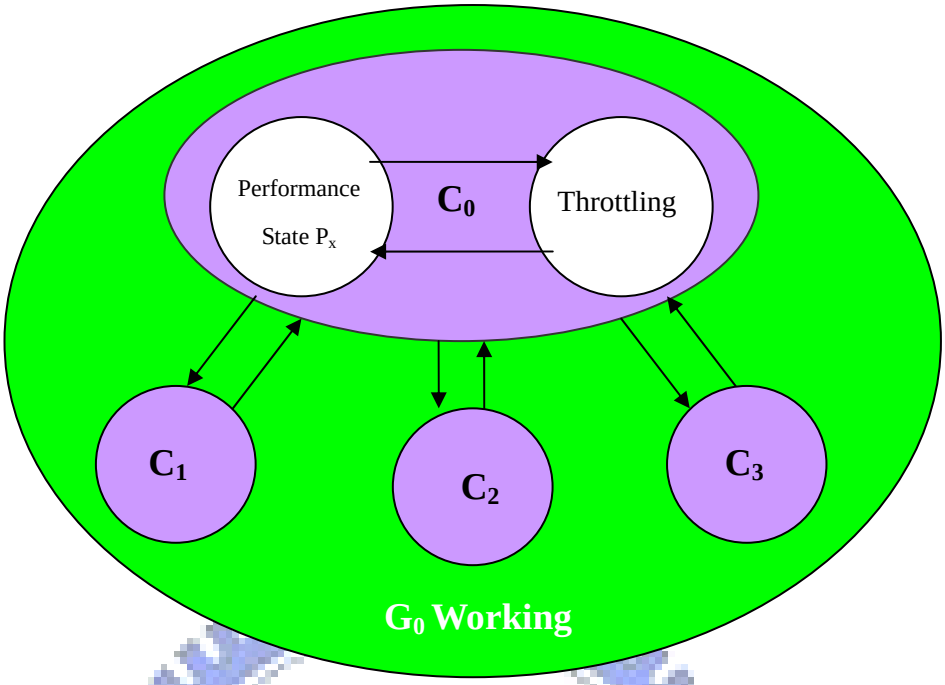


Fig. 3 Processor power states of ACPI [9].

And the hardware support constraint in our assumption with ACPI [9] is the same as that of [8]. All cores that follow ACPI in one physical package and reside in the same power domain must execute at the same performance state (P-state). It means if one core is busy running a task at P_0, other cores in that package can't enter lower P-states.

4.3 Problem statement

In this thesis, we propose a *longer common execution time* (LCET) algorithm to reduce the total execution time of tasks and total energy consumption in heterogeneous dual-core embedded real-time systems. From [4], we know that if tasks could execute more concurrently in the multi-core, the total execution time of the tasks and the total energy consumption can be decreased.

To illustrate our LCET algorithm we use an example which has two tasks, task 1 and task 2, with different priorities and execution times to be run in a dual-core, composed of ARM and DSP, as shown in Fig. 4.

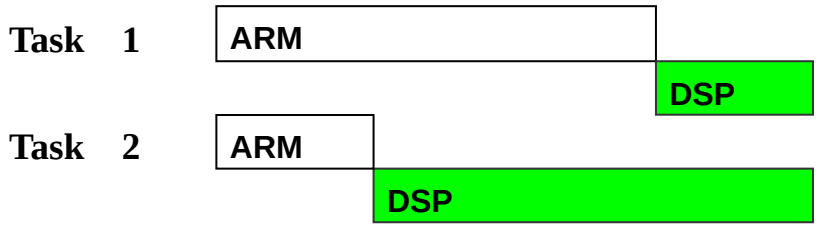
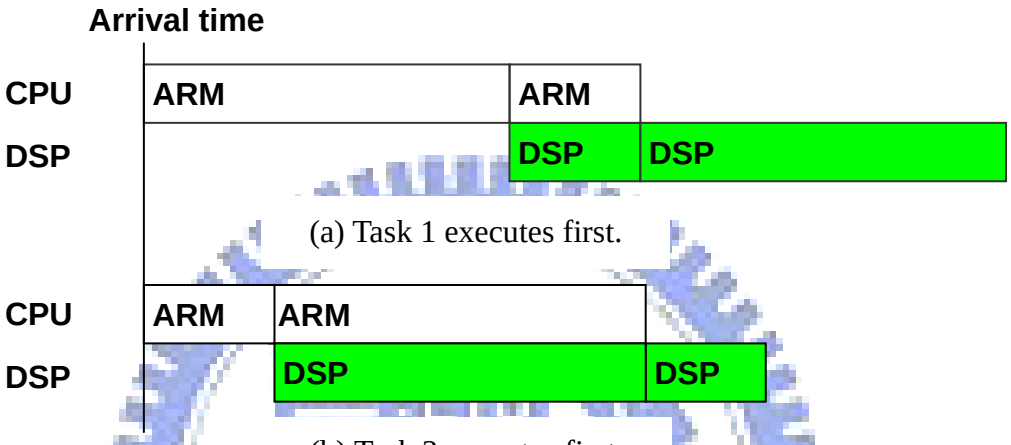


