國 立 交 通 大 學

網路工程研究所

碩 士 論 文

在多核心即時嵌入式系統下考量系統負載之高能源效率動態電壓調整排程演算法

Energy Efficient Workload-Aware DVS Scheduling for Multi-core Real-time Embedded Systems

研 究 生：林明翰

指導教授：王國禎　教授

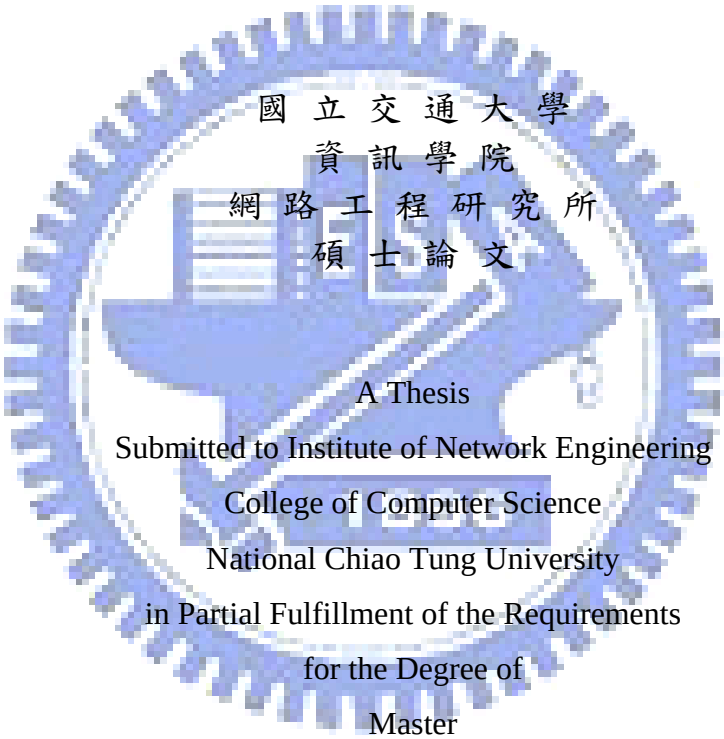
中 華 民 國　九 十 七　年 六 月

在多核心即時嵌入式系統下考量系統負載之高能源效率動態電壓調整排程演算法

Energy Efficient Workload-Aware DVS Scheduling for Multi-core Real-time Embedded Systems

研 究 生：林明翰　　Student：Ming-Ham 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 2007

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

在多核心即時嵌入式系統下考量系統負載之高能源效率動態電壓調整排程演算法

學生：林明翰　　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

在多核心即時嵌入式系統中，記憶體是一個重要的共享資源。因為需要等待記憶體的需求被服務，核心之間互相競爭記憶體會使得總執行時間增長。在本篇論文中，我們研究在考量記憶體競爭下之多核心即時嵌入式系統的任務分割排班問題。在考慮各任務的記憶體工作量特性下，我們提出一個改善現有基於最大工作量任務優先方法(LTF-MES)之高能源效率排程演算法，叫做考量系統負載之高能源效率動態電壓調整排程演算法。而我們提出的演算法和最大工作量任務優先之演算法最大的不同在於，我們考慮任務的執行順序，可因此減少核心之間相互競爭記憶體的頻率。實驗結果顯示，藉由減少任務與任務之間互相競爭記憶體，可以增加寬裕時間。

而且，我們的演算法可以利用這些寬裕時間來減少在擁有不同任務數量的多核心系統中所需的總執行時間和總能源消耗。在變動的工作數量以及核心數目在 2 到 16 個之間的環境中，使用本演算法相對於 LTF，可以在沒有使用動態調整電壓的情況下，降低 2 %到 10.3 %的總執行時間，並且可以在有支援動態調整電壓的情況下，相對於 LTF-MES，改善 3.85%到 19% 的總能源消耗。

關鍵詞：動態電壓調整，多核心，嵌入式即時系統，考量系統負載。

Energy Efficient Workload-Aware DVS Scheduling for Multi-core Real-time Embedded Systems

