GPScheDVS: A New Paradigm of the Autonomous CPU Speed Control for Commodity-OS-based General-Purpose Mobile Computers with a DVS-friendly Task Scheduling

By

Sookyoung Kim

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
In
Electric and Computer Engineering

Dr. Thomas L. Martin, Chair

Dr. Dong S. Ha, Committee Member

Dr. Michael S. Hsiao, Committee Member

Dr. William T. Baumann, Committee Member

Dr. Thurmon E. Lockhart, Committee Member

April 28, 2008
Blacksburg, Virginia

Keywords: Dynamic Voltage Scaling, Task Scheduling, Energy Management, Power Management, Thermal Management, General-Purpose Mobile Computers.

GPScheDVS: A New Paradigm of the Autonomous CPU Speed Control for Commodity-OS-based General-Purpose Mobile Computers with a DVS-friendly Task Scheduling

Sookyoung Kim

Abstract

This dissertation studies the problem of increasing battery life-time and reducing CPU heat dissipation without degrading system performance in commodity-OS-based general-purpose (GP) mobile computers using the dynamic voltage scaling (DVS) function of modern CPUs. The dissertation especially focuses on the impact of task scheduling on the effectiveness of DVS in achieving this goal. The task scheduling mechanism used in most contemporary general-purpose operating systems (GPOS) prioritizes tasks based only on their CPU occupancies irrespective of their deadlines. In currently available autonomous DVS schemes for GP mobile systems, the impact of this GPOS task scheduling is ignored and a DVS scheme merely predicts and enforces the lowest CPU speed that can meet tasks' deadlines without meddling with task scheduling. This research, however, shows that it is impossible to take full advantage of DVS in balancing energy/power and performance in the current DVS paradigm due to the mismatch between the urgency (i.e., having a nearer deadline) and priority of tasks under the GPOS task scheduling. This research also shows that, consequently, a new DVS paradigm is necessary, where a "DVS-friendly" task scheduling assigns higher priorities to more urgent tasks.

The dissertation begins by showing how the mismatch between the urgency and priority of tasks limits the effectiveness of DVS and why conventional real-time (RT) task scheduling, which is intrinsically DVS-friendly cannot be used in GP systems. Then, the dissertation describes the requirements for "DVS-friendly GP" task scheduling as follows. Unlike the existing GPOS task scheduling, it should prioritize tasks by their deadline. But, at the same time, it must be able to do so without a priori knowledge of the deadlines and be able to handle the various tasks running in today's GP systems, unlike conventional RT task scheduling. The various tasks include sporadic tasks such as user-interactive tasks and tasks having dependencies on each other such as a family of threads and user-interface server/clients tasks. Therefore, the first major result of this research is to

propose a new DVS paradigm for commodity-OS-based GP mobile systems in which DVS is performed under a DVS-friendly GP task scheduling that meets these requirements.

The dissertation then proposes GPSched, a DVS-friendly GP task scheduling mechanism for commodity-Linux-based GP mobile systems, as the second major result. GPSched autonomously prioritizes tasks by their deadlines using the type of services that each task is involved with as the indicator of the deadline. At the same time, GPSched properly handles a family of threads and user-interface server/clients tasks by distinguishing and scheduling them as a group, and user-interactive tasks by incorporating a feature of current GPOS task scheduling – raising the priority of a task that is idle most of the time – which is desirable to quickly respond to user input events in its prioritization mechanism.

The final major result is GPScheDVS, the integration of GPSched and a task-based DVS scheme customized for GPSched called GPSDVS. GPScheDVS provides two alternative modes: (1) the system-energy-centric (SE) mode aiming at a longer battery life-time by reducing system energy consumption and (2) the CPU-power-centric (CP) mode focusing on limiting CPU heat dissipation by reducing CPU power consumption.

Experiments conducted under a set of real-life usage scenarios on a laptop show that the best, worst, and average reductions of system energy consumption by the SE mode GPScheDVS were 24%, -1%, and 17%, respectively, over the no-DVS case and 11%, -1%, and 5%, respectively, over the state-of-the-art task-based DVS scheme in the current DVS paradigm. The experiments also show that the best, worst, and average reductions of CPU energy consumption by the SE mode GPScheDVS were 69%, 0%, and 43% over the no-DVS case and 26%, -1%, and 13% over the state-of-the-art task-based DVS scheme in the current DVS paradigm. Considering that no power management was performed on non-CPU components for the experiments, these results imply that the system energy savings achievable by GPScheDVS will be increased if the non-CPU components' power is properly managed. On the other hand, the best, worst, and average reductions of average CPU power by the CP mode GPScheDVS were 69%, 49%, and

60% over the no-DVS case and 63%, 0%, and 30% over the existing task-based DVS scheme. Furthermore, oscilloscope measurements show that the best, worst, and average reduction of peak system power by the CP mode GPScheDVS were 29%, 10%, and 23% over the no-DVS case and 28%, 6%, and 22% over the existing task-based DVS scheme signifying that GPScheDVS is effective also in restraining the peak CPU power.

On the top of these advantages in energy and power, the experimental results show that GPScheDVS even improves system performance in either mode due to its deadline-based task scheduling property. For example, the deadline meet ratio on continuous videos by GPScheDVS was at least 91.2%, whereas the ratios by the no-DVS case and the existing task-based DVS scheme were down to 71.3% and 71.0%, respectively.

Acknowledgments

I would like to express my deepest gratitude to my academic and research advisor Dr. Thomas L. Martin for his guidance and constant support in helping me to conduct and complete this work. I would also like to thank Dr. Dong S. Ha, Dr. Michael S. Hsiao, Dr. Thurmon E. Lockhart, and Dr. William T. Baumann for serving on my advisory committee, as well as their generous advices. In addition, I want to thank Dr. Jaehong Park and Dr. Sangmook Lee in Electrical and Computer Engineering Department for their precious advices on my research.

Many thanks to all the people I have come to know in Virginia Tech and Blacksburg, whose friendship and companionship I will always enjoy. I owe my sincere appreciation to my family and relatives who have supported and encouraged me over the years. I especially want to thank Mr. Younghan Jung and his family for his help on my living in Blacksburg. Finally, I want to extend my profound appreciation to my beloved parents for their love, affection, and invaluable support during my life and studies.

Thank you, all of you.

Sookyoung Kim

June 3, 2008

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile general-purpose (GP) personal computers (PC) such as laptops, tablet PCs, and ultra mobile PCs (UMPC) account for a large share of the entire PC market today, and that share is expected to keep growing for future years [1]. These commodity-OS-based GP mobile PCs are the type of personal mobile computing devices for which the demand for balancing power and performance is especially acute; on one hand, mobile PCs have to meet users' expectation for a far higher performance than other types of personal mobile computing devices such as personal digital assistants (PDA) or smart phones. But, on the other hand, as with other personal mobile computing devices, their power consumption must be low because their small form factor limits battery size. However, contemporary mobile PCs fail to successfully meet this demand resulting in a much shorter battery lifetime than other types of personal mobile computing devices as Table 1.1 shows [2].

Table 1.1 The battery lifetime of contemporary personal mobile computing devices available for sale on CNET as of March, 2008 [3]

Form factor	Maker/Model	Battery lifetime	Usage scenario
Laptop	Apple MacBook Air	4 hours 3 minutes	DVD playback
	Lenovo ThinkPad X300	3 hours 43 minutes	
	Toshiba Portege R500-S5003	2 hours 49 minutes	
	HP Compaq 2710p	2 hours 23 minutes	
UMPC	Sony Vaio UX390N	3 hours 32 minutes	DVD playback
	Fujitsu Lifebook U810	3 hours 15 minutes	
	WiBrain B1E	2 hours 35 minutes	
PDA	Sony Mylo 2	7 hours	Video playback
	Palm Tungsten E2	5 hours 30 minutes	
Smart phone	Apple iPhone	24 hours	MP3 playback
	Nokia Xpress Music 5300	9 hours 15 minutes	

The CPU is the key component inducing the imbalance between power and performance in contemporary mobile PCs; recent observations report that CPU has been the biggest power consumer in these systems [4, 5, 6]. Figure 1.1 confirms these observations by

showing that up to 50% of the total system power is consumed by CPU in a Dell Latitude D600 laptop, the experimental platform of this research.

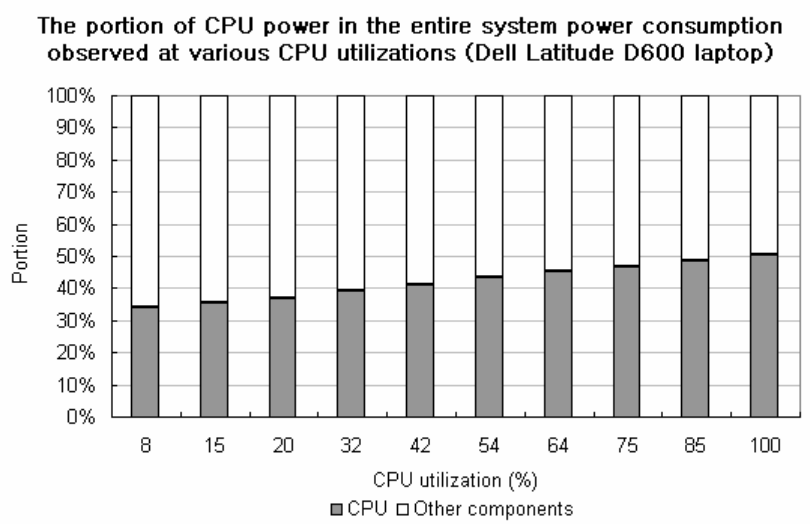


Figure 1.1 Portion of CPU power in the entire system power consumption observed on the experimental platform of this research

Moreover, the high CPU heat dissipation caused by the high CPU power consumption is another severe challenge in designing mobile PCs because these systems have limited choices for active CPU cooling solutions due to their small form factor [7, 8, 9, 10]. Therefore, reducing CPU power consumption has been a critical design goal today for commodity-OS-based GP mobile PCs.

Traditionally, CPU power reduction has been achieved through semiconductor process technology scaling. Complementary metal-oxide semiconductor (CMOS) has been the dominant process technology for CPUs since mid-1980s due to its low standby power consumption and the economy resulting from sharing a same process technology with dynamic random access memory (DRAM) [14]. For the past decades, CMOS CPUs have kept increasing their clock frequency and transistor count roughly by two times every two years for a higher performance [13]. The primary means to suppress the power needs of the improved performance CPUs for the period of time was to scale down supply voltage through CMOS process technology scaling. This approach is based on the CMOS features that supply voltage has a quadratic impact on the dynamic power consumption which has been the dominant portion of total power consumption today.

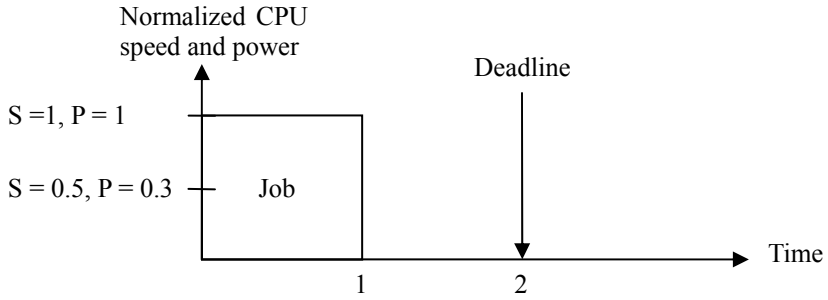
Notwithstanding the repeated CMOS process technology scaling, however, the reduction rate of supply voltage has failed to keep pace with the steep increase of clock frequency and transistor count, and the CPU power consumption has been increased continuously [12, 14, 15, 16, 17, 18, 19, 20]. Furthermore, as the threshold voltage, which has an exponential impact on sub-threshold leakage power, is lowered in accordance with the scaled-down supply voltage, the portion of sub-threshold leakage power in the total CPU power consumption keeps increasing diminishing the effectiveness of dynamic power reduction. For Intel CPUs, leakage power increased from being negligible in the 0.25 μm process to almost one-third of the total power in the 0.09 μm process [20]. The increase of sub-threshold leakage power is anticipated to be the primary factor that drives the surge of CPU power consumption in the near future [15, 16, 17, 18, 19, 20].

Consequently, the demand has been high for system-level techniques that alleviate the insufficient CPU power reduction in CMOS process technology scaling, especially for contemporary commodity-OS-based GP mobile systems that have high performance CPUs. Many logic-level, micro-architecture-level, system-level, compiler-level, and application-level techniques have been proposed for this purpose [21, 22, 23, 24, 25, 26].

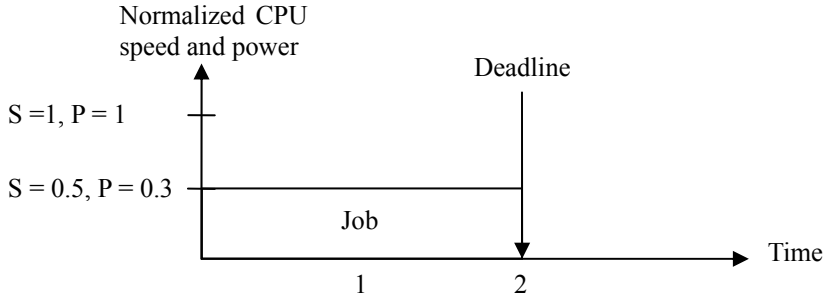
Dynamic voltage scaling (DVS) is a widely recognized system-level run-time technique to reduce the dynamic power consumption of CMOS CPUs without degrading system performance. In CMOS circuits, not only the dynamic power consumption but also the transistor switching speed reduces along with the decrease of supply voltage, extending the execution time of a given workload. Therefore, the supply voltage can be lowered for the reduction of CPU's dynamic power consumption as long as the workload can still complete within its deadline. With DVS, CPU clock frequency and CPU supply voltage are dynamically adjusted such that tasks are executed as slowly as possible, but without missing their deadlines [21, 23, 24, 25]. This *'just-in-time'* computation approach is depicted in Figure 1.2.

Note that, however, the reduction in power does not necessarily mean the reduction of CPU energy consumption, which actually determines battery lifetime, because energy is the average power consumption multiplied by execution time. DVS may increase tasks'

execution time more than it decreases the average power. If this is the case, DVS will merely degrade the battery lifetime. Fortunately, the CMOS property that dynamic power consumption reduces more quickly than the transistor switching speed along with the decrease of supply voltage allows DVS to improve the battery lifetime, as Figure 1.2 illustrates. Many DVS-enabled CPUs such as AMD K6-2, Intel Xscale 80200, and Intel Pentium-M processors are available in the market today [27, 28, 29, 30].



(a) Without DVS: Energy = Power × Execution time
= Power × Workload / Speed
= 1 × 1 / 1 = 1



(b) With DVS: Energy = 0.3 × 1 / 0.5 = 0.6

Figure 1.2 DVS energy saving mechanism based on the unbalance between power reduction rate and speed degradation rate (This example assumes no standby power. More energy will be saved if the standby power is not 0 because standby time reduces as active time is extended)

As described earlier, the rapid increase of sub-threshold leakage power consumption is weakening the usefulness of DVS in reducing the total CPU power consumption [32, 33, 34]. But in recent years, researchers have proposed dynamic threshold voltage scaling (DVTS) techniques where the threshold voltage is adjusted instead of the supply voltage aiming at the reduction of sub-threshold power consumption [35, 36, 37, 38, 39, 40]. Similar to DVS, DVTS allows the reduction of CPU energy consumption because the sub-threshold leakage power decreases more quickly than the transistor switching speed

along with the increase of threshold voltage. The analogy between DVS and DVTS in power-speed relationship was reported in [36] for a 70 nm CMOS process technology. With DVTS, the just-in-time computation approach is still making its way in reducing CPU power consumption without degrading system performance in deep-sub micron era.

Given a DVS/DVTS-enabled CPU, the condition to minimize CPU energy consumption without degrading system performance is to execute each task at the lowest CPU speed that can meet the task's deadline, i.e., the minimum feasible speed (MFS) [41, 42, 43, 44, 45, 46, 47, 48]. Therefore, a system-level scheme is necessary to find tasks' MFSs and control the DVS function of modern CPUs to enforce the found MFSs.

In a real-time (RT) system, the prediction of tasks' MFSs can be guided by a RT task scheduler. A RT task scheduler guarantees that future deadlines will be met as long as the total CPU utilization of the current task set does not exceed the schedulability bound. For example, earliest-deadline-first (EDF) scheduling rule guarantees the deadline meeting as long as the current task set's total CPU utilization is less than 1. Therefore, assuming that the current task set's total utilization is 0.5 and tasks are being scheduled by EDF, the uniform MFS that minimizes CPU energy consumption without missing any deadline is 0.5. This is the basic idea of so-called power-aware RT task scheduling, an extension of RT task scheduling with DVS.

In GP systems, however, such a RT task scheduling is impossible because the behavior of tasks such as each job's execution time, inter-arrival time, and the dependencies on each other are not known a priori. Therefore, most contemporary commodity GPOSs such as Microsoft Windows and Linux use a simple CPU-occupancy-based task scheduling that prioritizes tasks based only on their CPU occupancies irrespective of their deadlines. As a consequence, a DVS scheme for these commodity-OS-based GP systems has to predict tasks' MFSs autonomously. This type of DVS schemes is thus called *autonomous DVS scheme* in this dissertation.

In the current paradigm of the autonomous DVS schemes, the research focus has been on the accurate prediction of tasks' MFSs. In the early days, autonomous DVS schemes for

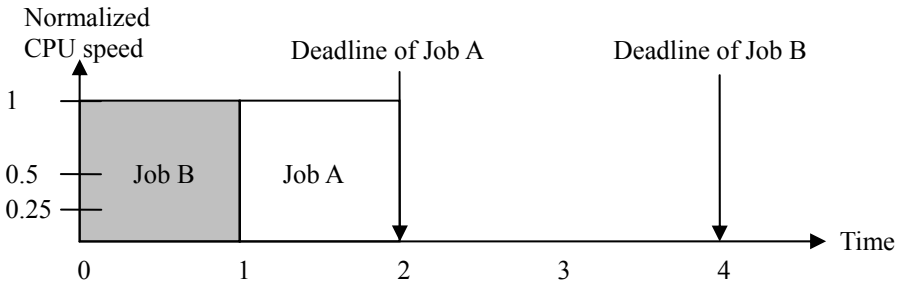
commodity-OS-based GP mobile systems predicted the MFSs of tasks simply based on the total CPU utilization of the entire task set measured at every fixed-length interval [69, 70, 71]. As for enhancing the battery lifetime, this simple approach called *interval-based autonomous DVS scheme* is very effective because the total-utilization-based CPU speed (i.e., the CPU speed whose ratio to the maximum CPU speed corresponds to the total CPU utilization) is the minimum possible CPU speed that does not increase the energy consumption of non-CPU components. If a given task set is executed at a lower CPU speed than the total-utilization-based CPU speed, the delay in the final completion time of the task set is physically inevitable. This, in turn, lengthens the active time of non-CPU components and causes them to consume more energy most likely offsetting the energy savings on CPU. Therefore, executing a given task set at the total-utilization-based CPU speed is the optimal way to minimize CPU energy consumption without increasing non-CPU components' energy consumption such that the entire system energy consumption is minimized.