Fig. 4 An example: two different tasks.



(a) Task 1 executes first.

(b) Task 2 executes first.

Fig. 5 Different scheduling policies.

Fig. 5 shows that different scheduling priorities have different total execution time and total energy consumption for task 1 and task 2. We assume the two tasks arrive at the same time. If task 1 has higher priority than task 2 and executes first, we can see that the total execution time would become longer than that if task 2 executes first, and thus have higher total energy consumption. Therefore, the tasks, which have a different structure of C_i^{pre} and C_i^{DSP}, at different scheduling priorities will affect the total execution time and total energy consumption. To deal with this problem, we propose a priority scheduling algorithm which intends to improve the total execution time and the total energy consumption in heterogeneous dual-core systems.

4.4 Proposed mechanisms: P-LCET and NP-LCET

Before describing our algorithm, we first define the scheduling priority as shown in equation (1):

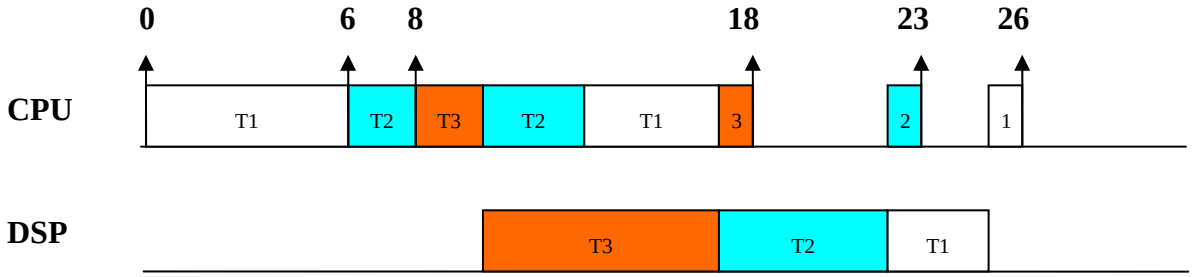
$$scheduling \quad priority : C_i^{pre} / C_i^{DSP} / C_i^{post} \quad \text{.....(1)}$$

We set task *i* with scheduling priority $C_i^{pre} / C_i^{DSP} / C_i^{post} < 1$ to have higher priority than task *j* with $C_j^{pre} / C_j^{DSP} / C_j^{post} > 1$. For those tasks with $C_i^{pre} / C_i^{DSP} / C_i^{post} < 1$, we set the shorter C_i^{pre} to have higher priority. Similarly, for tasks with $C_i^{pre} / C_i^{DSP} / C_i^{post} > 1$, we set the longer task C_i^{DSP} to have higher priority. The objective is that we want to find a scheduling solution of tasks with the shortest total execution time.