Student：Ming-Ham Lin u **Advisor：Dr. Kuochen Wang**

Department of Computer Science
National Chiao Tung University

Abstract

***Abstract*—**Memory is an important shared resource in a multi-core real-time embedded system. The memory contentions between cores will lengthen the total execution time due to waiting for memory requests being served. In this thesis, we focus on the tasks partition scheduling problem while considering memory contentions in multi-core real-time embedded systems. We propose an energy efficient scheduling mechanism with consideration to the memory workload of tasks, called WAS-DVS (workload-aware scheduling-dynamic voltage scaling), which is an improvement of an existing method, LTF-MES (Largest-Task-First-Minimize-Energy-Scheduling). The main difference between ours and LTF-MES is that we consider the execution order of tasks that may reduce the frequency of memory contentions. Simulation results show that by reducing memory contentions between tasks, the slack time will increase and the proposed WAS-DVS can use it to lower total execution time and total energy consumption on a variety of workloads in multi-core systems. The proposed WAS-DVS can lower the total execution time from 2% to 10.3% before applying DVS and improve the total energy consumption from 3.85% to 19% compared to LTF-MES, under various numbers of tasks and 2 to 16 cores after applying DVS.

Keywords: DVS, multi-core, real-time embedded system, workload-aware.

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-E009-140-MY3 and NSC96-2628-E-002-138-MY3 is also grateful acknowledged.

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to high computation performance and low energy consumption demands, the multi-core architecture which is the same as the chip-multiprocessor (CMP) has been proposed. With a slight increasing on the die size, multiple cores are mounted on a single die but with only comparatively low wire delay [13]. There are two categories of CMP architecture. The cores in a given chip package are symmetric, called a homogeneous CMP [1] [3] [4] [8]; otherwise, it is called a heterogeneous CMP for asymmetric processors in a chip package [10]. In this thesis, our focus is on homogeneous multi-core systems.

In multi-core (CMP) processor packages, each chip package contains two or more cores, and each core has its own resources (registers, execution units, some or all levels of caches, etc.) [1]. Design innovations of CMP architectures mainly span the area of shared resources (caches, power management, etc.) between cores, topologies (number of cores in a package, relationship between cores, etc.) [8]. To exploit optimal performance, the process scheduler needs to be aware of shared resources and task characteristics. Recent researches on multi-core (CMP) scheduling [10] [12] [13] seldom consider the shared memory behavior. However, accesses to the shared memory may take a significant fraction of the execution cycles, as well as of the total energy consumption.

DVS (dynamic voltage scaling) is the main technique to conserve power by scaling down the processor voltage and frequency when some unused idle periods exist in the schedule at run time. The voltage scheduler determines which voltage to be used by analyzing the state of the system [15]. That is, the voltage scheduler of the real-time system suggests the lowest possible level voltage without affecting the system performance (no deadline miss for

periodic tasks). A real-time task scheduling algorithm over multi-core with the capability of DVS is proposed in this thesis.

The rest of this thesis is organized as follows. Chapter 2 presents related work on energy-efficient scheduling. In Chapter 3, we formally define the problem, system model, assumptions, and notations. Our proposed energy efficient scheduling is presented in Chapter 4 and simulation results are discussed in Chapter 5. Finally, concluding remarks and future work are given in Chapter 6.

Chapter 2

Related work

Recently, as the multi-core processor [1] becomes popular, more and more people study the energy efficient scheduling problem on the multi-core (CMP) architecture. This problem is similar to the one on multiprocessors. Carpenter et al. [5] classified scheduling approaches on multiprocessors into *partitioning* and *global*. In partitioned scheduling algorithms, tasks are partitioned into many disjoint subsets, and each subset is associated with a unique processor. Aydin et al. [6] showed that balance workload of each processor has lower energy consumption. In global scheduling algorithms, all tasks are stored in a single priority-ordered queue and the scheduler selects the highest priority task for execution from this queue. Chen et al. [7] considered the task migration property and proposed a 1.13-approximation ratio scheduling algorithm on multiprocessors. In multi-core systems, Siddha et al. [7] showed some challenges on shared resources, task scheduling, etc. In task scheduling, Anderson et al. [4] proposed an algorithm that satisfies the real-time constraint. Miao et al. [12] used the TCSP (tri-dimensional coding based self-adaptive parallel) genetic algorithm to solve this NP-hard problem that each core can execute at a different frequency at the same time. And Yang et al. [13] also proposed an LTF (largest task first) algorithm with DVS to solve the partition scheduling problem that will achieve 2.371-approximation to the optimal solution. Unlike the above researches focusing on homogeneous CMP architecture, Leontyev et al. [10] proposed a mechanism which can be used for scheduling sporadic soft real-time task systems on asymmetric multi-core (heterogeneous) platforms with cores of different speeds. For task characteristics consideration, Yaldiz et al. [3] proposed an algorithm that considers the task co-relationship on homogeneous multi-core systems with soft real time tasks. Although this