The longer the interval an interval-based DVS scheme uses, the more accurately it can predict the total CPU utilization and, thus, the larger system energy savings it can achieve. However, no matter how the prediction of total CPU utilization is accurate, interval-based DVS schemes tend to miss many deadlines because the MFS of each task is not always lower than or same as the total CPU-utilization-based CPU speed. To avoid an excessive number of deadline misses, interval-based autonomous DVS schemes usually use a short interval length (about several tens of milliseconds). This solution improved the rate of meeting deadlines but reduces energy savings because it causes the CPU speed to rise quickly upon the arrival of a workload regardless of the workload's actual MFS. This dilemma is because an interval-based DVS scheme treats tasks in the aggregate. To solve this problem, researchers proposed *task-based autonomous DVS schemes* in which tasks' MFSs are predicted and enforced individually [75, 76, 77, 78, 79].

The goal of this work is to better realize the potential of DVS/DVTS by exploiting task scheduling. The motivating observation is that, as long as an autonomous DVS scheme is used under the native task scheduling of contemporary GPOSs, it is impossible to fully realize the potential of a DVS/DVTS technique even with a completely accurate

prediction of tasks' MFSs because of the unfriendliness of the GPOS task scheduling to DVS – the unnecessarily high and irregular CPU speed requirements after the tasks are scheduled together.

Figure 1.2 showed the case of just a single job. However, in real systems, a number of tasks exist and a job's ability to meet its deadline is affected by other jobs that can block it. In this sense, this dissertation redefines MFS into three different types. First, the MFS of a job is called *intrinsic minimum feasible speed (IMFS)*. IMFS is simply the ratio of a job's own workload to its deadline. Second, the effective MFS of a job that is affected by other jobs blocking the job is defined as *compound minimum feasible speed (CMFS)*. For example, the IMFS of Job *A* in Figure 1.3 is 0.5. But its CMFS is 1 because Job *A* can start execution only after Job *B* completes. Finally, the CPU speed corresponding to the total CPU utilization is called *ideal compound minimum feasible speed (ICMFS)*. ICMFS is ideal in the sense that it is the lowest speed that can complete a given task set without delaying the final completion time (in an ideal case where all CMFSs of individual tasks are same as the ICMFS). The delay in the final completion time lengthens the active time of non-CPU components, thus increasing their energy consumption. This may negate any energy savings of the CPU. Thus, the final completion time is an implicit deadline on the entire workload. When all CMFSs existing in the entire workload are the same as ICMFS, the ICMFS is the ideal speed in the sense that it is the lowest CPU speed which meets every deadline and causes no additional energy consumption of non-CPU components.



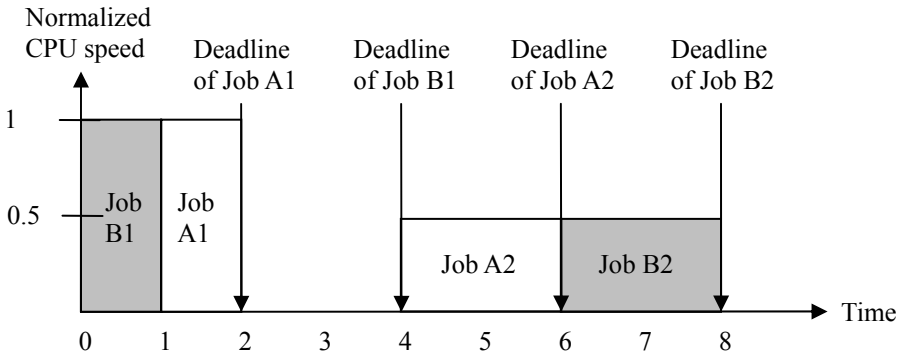
Release time: Jobs A and B = 0
Relative deadline: Jobs A = 2, Jobs B = 4
Execution time: All jobs = 1

Priority: Job A < Job B

IMFS: Job A = 0.5, Job B = 0.25
CMFS: Job A = 1, Job B = 0.25
ICMFS: 0.5

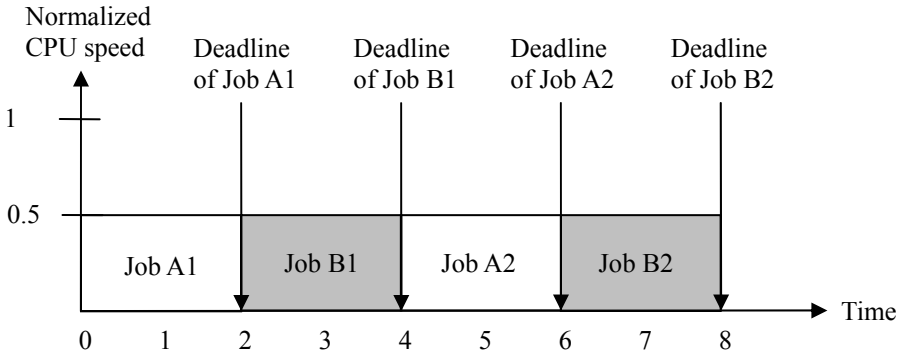
Figure 1.3 IMFS, CMFS, and ICMFS for an example task set

The idea of this work is to modify the native GPOS task scheduler such that tasks' CMFSs becomes close to the ICMFS. Figure 1.4 illustrates that prioritizing tasks based on their deadlines makes CMFSs close to ICMFS and thus further reduces the CPU energy consumption over even an ideal autonomous DVS scheme in the current paradigm.



(a) The ideal speed setting by a currently available task-based autonomous DVS scheme under the GPOS task scheduling

Release time: Jobs A1 and B1 = 0, Jobs A2 and B2 = 4
Priority: **Job A1 < Job B1**, Job A2 > Job B2
IMFS: Job A1 = 0.5, Job B1 = 0.25, Job A2 = 0.5, Job B2 = 0.25
CMFS: **Job A1 = 1, Job B1 = 0.25**, Job A2 = 0.5, Job B2 = 0.5
ICMFS: 0.5



(b) Speed setting by an interval-based autonomous DVS scheme under a DVS-friendly task scheduling

Release time: Jobs A1 and B1 = 0, Jobs A2 and B2 = 4
Priority: **Job A1 > Job B1**, Job A2 > Job B2
IMFS: Job A1 = 0.5, Job B1 = 0.25, Job A2 = 0.5, Job B2 = 0.25
CMFS: **Job A1 = 0.5, Job B1 = 0.5**, Job A2 = 0.5, Job B2 = 0.5
ICMFS: 0.5

Figure 1.4 DVS-friendly task scheduling effect: Reducing CMFSs

Moreover, the previous work on energy optimal CPU speed control showed that, given a DVS-enabled CPU whose power consumption is a convex function of speed, the CPU speed profile that minimizes CPU energy consumption among all possible CPU speed profiles that complete a given workload exactly on its deadline is to run the CPU constantly at the minimum feasible speed of the workload. However, real DVS-enabled

CPUs typically provide discrete CPU speed levels. Thus, if the energy-optimal constant CPU speed is not supported in the speed set, the energy optimal CPU speed profile is a combination of the two neighboring high and low CPU speeds. In other words, generally speaking the energy-optimal CPU speed profile is to keep the CPU speed uniform close to the ICMFS. Under the CPU-occupancy-based task scheduling this condition cannot be met because of the irregular CMFS due to the mismatches between tasks' CPU speed requirements and priorities. Scheduling tasks according to their deadlines – the tightness and firmness – alleviates this problem and yields more energy savings.

Based on this idea, the dissertation proposes a new paradigm of the autonomous CPU speed control for commodity-OS-based GP PCs in which an autonomous DVS scheme is performed under a DVS-friendly task scheduling. To verify the effectiveness of the suggested paradigm, the dissertation implements a working example of an autonomous DVS scheme called GPScheDVS designed in the new paradigm.

The rest of this introductory chapter is organized as follows. In Section 1.1, the scope of this research is defined. Section 1.2 briefly describes the organization of this dissertation. Finally, the contributions of this research are summarized in Section 1.3.

1.1 Scope of this research

This dissertation focuses on designing an autonomous CPU speed control scheme which maximizes a given DVS/DVTS-enabled CPU in commodity-OS-based GP PCs. Unlike autonomous CPU speed control schemes that focus only on the accurate prediction of tasks' CMFSs, the main focus of this research is on converting tasks' IMFSs to CMFSs in a DVS-friendly fashion such that the CPU operating point can be set to low levels for a longer time. Designing an autonomous DVS scheme that is tightly coupled with the DVS-friendly task scheduler to take full advantage of the scheduler support is another important goal of this dissertation. Figure 1.5 depicts the focus of this work as it relates to the whole mapping procedure between tasks' IMFSs and CPU operating points.

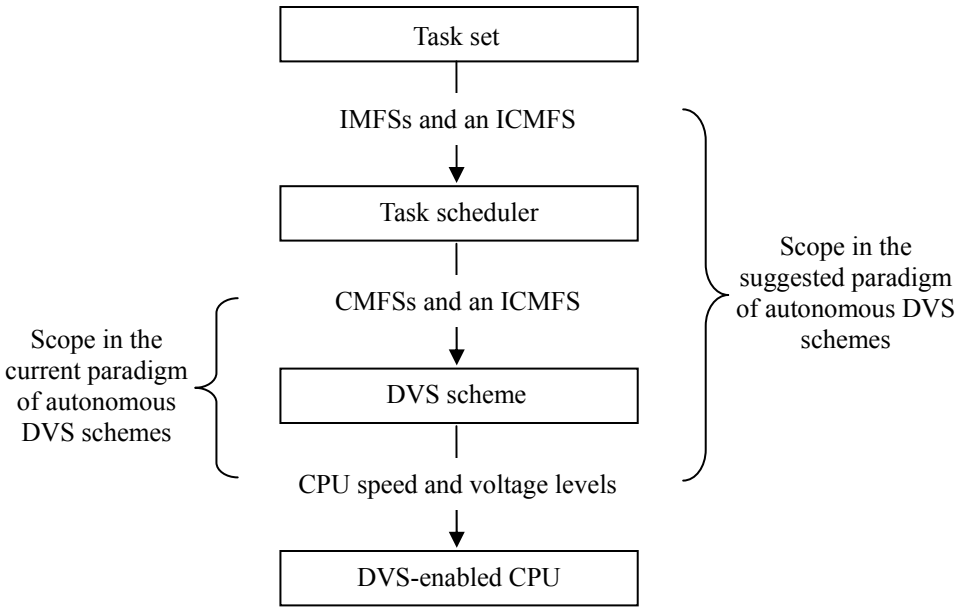


Figure 1.5 Mapping procedure between tasks' IMFSs and CPU speed/voltage levels

1.2 Organization

The organization of the remainder of this dissertation is as follows.

Chapter 2 presents the previous work in the areas closely related to this work. The two main areas are task scheduling and autonomous DVS schemes for commodity-OS-based GP PCs. Traditional priority-driven RT task scheduling methods are summarized and the problems in applying them to commodity-OS-based GP PCs are outlined. Then, the CPU-occupancy-based non-real-time (NRT) task scheduling of contemporary GPOSs is described using the Linux task scheduling mechanism as an example. The idea of power-aware RT task scheduling, the DVS extension of the traditional RT task scheduling, is explained. Finally, the autonomous DVS schemes in the current paradigm are reviewed.

Chapter 3 proposes the new paradigm of autonomous CPU speed control for commodity-OS-based GP PCs in which an autonomous DVS scheme predicts and enforces tasks' CPU speed requirements under a DVS-friendly task scheduling. The chapter begins by showing how much the current Linux native task scheduling unnecessarily increases the CMFSs of tasks and, then, how the problem can be alleviated by applying a DVS-friendly task scheduling. The chapter describes the complete set of requirements for an eligible

DVS-friendly task scheduling.

Chapter 4 implements a working DVS-friendly task scheduler called GPSched for commodity-Linux-based GP PCs. The underlying ideas are explained first and a complete description of the scheduler follows.

Chapter 5 describes GPScheDVS, an integration of GPSched and a task-based DVS scheme customized to GPSched. Implementation issues involved with GPScheDVS are explained in detail.

Chapter 6 presents the experimental results obtained from a real Linux-based laptop under a set of real-life usage scenarios. The chapter describes the experimental platform, benchmarks, and the evaluation metrics. Then, a Linux-based tool for the experiments, DVS-suite, is described. Finally, the experimental results are presented and discussed.

Finally, Chapter 7 discusses possible future work and concludes this dissertation.

1.3 Research contributions

The major contributions of this research are, first, to show that the appropriate support of task scheduling is necessary to take full advantage of the DVS technique in reducing CPU energy and power consumption without degrading system performance and, second, to propose a new paradigm for autonomous CPU speed control for commodity-OS-based GP mobile PCs in which DVS is performed under a *DVS-friendly* task scheduling.

The specific contributions to the areas of autonomous DVS schemes for commodity-OS-based GP mobile PCs include the following:

1. *GPSched*, a DVS-friendly GP best-effort (BE) task scheduler for commodity-Linux-based GP PCs, described in Chapter 4.

2. *GPScheDVS*, an integration of GPSched and a task-based DVS scheme customized to GPSched, described in Chapter 5.

3. *AIDVS*, an improved interval-based algorithm, which can save a larger amount of energy saving than existing interval-based algorithms, described in Section 5.3.

4. *DVS-suite*, a Linux-based tool to perform the experiments on autonomous DVS schemes in commodity-Linux-based GP mobile PCs. DVS-suite includes implementations of several representative existing autonomous DVS schemes and GPScheDVS and provides facilities to measure, store, and report a set of DVS performance metrics. DVS-suite allows a user to switch between different autonomous DVS schemes and activate/deactivate the metric trace facilities on-the-fly while the system is running. The design of DVS-suite is described in Section 6.4.

5. *Tweak-PM*, a Linux-based tool to probe and set the features of an equipped Intel Pentium-M processor.

Chapter 2

Related Work

This chapter describes the previous work in the two areas that are closely related to the topic of this dissertation: task scheduling and dynamic supply and/or threshold voltage scaling (DVS/DVTS).

2.1 Introduction

The chapter begins by summarizing the previous work in real-time (RT) task scheduling in Section 2.2.1. The section then shows why RT task scheduling is difficult to use for the open general-purpose (GP) systems such as commodity-OS-based GP PCs. The difficulty caused most contemporary GPOSs to use CPU-occupancy-based task scheduling, a non-real-time (NRT) task scheduling mechanism in which tasks are prioritized based only on their CPU occupancies irrespective of their deadlines. Section 2.2.2 exemplifies the CPU-occupancy-based task scheduling with the native Linux task scheduling mechanism.

The chapter then provides the background knowledge of DVS and DVTS in Section 2.3.1. Given a DVS/DVTS-enabled CPU, the condition to minimize CPU energy consumption has been studied by researchers. Then, Section 2.3.2 explains the two categories of CPU speed control approaches: power-aware RT task scheduling and autonomous DVS schemes. Section 2.3.2.1 describes the underlying idea of the power-aware RT task scheduling with two examples. The section shows the power-aware RT task scheduling is also difficult to use for commodity-OS-based GP PCs for the same reason as the RT task scheduling. Section 2.3.2.2 summarizes how the autonomous DVS schemes have evolved from interval-based DVS schemes to task-based DVS schemes.

2.2. Task scheduling

2.2.1. Real-time task scheduling

As defined in [53], RT systems are such systems where the correctness of the system depends not only on the logical results of computation, but also on the time at which the results are produced. In order to meet the timeliness constraint, a RT system has to be fast and predictable. Task scheduling, resource allocation, and bounded OS primitives are the three necessities for a predictable system [52]. Among the three issues, this section focuses on task scheduling.

Before Liu and Layland proposed rate-monotonic (RM) and earliest-deadline-first (EDF) task scheduling rules in their seminal paper announced in 1973 [58], the task schedule for RT systems had been constructed in an ad-hoc manner. In this approach, when and how long each task is executed is decided by a designer before the tasks are actually executed and recorded in a table. Naturally, the biggest problem was that they are inflexible. Unlike these early days' approach, when and how long each task is executed is determined in runtime based on the tasks' priorities in RM and EDF. Therefore, they are called priority-driven preemptive RT task scheduling.

Under RM, each task is assigned with a priority according to its period; the shorter period has a task, the higher priority the task is assigned with. Once a task's priority is determined, all jobs of the task share the same unique priority. For this reason, RM is called fixed-priority task scheduling. Unlike RM that assigns priority on a per-task basis, EDF assigns priority on a per-job basis based on each job's deadline. Even the jobs of a same task can have different priorities under EDF. Therefore, EDF is called dynamic-priority task scheduling.

Liu and Layland showed that, with a set of assumptions, RM and EDF guarantee that every future deadline will be met as long as the total CPU utilization of a given task set does not exceed their intrinsic schedulability bounds [58]. The important assumptions include followings.

- All tasks are periodic
- All tasks have a (job) deadline equal to their period
- All tasks are independent from each other in terms of resource or precedence constraints
- All tasks have a fixed (job) execution time or, at least, a fixed upper bound on the execution time and the execution time is less than or equal to their period

The schedulability bound of RM is defined by Equation 2.1.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1), \qquad \text{(Equation 2.1)}$$

where C_i is the execution time of the each job of task i, T_i is the period of task i, and n is the total number of tasks. When n becomes infinite, the schedulability bound converges to 69.31%. On the other hand, the schedulability bound of EDF is 100%.

Liu and Layland showed that RM and EDF are the optimal fixed-priority task scheduling and the optimal dynamic-priority scheduling, respectively, in the sense that no other fixed-priority task scheduling can have a higher schedulability bound than RM and that no other dynamic-priority task scheduling can have a higher schedulability bound than EDF [58].

Since the RM and EDF were proposed, the main stream in RT task scheduling area has been the relaxation of the assumptions made in [58]. Efforts have been made for the exact feasibility analysis, the analysis of the interaction, the inclusion of aperiodic tasks, overload management, implementation simplification, and multiprocessors and distributed systems [55].