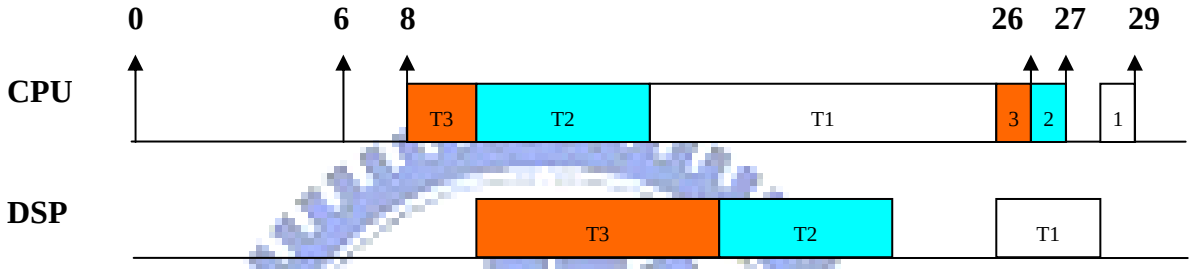
We propose two different scheduling algorithms: *preemptive LECT* (P-LCET) and *non-preemptive LECT* (NP-LCET) with/without consideration of the preemptive policy, which are shown in Fig. 8 and Fig. 9, respectively. In the P-LCET algorithm, a high priority task can preempt the running task with low priority. However, in the NP-LCET algorithm, we need to wait for all the tasks arrived before executing NP-LCET.

Table 2 Example task set.

Task	C^{pre}	C^{DSP}	C^{post}	Period
T1	10 ms	3 ms	1 ms	40 ms
T2	5 ms	5 ms	1 ms	40 ms
T3	2 ms	7 ms	1 ms	40 ms



(a). An example of P-LCET.



(b). An example of NP-LCET.

Fig. 6 Task scheduling for P-LCET and NP-NCET.

We use an example with three tasks, as shown in Table 2, to illustrate the P-LCET and NP-LCET algorithms. Fig. 6 (a) shows the scheduling result of the P-LCET algorithm. First, task 1 arrives and starts running at t = 0. When task 2 arrives at t = 6, task 2 has a higher priority (5/5) than task 1 (4/3) and it preempts task 1. When task 3 arrives, due to having a higher priority (2/7) than task 2 (3/5), task 2 will be preempted, and the total execution time is 25 *ms*. In Fig. 6 (b), because the NP-LCET algorithm needs to wait for all the tasks arrived, the execution order of the three tasks is, task 3, task 2 and task 1, based on their scheduling priorities 2/7, 5/5 and 10/3, and the total execution time is 21 *ms*.

Fig. 6 (b) shows that the scheduling result of NP-LCET has the shortest total execution time of CPU and DSP and the lowest total energy consumption. Although we know that the NP-LCET algorithm has the better scheduling result, it will increase the average waiting time due to waiting for the next task arrived. In embedded real-time

systems, in order to ensure that each task can complete its work before its deadline and avoid having longer average waiting time, we use two different thresholds, called *Task-Threshold* (T_{TH}) and *Timer-Threshold* (T_{TR}), to solve these problems, as shown as Fig. 7, where T_{TH} is the bound of tasks number which can wait and be scheduled in the ready queue and T_{TR} is the time interval which can wait until the next task arrival.

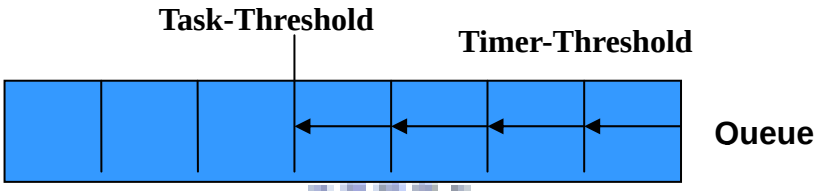


Fig. 7 Two thresholds used in the ready queue.

NP-LCET **:**

Input: $T_i = \{T_1, T_2, T_3, ..., T_n\}$, $U_{worst} = \sum_{i=1}^{n} w_i / p_i$.

Output: Total execution time (T_{ET}) and average waiting time (T_{WT}) of tasks.

1. Set T_{TH} and T_{TR}.
2. for (i = *1* to n)

    if (T_i) arrives;

        task number++;

        Schedule (T_i) by the scheduling priority in the ready queue.

        if (task number == T_{TH} || T_{TR} expire)

            T_{ET} = total execution time of the tasks in the queue;

            T_{WT} = average waiting time of the tasks in the queue;

            $f_{ar} = (T_{ET} + T_{WT}) / \sum_{i=1}^{n} w_i$

            $f = f_{ar} \times U_{worst} \times f_{max}$

Fig. 8 Algorithm of NP-LCET.