mechanism will increase a little deadline miss rate, it will improve energy consumption. While discussing about shared resources, Ozturk et al. [14] proposed a two-step approach: (1) determining the amount of data that are shared by cores and the amount of data that are private to each core and (2) allocating memory space across private and shared data and over all cores, to improve the utilization of shared memory.

Table 1 shows the comparison of several existing energy efficient scheduling algorithms on multi-core systems and the proposed algorithm WAS-DVS. The metric of *multi-core type* describes that if the multi-core architecture is homogeneous or heterogeneous. The metric of *real-time* indicates that which real-time environment the algorithm could apply. There are two categories of real-time systems: hard and soft. In hard real-time system all the task need to be finished before its deadline constraint but in soft ones we could tolerate a little deadline miss. The metric of *memory contention* indicates that if the algorithm will consider or reduce the memory contentions. The metric of *core or chip* indicates that if the cores need to execute at the same frequency. In chip level all the cores need to execute at the same frequency but it is not necessary in core level.

In this thesis, we explore real-time energy-efficient scheduling on a multi core system with dynamic voltage scaling, which is based on LTF [13] and consider the workload characteristics of tasks. By reducing the shared memory access contentions, we can lower total execution time or execute all tasks with lower frequency than LTF-MEM [13] with the same deadline due to that ours has more slack time than compared.

Algorithm	Multi-core type	Real-time	memory contention	Core or Chip level
Anderson et al. [4]	Homogeneous	Hard	None	None
TCSP [12]	Homogeneous	Hard	N/A	Core
LTF-MES [13]	Homogeneous	Hard	N/A	Chip
Leontyev et al. [10]	Heterogeneous	Soft	N/A	Core
Yaldiz et al. [3]	Homogeneous	Soft	N/A	Chip
Ozturk et al. [14]	Homogeneous	N/A	Yes	N/A
WAS-DVS (Proposed)	Homogeneous	Hard	Yes	Chip

Table 1. Comparison of related works

Chapter 3

Problem Statement and System Model

3.1 Problem Statement

Most energy efficient scheduling schemes for CPUs simply ignore the memory behavior and assume that the execution time and energy consumption of the system are only determined by the CPUs. For real cases, the memory does contribute both execution time and energy consumption. In multi-core systems, each core shares with limited buses, shared caches and external memory [1][8]. A recent research shows that the proportion of total execution time will vary with shared memory access and contention [8]. Besides, tasks execution order will affect the energy consumption, system performance (deadline miss, and total execution time etc.) and memory access contentions. For an example in Fig. 1, the two processors, core 1 and core 2, have their own tasks T_1 and T_2, respectively. If T_1 and T_2 send memory access requests to the memory controller at time 5 and each request needs 5 time units. Then the controller schedules the requests with a larger core index first policy and the scheduling result is (T_2, T_1). The request of T_1 needs to wait for the one of T_2 being complete at time 10. The contention in this situation increases 5 time units delay for T_1 and core 1 just does nothing and waits for the request being severed during this time interval.

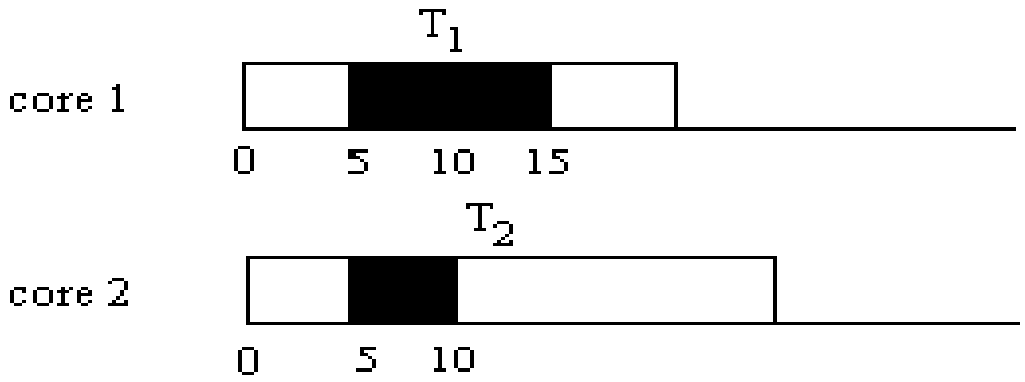


Fig 1 An example of memory contention with 2-core.