However, using a conventional RT task scheduling in GP systems is still a great challenge as many evidences designate for the following reasons [54, 55, 57].

- A priori knowledge of tasks' behavioral parameters, which is typically not available in actual GP systems, is required (for static table-driven and static priority-based preemptive), at least at a new task's arrival time (for dynamic plan-based).

- If the worst case execution times and the minimum inter-arrival times are used to apply RT scheduling to a set of irregularly behaving tasks, which is the case for most GP systems, CPU utilization becomes worse. If the worstcase parameters are not used, on the other hand, the merit of RT scheduling vanishes.

- The weakness in scheduling the task sets in which tasks have dependencies (control dependencies due to multi-threading, i.e., the precedence constraints, data dependencies due to IPC, or blocking due to resource contention) to each other.

- Implementation overhead.

These problems make today's most commodity-OS-based GP PCs continue using non-real-time (NRT) CPU-occupancy-based task scheduler.

2.2.2. Native task scheduling of contemporary GPOSs

This section outlines Redhat Linux 9's task scheduling mechanism focusing on the features for single processor systems which were implemented in the event-driven simulator and used to generate on-line task schedules during simulations. The task scheduler of Redhat Linux 9 is basically a non-real-time dynamic-priority-based task scheduler. It updates the tasks' priorities dynamically based on their CPU occupancies then selects the next task to run based on the priorities of the tasks. The three major kernel functions that realize the traditional task scheduling in Redhat Linux 9 are schedule(), scheduler_tick(), and try_to_wake_up(). All of these functions are implemented in linux/kernel/sched.c source file.

Schedule() illustrated in Figure 2.1 selects the next task to run. In Figure 2.1, 'prev' and 'next' mean the precedent task to be scheduled out and the next task to be scheduled in, respectively. Schedule() first records the time when prev is scheduled out for the later

calculation of the priority of prev (line 23). It then removes prev from run queue if prev's state is not 0, i.e. running state (line 33). The run queue of Redhat Linux 9 consists of actually two queues; the active array and the expired array. The expired array contains all the tasks that want to run but have run out of their time slices. The variable rq_nr_running indicates the number of tasks in the entire run queue, whereas the variable array_nr_active indicates the number of tasks only in the active array. If there is no eligible task throughout the run queue, schedule() selects swapper, the idle task, as the next task (lines 46~49). If there are eligible tasks in run queue's active array, schedule() chooses the highest priority of task among them (lines 62~64). If there are runnable tasks only in the expired array, schedule() simply swaps the active array and the expired array (line 51~60), then chooses the highest priority of task from the new active array which was expired array before the swapping. The swapping is a feature of so-called O(1) scheduler of Redhat Linux 9.

Figure 2.1 schedule() kernel function (Lines 1~29)

```
1     /*
2      * schedule() is the main scheduler function.
3      */
4     asmlinkage void schedule(void)
5     {
6             task_t *prev, *next;
7             runqueue_t *rq;
8             prio_array_t *array;
9             struct list_head *queue;
10            int idx;
11
12            BUG_ON(in_interrupt());
13
14    need_resched:
15            prev = current;
16            rq = this_rq();
17
18            release_kernel_lock(prev, smp_processor_id());
19            /*
20             * Ok, we are leaving the CPU now, lets update the 'last run'
21             * timestamp:
22             */
23            prev->last_run = jiffies;
24            spin_lock_irq(&rq->lock);
25
26            switch (prev->state) {
27            case TASK_INTERRUPTIBLE:
28                    if (unlikely(signal_pending(prev))) {
29                            prev->state = TASK_RUNNING;
```

Figure 2.1 schedule() kernel function (Lines 30~81)

```
30                          break;
31                  }
32          default:
33                  deactivate_task(prev, rq);
34          case TASK_RUNNING:
35                  ;
36          }
37  #if CONFIG_SMP
38  pick_next_task:
39  #endif
40          if (unlikely(!rq->nr_running)) {
41  #if CONFIG_SMP
42                  load_balance(rq, 1);
43                  if (rq->nr_running)
44                          goto pick_next_task;
45  #endif
46                  next = rq->idle;
47                  rq->expired_timestamp = 0;
48                  goto switch_tasks;
49          }
50
51          array = rq->active;
52          if (unlikely(!array->nr_active)) {
53                  /*
54                   * Switch the active and expired arrays.
55                   */
56                  rq->active = rq->expired;
57                  rq->expired = array;
58                  array = rq->active;
59                  rq->expired_timestamp = 0;
60          }
61
62          idx = sched_find_first_bit(array->bitmap);
63          queue = array->queue + idx;
64          next = list_entry(queue->next, task_t, run_list);
65
66  switch_tasks:
67          prefetch(next);
68          clear_tsk_need_resched(prev);
69
70          if (likely(prev != next)) {
71                  struct mm_struct *prev_mm;
72                  rq->nr_switches++;
73                  rq->curr = next;
74
75                  prepare_arch_switch(rq, next);
76
77                  prev = context_switch(rq, prev, next);
78                  barrier();
79                  rq = this_rq();
80                  prev_mm = rq->prev_mm;
81                  rq->prev_mm = NULL;
```

Figure 2.1 schedule() kernel function (Lines 82~91)

```
82                      finish_arch_switch(rq, prev);
83                      if (prev_mm)
84                              mmdrop(prev_mm);
85              } else
86                      spin_unlock_irq(&rq->lock);
87
88              reacquire_kernel_lock(current);
89              if (need_resched())
90                      goto need_resched;
91      }
```

Figure 2.1 schedule() kernel function

There are three cases when schedule() is called. First, when the current task has to leave the run queue either willingly or unwillingly, the task sets the need_sched flag in its task data structure which causes the activation of schedule().

The second case schedule() can be called is when a task having a higher priority than the current task gets into run queue. When a task needs to get into run queue, the kernel calls try_to_wake_up() function given in Figure 2.2. Focusing on the mechanism for single processor systems, the work this function performs is very simple; it activates the task that wants to get into run queue (i.e. calculates how long the task has slept in jiffies that is the number of timer ticks, updates the task's priority based on its sleep time, and inserts the task into the run queue), sets the woken task's state running, and calls schedule() if the woken task's priority is higher than the priority of current task (lines 44~46 and line 52).

Figure 2.2 try_to_wakeup() kernel function (Lines 1~15)

```
1       /***
2        * try_to_wake_up - wake up a thread
3        * @p: the to-be-woken-up thread
4        * @state: the mask of task states that can be woken
5        * @sync: do a synchronous wakeup?
6        *
7        * Put it on the run-queue if it's not already there. The "current"
8        * thread is always on the run-queue (except when the actual
9        * re-schedule is in progress), and as such you're allowed to do
10       * the simpler "current->state = TASK_RUNNING" to mark yourself
11       * runnable without the overhead of this.
12       *
13       * returns failure only if the task is already active.
14       */
15      static int try_to_wake_up(task_t * p, unsigned int state, int sync)
```

Figure 2.2 try_to_wakeup() kernel function (Lines 16~15)

```
16   {
17       unsigned long flags;
18       int success = 0;
19       long old_state;
20       runqueue_t *rq;
21
22       sync &= SYNC_WAKEUPS;
23   repeat_lock_task:
24       rq = task_rq_lock(p, &flags);
25       old_state = p->state;
26       if (old_state & state) {
27          if (!p->array) {
28              /*
29               * Fast-migrate the task if it's not running or runnable
30               * currently. Do not violate hard affinity.
31               */
32              if (unlikely(sync && !task_running(rq, p) &&
33                 (task_cpu(p) != smp_processor_id()) &&
34                 (p->cpus_allowed & (1UL << smp_processor_id())))) {
35                        set_task_cpu(p, smp_processor_id());
36                        task_rq_unlock(rq, &flags);
37                        goto repeat_lock_task;
38              }
39              if (old_state == TASK_UNINTERRUPTIBLE)
40                 rq->nr_uninterruptible--;
41              if (sync)
42                 __activate_task(p, rq);
43              else {
44                 activate_task(p, rq);
45                 if (p->prio < rq->curr->prio)
46                        resched_task(rq->curr);
47              }
48              success = 1;
49          }
50          if (p->state >= TASK_ZOMBIE)
51                 BUG();
52          p->state = TASK_RUNNING;
53       }
54       task_rq_unlock(rq, &flags);
55
56       return success;
57   }
```

Figure 2.2 try_to_wakeup() kernel function

The last case schedule() can be called is when the current task runs out of its time slice upon a timer tick. Upon every timer tick, kernel calls scheduler_tick() function shown in Figure 2.3. The fraction related to task scheduling in scheduler_tick() spans from line 59 to line 74. Scheduler_tick() first decrements current task's sleep_avg that indicates how often the task occupies CPU and time_slice that indicates how longer the task can run. If

the decremented time_slice became 0, scheduler_tick() once gets the task out of the active array then recalculates the task's priority and recharges the task's time slice. Before inserting the task to expired array, scheduler_tick() check if the task deserves to be reinserted into the active array as it has paid the penalty in occupying CPU. If the task does, scheduler_tick() reinserts the task into the active array. If the task does not, the function inserted the task into the expired array.

Figure 2.3 scheduler_tick() kernel function (Lines 1~41)

```
1     /*
2      * This function gets called by the timer code, with HZ frequency.
3      * We call it with interrupts disabled.
4      *
5      * It also gets called by the fork code, when changing the parent's
6      * timeslices.
7      */
8     void scheduler_tick(int user_ticks, int sys_ticks)
9     {
10            int cpu = smp_processor_id();
11            runqueue_t *rq = this_rq();
12            task_t *p = current;
13
14            if (p == rq->idle) {
15                    if (local_bh_count(cpu) || local_irq_count(cpu) > 1)
16                            kstat.per_cpu_system[cpu] += sys_ticks;
17    #if CONFIG_SMP
18                    idle_tick(rq);
19    #endif
20                    return;
21            }
22            if (TASK_NICE(p) > 0)
23                    kstat.per_cpu_nice[cpu] += user_ticks;
24            else
25                    kstat.per_cpu_user[cpu] += user_ticks;
26            kstat.per_cpu_system[cpu] += sys_ticks;
27
28            /* Task might have expired already,
29    but not scheduled off yet */
30            if (p->array != rq->active) {
31                    set_tsk_need_resched(p);
32                    return;
33            }
34            spin_lock(&rq->lock);
35            if (unlikely(rt_task(p))) {
36                    /*
37                     * RR tasks need a special form of timeslice management.
38                     * FIFO tasks have no timeslices.
39                     */
40                    if ((p->policy == SCHED_RR) && !--p->time_slice) {
41                            p->time_slice = task_timeslice(p);
```

Figure 2.3 scheduler_tick() kernel function (Lines 42~81)

```
42                          p->first_time_slice = 0;
43                          set_tsk_need_resched(p);
44
45                          /* put it at the end of the queue: */
46                          dequeue_task(p, rq->active);
47                          enqueue_task(p, rq->active);
48                  }
49                  goto out;
50          }
51          /*
52           * The task was running during this tick - update the
53           * time slice counter and the sleep average. Note: we
54           * do not update a thread's priority until it either
55           * goes to sleep or uses up its timeslice. This makes
56           * it possible for interactive tasks to use up their
57           * timeslices at their highest priority levels.
58           */
59          if (p->sleep_avg)
60                  p->sleep_avg--;
61          if (!--p->time_slice) {
62                  dequeue_task(p, rq->active);
63                  set_tsk_need_resched(p);
64                  p->prio = effective_prio(p);
65                  p->time_slice = task_timeslice(p);
66                  p->first_time_slice = 0;
67
68                  if (!TASK_INTERACTIVE(p) || EXPIRED_STARVING(rq)) {
69                          if (!rq->expired_timestamp)
70                                  rq->expired_timestamp = jiffies;
71                          enqueue_task(p, rq->expired);
72                  } else
73                          enqueue_task(p, rq->active);
74          }
75  out:
76  #if CONFIG_SMP
77          if (!(jiffies % BUSY_REBALANCE_TICK))
78                  load_balance(rq, 0);
79  #endif
80          spin_unlock(&rq->lock);
81  }
```

Figure 2.3 scheduler_tick() kernel function

The traditional task scheduler of Redhat Linux 9 is, on the other hand, activated upon one of the following three conditions:

- The currently running task suspends either willingly as it completed its work for now or unwillingly as a necessary resource is unavailable at present. In this case, the task calls the kernel scheduler, leaves run queue setting its state 1 or 2, i.e. ‘interruptible’

or ‘uninterruptible,’ and gets into a wait queue.

- A task that was just spawned or woken up from a wait queue gets into run queue, and the priority of this task is higher than the priority of the currently running task. In this case, the kernel activates the task scheduler, and the new task preempts the current task. The preempted task holds its ‘running’ state and is left in the active array of run queue.

- The currently running task ran out of its time slice upon the system timer tick. Limiting a task’s continuous execution with its time slice is a well-known mechanism of all Linux OSs to accomplish the fair schedule among tasks. In this case, the current task is moved from the active array of run queue to the expired array of run queue keeping its state still 0, i.e. ‘running,’ and kernel activates the task scheduler to choose the next task to run.

The following briefly explains the mechanisms of Linux, Redhat Linux 9 Shrike OS based on Linux kernel version 2.4.20-8, for achieving fairness among tasks’ CPU occupancies and determine tasks’ priorities.

There is one thing to note. Starting with version 2.6.x, Linux kernels come with a brand new task scheduler capable of limiting scheduling latency to a constant time, i.e., O(1) time. It has been reported that, compared with the O(n) task scheduler of version 2.4.x kernels (where n is the number of tasks), the new scheduler improved both the system’s fitness for real-time applications and the response-time by virtue of the reduced uncertainty in scheduling latency as well as the decreased scheduling latency itself. Interestingly, Linux of this work has adopted an early work of the O(1) scheduler; although there are some trivial differences, Linux still shares the key features of the O(1) scheduler with version 2.6.x Linux kernels. Therefore, it is recommended not to be confused with the discordance of the explained features of Linux’s task scheduler with that usually known for most version 2.4.x kernels’ task scheduler.

Linux achieves the fairness among tasks’ CPU occupancies by using a per-task variable

‘time_slice.’ Time_slice indicates a task’s remained time quanta, for which amount of time a task is allowed to run without waiting for all other tasks’ running out of time quanta. The run queue of Linux consists of two sub queues: ‘active_array’ which holds all the tasks that have reserved their time_slice values and ‘expired_array’ which keeps the tasks that have run out of their time_slice values. A newly created task is assigned with an initial time_slice value, which is usually 10 ‘jiffies’ for normal (non-real-time) tasks, where jiffies is a kernel’s global variable indicating the cumulative occurrence number of system timer ticks since the system booted up. The task looses its time_slice as it runs by 1 jiffy at every system timer tick occurred in the middle of the task’s execution. Upon the expiration of its time_slice, the task is forced to stop execution, moved from active_array to expired_array, and waits until all other tasks in active_array run out of their time_slice. When all other tasks in active_array finally expire their time_slice, the task scheduler swaps the two arrays simply by switching the pointers to the arrays; this is a function of the O(1) task scheduler to reduce scheduling latency. Throughout this process, every task in run queue becomes to have a fair share of CPU time.

As a traditional non-real-time dynamic-priority-based task scheduler, the scheduler of Linux determines tasks’ priorities based on the tasks’ CPU occupancies. In this mechanism, the scheduler refers each task’s ‘sleep_avg,’ a per-task variable to keep track of tasks’ CPU occupancy ratios. The sleep_avg of a task is updated every time when the task is activated again with the ‘sleep_time’, a per-task variable that holds the task’s starved time. Finally, the sleep_time is calculated using the task’s ‘last_run’ variable immediately before it is used to update the sleep_avg. A task’s last_run is the jiffies when the task was scheduled out last time.

Interactive task reinsertion is another unique mechanism of Linux’s O(1) task scheduler to improve the system response time. The underlying idea is that giving one more chance to get the CPU for interactive tasks by not moving it into expired_array but leaving it in active_array upon its running out of time_slice. The decision of which array an expired task should be inserted is made based on two conditions: how the task is interactive and how much starved the tasks in the expired array, if any. These conditions are checked respectively by using TASK_INTERACTIVE() macro and EXPIRED_STARVING()

macro in the kernel function linux/kernel/sched.c:scheduler_tick() as follows.

```
if (!TASK_INTERACTIVE(p) || EXPIRED_STARVING(rq)) {
        if (!rq->expired_timestamp)
                rq->expired_timestamp = jiffies;
        enqueue_task(p, rq->expired);
} else
        enqueue_task(p, rq->active);
```

The kernel variable expired_timestamp used in this mechanism indicates the jiffies when a task in run queue ran out of its time_slice for the first time among the tasks in the run queue since the last swap between the run queue's active array and expired array. In other words, expired_timestamp indicates that how long the first expired task has waited in the expired_array. The more starved the expired task is, the narrower chance to be inserted back to the active array an interactive task has.

2.3 Dynamic supply/threshold voltage scaling

2.3.1 Fundamentals

Complementary metal-oxide semiconductor (CMOS) has been the dominant very large scale integrated (VLSI) circuit process technology since 1980s due to its suitability for high integrity and lower power consumption [14]. As CMOS CPUs continue increasing their clock frequency and transistor count, however, their power consumption has been increased rapidly.

CMOS CPUs consume power in either the active mode or standby (also known as idle) mode. Equation 2.2 shows the total power consumption of CMOS circuits, which consists of dynamic power dissipation, short-circuit power dissipation, and static (leakage) power dissipation.

$$P_{active} = P_{dynamic} + P_{short} + P_{leakage} = \alpha C V^2 f + \tau \alpha V I_{short} f + V I_{leakage}, \qquad \text{(Equation 2.2)}$$

where, α is the activity factor to the clock rate, C is the total capacitance of the circuit, V is supply voltage, f is clock frequency, τ is short-circuit duration, I_{short} is short-circuit current, and $I_{leakage}$ is leakage current [11].

And the sub-threshold leakage current, the dominant part of the total leakage current, is given by Equation 2.3.

$$I_{leakage,subthreshold} = K_1 W e^{\frac{-V_{th}}{nV_T}} (1 - e^{\frac{-V}{T}}), \qquad \text{(Equation 2.3)}$$

where W is the gate width, K_1 and n are constants, V is supply voltage, V_{th} is threshold voltage, and T is temperature [24].