P-LCET **:**

Input: $T_i = \{T_1, T_2, T_3, ..., T_n\}$, $U_{worst} = \sum_{i=1}^{n} w_i / p_i$.

Output: Total execution time (T_{ET}) and average waiting time (T_{WT}) of tasks.

1. for (i = *1* to n)

    if (T_i) arrives;

        if (priority of (T_i) > priority of the running task (T_j))

            Preempt and schedule (T_j) by the scheduling priority in the queue.

        else

            Schedule (T_i) by the scheduling priority in the queue.

T_{ET} = total execution time of the tasks in the queue;

T_{WT} = average waiting time of the tasks in the queue;

$f_{ar} = (T_{ET}) / \sum_{i=1}^{n} w_i$

$f = f_{ar} \times U_{worst} \times f_{max}$

Fig. 9 Algorithm of P-LCET.

Before applying DVS to our proposed algorithms, NP-LCET and P-LCET, we first need to estimate the worst-case utilization U_{worst}, which can be computed by equation (2), as follows:

$$U_{worst} = \sum_{i=1}^{n} w_i / p_i \quad \text{.....(2)}$$

where n is the number of tasks in the task set, w_i is the worst-case execution time of task T_i, and p_i is the period of task T_i. Based on the scheduling results of the P-LCET and NP-LCET algorithms, we use the total execution time (T_{ET}), average waiting time (T_{WT}) and the worst-case execution time ($\sum_{i=1}^{n} w_i$) to derive a frequency adjustment ratio (f_{ar}), as shown in equations (3) and (4), respectively:

In NP-LCET:

$$f_{ar} = (T_{ET} + T_{WT}) / \sum_{i=1}^{n} w_i \quad \text{.....(3)}$$

In P-LCET:

$$f_{ar} = (T_{ET}) / \sum_{i=1}^{n} w_i \quad \text{.....(4)}$$

And the operating frequency (*f*) is set according to equation (5)

$$f = f_{ar} \times U_{worst} \times f_{\max} \quad \text{.....(5)}$$

where $f_{\max}$ is the maximum CPU frequency.

For the task set in Table 2, if we use the maximum frequency and set $f_{\max} = 1$ to run the task scheduling as shown in Fig. 6 (a), the total energy consumption is 26 *mJ*. If we use the minimum constant speed, $f = U_{worst} \times f_{\max}$, and $U_{worst} = \sum_{i=1}^{n} w_i / p_i = 35/40$, we have the operating frequency $f = 35/40$. Notice that by lowering the frequency, the total execution time of tasks will increase. So in Fig. 6 (a), the total execution will become 29.71 *ms* and the total energy consumption will become 22.75 *mJ*. Using our P-LCET in Fig. 6 (a), we can get $f_{ar} = (T_{ET}) / \sum_{i=1}^{n} w_i = 26/35$. So the operating frequency can be changed to $f = f_{ar} \times U_{worst} \times f_{\max} = 26/40$ and the total execution time will become 40 *ms*. The total energy consumption is 19.31 *mJ*. In Fig. 6 (b), our algorithm, NP-LCET, can get the shortest total execution time of 21 *ms* and the average waiting time of 11 *ms* (T1 = 15 *ms*, T2 = 10 *ms*, T3 = 8 *ms* and average waiting time T_{WT} = (15 + 10 + 8) / 3 = 11 *ms*). So we can get $f_{ar} = (T_{ET} + T_{WT}) / \sum_{i=1}^{n} w_i = 32/35$ and $f = f_{ar} \times U_{worst} \times f_{\max} = 32/40$. After we change the frequency, the total execution will become 26.25 *ms* and the total energy consumption will become 16.8 *mJ*.

Chapter 5
Simulation Results and Discussion

5.1 Simulation model

In the simulation, we assume the heterogeneous dual-core can change its operating frequency and supply voltage continuously within its operational ranges, $[f_{\min}, f_{\max}]$ and $[V_{\min}, V_{\max}]$, and all cores need to be execute at the same frequency [8]. The task sets were generated using random parameters with uniform distribution with the following characteristics [5] [6]:

- The number of tasks was chosen as a random variable from 10 to 50.
- Task periods were generated from 10 to 100 *ms*.
- Tasks which perform at most one DSP request needs to be executed.
- The worst-case execution times were selected in such a way that the worst-case utilization $\sum_{i=1}^{n} w_i / p_i$ varied from 0.01 to 0.99.
- C_i^{DSP} was generated to be a random variable with uniform distribution in the range of 10% to 80% of tasks.
- T_{TR} was set to 10 *ms*.
- T_{TH} was ranging from 2 to 6.