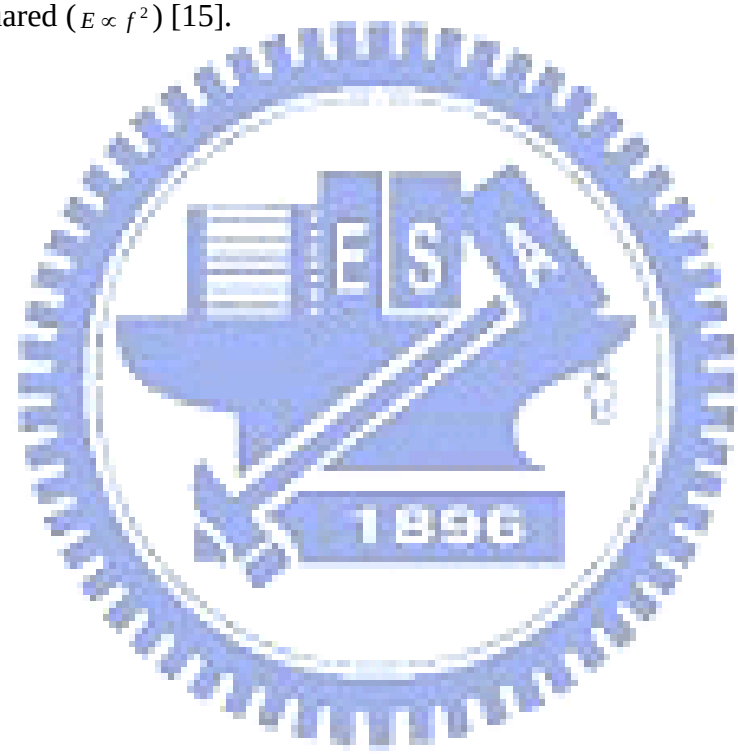
To resolve this problem, our proposed mechanism schedules tasks with consideration of its workload characteristics to reduce the total execution time, the memory access contentions, and energy consumption using DVS.

3.2 System Model, Assumptions, and Notations

We have a target m homogeneous cores [C_1, C_2,…, C_m] which share with an external memory [1] [13], a common timer t, and have their own job queue [Q_1, Q_2,…, Q_m]. These cores can change their supply voltage (V) and clock speed (S_{clk}) (or frequency) continuously within their operational ranges, [V_{cmin},V_{cmax}], and [S_{cmin},S_{cmax}]. In addition, all the cores must operate at the same supply voltage (frequency) and could go to idle state immediately [1] [8] [13] [15]. And each core has two variables TW_{Cj} and TW_{Mj}, where TW_{Cj} and TW_{Mj} are the total CPU and memory workloads of assigned tasks, respectively, where j is the index of a core from 1 to m. A set of n periodic tasks [T] = {T_1, T_2…..T_n} are ready at time 0 [13].

A periodic task can be specified as T_i (P_i, W_i, w_m), where P_i is the period, W_i is the WCET (worst case execution time), and w_m is the memory workload. Based on [2] [3] [9], the arrival time, period, WCET, and memory access times of periodic tasks are known in advance.

The relative deadline (D_i) of each periodic task instance i is assumed equal to a common deadline D and all tasks are mutually independent [13]. And the well known energy consumption E of a CMOS circuit we used is dominated by its dynamic supply voltage and is proportional to the square of its supply voltage, which is defined as $E = C_{eff} \cdot V_{dd}^2 \cdot C$, where C_{eff} is the effective switched capacitance, V_{dd} is the supply voltage, and C is the number of execution cycles. Degrading the supply voltage also drops the operating frequency proportionally ($V_{dd} \propto f$). Thus E could be approximated as being proportional to the operating frequency squared ($E \propto f^2$) [15].