An important aspect of the dynamic power dissipation is that the supply voltage has a quadratic impact on the resultant power dissipation. The supply voltage also has linear impacts on the short-circuit power dissipation and the static power dissipation. Therefore, lowering the supply voltage is one of the most effective ways to reduce the energy consumption of CMOS circuits.

However, lowering the supply voltage increases the CMOS circuit delay, and therefore degrades the circuit's timeliness performance. Conversely, decreasing the supply voltage should be allowed only when the slower operation of a circuit does not degrade the system performance. This is the basic rule of DVS.

$$f = \frac{(V - V_{th})^{\alpha}}{kV}, \qquad \text{(Equation 2.4)}$$

where, V is supply voltage, V_{th} is threshold voltage, and k and α are the constants determined by CMOS process technology [24].

The ultimate goal of DVS is to reduce the energy consumption of a CMOS circuit without violating any timeliness-constraint degrading the temporal performance of the circuit.

Given the properties of CMOS circuits, DVS can afford an opportunity to reduce the energy consumption while preserving the performance. The idea behind the DVS arose from the observation that the workload of a CPU does not always require the maximum speed, which means that if one can dynamically adjust the CPU speed along with the

operation voltage just enough to meet the required level of performance, chances to a save considerable amount of energy exist. CPUs capable of dynamically changing their clock speed along with the operation voltage are already available. For example, AMD's *K6-2* processor can dynamically change its clock speed along with the operation voltage between nine levels within the range from 266MHz to 475MHz (OPN Suffixes AHX, 400AFQ, and AFR) [27]. Intel's Xscale™ 80200 processor and Pentium-M processor are other examples of such processor that supports DVS [28, 30].

The biggest challenge in applying DVS is how to accurately schedule the CPU speed to minimize the CPU energy consumption while meeting the required level of performance. If the future CPU performance requirements are underestimated, performance may be degraded. On the other hand, if it is overestimated, the CPU will consume energy unnecessarily [73]. On this, the key issue for DVS approaches is how to accurately estimate the future CPU performance requirements.

As CMOS feature size shrinks, however, the leakage power consumption is expected to overwhelm the dynamic power consumption. In recent years, researchers have proposed dynamic threshold voltage scaling (DVTS) techniques [24, 25, 35, 36, 37, 38, 39, 40]. As DVS scales the supply voltage to reduce the dynamic power, DVTS scales the threshold voltage to reduce the sub-threshold leakage power. Similar to DVS, the sub-threshold voltage has an exponential impact on the sub-threshold power consumption. On the other hand, the degradation of circuit speed is near the linear function of the sub-threshold voltage. This analogy makes it possible to use an existing system level DVS scheme as it is for a DVTS CPU instead of a DVS CPU.

2.3.2. Existing DVS approaches for general-purpose systems

2.3.2.1. Power-aware real-time task scheduling

Power-aware RT scheduling is on the foundation of traditional RT scheduling theory. The basic principle of power-aware RT scheduling is as follows: if there is slack in the existing schedule, compose a new RT schedule taking account of the tasks' execution times that will be extended by lowering the CPU speed. As the task scheduling must be

done taking CPU speed into account, any power-aware RT scheduling scheme is an integrated form of task scheduling and DVS.

As one expects, however, that power-aware RT scheduling inherits the problems of traditional RT scheduling theory – it is as yet impractical for contemporary GP systems with the practical characteristics summarized in section 2.2.1.

2.3.2.2. Autonomous DVS schemes

Unlike the power-aware RT scheduling, there is another category of DVS schemes proposed to be used in the GP systems which continue using the traditional NRT BE task scheduling. In this dissertation, this category of DVS schemes are called 'autonomous DVS' as they work ‘in tandem’ with the traditional task schedulers.

In autonomous DVS approaches, task scheduling does not take account of CPU speed and a separate DVS scheme takes the current schedule as an its input and then determines the next CPU speed; the task scheduler and the DVS scheme work in tandem.

Existing autonomous DVS schemes solve the problems of power-aware RT scheduling in the application to GP systems by thoroughly leaving task scheduling decisions to the task scheduling mechanism of commodity OS. The task scheduling mechanism is the CPU-occupancy-based NRT BE task scheduling discipline, which is robust against the characteristics of GP systems and thus a *de facto* standard in contemporary GP systems.

The first autonomous DVS schemes are called ‘interval-based DVS (IDVS)’ as they determine the next CPU speed according to tasks’ aggregated CPU utilization measured at every fixed interval [69, 70, 71, 72]. The two main problems of IDVS are that (1) it cannot cope with a particular section of workload which requires a higher CPU speed than the CPU speed determined with the aggregated CPU utilization and (2) its fixed interval size is poor in accurately measuring the CPU utilization; the scheme will see a fully busy duration or a fully idle duration if the interval size is too short and it will fall behind in updating the measurement if the interval size is too long compared to the utilization transition rate [73, 74, 75, 76, 77, 78, 79].

A new autonomous DVS approach, so-called ‘task-based DVS (TDVS),’ has been proposed to solve these problems of the old-fashioned autonomous DVS schemes and become the current paradigm of autonomous DVS approach. The underlying idea of TDVS is, first, to categorize tasks according to their behavioral characteristics (whatever they are, but distinguishing from others) and, then, to apply a customized policy to each category of tasks. In this way, TDVS can predict tasks’ CPU speed requirements more accurately and thus adjust CPU speed more adequately. A category of tasks that most TDVS schemes commonly focus on is the interactive application tasks which is stimulated from user input events such as keyboard strokes and mouse button clicking [74, 75, 76, 77, 78, 79].

Most existing TDVSs focus on the interactive episode, which is a section of workload involved with an interactive application and begins with a user input event and ends with the corresponding interactive application task's graphical data output to X. As 50 ms of human perception threshold is widely accepted to be the deadline for interactive episodes, the decision on the CPU speed to complete interactive episodes can be achieved relatively easily compared to other time-constrained section of workload since not the deadline but only the average amount of workload imposed during the past interactive episodes needs to be acquired.

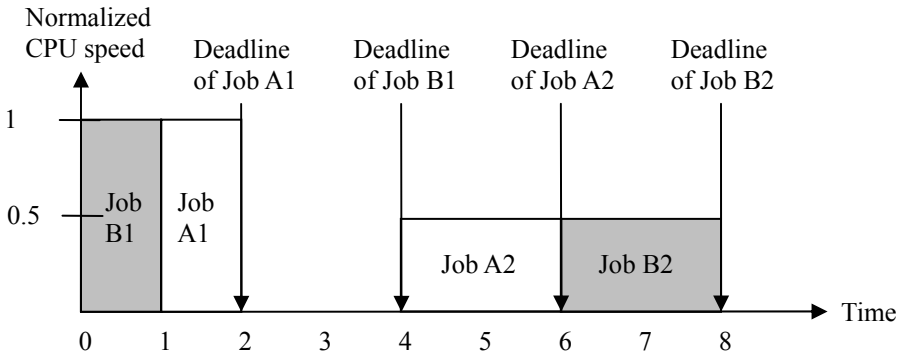
For the rest of the workload, existing TDVSs typically use a traditional IDVS having a fixed interval size [75, 77, 78]. Unlike other existing TDVSs, Vertigo tries to determine individual tasks' workloads and deadlines and finally the required CPU speeds particular to the individual tasks on a per-task basis. However, Vertigo still does it in such a way that every task is treated with the same policy regardless whether the task is actually time-constrained or not [75].

Chapter 3

Toward a New Paradigm of the Autonomous CPU Speed Control for Commodity-OS-Based GP Mobile PCs

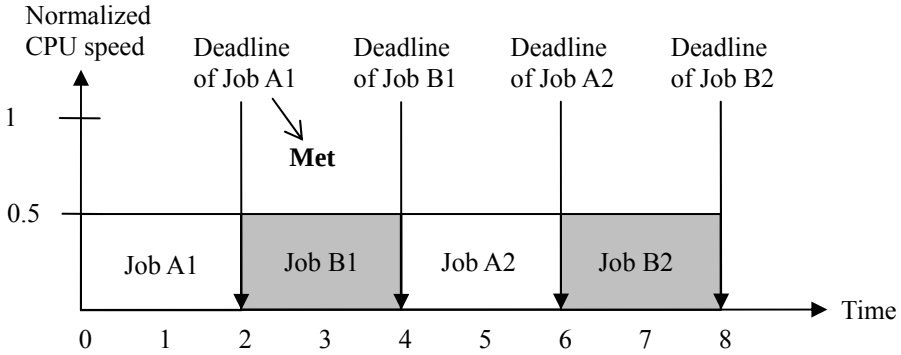
This chapter proposes a new paradigm of the autonomous CPU speed control for commodity-OS-based GP mobile PCs.

Traditionally, the research focus of autonomous DVS has been on how to accurately predict future MFSs (Minimum Feasible Speed, i.e., the lower bound of the CPU speed required to complete a given workload before its deadline). However, the CPU-occupancy-based NRT task scheduling of commodity GPOSs, under which autonomous DVS schemes have been proposed to work, makes this goal harder to achieve. This is because it often assigns less urgent tasks such as background batch jobs with higher priorities than more urgent tasks such as continuous media jobs. This task prioritization means that the more urgent workloads' MFSs tend to be affected by the less urgent workloads whose behavior is typically more irregular and thus unpredictable. A more essential problem posed by the NRT task scheduling is that, even with the completely accurate prediction of future MFSs, the CPU speed cannot be kept close to the energy optimal constant speed because the MFS distribution across the schedule becomes unnecessarily uneven as the less urgent workloads crowd into the more urgent workloads. This problem of autonomous DVS schemes implies that the integration of DVS and time constraint-based task scheduling is a necessity in achieving the optimal energy-performance trade-off even for GP systems. Figure 3.1 illustrates this idea.



(a) Ideal speed setting by a task-based autonomous DVS scheme

Release time: Jobs A1 and B1 = 0, Jobs A2 and B2 = 4
Priority: **Job A1 < Job B1**, Job A2 > Job B2
IMFS: Job A1 = 0.5, Job B1 = 0.25, Job A2 = 0.5, Job B2 = 0.25
CMFS: **Job A1 = 1, Job B1 = 0.25**, Job A2 = 0.5, Job B2 = 0.5
ICMFS: 0.5



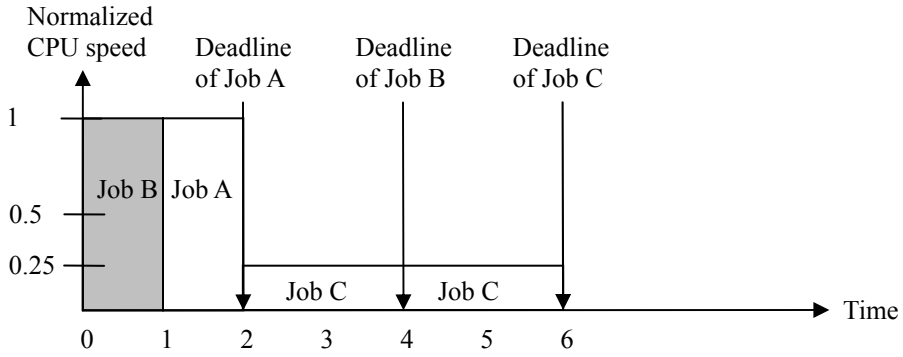
(b) Speed setting by an interval-based autonomous DVS scheme under a DVS-friendly task scheduling

Release time: Jobs A1 and B1 = 0, Jobs A2 and B2 = 4
Priority: **Job A1 > Job B1**, Job A2 > Job B2
IMFS: Job A1 = 0.5, Job B1 = 0.25, Job A2 = 0.5, Job B2 = 0.25
CMFS: **Job A1 = 0.5, Job B1 = 0.5**, Job A2 = 0.5, Job B2 = 0.5
ICMFS: 0.5

Figure 3.1 DVS-friendly task scheduling effect: Reducing CMFSs

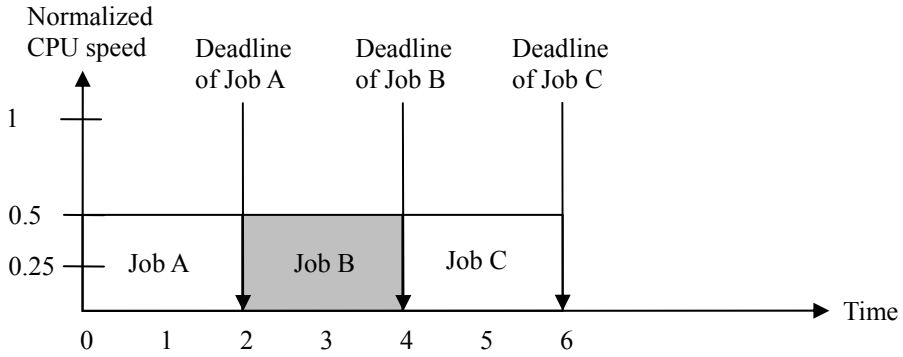
The previous work on the energy optimal CPU speed control showed that the CPU speed profile that minimizes CPU energy consumption whose power consumption is a convex function of speed among all possible CPU speed profiles that complete a given workload exactly on its deadline is to run the CPU constantly at the minimum feasible speed of the workload. However, real DVS-enabled CPUs typically provides a set of discrete CPU speed levels. Thus, if the MFS is not supported in the speed set, the energy optimal CPU speed profile is a combination of the two neighboring high and low CPU speeds. In other

words, generally speaking the energy-optimal CPU speed profile is to keep the CPU speed uniform close to the MFS. Under the CPU-occupancy-based task scheduling this condition cannot be met because of the irregular CMFS due to the mismatches between tasks' CPU speed requirements and priorities. Scheduling tasks according to their deadlines – the tightness and firmness – alleviate this problem and yields more energy savings. Figure 3.2 illustrates this idea.



(a) Ideal speed setting by a task-based autonomous DVS scheme

Release time: All jobs at 0
Priority: **Job B > Job A > Job C**
IMFS: Job A = 0.5, Job B = 0.25, Job C = 1/6
CMFS: **Job A = 1, Job B = 0.25, Job C = 0.25**
ICMFS: 0.5



(b) Speed setting by an interval-based autonomous DVS scheme under a DVS-friendly task scheduling

Release time: All jobs at 0
Priority: **Job A > Job B > Job C**
IMFS: Job A = 0.5, Job B = 0.25, Job C = 1/6
CMFS: **Job A = 0.5, Job B = 0.5, Job C = 0.5**
ICMFS: 0.5

Figure 3.2 DVS-friendly task scheduling effect: Keeping CMFSs uniform near the ICMFS that is the energy optimal CPU speed profile

Power-aware RT task scheduling is such an integration of DVS and the traditional RT task scheduling theory. For GP systems, however, it is still a long way off from the optimal trade-off due to the fundamental incongruity of its underlying task model – tasks have regular periods and job execution times (or their minimum inter-arrival times and worst case execution times are known) and are independent from each others – with the actual tasks running in GP systems.

First, the power-aware HRT task scheduling cannot be applied to GP systems because it is based on the assumption that all the time constrained tasks it will schedule are in accordance with the underlying task model, which is not the case in real GP systems, to save energy without hurting the a priori guarantee that all future hard deadlines will be met. On the other hand, the power-aware SRT task scheduling relaxes the assumption by using aperiodic server techniques and the averages inter-arrival time and execution time aiming at the energy savings with a statistical guarantee on soft deadlines rather than the complete guarantee on hard deadlines. An aperiodic server itself is a task having a regular period (defined by a server deadline) and execution time (defined by the server budget). By running an irregularly behaving SRT task through an aperiodic server whose deadline and budget are the served task's average inter-arrival time and execution time, the task's irregular behavior is tailored to the task model of the RT task scheduling theory and the tasks' soft deadlines are statistically guaranteed.

However, the power-aware SRT task scheduling essentially limits the performance of irregularly behaving SRT tasks such as continuous media applications or interactive applications because of the residuals between the averages and the served task's run-time execution time and inter-arrival time; if a job of the served task requires a longer CPU time than the current server budget, the job should wait for the next time slot allocated to the server. Increasing the server bandwidth would improve the performance but will limit the number of tasks that will be scheduled simultaneously and thus the system utilization eventually. The lowered system utilization increases energy consumption because the pending tasks that were rejected due to the increased bandwidths of existing aperiodic servers will lengthen the system service duration increasing the energy consumption by non CPU system components. A critical drawback of the power-aware SRT task

scheduling for GP systems is that it cannot properly copes with the dependencies among tasks which is easily found in GP systems between a GUI server task and its clients and among the family threads of a multi-threaded application. As a result, the power-aware SRT task scheduling is practical only for a proprietary system running a single threaded continuous media application without using a GUI server task to date.

Based on these findings, this dissertation suggests a new DVS paradigm for GP systems, in which DVS is integrated with a so-called DVS-favor GP task scheduling.

A task scheduling must meet the following requirements to be an eligible DVS-friendly GP task scheduling:

It must be a DVS-friendly task scheduling: Existing autonomous DVS schemes have been designed to work under the traditional CPU-occupancy-based BE (Best Effort) task scheduling mechanisms of GPOSs (General Purpose OSs).

Unlike the traditional CPU-occupancy-based NRT (Non Real Time) task scheduling mechanisms of GPOSs (General Purpose OSs) under which existing autonomous DVS schemes have been designed to work, a DVS-friendly GP task scheduling has to be a time constraint-based task scheduling that assigns a higher priority to a more urgent task such that the *CMFS (Compound Minimum Feasible Speed)* becomes easier to predict as well as is kept closer to the energy-optimal CPU speed (i.e., the *ICMFS (Ideal CMFS)*) across the schedule.

It must be a GP task scheduling: However, the DVS-friendly GP task scheduling must be able to preserve the QoS of the actual tasks running in a commodity-OS-based GP system that behave irregularly (especially as interactive tasks do) and often have precedence constraints among them (e.g., the dependencies between an UI (User Interface) server and client pair or among cooperating family threads) without degrading the system utilization for the optimal energy-performance trade-off.

Chapter 4

GPSched: A DVS-friendly general-purpose best-effort task scheduler for Linux

This chapter describes GPSched, a best-effort (BE) task scheduler for commodity Linux which meets the requirements for an eligible DVS-friendly GP task scheduler described in the previous chapter.

4.1 Underlying ideas

The basic priority assignment rule of the proposed task scheduler is to assign a higher priority to the task having a tighter and harder time constraint. By doing so, GPSched can prevent less urgent tasks from crowding into more urgent tasks. As more urgent tasks do not need to wait until the completion of less urgent tasks, their CPU speed requirements to meet deadlines are kept low – closer to the ICMFS of the given task set – reducing the CPU energy consumption. Moreover, this priority assignment rule naturally improves the system performance because more urgent tasks will be executed first. On the top of this, the approach makes the CPU speed scaling decision easier because most deadlines of urgent tasks will be met at the ICMFS that is, in turn, easy to predict even with a simple CPU utilization-based DVS algorithm.