In the following discussions, we have normalized all the simulation results of total energy consumption to that of all tasks with the worst-case execution time $\sum_{i=1}^{n} w_i$ using the maximum frequency, $f_{\max}$.

5.2 Effects of worst-case utilization on total energy consumption

Fig. 10 compares the total energy consumption of NP-LCET under different T_{TH} and of P-LCET under different worst-case utilization U_{worst}.

- P-LCET reduces the total energy consumption by an average of 8% compared with Kim et al. [6] using minimum constant speed.
- NP-LCET reduces the total energy consumption by an average of 16%, 25%, 20%, compared with Kim et al. [6] using minimum constant speed as T_{TH} was set to be 2, 4, and 6, respectively.
- The result with T_{TH} = 6 has the worse total energy consumption than that with T_{TH} = 4 because higher average waiting time will affect the derivation of f_{ar}.

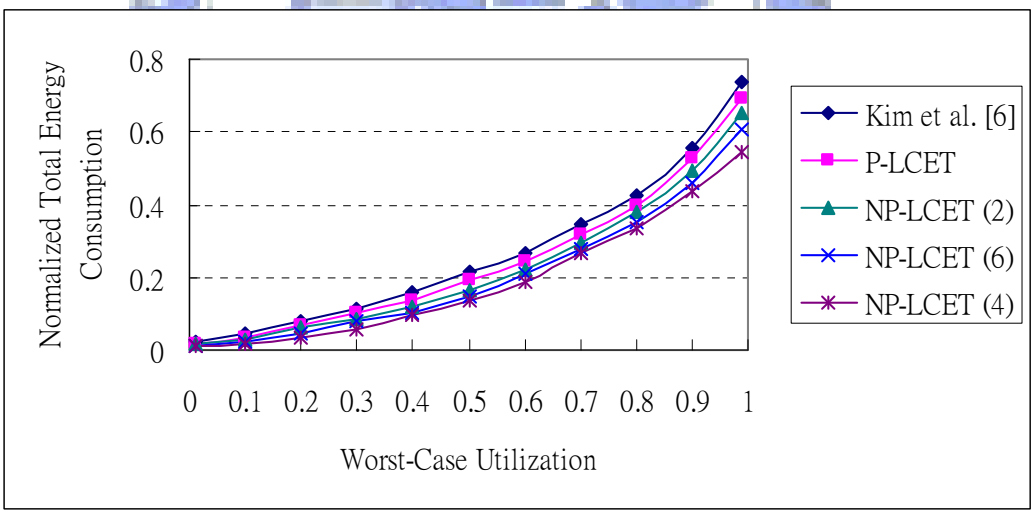


Fig. 10 Total energy consumption under different worst-case utilization.

5.3 Effects of worst-case utilization on average waiting time

Fig. 11 compares the average waiting time of NP-LCET under different T_{TR} and of P-LCET under different worst-case utilization U_{worst}.

- NP-LCET increased the average waiting time by an average of 40%, 71% and 113%, compared with Kim et al. [6], as T_{TR} was set to be 2, 4, and 6, respectively.
- From simulation results, we know that NP-LECT will have more average waiting time due to that NP-LCET needs to wait until the number of arrival tasks is equal to T_{TR}.