Chapter 4

Proposed Energy Efficient Workload Aware DVS Scheduling Mechanism

Memory contentions vary with the tasks execution order during the same period of all cores. We take Fig 2 as an example to illustrate the situation. In Fig 2, core 1 and core 2 have their own tasks [T_1, T_3] and [T_2, T_4], respectively, which all start at time 0. The execution time of T_1, T_2, T_3, and T_4 are W_1 (50 cycles), W_2 (60 cycles), W_3 (30 cycles), and W_4 (30 cycles) while T_1, T_2, T_3, and T_4 have the numbers of memory requests m_1 (10 times), m_2 (12 times), m_3 (3 times), and m_4 (3 times), respectively. The probability p_1 of memory access request sent by T_1 is m_1 / W_1 in each cycle while p_2 is m_2 / W_2, p_3 is m_3 / W_3, and p_4 is m_4 / W_4. In Fig 2 (a), the expected value of memory contention is 0.2*0.2*50 + 0.1*0.1*10 +0.1*0.1*20 = 2.4 (times). However, in Fig 2 (b), if we exchange the task execution order of the job queue of core 1, the expected value of memory contention is 0.1*0.2*30 + 0.2*0.2*20 + 0.2*0.1*20 = 1.8 (times).

The variance of the task execution order will change the contention probability during the same period of all cores. We want to reduce memory contentions by reordering the execution order of tasks in the job queue of each core. However, the main challenges before proceeding scheduling are to identify and predict the resource needs of each task, and to schedule them with an aim to reduce shared resource contention and minimize energy consumption.

To achieve this, we propose a WAS (workload aware scheduling) algorithm which is

shown in Fig 3. At the beginning of the algorithm we need to sort the tasks by the information T_i (P_i, W_i, w_m), where P_i has the highest weight, w_m has the smallest. Our proposed WAS algorithm includes two parts. The first part is based on LTF [13]. LTF (largest task first) is an algorithm that partition [T] into M disjoint sets and each disjoint set corresponds to a job queue. According to the sorting result we find the task with the largest CPU workload and assign it to the core with the smallest load. And we then sort the cores by a non-increasing order with TW_c. To reduce memory access contentions we reorder the task execution order in each job queue. If index j of C is odd, reorder the task execution order in an increasing order with w_m. If an index is even, reorder by a non-increasing order. Our proposed WAS algorithm expects to avoid two tasks with high w_m executed at the same period. The complexity of LTF is O[n ($log\ n$ + $log\ m$) + m] and ours is O[n ($log\ n$ + $log\ m$) + m] + O[$n\ log\ (n/m)$].

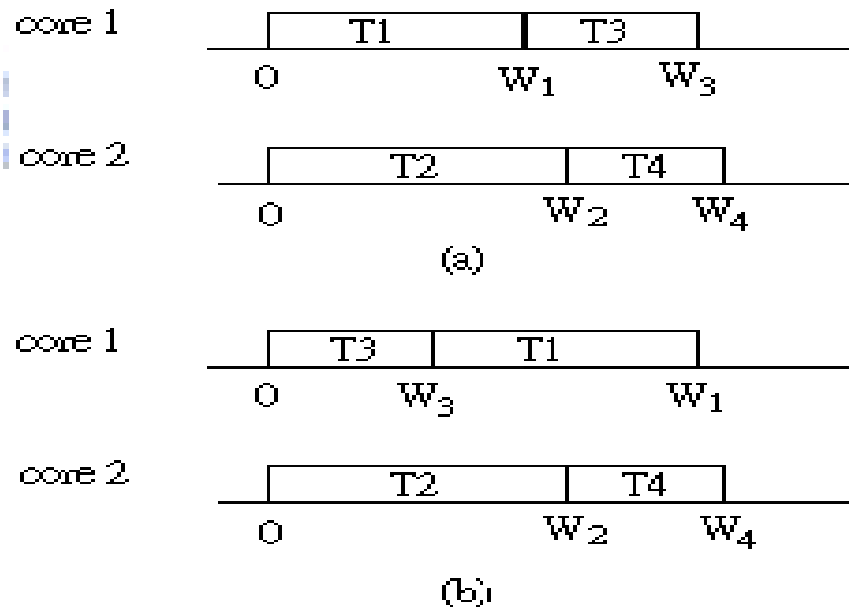


Fig 2. A motivation example: (a) original task execution order (b) new task execution order.