The two key questions in realizing the above priority assignment rule are (1) how to distinguish more urgent tasks from less urgent tasks and (2) how to cope with the possible CPU speed requirements that are higher than the ICMFS of the given task set.

These questions can be answered based on the following insights into the tasks running in modern GP systems.

First, the activities that all the tasks running in modern GP systems after all fall into one

of the followings: (1) User interface activities such as watching a movie, listening music, playing game, writing documents, and so on, (2) house keeping activities such as system resource management like periodic memory swapping, (3) CPU-bound batch works such as program installations or upgrading, anti-virus checking, scientific calculations, high-level language compilations, and so on, and (4) specific hardware control such as printer driver.

Second, each category of the activities has a regular type and degree of timeliness requirement intrinsic to its purpose. For example, the user interface activities are typically deadline-based and the deadline is after all up to the human perception threshold. House keeping activities are by nature rate-based rather than deadline-based; they do not need to complete each job by a specific deadline but need a certain level of average progress such as one execution at every several seconds. CPU-bound batch works typically do not require any deadline for proper results, but it is desirable not to delay their final completion time because the later completion may cause more energy consumption of other system components but the CPU that should be kept on. The deadline of a hardware control activity is the one difficult to estimate; one solution is to execute such activities at the full CPU speed with the highest priority.

Thus the answer to the first question above, how to distinguish more urgent tasks from less urgent tasks, is that the GPSched task scheduler automatically detects which category of activity each task is involved with by observing tasks' behavior and prioritizes the tasks according to the involved activities as the indicator of task priority. On the other hand, the answer to the second question above, how to cope with possible time constrained sections, is that GPSched speeds up the execution of tasks if the tasks are not completed after some timeouts which are pre-described based on the intrinsic timeliness requirements of the involved activities. In this way, the GPSched reduces the delay in completing an activity which requires a higher CPU speed than the ICMFS of the given task set.

On the other hand, GPSched prevents the less urgent tasks from being starved for an unduly long time by incorporating the GPOS task scheduling rule into it. Once a task's

type is determined and the task is assigned with a priority range correspondingly, the specific priority of the task within the priority range is determined by the GPOS task scheduling rule; the longer time a task is starving, the higher the task's priority becomes. Although tasks are usually not allowed to get a priority beyond its own priority range, RBSRT tasks (kernel threads and daemon processes) are allowed to get as high of a priority as DBSRT tasks (UI server tasks, continous media application tasks, and user interactive tasks) so as to prevent system management jobs from being blocked for an unduly long time. As for NRT tasks, no matter how long the task has been starved, having a priority beyond the NRT tasks' priority range is not allowed; this makes sense because the situation means that there still are some time constrained tasks to do in the system.

In the following section, the detailed task model that GPSched uses based on this ideas are presented.

4.2 Task model

The scope of the time constraint imposed in a commodity-OS-based GP system is not the individual jobs (of tasks) but a transaction of the services that the system provides. Moreover, the precedence constraints among the tasks cooperating to provide a service are naturally bounded within a service transaction. For example, a transaction of the service that responds to a user input event consists of a series of jobs having precedence constraints among them; the user interface (UI) server first detects the user input event then redirect it to the aimed application task, the application task receives the event, processes it, then replies back to the UI server, and finally the UI server puts the response into the video memory. In this series of operations, the time points when the involved jobs are completed do not matter as long as the whole transaction is completed within the human perception threshold known as several tens of milliseconds. This aspect implies that prioritizing individual service transactions instead of the individual jobs can alleviate the difficulty due to the precedence constraints among tasks in a time constraint-based task scheduling without hurting the system performance. The following characteristics of commodity-OS-based GP systems provide an easy way to prioritize service transactions.

- In commodity-OS-based GP systems, any time constraint is after all originated from

either the necessity to meet the human perception threshold in completing a UI service transaction such as processing a movie frame or a sound sample or responding to a user input event, the necessity to provide a timely transaction of system management services such as a memory swapping, or the necessity to complete an operation of the hardware that is not related with any UI activities, such as sending some data to the CD-RW drive burning a CD, on time (note that any operation of the hardware involved with a UI activity is subject to the human perception threshold). Therefore, any service that a commodity-OS-based system provides can be classified broadly into a UI service, a system management service, a non-UI hardware service, or one of the rest non-time constrained services and any job of the task running in a commodity-OS-based system is a part of a transaction of one of the four broad types of service.

- Except the non-UI hardware service transactions whose time constraints depend on particular hardware, the order in urgency among the UI service transactions, the system management service transactions, and the non-time constrained service transactions, which account for the dominant portion of the total service transactions provided by a commodity-OS-based GP system throughout a service duration, is obvious; the human perception threshold is known as several tens of milliseconds, the system management services are set to run typically once a few seconds in contemporary commodity-Linux-based GP systems and, unlike the UI service transactions, are intrinsically rate-based rather than deadline-based because the specific point of time when they execute does not matter as long as a required number of transactions are completed on average, and the non-time constrained services do not have any explicit time constraint. Moreover, the order in urgency hardly switches from system to system, OS to OS, or application to application because the time constraints are originated not from the particular technical methods realizing the services but from their real natures.

Based on these ideas, the GPSched BE task scheduling assigns tasks with one of HRT priority range, *DBSRT (Deadline-Based Soft Real Time)* priority range, *RBSRT (Rate-*

Based Soft Real Time) priority range, and NRT priority range based on the type of the service that the task is involved with. The priority ranges are graded in the listed order and are respectively mapped to *non-UI hardware control services, UI services, system management services*, and the rest non-time constrained services, the four broad types of service that commodity-OS-based GP systems provide. It can be thought that the GPSched BE task scheduler deals with each of the four broad types of service like a task having a known static priority and each transaction of a service like a job of the task. With this approach, the GPSched BE task scheduling can avoid the difficulties due to the existence of precedence constraints among tasks and the irregular and unknown inter-arrival times and execution times of tasks in providing a time constraint-based task scheduling for a commodity-OS-based GP system. It also better preserves system utilization than conventional RT task scheduling mechanisms as it does not tailor the irregular behavior of the tasks running in commodity-OS-based GP systems. This simple time constraint-based task scheduling is, however, effective in improving the energy-performance trade-off for commodity-OS-based GP systems over existing DVS approaches as Figure 3.1 and 3.2 in the previous chapter conceptually illustrated and the experimental results will show.

In this approach, the problem of time constraint-based task scheduling reduces to the problem of finding the type of service that each job is involved with. And the information about the involved service type is provided by the task type detector.

4.3 Task type detector

The task type detector classifies tasks into non-UI hardware control tasks having the HRT priority range, a GUI (Graphic UI) server task, a sound server task, continuous video application tasks, continuous audio application tasks, and interactive application tasks that have the DBSRT priority range, kernel threads and daemons having the RBSRT priority range, and the rests tasks having the NRT priority range based on a variety of task type indicators such as the names of a task and its parent, the memory address space of a task, whether a task has an open socket to the GUI server task or the sound server task, whether a task has directly opened the audio driver, whether a task received a user input

event from the GUI server, and the interval between each task's consecutive video/audio outputs or the user input event receptions.

Task type detector first traces the behavior of every task in the system such as what is the destination of the task's outputs (particular hardware such as CDRW or specific user interface servers such as X or ESD (Enlightened Sound Daemon) of Linux), whether it receives user input events or not, whether it runs in kernel memory space or not, whether it is a well-known user interface server task like X or ESD, and which task is the parent of it (for example, the tasks created by the init process are typically kinds of daemon processes in Linux) to determine the type of activities the task involved with. Task type detector then categorizes tasks into HRT tasks, DBSRT tasks, RBSRT tasks, and NRT tasks if the tasks are involved with some non user interface hardware control activities, user interface activities, house keeping activities, and CPU-bound batch work, respectively. In this classification, Task type detector traces all the family threads and assigns them with the representative type in HRT, DBSRT, RBSRT, and NRT order. NRT tasks are further classified into HINRT (Hard Idling NRT) tasks and SINRT (Soft Idling NRT) tasks by observing whether a NRT task sleeps with explicit timeout or not. HINRT tasks should be executed at the full CPU speed to avoid the increase of system energy consumption due to the extended active durations of the other system components but CPU.

The GPSched then assigns the higher priority range (note that not a particular priority) to the tasks (and all the family threads of a task) in the order of HRT tasks (non user interface hardware controlling tasks), DBSRT (continuous media, interactive, graphic server, and sound server tasks) and HINRT tasks (same priority range), RBSRT tasks (kernel threads and the daemon processes which are not the graphic server or sound server task), and SINRT tasks. The specific priority of each task within a same priority range is determined in the same way of the traditional CPU-occupancy-based priority assignment. By doing so, interactive application tasks are still received the highest priority and show good responsiveness as they do under the GPOSs' native task scheduling. This scheduling approach also provides the robustness in scheduling a set of tasks having precedence constraints from each others because all the family threads of a

deadline-based task or a user interface server and all of its clients are assigned with a same high priority range and thus a successor task can quickly follow its predecessor task. These two features, together with the feature that the GPSched provides some degree of time constraint aware task scheduling while not requiring the knowledge of every job's deadline, make GPScheDVS clearly distinguishable from the power-aware SRT task scheduling.

4.4 Hard-idling NRT task treatment

The task type detector provides not only the information about the type of the service involved with a task but also the information whether a task used to cause the hard idle, the CPU idle time whose length is fixed regardless of the CPU speed (e.g., the CPU idle time during a hard disk access by a task when there is not any other task to run), or the soft idle, the CPU idle time that disappears under a lower CPU speed (e.g., the CPU idle time when there is no task in the run queue). The GPSched BE task scheduler additionally uses this information in the priority range assignment; it assigns the *HINRT (Hard Idling NRT)* tasks, the NRT tasks that sleep causing hard idling, with the DBSRT priority range instead of the NRT priority range such that the system utilization is improved under a low CPU speed as other tasks are allowed to use the hard idle time caused by HINRT tasks.

Chapter 5

GPScheDVS

This chapter describes GPScheDVS, the integrated approach of GPSched and a task-based autonomous DVS scheme called GPSDVS which is customized to GPSched.

5.1. Introduction

Any autonomous DVS schemes can be used under the GPSched which is described in Chapter 4. However, for a maximum benefit, a customized DVS scheme is necessary. For this purpose a task-based DVS scheme customized to GPSched called GPSDVS was designed. GPScheDVS is, therefore, an integrated approach of GPSched and GPSDVS.

5.2. GPSDVS: A task-based DVS scheme customized to GPSched

Under the support of the DVS-friendly GP task scheduling, future CMFSs are predicted by the GPSDVS which consists of the four instances of adaptive-interval-based DVS (AIDVS) [93], an interval-based DVS algorithm that adapts its interval length to the arrival pattern of workload, respectively devoted to the four priority ranges. Each instance of AIDVS predicts the future CMFSs required by the service transactions for the priority range that it is devoted and the final decision on the CPU operation point is made according to the most stringent one among the respectively predicted CMFSs by the four instances of AIDVS.

5.2.1. AIDVS: Adaptive interval-based DVS algorithm

The use of AIDVS is another important feature that makes GPScheDVS distinguishable. AIDVS outperforms existing interval-based DVS algorithms that have a fundamental dilemma of choice between the energy efficiency and system performance as they are based on a fixed interval length; too long interval length causes the poor responsiveness

in CPU speed adjustment resulting in the QoS degradation, whereas too short interval length often causes an interval to be located in the middle of CPU busy duration or CPU idle duration such that the DVS algorithm is mislead to overestimate or underestimate future CMFSs resulting in an unnecessarily large amount of energy consumption or the QoS degradation, respectively. AIDVS overcomes this problem by adapting its interval length to the arrival pattern of workload such that an interval begins upon the arrival of a workload (the beginning of busy half), continues upon the completion of the workload for the following CPU idle duration (the beginning of idle half), and finally ends upon the arrival of the next workload. In this way, an interval of AIDVS is never located in the middle of CPU busy duration or CPU idle duration regardless of the arrival pattern of workloads, allowing AIDVS to avoid either the overestimation or underestimation of future CMFSs. Moreover, AIDVS can keep the proper level of responsiveness particular to the arrival pattern of workload as it is always given a chance to adjust the CPU speed after the current workload is completed and before the next workload arrives. For a long lasting workload, AIDVS gradually increases the CPU speed using a time out that is set according to the typical time constraint particular to each broad type of service.

Note that the CMFS required by a time constrained service transaction is given by the following equation.

$$\frac{(W_{own} + W_{blocking})}{D_{own}}, \qquad \text{(Equation 5.1)}$$

where W_{own} is the total workload of the jobs that consist of the transaction, $W_{blocking}$ is the total workload of the jobs that block the involved jobs by having higher priorities than them, and D_{own} is the deadline of the transaction.

To take the $W_{blocking}$ term into account, each instance of AIDVS counts also the workloads of the tasks assigned with a higher priority range than the priority range that the AIDVS is devoted. The instance of AIDVS that is devoted to the NRT priority range, the lowest priority range, counts all tasks' workloads and works as a total CPU utilization tracer. Upon overload situation, this instance of AIDVS leads the CPU speed to the maximum

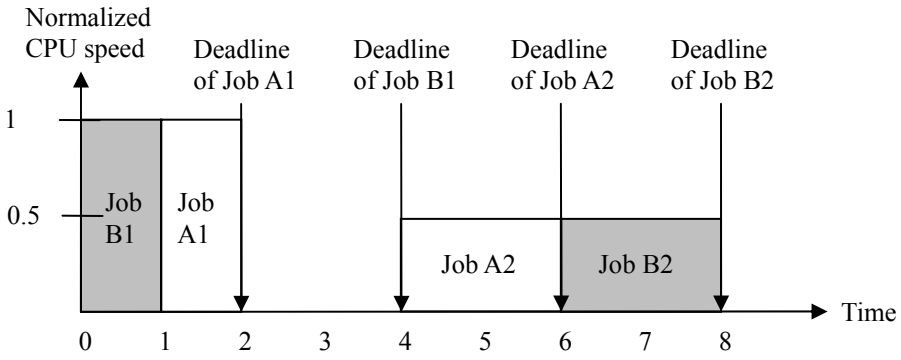
such that the total system energy consumption is not increased by preventing the extension of system service duration.

Each instance of AIDVS also relies on the intrinsic tightness of the classified tasks' involved activities to cope with the possible time constrained sections within the entire interval that require the higher CPU speeds than the ICMFS. For example, the second level AIDVS devoted to DBSRT, HINRT, and HRT tasks increases the CPU speed by one level if a busy duration lasts over 20 milliseconds of FSE (Full Speed Equivalent) timeout and upon every 10 milliseconds after the first speed up. The FSE timeouts for the first level, third level, and the lowest level AIDVSs are defined as (5 milliseconds, 2.5 milliseconds), (100 milliseconds, 50 milliseconds), and (500 milliseconds, 250 milliseconds), respectively, also considering the intrinsic time constraints of the involved activities.

Note that these customized policies to the individual types of tasks with a single AIDVS algorithm are possible because the GPSched and GPSDVS share the task type information. In contrast, an existing DVS approach – even a task-based DVS scheme which adjusts CPU speed on a per-task basis – cannot realize properly customized policies to each type of tasks even though it is used under GPSched because it does not share the task type information with GPSched.

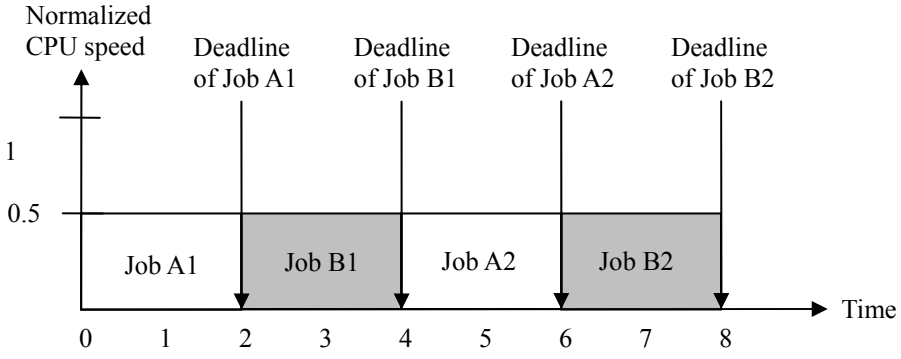
5.3. GPScheDVS: The integration of GPSched and GPSDVS

GPScheDVS is the integrated approach of GPSched and GPSDVS [94]. Figures 5.1 and 5.2 show how both GPSched and GPSDVS make GPScheDVS superior over existing autonomous DVS schemes in reducing energy consumption without degrading system performance. As Figure 5.1 depicts, an autonomous DVS scheme can not fully realize the potential of DVS no matter how accurately it predicts tasks' CMFSs as long as it runs under GPOS task scheduling because this task scheduling causes tasks' CMFSs to be unnecessarily high obstructing the DVS scheme in lowering CPU speed for a reduced energy consumption. Unlike the GPOS task scheduling, GPSched, as a DVS-friendly GP task scheduling, keeps tasks' CMFSs low and uniform and, thus, allows a DVS scheme to lower CPU speed for a reduced energy consumption.



(a) The combination of GPOS task scheduling and GPSDVS

The unnecessarily high CMFSs of job A1 and B1 due to the mismatch of priority and urgency under GPOS task scheduling limits the achievable amount of energy savings.



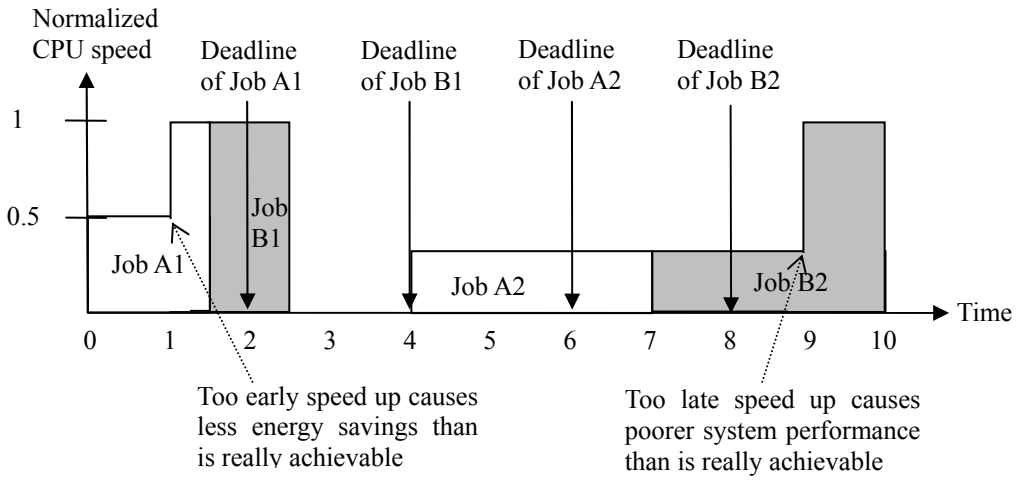
(b) The combination of GPSched and GPSDVS

By matching tasks' urgency and priority, GPSched can keep the CMFSs low and uniform maximizing the achievable amount of energy savings.