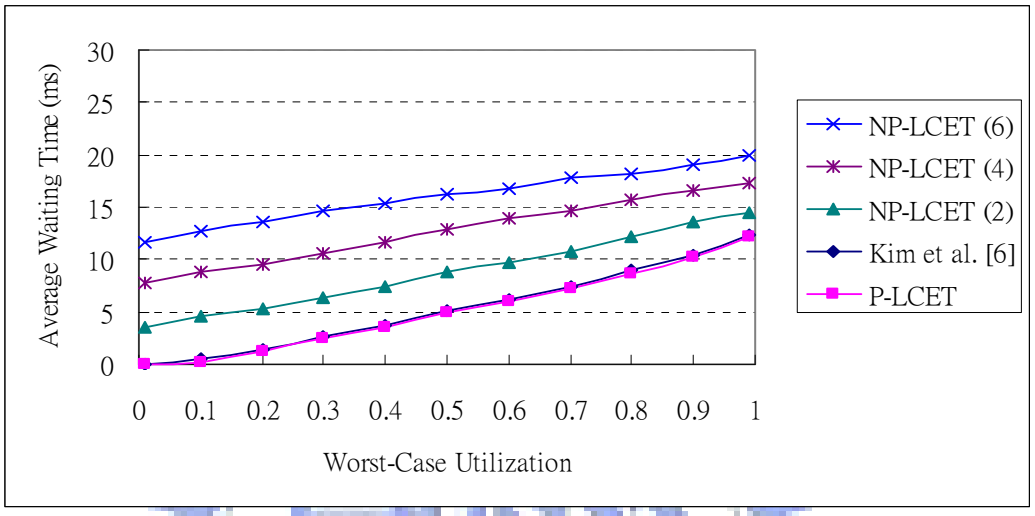


Fig. 11 Average waiting time under different worst-case utilization.

5.4 Effects of worst-case utilization on deadline miss

Fig. 12 compares the deadline miss of NP-LCET under different T_{TR} and of P-LCET under different worst-case utilization U_{worst}.

- The percentage of tasks completed before the deadline using NP-LCET decreases slightly when the worst-case utilization increases. This is because that when T_{TH} or the worst-case utilization increases, the average waiting time of each task will also increase. So it will result in the increase of deadline miss.

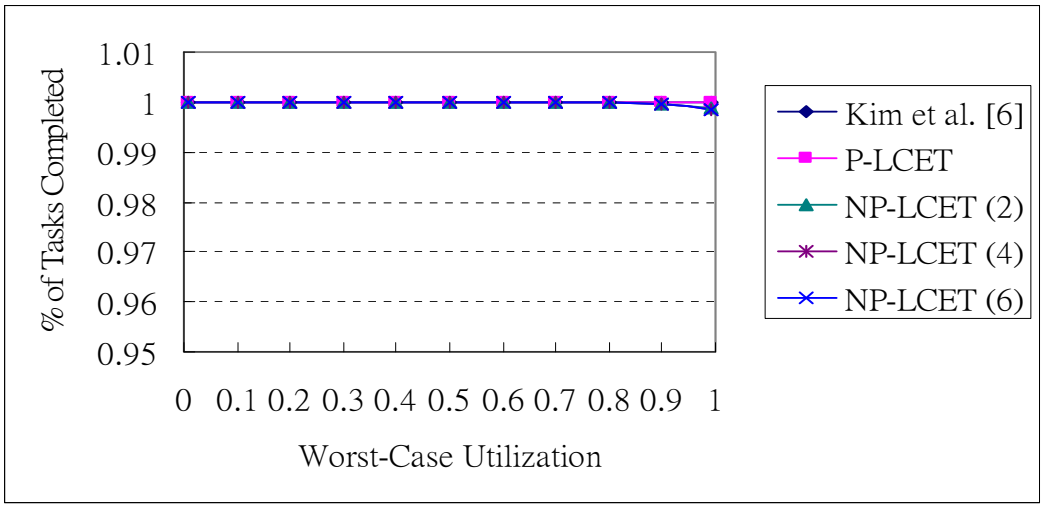


Fig. 12 Deadline miss under different worst-case utilization.

5.5 Effects of task number on total energy consumption

Fig. 13 compares the total energy consumption of NP-LCET under different T_{TR} and of P-LCET under different task numbers. The worst-case utilization was set to 95%.

- We observed that the total energy consumption improvement of the Kim et al. [6] and P-LCET are almost no change when the number of tasks increases. It is because that the DVS algorithm relies on the worst-case utilization, but not on the task number.
- The total energy consumption improvement using NP-LCET decreases conspicuously when the number of tasks in a task set increases. It is because the task arrival frequency increases duo to lower average waiting time and a lower frequency used.