WAS :

Input : (T, C, D, Q) /* T: the task sets
Output : A feasible scheduling result Q /* C: the core sets
/* D: the common deadline
/* Q: the queue sets

1. sort T in a non-increasing order of $T_i(W_i)$;

2. for $i = 1$ to [T] do /* assign each task to
find the core C_j with the smallest load /* each core by LTF
$TW_{Cj} = TW_{Cj} + T_i(W_i)$
$TW_{Mj} = TW_{Mj} + T_i(w_m)$
$Q_i \leftarrow Q_i + T_i$

3. sort C by a non-increasing order with TW_{Mj}

4. for $j = 1$ to M /* resort the order of each
if j is even /* core by $w_{\backslash m}$
reorder the tasks in Q_j of C_j in
increasing order with w_m
else
reorder the tasks in Q_j of C_j in
non-increasing order with w_m

Fig 3. Algorithm of WAS.

In order to reduce the total energy consumption, we propose an energy efficient scheduling algorithm WAS-DVS, which is based on WAS and use DVS. Because each core needs to execute at the same supply voltage [1], the WAS-DVS needs to find the lowest frequency that satisfies all the deadline constraints. In DVS, due to the use of the online adjustment mechanism that dynamic collection of slack time for DVS using, we need to define two events: one is that all cores start to run at time 0 and the other is any task is finished. When any of the two events occurs, it will trigger the WAS-DVS algorithm to reassign the frequency to each core. An example is shown in Fig 4. In this example t_1 is the finished time of task 1; …; t_5 is the finished time of task 5. At starting time 0 we assign a frequency, which satisfies the deadline constraint of core 2 with the largest workload, if deadline miss of the remaining TW_{Cj} of each core after time t_1 will not arise due to this change. When task 1 is finished at t_1, we will reassign the frequency to each core while satisfying

deadline constraints of the core which has the largest remaining TW_{Cj}. The WAS-DVS algorithm is shown in Fig 5.

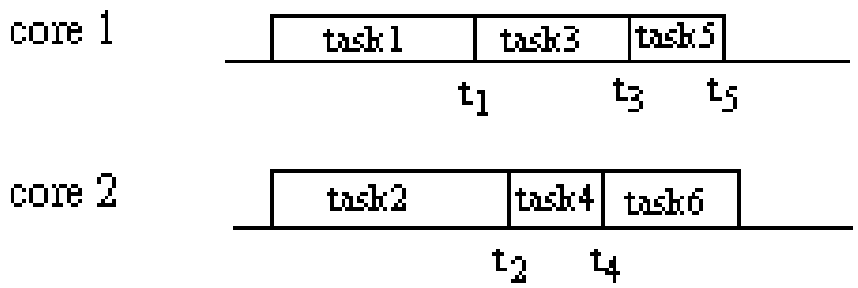


Fig 4. A task execution example of a dual core.

WAS-DVS :

Input : (*D*, *C*, *t*) /* D: the common deadline
/* C: the core sets
/* t : the current time

Output: A feasible *Q*-based schedule R with the minimum energy consumption

5. for *i* = *1* to [*T*] do **/*** estimate the remaining
   temp[TW_{Cj}] = TW_{Cj} **/*** workload of each core
   temp[TW_{Cj}] = temp[TW_{Cj}] - *t*

6. find the C_i with maximum temp[TW_{Cj}]

7. assign the $S \geqq$ temp[TW_{Cj}] / (*D* - *t*) to each C_j /* estimate the frequency
   /* and assign to each core

Fig 5. Algorithm of WAS-DVS.

Chapter 5

Simulation Results and Discussion

The purpose of this chapter is to provide performance comparison of algorithm WAS and WAS-DVS against with LTF and LTF-MEM (LTF-DVS). To evaluate our proposed algorithms, we use the real-time benchmark information, obtained from [11], which is shown in Table 2, where each row of Table 2 represents a category of tasks. Assume that all the arrival times of tasks are at time zero and all tasks share with a common deadline [13]. Processors can adjust their frequency continuously within [S_{min} , S_{max}] and share with an external memory with 20 cycles of access latency [11] [13]. In order to avoid the situation that the numbers of task is fewer than the numbers of cores, we randomly select the number ($p \times m + 1$) of tasks from all categories, where p is a rational positive integer and m is the numbers of cores. The p value stands for an average anticipant number of tasks in each job queue. In our simulation, all the simulation results of WAS and WAS-DLVS are normalized to those of LTF and LTF-MEM (LTF-DVS).