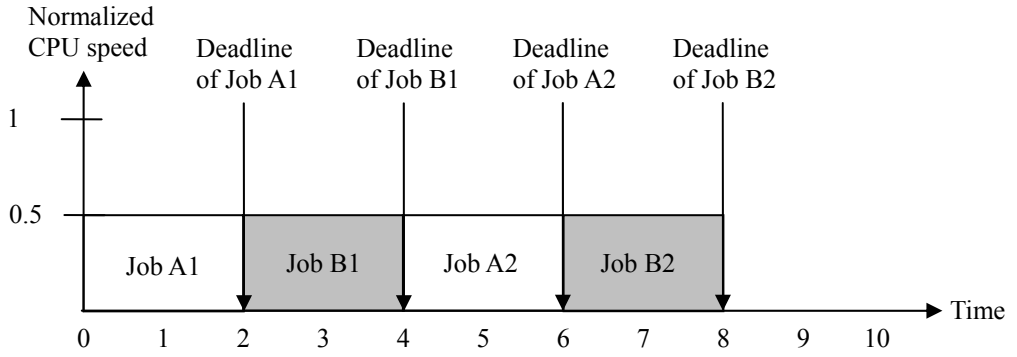
Figure 5.1 The superiority of GPScheDVS in power-performance trade-off over existing autonomous DVS schemes due to GPSched

As Figure 5.2 illustrates, however, using GPSched alone still does not result in the minimized energy consumption. When an existing autonomous DVS scheme is used, it tends to cause the CPU to speed up either too early or too late, resulting in an unnecessarily larger energy consumption or poorer system performance than are really achievable even under GPSched taks scheduling. This is because existing DVS schemes can not share task information such as task type and time constraint with GPSched. When the DVS scheme overestimate tasks' CMFSs, it will raises the CPU speed prematurely as Figure 5.2 (a) examplifies. On the other hand, when the scheme underestimates the

CMFSs, the CPU speed will be kept low for a unduly long time. This situation is also depicted in Figure 5.2 (a). Unlike existing DVS schemes, GPSDVS shares the task information with GPSched and, thus, does not raise the CPU speed either too early or too lately.



(a) The combination of GPSched and an existing autonomous DVS scheme



(b) The combination of GPSched and GPSDVS

As sharing task type and time constraint with GPSched,
GPSDVS can avoid too early and too late speed up.

Figure 5.2 The superiority of GPScheDVS in power-performance trade-off over existing autonomous DVS schemes due to GPSDVS

GPSched first makes the CPU speed requirements of tasks low, closer to the ICMFS of the given task set. Under the GPSched task scheduling, GPSDVS detects the CPU speed requirements of each priority range of tasks by using the four instances of AIDVS, then,

adjusts the CPU speed based on the most stringent one among the four CPU speed requirements.

GPScheDVS works in either system energy centric (SE) mode (GPScheDVS-SE) aiming at system energy-performance trade-off or CPU power centric (CP) mode (GPScheDVS-CP) pursuing CPU heat dissipation-performance trade-off. Both modes of GPScheDVS work in a same way but the treatment of NRT tasks.

In the system energy centric mode, GPScheDVS assigns HINRT tasks with the DBSRT priority range and keep the CPU speed to the maximum as long as such a HINRT task is in the run queue so as to prevent the lengthened system service time that increases non-CPU components' energy consumptions and eventually the entire system energy consumption. The result is a larger CPU energy consumption but a less system energy consumption than existing DVS schemes.

In the CPU power centric mode, GPScheDVS executes every task involved with non-time constrained services at the critical CPU speed, i.e., the CPU speed that yields the lowest CPU energy consumption. GPScheDVS, however, still executes the tasks involved with time constrained services at the appropriate CPU speeds that can meet the deadlines. In this way, GPScheDVS-CP limits the CPU heat generation while preserving system performance at the cost of larger system energy consumption. The GPScheDVS-CP is especially for the commodity-OS-based GP systems such as desktop where the primary concern of user is often the CPU heat generation rather than the system energy consumption.

Note that the selective application between these two operation modes – another customized policy to the target system type – is also possible because the GPSched and the GPSDVS share the task type information – whether the task is NRT or not and the NRT task is HINRT task or not – provided by the task type detector.

5.4. Implementation issues

5.4.1. Fixed point arithmetic

All DVS schemes measure the fraction of CPU busy and idle times and calculate their ratio so as to determine the most appropriate level of CPU OP. This common mechanism requires the arithmetic on fractional numbers. However, the original Linux kernel does not support the arithmetic on fractional numbers with floating point types. For this reason, we implemented a set of fixed point arithmetic functions.

The arithmetic functions were designed for an U(20, 12) fixed point fractional number format, i.e., an unsigned 32-bit number format in which the upper 20 bits (bit31~bit12) are used to represent the integer part and the lower 12 bits (bit11~bit0) are used to represent the fractional part. The bit12 has the value of $2^0 = 1$, and the bit13 has the value of $2^1 = 2$, and so forth. On the other hand, the bit11 has the value of $2^{-1} = 0.5$, and the bit10 has the value of $2^{-2} = 0.25$, and so on. The range of the value that the format can represent is from 0 to 2^{20}-2^{12} = 1048575.999755859375. With microsecond of unit, this range corresponds to 0~1.048 seconds which range is sufficient for most DVS schemes (for example the largest DVS interval, which is used in SpeedStep, is 0.5 seconds).

This work implemented fpmul12() and fpdiv12() functions in in-line assembly codes which automatically matches the bit lengths for fractional part after and before the multiplication and division, respectively. For addition and subtraction, we don't need to any particular arithmetic functions because such types of arithmetic do not modify the bit numbers for fractional parts; we only need to confirm that the operand and operator are in U(20, 12) fixed point format in these cases (we can convert an integer to U(20, 12) fixed format by 12 bit left shift).

5.4.2. Time counters

Flautner, the first author of Vertigo, directly used the RDTSC instruction, which returns the current CPU time stamp counter (TSC) value, to get the timing information required for the operation of Vertigo. However, the kernel programmers' community recommends not to use RDTSC directly but to use the do_gettimeofday kernel function which also

uses RDTSC internally and returns the current time in microseconds. Briefly speaking, directly using RDTSC is risky, whereas using do_gettimeofday is safe.

Thus, if the two approaches provide a same degree of time resolution and availability (i.e., whether we can call it whenever we want), there is no reason for persisting in directly using the RDTSC. According to my test, using do_gettimeofday does not have any disadvantage compared to using RDTSC. Therefore, GPScheDVS is based on do_gettimeofday rather than RDTSC.

Chapter 6

Experiments

A set of experiments was performed on GPScheDVS described in the previous chapter. This chapter first describes the goal and method of the experiments and then presents and discusses the experimental results.

6.1 Introduction

The fundamental goal of the experiments is to verify the effectiveness of the suggested autonomous CPU speed control paradigm in extending battery lifetime and reducing CPU heat dissipation while improving system performance over the current paradigm under the real-life usage scenarios of typical contemporary GP mobile PCs. For this purpose, the performances of GPScheDVS and the existing autonomous DVS schemes that follow the current paradigm were compared.

The specific questions to deal with in the experiments are how much system energy consumption, the pertinent metric to battery lifetime, the system-energy-centric mode GPScheDVS (GPScheDVS-SE) will reduce and how much average CPU power consumption, the most relevant metric to CPU heat dissipation, the CPU-power-centric mode GPScheDVS (GPScheDVS-CP) will reduce both while improving the QoS of SRT applications such as continuous media and interactive applications over the existing autonomous DVS schemes under the following usage scenarios:

- The typical usage scenarios where a SRT application runs alone or with the NRT applications that impose an amount of workload spanning from light to moderate to heavy in the background: The experimental results under this type of usage scenarios will be presented in Section 6.3.1.

- The special usage scenarios where a SRT application runs with a HINRT application:

As Chapter 4 pointed out, executing HINRT tasks at a low CPU operating point extends the running time of non-CPU components and, accordingly, increases their energy consumptions. The GPScheDVS-SE prevents this problem by executing them at the highest CPU operating point as like no-DVS case. The point of interest under this type of usage scenarios is, thus, how much system energy consumption the GPScheDVS-SE reduces over not only the existing autonomous DVS schemes but also the no-DVS case while preserving the QoS of SRT applications. The results of these experiments will be given in Section 6.3.2.

- The special usage scenarios where only SRT applications are run without having any NRT application in the background such that the workload is mostly made up of SRT tasks: As a DVS-friendly task scheduler, GPSched provides a time constraint-based task prioritization to make tasks' CMFSs uniform near the ICMFS of the task set. But, as described in Chapter 4, the time constraint-based task prioritization of GPSched is simply among the four types of tasks, i.e., HRT, DBSRT, RBSRT, and NRT tasks. A naturally arising question is, thus, can GPScheDVS further reduce system energy consumption and CPU power consumption over existing autonomous DVS schemes when the workload is mostly made up of SRT tasks? This type of usage scenarios are to deal with this question and the experimental results will be shown in Section 6.3.3.

For these goals, GPScheDVS and three representative existing autonomous DVS schemes were implemented in the kernel of a commodity Linux and experimented on a real laptop equipped with an Intel Pentium-M, a DVS-enabled CPU. Two interval-based DVS schemes, the OnDemand cpufreq governor (OCG) for Enhanced Intel SpeedStep Technology (EIST) [40] and AVG3 [41], and a task-based DVS scheme, Vertigo [45], were selected as the existing autonomous DVS schemes to compare with GPScheDVS. The detailed features of the experimental platform will be presented in Section 6.2.1. Section 6.2.2 will describe the implementation details of these existing autonomous DVS schemes.

For the realistic assessment, the experiments were conducted with various common Linux applications including two continuous media players, a web-browser, a game, a document

editor, an anti-virus scanner, and a programming language compiler. The benchmarks and their usage scenarios will be described in Section 6.2.3.

Experimented DVS schemes were compared in five performance metrics: system energy consumption, CPU energy consumption, average CPU power consumption, the QoS of SRT applications, and the overhead of DVS schemes in time, system energy consumption, and the number of CPU operating point transitions. Section 6.2.4 will define the metrics and the methods to measure them in the concrete.

In this research, the energy and power consumptions for each experiment were estimated by using energy models due to the practical difficulties in the on-line measurement with electrical instruments. The models have the power consumptions of the experimental platform measured beforehand with an electrical instrument at each CPU operating point as its coefficients and the fractions of time spent at each CPU operating point during the experiment as its explanatory variables. The models were validated by comparing the values calculated using it and measured with an electrical instrument for 168 experiments. Section 6.2.4.1 describes the models and shows the validation process.

The experiments were conducted using two Linux-based tools developed for this research, *Tweak-PM* and *DVS-suite*. Tweak-PM is a Linux loadable kernel module (LKM) which probes the properties of a given Intel Pentium-M CPU and allows a user to manually control the CPU functions such as the thermal monitors (TM1/TM2), on-demand clock modulation (ODCM), and EIST. Tweak-PM was originally devised as a pilot tool for DVS-suite but used to manually set the CPU operating point while measuring the power consumptions of the experimental platform to build the energy models. DVS-suite is the collection of the Linux kernel extensions that implement the DVS schemes to test and the facilities to measure, record, and report the experimental metrics. The implemented DVS schemes can be switched on-the-fly, and the metric trace facilities can be turned on or off individually on demand of users through a LKM called DVS-suite module. Section 6.2.5 will describe the functions and structures of Tweak-PM and DVS-suite.

The experimental results will be presented and discussed in Section 6.3. Finally, Section

6.4 will conclude this chapter.

6.2 Experimental method

This section describes the experimental method. Specific issues include the experimental platform, the implementation of selected existing autonomous DVS schemes, the selected benchmarks and their usage scenarios, DVS performance metrics and their measurement methods, and the two tools developed in this research for the experiments – Tweak-PM and DVS-suite.

6.2.1 Platform

The experiments were conducted on a Dell Latitude D600 laptop which is equipped with an Intel Pentium-M processor and runs a Red Hat Linux 9 Shrike OS. Figure 6.1 shows the experimental platform and the system power consumption measurement environment based on a HP 54602B oscilloscope [89] and a LEM PR30 current probe [90]. Table 6.1 describes the detailed features of the experimental platform.

Table 6.1 Features of the experimental platform (Dell Latitude D600 laptop)

Processor	Intel Pentium-M 725 processor 1.6GHz (Dothan core, Rev. B1, signature 0x6D6)
Main memory size	512 MB
Graphic adaptor	ATI Radeon Mobility 9000 (32 MB of video RAM, No Genlock or Frame lock support [87, 88])
Audio device	The audio hardware integrated in Intel I810 chipset (65536 Bytes of audio buffer)
Wireless network device	Lucent Orinoco PC card silver (11 Mb/second of maximum data transfer rate)
Operating system	Red Hat Linux 9 Shrike OS extended with DVS-suite (Linux kernel version 2.4.20-8, No dynamic power management (DPM) support)

Pentium-M is a DVS-enabled CPU whose clock frequency and supply voltage can be adjusted on-the-fly while running programs. The Pentium-M of the experimental platform supports six CPU operating points, i.e., the pairs of clock frequency and supply voltage [30]. The Pentium-M hardware mechanism that allows software to adjust the CPU operating point is called Enhanced Intel SpeedStep Technology (EIST) [29]. With EIST, the current CPU operating point can be probed by reading IA32_PERF_STATUS (MSR

index 0x198), a model specific register (MSR) particular to Pentium-M. And, a new CPU operating point can be enforced by updating IA32_PERF_CTL MSR (MSR index 0x199). These MSRs can be read and written by OS or even a user-space application through RDMSR and WRMSR instructions, respectively. Table 6.2 describes the six CPU operating points supported by the Pentium-M of the experimental platform with the corresponding MSR values.

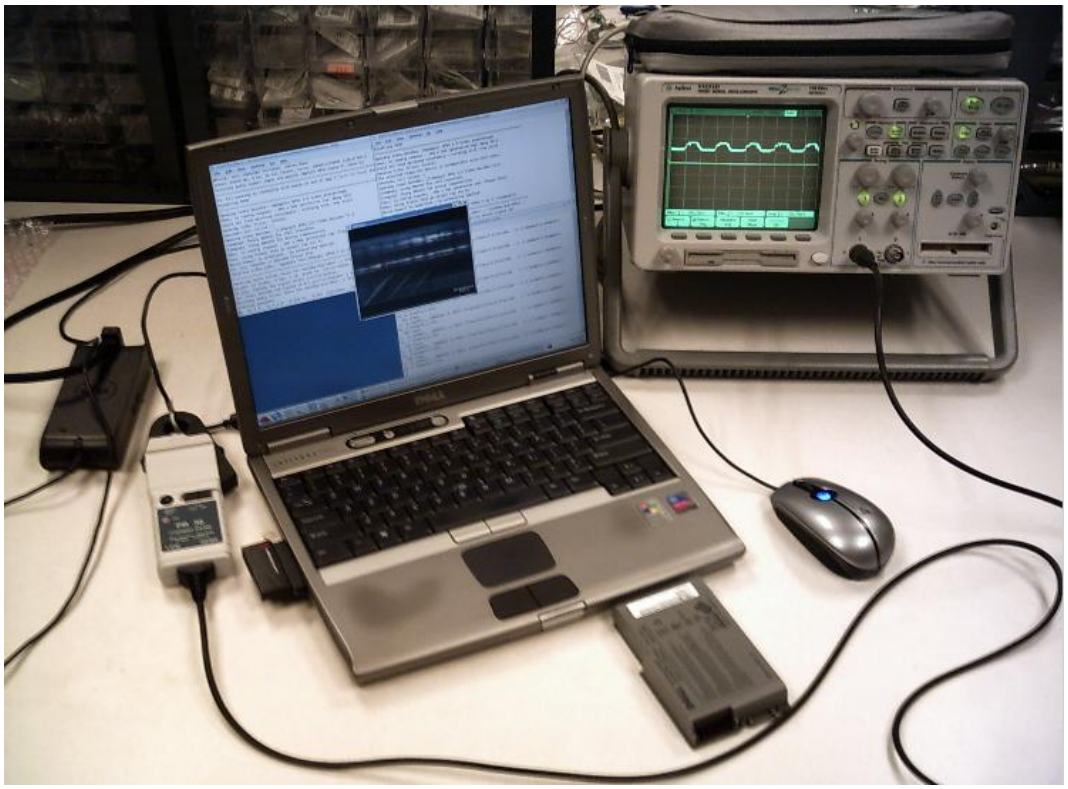


Figure 6.1 Experimental platform and the system power consumption measurement setup (the built-in battery of the experimental platform was removed to avoid the impact of the battery on the measured results)

Table 6.2 Intel Pentium-M 725 processor operating points.

CPU operating point	CPU clock frequency (MHz)	CPU supply voltage (V)	MSR value
CPU_OP0	1598.6960	1.340	0x1028
CPU_OP1	1398.8590	1.276	0xE24
CPU_OP2	1199.0220	1.212	0xC20
CPU_OP3	999.1850	1.132	0xA1B
CPU_OP4	799.3480	1.068	0x817
CPU_OP5	599.5110	0.988	0x612

EIST is not exclusively used by OS or a user-space application; Intel thermal monitor 2 (TM2), another hardware mechanism particular to the Pentium-M, uses EIST to protect the CPU from the physical damage due to overheating. TM2 keeps observing the CPU junction temperature, i.e., the temperature at the P-N junction of the CMOS CPU through a built-in thermal sensor. If the junction temperature exceeds the thermal threshold particular to Pentium-M (100 degrees Celsius), TM2 keeps the CPU operating point at CPU_OP4 until the junction temperature falls down below the thermal threshold. Note that the CPU operating point CPU_OP4 is just the default setting. The TM2 target CPU operating point can be changed by setting bit 15:0 of MSR_THERM2_CTL MSR (MSR index 0x19D).