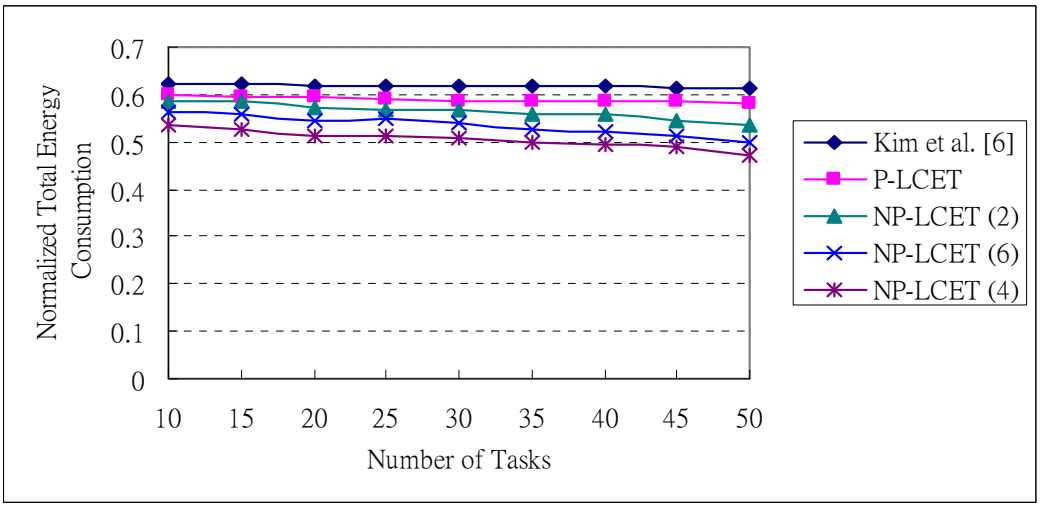


Fig. 13 Total energy consumption under different number of tasks.

5.6 Effects of task number on average waiting time

Fig. 14 compares the average waiting time of NP-LCET under different T_{TH} and of P-LCET under different task numbers. The worst-case utilization was set to 95%.

- The average waiting time of NP-LCET algorithms decreased clearly when the number of task increases due to the increased task arrival frequency.

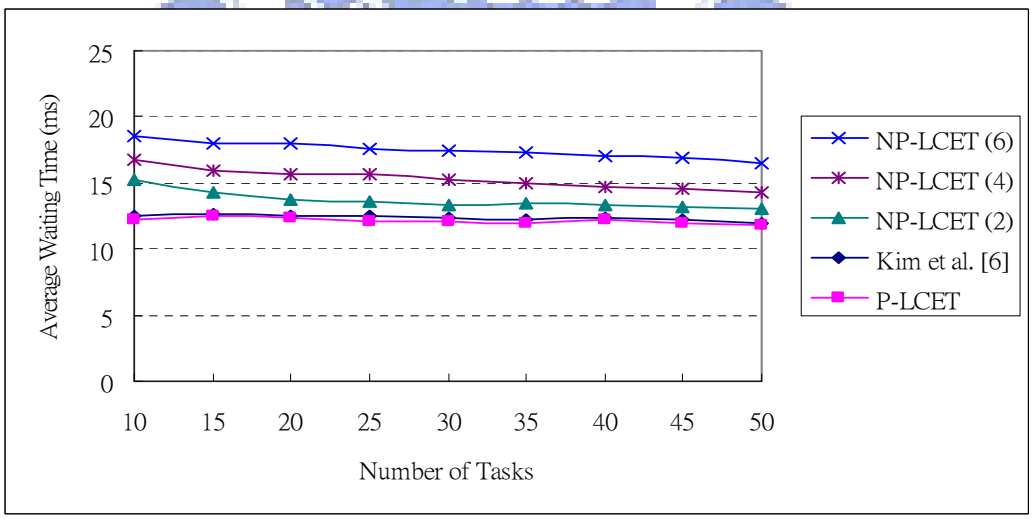


Fig. 14 Average waiting time under different number of tasks.