In order to observe the influence of different numbers of tasks on each environment with a different number of processors (cores), we vary the p value from 2 to 7. Simulation results in Fig 6 show that the execution time improvement of our proposed WAS-DVS is 2% to 10.3% better than that of LTF [13] under varies numbers of cores.

Task	WCET (cycles)	M (memory access times)
Fibcall	9536	114
Qsort	13309	97
Matmul	13985	26
IDCT	16131	193
FIR	33983	133
CRC	42907	99
FFT2	60234	2820
LUD	255998	102
LUD2	255998	1364
LMS	365893	123
LMS2	365893	14741
FFT	515771	38404
FIR2	557589	405
ADPCM	2486633	4053

Table 2. Offline information of tasks [11].

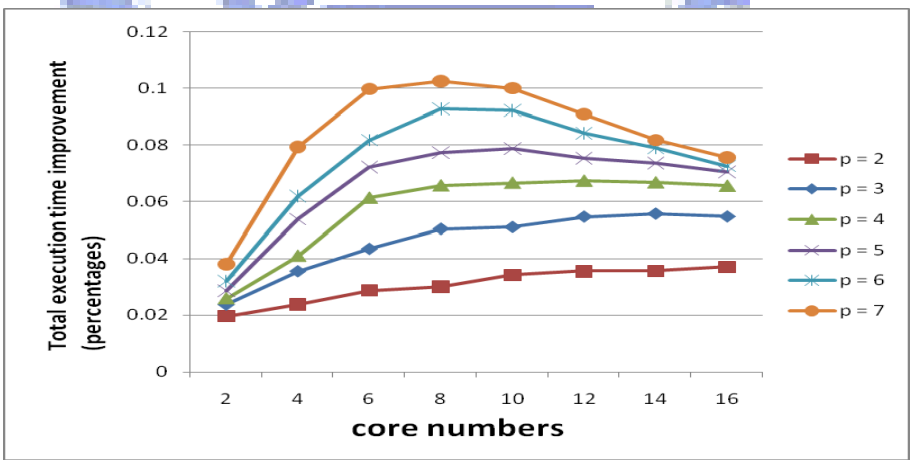


Fig 6. Total execution time improvement of WAS over LTF.

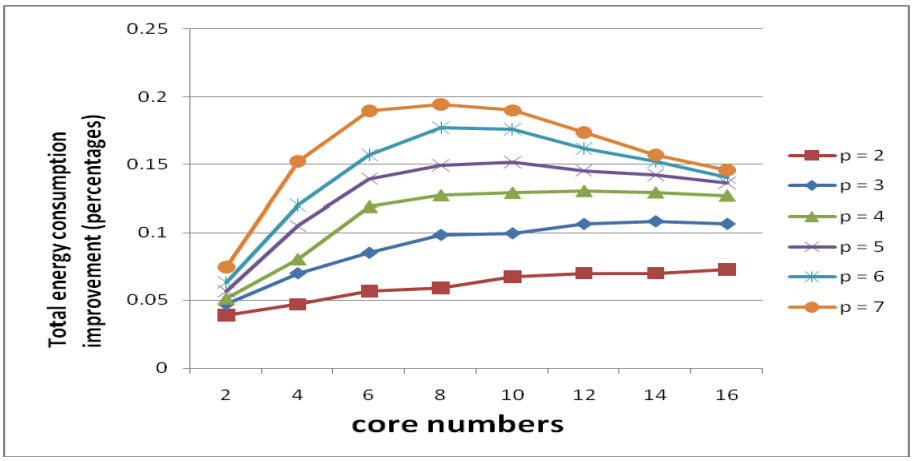


Fig 7. Total energy consumption improvement of WAS-DVS over LTF-DVS.