Pentium-M has another thermal monitor TM1. Unlike the TM2 that uses EIST, TM1 uses the thermal control circuit (TCC) to reduce the junction temperature. Upon activation, TCC first reduces the clock ratio by half by turning off every other original clock tick and keeps the reduced clock ratio until the junction temperature falls down below the threshold. Similar to EIST, TCC can be programmed by software. And, in that case, the clock modulation ratio has not necessarily to be 50%; the ratio can be chosen among 12.5, 25, 37.5, 50, 62.5, 75, and 87.5%. This software-based clock modulation technique is called on-demand clock modulation (ODCM). TM1 and ODCM are the legacy mechanisms introduced to the early Intel processors that were not equipped with EIST. Pentium-M does not allow both of the thermal monitors to be turned on. If TM2 is activated, TM1 is deactivated automatically. And, if TM2 is deactivated again, TM1 is activated [30].

The experiments were conducted with TM2 activated (and TM1 deactivated) to protect the Pentium-M from the possible damage due to overheating. To activate any thermal monitor, the TM1E flag (bit 03) of IA32_MISC_ENABLE MSR (MSR index 0x1A0) has to be set first. Then, TM2 is activated by setting the TM_SELECT flag (bit 16) of MSR_THERM2_CTL MSR. Resetting TM_SELECT flag deactivates TM2 and activates TM1. Note that switching the thermal monitors in this way applies to only the early versions of Pentium-M whose signature is either 0x69n or 0x6Dn such as the Pentium-M used in the experiments. For the recent versions of Pentium-M, the switch from TM1 to

TM2 is performed simply by setting the TM2E flag (bit 13) of IA32_MISC_ENABLE MSR. Again, resetting the TM2E flag switches TM2 to TM1 [31].

A notable characteristic of the Red Hat Linux 9 used in the experiments is that it does not support any DPM for non-CPU components unless it is extended with additional DPM patches. This OS feature allows the energy models mentioned in Section 6.1 to accurately approximate the actual system and CPU energy consumptions. Section 6.2.4.1 will show the reason.

Another system feature worth noting is that the graphic adaptor does not support the Genlock extension that synchronizes the monitor refresh rate to the frame rate of the video source in order to avoid the judders in a video playback [87, 88]. This system feature causes the video output period of a continuous media application to deviate from the nominal video frame rate affecting the video deadline meet ratio measurement method as Section 6.2.4.4 will show.

6.2.2 Implementation of the existing autonomous DVS schemes

As Section 6.1 explained, OCG, AVG3, and Vertigo were selected as the representative existing DVS schemes that follow the current DVS paradigm for commodity-OS-based GP PCs and implemented in the Linux kernel for the comparison with GPScheDVS.

The interval-based DVS scheme OCG has a simple mechanism. It keeps sampling the CPU utilization at every 110 milliseconds of interval. If the last interval's CPU utilization was higher than the 80% of upper bound threshold, OCG rises up the CPU operating point by one level from the current level to the next higher level. Similarly, if the last interval's CPU utilization was less than the 20% of lower bound threshold, OCG decreases the CPU operating point to the next lower level. If the next interval's CPU utilization is still out of the bounds, OCG keeps rising up or lowering down the CPU operating point by one level every interval until the CPU utilization is within the bounds. Figure 6.2 shows the pseudo code of OCG.

Unlike OCG that predicts the next CPU operating point based only on the last interval's

CPU utilization, another interval-based DVS scheme AVG3 predicts the next CPU operating point based on the moving average of all the past intervals' CPU utilizations defined by Equation 6.1. By using the moving average, AVG3 can have a long-term view of CPU utilization and more accurately predict the next CPU operating point than OCG. The moving average and the CPU operating point are updated at every 20 milliseconds [72].

```
1     OCG main function:
2
3     Increment interval by ds_counter.elapsed - interval_base
4     Initialize interval_base to ds_counter.elapsed
5     Increment busy by ds_counter.busy_total - busy_base
6     Initialize busy_base to ds_counter.busy_total
7     IF interval >= 110 msec
8        Calculate cpu_utilization by busy / interval
9        IF cpu_utilization > 0.8
10          CASE Current CPU_OP
11             CPU_OP0: Set next CPU_OP to CPU_OP0
12             CPU_OP1: Set next CPU_OP to CPU_OP0
13             CPU_OP2: Set next CPU_OP to CPU_OP1
14             CPU_OP3: Set next CPU_OP to CPU_OP2
15             CPU_OP4: Set next CPU_OP to CPU_OP3
16             CPU_OP5: Set next CPU_OP to CPU_OP4
17          ENDCASE
18          ELSE IF cpu_utilization < 0.2
19             CASE Current CPU_OP
20                CPU_OP0: Set next CPU_OP to CPU_OP1
21                CPU_OP1: Set next CPU_OP to CPU_OP2
22                CPU_OP2: Set next CPU_OP to CPU_OP3
23                CPU_OP3: Set next CPU_OP to CPU_OP4
24                CPU_OP4: Set next CPU_OP to CPU_OP5
25                CPU_OP5: Set next CPU_OP to CPU_OP5
26             ENDCASE
27          ELSE
28             Keep the current CPU_OP
29          ENDIF
30          Initialize interval to 0
31          Initialize busy to 0
32       ENDIF
33    ENDIF
```

Figure 6.2 OCG pseudo code

$$AVG\_CPU\_U_{New} = \frac{AVG\_CPU\_U_{Old} \times W + CPU\_U_{Last}}{1 + W}, \quad \text{(Equation 6.1)}$$

where CPU_U_{Last} is the last interval's CPU utilization, $AVG\_CPU\_U_{Old}$ is the moving

average as yet not updated with CPU_U_{Last}, $AVG\_CPU\_U_{New}$ is the moving average updated with CPU_U_{Last}, and W is the weight whose value is 3 for AVG3 [72].

AVG3 is distinguishable from OCG also in that it switches the CPU operating point to the target level at once without passing through the intermediate levels between the current level and the target level. For example, if the current moving average is 0.5, the next CPU operating point will be that the ratio of its speed to the highest CPU speed is 0.5. This feature makes AVG3 enforce the CPU operating point more precisely than OCG. Figure 6.3 shows the pseudo code of AVG3.

```
1     AVG3 main function:
2
3     Increment interval by ds_counter.elapsed - interval_base
4     Initialize interval_base to ds_counter.elapsed
5     IF interval >= 20 msec
6       Calculate busy by ds_counter.busy_total - busy_base
7       Initialize busy_base to ds_counter.busy_total
8       Calculate cpu_u_last by busy / interval
9       Update avg_cpu_u by (avg_cpu_u * w + cpu_u_last) / (1 + w)
10      CALL cpu_utilization_to_cpu_op WITH avg_cpu_u RETURNING cpu_op_new
11      IF the current CPU_OP is not cpu_op_new
12        Set the next CPU_OP to cpu_op_new
13      ENDIF
14      Initialize interval to 0
15    ENDIF
16
17    cpu_utilization_to_cpu_op (cpu_u):
18
19    IF cpu_u >= 0.875     # 1400 MHz / 1600 MHz
20      cpu_op_new = CPU_OP0
21    ELSE IF cpu_u > 0.750 # 1200 MHz / 1600 MHz
22      cpu_op_new = CPU_OP1
23    ELSE IF cpu_u > 0.625 # 1000 MHz / 1600 MHz
24      cpu_op_new = CPU_OP2
25    ELSE IF cpu_u > 0.500 # 800 MHz / 1600 MHz
26      cpu_op_new = CPU_OP3
27    ELSE IF cpu_u > 0.375 # 600 MHz / 1600 MHz
28      cpu_op_new = CPU_OP4
29    ELSE
30      cpu_op_new = CPU_OP5
31    ENDIF
32    RETURN cpu_op_new
```

Figure 6.3 AVG3 pseudo code

OCG and AVG3 are activated every time a context switch, a system call, or an interrupt is completed by DVS-suite as is GPScheDVS. Upon activation, they first increment the

elapsed time and the busy duration since the last time the CPU utilization and the moving average (AVG3 only) were calculated and, then, check if the end of the current interval is reached. And, if so, they calculate the CPU utilization for the last interval, update the moving average (AVG3 only), and adjust the CPU operating point if necessary. For the calculation of fractional numbers, they share the fixed-point arithmetic primitives described in Chapter 5 that were devised for GPScheDVS.

Unlike these interval-based DVS schemes that treat tasks in the aggregate, the task-based DVS scheme Vertigo treats tasks individually both in predicting and enforcing the appropriate future CPU operating point and fundamentally improves the accuracy in the prediction and the precision in the enforcement. Vertigo predicts the future CPU operating points in two steps, first, with the perspectives-based algorithm and, then, the algorithm for interactive tasks. The perspectives-based algorithm keeps updating the moving average of CPU utilization on a per-task basis. The moving average of a task's CPU utilization is updated when the task is scheduled-in again since the last time it willingly relinquished the CPU. At such a schedule-in time point, the task's own CPU utilization is measured for the interval of time between the last and current such schedule-in times and used to update the moving average.

In addition to each task's average CPU utilization, Vertigo traces the average amount of workload required to respond to user input events. When there is an ongoing response procedure, Vertigo predicts the future CPU operating point to complete the procedure in 50 milliseconds based on the average amount of workload observed in the past. This CPU operating point overrides the CPU operating points predicted based on the per-task average CPU utilizations. If there is no ongoing response procedure, the future CPU operating points are set thoroughly based on the per-task average CPU utilizations. Figure 6.4 shows the pseudo code for the Vertigo implemented in this research.

Figure 6.4 Vertigo pseudo code (Lines 1~5)

```
1    vertigo main function:
2
3    CALL perspectives_based_algorithm RETURNING cpu_op_new_p
4    CALL algorithm_for_interactive_applications RETURNING cpu_op_new_i
5
```

Figure 6.4 Vertigo pseudo code (Lines 6~57)

```
6     IF vertigo_status.flag_force_interactive_perf is set to 1
7       RETURN cpu_op_new_i
8     ELSE
9       RETURN cpu_op_new_p
10    ENDIF
11
12    perspectives_based_algorithm:
13
14    # Basic per-task operation at every context switch
15    IF ds_parameter.entry_type is DS_ENTRY_SWITCH_TO
16
17      IF prev_p is not swapper
18        IF vertigo_status.pds_table[prev_p->pid] exists
19          IF prev_p terminated
20            Deallocate vertigo_status.pds_table[prev_p->pid]
21          ELSE
22            Increment vertigo_status.pds_table[prev_p->pid]->work_fse by \
23            ds_counter.busy_fse - vertigo_status.job_work_fse_base
24            IF prev_p went to sleep
25              Reset vertigo_status.pds_table[prev_p->pid]->run_bit to 0
26            ENDIF
27          ENDIF
28        ENDIF
29      ENDIF
30
31      IF next_p is not swapper
32        IF vertigo_status.pds_table[next_p->pid] does not exist
33          Allocate and initialize vertigo_status.pds_table[next_p->pid]
34          Set vertigo_status.pds_table[next_p->pid]->run_bit to 1
35        ELSE
36          IF vertigo_status.pds_table[next_p->pid]->run_bit is 0
37            Set vertigo_status.pds_table[next_p->pid]->run_bit to 1
38            Update work_est, the moving average of work_fse, of next_p \
39            as follows:
40              vertigo_status.pds_table[next_p->pid]->work_est = \
41              (vertigo_status.pds_table[next_p->pid]->work_est * weight + \
42               vertigo_status.pds_table[next_p->pid]->work_fse) \
43               / (1 + weight)
44            Calculate idle within the last interval of next_p as follows:
45              idle = \
46              ds_counter.idle_total - \
47              vertigo_status.pds_table[next_p->pid]->idle_base
48            Update deadline, the moving average of work_fse + idle, \
49            of next_p as follows:
50              vertigo_status.pds_table[next_p->pid]->deadline = \
51              (vertigo_status.pds_table[next_p->pid]->deadline * weight + \
52               (vertigo_status.pds_table[next_p->pid]->work_fse + \
53                vertigo_status.pds_table[next_p->pid]->idle)) \
54               / (1 + weight)
55            Calculate perf, the new performance requirement, \
56            of next_p as follows:
57              vertigo_status.pds_table[next_p->pid]->perf = \
```

Figure 6.4 Vertigo pseudo code (Lines 58~109)

```
58                vertigo_status.pds_table[next_p->pid]->work_est \
59                / vertigo_status.pds_table[next_p->pid]->deadline
60              Initialize vertigo_status.pds_table[next_p->pid]->work_fse to 0
61              Initialize vertigo_status.pds_table[next_p->pid]->idle_base \
62              to ds_counter.idle_total
63              CALL cpu_utilization_to_cpu_op WITH perf RETURNING cpu_op_new
64            ENDIF
65          ENDIF
66        ELSE
67          RETURN the current CPU_OP
68        ENDIF
69
70        Initialize vertigo_status.job_work_fse_base to ds_counter.busy_fse
71        Initialize vertigo_status.non_preemption_lasting to 0
72        Initialize vertigo_status.non_preemption_lasting_base \
73        to ds_counter.elapsed
74
75      # Additional operation to improve the response time of long lasting jobs
76      # If ds_parameter.entry_type is DS_ENTRY_RET_FROM_SYSTEM_CALL
77      ELSE
78
79        IF current is not swapper
80          Increment vertigo_status.non_preemption_lasting by \
81            ds_counter.elapsed - vertigo_status.non_preemption_lasting_base
82          Initialize vertigo_status.non_preemption_lasting_base \
83            to ds_counter.elapsed
84          IF vertigo_status.non_preemption_lasting >= 100 msec of \
85          NON_PREEMPTION_THRESHOLD
86            Initialize vertigo_status.non_preemption_lasting to 0
87            Increment vertigo_status.pds_table[current->pid]->work_fse by \
88              ds_counter.busy_fse - vertigo_status.job_work_fse_base
89            Update work_est, the moving average of work_fse, \
90            of current as follows:
91              vertigo_status.pds_table[current->pid]->work_est = \
92              (vertigo_status.pds_table[current->pid]->work_est * weight + \
93               vertigo_status.pds_table[current->pid]->work_fse) \
94              / (1 + weight)
95            Calculate idle since the schedule-in of current as follows:
96              idle = \
97              ds_counter.idle_total - \
98              vertigo_status.pds_table[current->pid]->idle_base
99            Update deadline, the moving average of work_fse + idle, \
100           of current as follows:
101             vertigo_status.pds_table[current->pid]->deadline = \
102             (vertigo_status.pds_table[current->pid]->deadline * weight + \
103              (vertigo_status.pds_table[current->pid]->work_fse + \
104               vertigo_status.pds_table[current->pid]->idle)) \
105              / (1 + weight)
106           Calculate perf, the new performance requirement, \
107           of current as follows:
108             vertigo_status.pds_table[current->pid]->perf = \
109             vertigo_status.pds_table[current->pid]->work_est \
```

Figure 6.4 Vertigo pseudo code (Lines 110~161)

```
110             / vertigo_status.pds_table[current->pid]->deadline
111           Initialize vertigo_status.pds_table[current->pid]->work_fse to 0
112           Initialize vertigo_status.pds_table[current->pid]->idle_base \
113           to ds_counter.idle_total
114           Initialize vertigo_status.job_work_fse_base to ds_counter.busy_fse
115           CALL cpu_utilization_to_cpu_op WITH perf RETURNING cpu_op_new
116         ENDIF
117       ELSE
118         RETURN the current CPU_OP
119       ENDIF
120
121   ENDIF
122   RETURN cpu_op_new
123
124   algorithm_for_interactive_applications:
125
126   # If no ongoing interactive episode exists
127   IF vertigo_status.ids.flag_episode_on is 0
128
129     IF ds_parameter.entry_type is DS_ENTRY_RET_FROM_SYSTEM_CALL
130       # If current received a user input event from X
131       IF ds_status.just_received_user_event_x_opcode is true
132         Set vertigo_status.ids.flag_episode_on to 1
133         Set vertigo_status.ids.flag_marked[current->pid] to 1
134         Set vertigo_status.ids.flag_marked[X->pid] to 1
135         Set vertigo_status.ids.flag_preempted[current->pid] to 1
136         Set vertigo_status.ids.flag_preempted[X->pid] to 1
137         Increment vertigo_status.ids.preempted_task_no by 2
138         Initialize vertigo_status.ids.work_fse_base to ds_counter.busy_fse
139         Initialize vertigo_status.ids.elapsed_base to ds_counter.elapsed
140       ENDIF
141     ENDIF
142     Reset vertigo_status.flag_force_interactive_perf to 0
143     RETURN the current CPU_OP
144
145   # If an ongoing interactive episode exists
146   ELSE
147
148     Increment vertigo_status.ids.work_fse by \
149       ds_counter.busy_fse - vertigo_status.ids.work_fse_base
150     Increment vertigo_status.ids.elapsed by \
151       ds_counter.elapsed - vertigo_status.ids.elapsed_base
152     Initialize vertigo_status.ids.work_fse_base to ds_counter.busy_fse
153     Initialize vertigo_status.ids.elapsed_base to ds_counter.elapsed
154
155     # If current is not marked
156     IF ds_parameter.entry_type is DS_ENTRY_RET_FROM_SYSTEM_CALL
157       IF vertigo_status.ids.flag_marked[current->pid] is 0
158         # If current received an IPC
159         IF ds_status.just_read_ipc_writer_pid
160           # If the task that has written to current is a marked task
161           IF vertigo_status.ids.flag_marked[ \
```