Chapter 6
Conclusions and Future Work

6.1 Concluding remarks

In this thesis, we have presented two efficient low power scheduling techniques, called P-LCET and NP-LCET, for heterogeneous dual-core embedded real-time systems. Our design approach was motivated by the heterogeneous dual-core architecture, task scheduling and DVS techniques. The main contribution of the P-LCET and NP-LCET is that the two proposed scheduling algorithms for heterogeneous dual-core systems have better power saving and less total execution time than existing approaches. The proposed NP-LCET has better power saving and no preemptive overhead, but has higher average waiting time and deadline misses. The proposed P-LCET has less power saving than NP-LCET, but the average waiting time will be lower and no deadline miss. Besides, the overhead of P-LCET is an increase in the number of preemptions, which results in increased energy consumption and clock cycles. Fortunately, these overheads are usually small enough and can be neglected.

6.2 Future work

In this thesis, the proposed LCET has two different policies, preemptive LCET (P-LCET) and non-preemptive LCET (NP-LCET). We know that to have shorter total execution time and lower total energy consumption for all tasks is using NP-LCET. But it will have higher average waiting time. The other scheduling policy, P-LCET, may not have better power saving than NP-LCET, but the average waiting time will be lower. How to integrate these two policies to have low total energy consumption, low waiting time, and considering the mixed workload as well is our future work. The performance and energy

consumption of the integrated approach deserve to further study.

Bibliography

[1] L. Miao, Y. Qi, D. Hou, C. I. Wu, Y. H. Dai, “Dynamic power management and dynamic voltage scaling in real-time CMP systems,” in *Proceedings of International Conference on Networking, Architecture, and Storage*, pp. 249 – 250, July 2007.

[2] B. Moyer, “Low-power design for embedded processors,” in *Proceedings of IEEE*, Volume 89, Issue 11, pp. 1576-1587, November 2001.

[3] W. Kim, D. Shin, H. S. Yun, J. Kim and S. L. Min, “Performance comparison of dynamic voltage scaling algorithms for hard real-time systems,” in *Proceeding of the Eighth IEEE on Real-Time and Embedded Technology and Applications Symposium*, pp. 219 – 228, Sept. 2002.

[4] “Intel dual-core” http://www.intel.com/technology/itj/2006/volume10issue02/art03_Power_and_Thermal_Management/p02_intro.htm.

[5] P. Gai, L. Abeni and G. Buttazzo, ”Multiprocessor DSP scheduling in system-on-a-chip architectures,” in *Proceedings of 14th Euromicro Conference on Real-Time Systems*, pp. 231 – 238, June 2002.

[6] K. Kim, D. Kim and C. Park, “Real-time scheduling in heterogeneous dual-core architectures,” in *Proceedings of 12th International Conference on Parallel and Distributed Systems*, pp. 1 - 6, July 2006.

[7] P. Pillai and K.G. Shin. “Real-time dynamic voltage scaling for low-power embedded operating systems,” in *Proceedings of 18th ACM symposium on Operating Systems*, pp. 89-102, October 2001.

[8] Intel multi-core, http://www.intel.com/technology/itj/2007/v11i4/9-process/6-linux-scheduler.htm.

[9] ACPI spec., http://www.acpi.info/DOWNLOADS/ACPIspec30a.pdf.

[10] M.T. Schmitz, B.M. Al-Hashimi and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in *Proceedings of Europe Conference and Exhibition on Design, Automation and Test*, pp. 514 – 521, March 2002.

[11] J. Luo and J. N, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," in *Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design*, pp. 719 – 726, Jan. 2002.

[12] OMAP processor, http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11991&contentId=4670.

[13] Freescale i.300-30 processor, http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.300-30&nodeId=01J4Fsm6cyDbFf.

[14] J. M. Chen, K. Wang and M. H. Lin, "Energy efficient scheduling for real-time systems with mixed workload", in *Proceedings of International Federation for Information Processing*, pp. 33–44, Dec. 2007.

[15] Y. Chen, Z. Shao, Q. Zhuge, C. Xue, B. Xiao and E.H.-M. Sha, "Minimizing energy via loop scheduling and DVS for multi-core embedded systems" in *Proceedings of 11th International Conference on Parallel and Distributed Systems*, pp, 2–6, July 2005.

[16] C. Yuan, S.M. Reddy, I. Pomeranz and B.M. Al-Hashimi, "Battery-aware dynamic voltage scaling in multiprocessor embedded system" in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 616 – 619, May 2005.