Note that the performance (in terms of total execution time) of our proposed WAS-DVS is close to that of LTF when *p* is small. This is because that there are few tasks in the job queue to be reordered using our proposed mechanism. On the contrary, the total execution time improvement percentage increases as *p* increases. But the increasing degree is fewer and fewer due to that the proportion of the amount of memory contention delay to total execution time decreases when the number of tasks increases. Furthermore, Fig 7 shows that in terms of total energy consumption, our proposed WAS-DVS can save 3.85 % to 19 % more than LTF-DVS under varies number of cores. Note that the total energy consumption improvement percentages are a little bit lower than the total execution time improvement percentages squared in Fig 6. This is due to the constraint that each core needs to execute at the same frequency level.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have presented an energy efficient scheduling algorithm, called WAS-DVS, for multi-core real-time embedded systems. The WAS-DVS not only considers the CPU workload of tasks but also the memory workload. The basic idea of WAS-DVS is to balance the load of each core and to reorder the execution order in each job queue to reduce memory contentions. By reducing memory contentions, our proposed mechanism can lower the total execution time from 2% to 10.3% before applying DVS and improve the total energy consumption from 3.85% to 19% after applying DVS, compared to LTF-DVS under different CMP environments with 2 to 16 cores.

6.2 Future work

For the future work, we will explore energy-efficient scheduling for threads with arbitrary deadlines, thread dependency, and arrival times on multi-core real time systems.

Chapter 7 Bibliography

[1] A. Naveh, E. Rotem, “Power and Thermal Management in the Intel® Core Duo Processor”, in *Proceedings of Intel Technology Journal*, Volume 10, Issue 2, 2006 and Test in Europe, pp. 634 – 639, 2005

[2] L. F. Leung, C. Y. Tsui, Hu, X.S., “Exploiting Dynamic Workload Variation in Low Energy Preemptive Task Scheduling”, in *Proceedings of Design, Automation and Test in Europe*, pp.634 - 639, 2005

[3] S. Yaldiz, A. Demir, S. Tasiran, P. Ienne, Y. Leblebici, “Characterizing and exploiting task load variability and correlation for energy management in multi core systems”, in *Proceedings of Multimedia 3rd Workshop on Embedded Systems for Real-Time*, pp.135 - 140, 2005

[4] J.H. Anderson, J.M. Calandrino, “Parallel Real-Time Task Scheduling on Multicore Platforms”, in *Proceedings of 27th IEEE International Real-Time Systems Symposium*, pp.89 – 100, Dec. 2006.

[5] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, S. Baruah, “A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms”, *in Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Joseph* Y-T Leung (ed.), Chapman Hall/CRC Press, 2003

[6] T.A. AlEnawy, H. Aydin, “Energy-aware task allocation for rate monotonic scheduling”, in *Proceedings of 11th IEEE Real Time and Embedded Technology and Applications Symposium*, pp. 213-223, March 2005

[7] J.-J. Chen, K.-H. Chuang, and T.-W. Kuo. “Multiprocessor energy-efficient scheduling with task migration considerations”, in *Proceedings of EuroMicro Conference on*

Real-Time Systems, pp. 101–108, July 2004.

[8] S. Siddha, V. Pallipadi, A. Mallick, “Process Scheduling Challenges in the Era of Multi-core Processors”, in *Proceedings of Intel Technology Journal*, Volume 11, Issue 4, Nov. 2007

[9] K. Karuri, C. Huben, R. Leupers ,“Memory Access Micro-Profiling for ASIP Design”, in *Proceedings of the Third IEEE International Workshop on Electronic Design, Test and Applications*, Jan. 2005

[10] H. Leontyev; J.-H. Anderson, “Tardiness Bounds for EDF Scheduling on Multi-Speed Multicore Platforms”, in *Proceedings of 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 103 – 110, Aug 2007

[11] H.B. Fradj, C. Belleudy, M. Auguin, ”Scheduler-based Multi-Bank Main Memory Configuration for Energy Reduction”, in *Proceedings of International Symposium on Industrial Embedded Systems,.*, pp. 1-7 , Oct. 2006

[12] Lei Miao , Yong Qi , Di Hou, Y. Dai, “Energy-Aware Scheduling Tasks on Chip Multiprocessor”, in *Proceedings of the Third International Conference on Natural Computation*, 2007

[13] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, ”An Approximation Algorithm for Energy-Efficient Scheduling on A Chip Multiprocessor”, in *Proceedings of the Design, Automation and Test 2005 in Europe*, Vol. 1, pp. 468 – 473, 2005

[14] O. Ozturk and M. Kandemir, “Customized On-Chip Memories for Embedded Chip Multiprocessors”, in *Proceedings of the Design Automation Conference. in Asia and South Pacific*, Volume 2, Jan. 2005

[15] 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, 2007