Figure 6.4 Vertigo pseudo code (Lines 162~213)

```
162            ds_status.just_read_ipc_writer_pid] is 1
163               Set vertigo_status.ids.flag_marked[current->pid] to 1
164               Set vertigo_status.ids.flag_preempted[current->pid] to 1
165               Increment vertigo_status.ids.preempted_task_no by 1
166            ENDIF
167          ENDIF
168        ENDIF
169
170      # If ds_parameter.entry_type is DS_ENTRY_SWITCH_TO
171      ELSE
172        # If prev_p is marked
173        IF vertigo_status.ids.flag_marked[prev_p->pid] is 1
174          IF prev_p has not been preempted but gone to sleep
175            Reset vertigo_status.ids.flag_preempted[prev_p->pid] to 0
176            Decrement vertigo_status.ids.preempted_task_no by 1
177            # If the ongoing episode is completed
178            IF vertigo_status.ids.preempted_task_no is 0
179               Update work_est, the moving average of work_fse, \
180               of interactive episodes as follows:
181                  vertigo_status.ids.work_est = \
182                  (vertigo_status.ids.work_est * weight + \
183                   vertigo_status.ids.work_fse) \
184                  / (1 + weight)
185               Update perf, the required normalized CPU speed, \
186               for the next interactive episode as follows:
187                  vertigo_status.ids.perf = \
188                  vertigo_status.ids.work_est \
189                  / 50 msec of PERCEPTION_THRESHOLD
190               Reset vertigo_status.ids.flag_episode_on to 0
191               FOR all pids
192                  Reset vertigo_status.ids.flag_marked[pid] to 0
193                  Reset vertigo_status.ids.flag_preempted[pid] to 0
194               ENDFOR
195               Initialize vertigo_status.ids.work_fse to 0
196               Initialize vertigo_status.ids.elapsed to 0
197            ENDIF
198          ENDIF
199        ENDIF
200
201        # If next_p is marked
202        IF vertigo_status.ids.flag_marked[next_p->pid]
203          # If the ongoing interactive episode was not completed by prev_p
204          IF vertigo_status.ids.flag_episode_on is 1
205            IF vertigo_status.ids.flag_preempted[next_p->pid] is 0
206               Set vertigo_status.ids.flag_preempted[next_p->pid] to 1
207               Increment vertigo_status.ids.preempted_task_no by 1
208            ENDIF
209          ENDIF
210        ENDIF
211
212        # If The ongoing interactive episode was not completed by prev_p
213        IF vertigo_status.ids.flag_episode_on is 1
```

Figure 6.4 Vertigo pseudo code (Lines 214~234)

```
214
215          IF vertigo_status.ids.elapsed < 5 msec of SKIP_THRESHOLD
216            Reset vertigo_status.flag_force_interactive_perf to 0
217            RETURN the current CPU_OP
218          ELSE IF vertigo_status.ids.elapsed < 50 msec of PANIC_THRESHOLD
219            CALL cpu_utilization_to_cpu_op WITH vertigo_status.ids.perf
220            RETURNING cpu_op_new
221            Set vertigo_status.flag_force_interactive_perf to 1
222            RETURN cpu_op_new
223          ELSE
224            Set vertigo_status.flag_force_interactive_perf to 1
225            RETURN CPU_OP0
226          ENDIF
227        # If the ongoing interactive episode was completed by prev_p
228        ELSE
229          Reset vertigo_status.flag_force_interactive_perf to 0
230          RETURN the current CPU_OP
231        ENDIF
232
233      ENDIF
234    ENDIF
```

Figure 6.4 Vertigo pseudo code

How closely these existing autonomous DVS schemes were implemented in this research to the original implementations by the respective authors was assessed. Because of the difference in the platform used in this work and in the authors' work, just the tendency was compared. Flautner et. al., the authors of Vertigo, compared the fraction of time that Vertigo and LongRun, an interval-based autonomous DVS scheme for Transmeta Crusoe CPU that is similar to OCG, spent at each level of CPU speed [75] (Due to the lack of any other literature that provides the comparison result among Vertigo, AVG3, and OCG, only this comparison result was used for this assessment). The result shows that Vertigo spent around 2 to 9 times longer time at the lowest CPU speed than LongRun for continuous movie application usage scenarios and around 4 times longer time for Web browser usage scenarios. As Figure 6.18 and 6.29 will show in the following sections, the Vertigo implemented in this work spent around 4 times longer time at the minimum CPU speed for continous movie usage scenarios and also 4 times longer time for Web browser usage scenarios than OCG. Although this is a rough assessment, the coincidence between the tendency in the improvement by Vertigo over existing interval-based DVS schemes implies that, in this work, the existing autonomous DVS schemes were implemented closely to their original implementations.

6.2.3 Benchmarks and usage scenarios

The benchmarks used in the experiments were selected from the existing common Linux applications so as to conduct realistic experiments. The benchmarks include continuous media SRT applications, interactive SRT applications, and NRT applications. Table 6.3 describes the benchmarks and Table 6.4 and 6.5 describes the chosen multimedia contents for the continuous media applications.

Table 6.3 Selected benchmarks for the experiment

Time constraint type	Application name	Version	Description	Features
SRT (Continuous media)	Mplayer	090rc5-3.2.2	Continuous media player	Single-threaded
	Avifile aviplay	0.7.43	Continuous media player	Multi-threaded Unable to decode MPEG movies
SRT (Interactive)	Mozilla	1.2.1	Web browser	Multi-threaded
	Gnect	2.2.0	Game	Single-threaded
	OpenOffice word	1.0.2	Document editor	Single-threaded
NRT	Gcc	4.2.2	GNU C and C++ compiler	Causes a series of different tasks
	AVG avgscan	7.1flr-r30	Virus scanner	Causes a large number of hard idles

In the choice of benchmarks, an additional point taken into consideration was whether they are multi-thread applications or single-threaded applications. Typically, a multi-threaded application is heavier than a single-threaded application. However, as it divides its workload into multiple threads, a multi-threaded application can keep higher priorities under the native GPOS task scheduling – the CPU-occupancy-based task scheduling. This aspect makes multi-threaded applications typically better in QoS than single-threaded applications at the cost of a larger amount of energy consumption. To see the impact of this aspect on the performance of GPScheDVS, the choice of the SRT benchmarks was balanced between multi-threaded applications and single-threaded applications.

Table 6.4 Movie files used in the experiment

Name		MPEG1	MPEG2	AVI
Length (seconds)		67.600	99.900	160.000
Video specification	Encoding format	MPEG	MPEG	Microsoft AVI
	Resolution (pixels)	480×368	480×360	640×480
	Frame per second (fps)	29.970	29.970	23.976
	Frame period (msec)	33.367	33.367	41.708
Audio specification	Encoding format	MP3	MP3	MP3
	Compressed bit rate (Kbps)	128	64	128
	Sampling method (bits/Hz/channels)	16/44100/2	16/48000/2	16/44100/2
	Uncompressed bit rate (Kbps) (See Note 1)	1411.2	1536	1411.2

Note 1—Uncompressed bit rate = sampling bit width × sampling frequency (Hz) × sampling channel number

Table 6.5 Music files used in the experiment

Name	MP31	MP32	WAV
Length (seconds)	235.400	262.000	257.600
Encoding format	MP3	MP3	Microsoft WAV
Compressed bit rate (Kbps)	128	320	172
Sampling method (bits/Hz/channels)	16/44100/2	16/44100/2	16/44100/1
Uncompressed bit rate (Kbps)	1411.2	1411.2	705.6

Gcc, the GNU C and C++ compiler, was selected as the representative NRT benchmark. An important nature of gcc is that it keeps creating and destroying a series of different tasks in compiling a program; some of the tasks are CPU-bound and some of them are IO-bound. This nature of gcc makes it possible to get insight into the results with various types of NRT tasks.

AVG avgscan, the other NRT application, was selected as the benchmark to verify the performance of GPScheDVS-SE on HINRT tasks. Unlike gcc, avgscan consists of a single task that repeats making a system call to read hard disk, sleeping until the system call returns, and waking up to analyze the read data. The frequent sleep on system calls in this procedure causes a lot of hard idle times especially when there is a little amount of workload.

Using the selected benchmarks, various real-life usage scenarios were composed for the experiment. The typical real-life usage scenarios where a SRT application is run alone or

with various amounts of background NRT workloads are described in Table 6.6 and 6.7. Table 6.8 and 6.9 describe the special usage scenarios in which a SRT application is run with avgscan, the HINRT application, in the background. Finally, Table 6.10 and 6.11 list the special usage scenarios to see the performance of GPScheDVS upon the workload mostly made up of SRT tasks. Under the usage scenarios described in Table 6.10, an interactive application was executed while a continuous media application is playing a movie or music data. The usage scenarios described in Table 6.11 make a more extreme situation; under the usage scenarios, a multiple number of continuous media applications were executed simultaneously. The number of applications was increased from 1 to 6. With 6 simultaneous continuous media application, the CPU utilization is filled up to its capacity.

Table 6.6 Usage scenarios with a continuous media application and gcc compilation

<table>
<tr><th>Usage scenario name (See Note 1)</th><th>Duration (seconds)</th><th>Description</th></tr>
<tr><td>mpeg1_m_gcc_u[n]</td><td rowspan="4">190</td><td rowspan="8">Playing a movie or music file with a continuous media player while running a gcc compilation in the background.

Each usage scenario was repeated with different gcc compilation workloads from 0 to the entire remaining portion of the total CPU utilization.</td></tr>
<tr><td>mpeg2_m_gcc_u[n]</td></tr>
<tr><td>avi_m_gcc_u[n]</td></tr>
<tr><td>avi_a_gcc_u[n]</td></tr>
<tr><td>mp31_m_gcc_u[n]</td><td rowspan="4">310</td></tr>
<tr><td>mp31_a_gcc_u[n]</td></tr>
<tr><td>mp32_m_gcc_u[n]</td></tr>
<tr><td>mp32_a_gcc_u[n]</td></tr>
<tr><td>wav_m_gcc_u[n]</td><td rowspan="2">310</td><td rowspan="2"></td></tr>
<tr><td>wav_a_gcc_u[n]</td></tr>
<tr><td colspan="3">Note 1—In the scenario names, mpeg1, mpeg2, avi, mp31, mp32, and wav stand for the MPEG movie 1, MPEG movie 2, AVI movie, MP3 music 1, MP3 music 2, and WAV music, respectively. The term _m_ and _a_ stand for the mplayer and aviplay, respectively. And, the term _u[n] means the ***total*** CPU utilization under each usage scenario. For example, _u100 means that the total CPU utilization is 100 %.</td></tr>
</table>

Table 6.7 Usage scenarios with an interactive application and gcc compilation

<table>
<tr><th>Usage scenario name</th><th>Duration (seconds)</th><th>Description</th></tr>
<tr><td>mozilla_gcc_u[n]</td><td rowspan="3">190</td><td rowspan="3">Web browsing with mozilla, Playing gnect game, or documenting with OpenOffice word while running gcc compilation in the background.
Each usage scenario was repeated with different gcc compilation workloads from 0 to the entire remaining portion of the total CPU utilization.</td></tr>
<tr><td>gnect_gcc_u[n]</td></tr>
<tr><td>ooffice_gcc_u[n]</td></tr>
</table>

Table 6.8 Usage scenarios with a continuous media application and avgscan

Usage scenario name	Duration (seconds)	Description
mpeg1_m_avgscan	180	Playing a movie or music file with a continuous media player while running an AVG avgscan virus scanner in the background.
mpeg2_m_avgscan		
avi_m_avgscan		
avi_a_avgscan		
mp31_m_avgscan	300	
mp31_a_avgscan		
mp32_m_avgscan		
mp32_a_avgscan		
wav_m_avgscan		
wav_a_avgscan		

Table 6.9 Usage scenarios with an interactive application and avgscan

Usage scenario name	Duration (seconds)	Description
mozilla_avgscan	180	Web browsing with mozilla, Playing gnect game, or documenting with OpenOffice word while scanning hard disk for virus with AVG avgscan virus scanner in the back ground.
gnect_avgscan		
ooffice_avgscan		

Table 6.10 Usage scenarios with an interactive application and a continuous media application

Usage scenario name	Duration (seconds)	Description
mozilla_avi_m	180	Web browsing with mozilla, Playing gnect game, or documenting with OpenOffice word while playing AVI movie with mplayer.
gnect_avi_m		
ooffice_avi_m		
mozilla_mp31_m	240	Web browsing with mozilla, Playing gnect game, or documenting with OpenOffice word while playing MP3 music 1 with mplayer.
gnect_mp31_m		
ooffice_mp31_m		

Table 6.11 Usage scenarios with a multiple number of continuous media applications

Usage scenario name	Duration (seconds)	Description
1_mpeg1_m	67.6	Playing 1 instance of MPEG movie 1 with mplayer.
2_mpeg1_m	72.6	Playing 2, 4, or 6 instances of MPEG movie 1 with mplayer simultaneously.
4_mpeg1_m	82.6	
6_mpeg1_m	92.6	

Figure 6.5 shows the screen snap shots captured while executing the benchmarks.

(a) Playing MPEG movie 1 with mplayer

(b) Playing MPEG movie 2 with mplayer

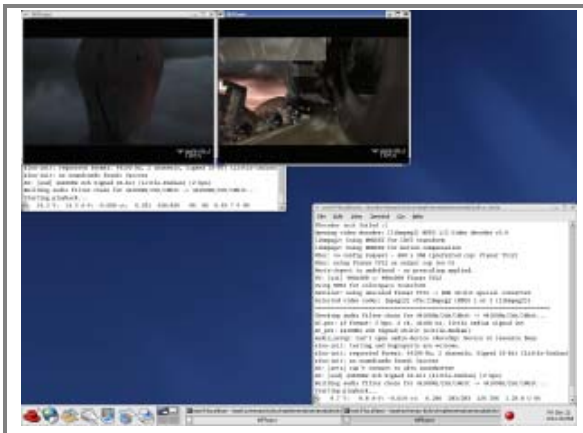
(c) Playing AVI movie with mplayer

(d) Playing AVI movie with aviplay

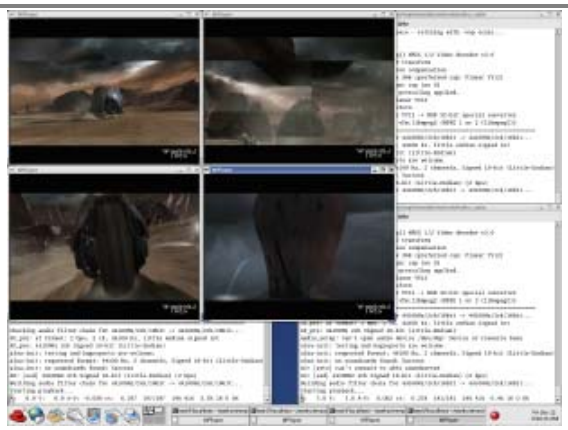
(e) Playing MP3 music 1 with mplayer

(f) Playing MP3 music 2 with aviplay

Figure 6.5 Screen snap shots of the representative usage scenarios

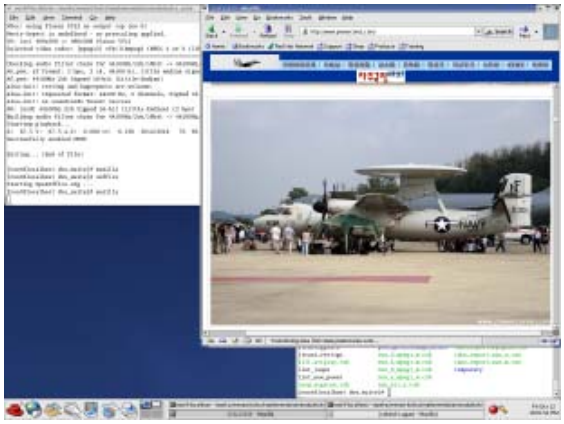


(g) 2 instances of MPEG movie 1 with mplayer



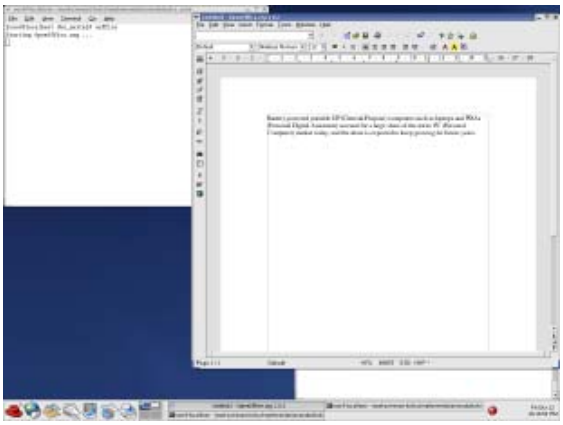
(h) 4 instances of MPEG movie 1 with mplayer

(i) 6 instances of MPEG movie 1 with mplayer



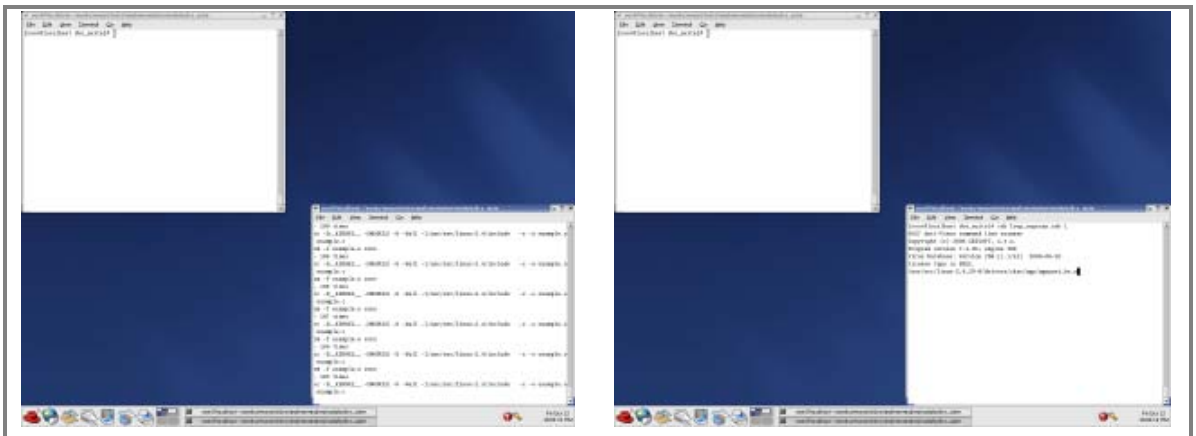
(j) Web browsing with Mozilla

(k) Playing gnect game



(l) Documenting with OpenOffice word

Figure 6.5 Screen snap shots of the representative usage scenarios (Cont'd)



(m) Gcc compilation (n) Scanning hard disk for virus with AVG avgscan

Figure 6.5 Screen snap shots of the representative usage scenarios (Cont'd)

6.2.4 Metrics

In the experiment, the performance of GPScheDVS and the existing autonomous DVS schemes were measured and compared in three major metrics and two auxiliary metrics. The three major metrics are as follows.

- The system energy consumption in Joules.
- The average CPU power consumption in Watts.
- The QoS of SRT applications such as continuous media applications and interactive applications in deadline meet ratio.

The system energy consumption is the relevant metric to the battery life-time. And, the average CPU power consumption is the pertinent metric to the CPU temperature [7] [8] [9] [10]. Finally, the QoS of SRT applications gives insight into the system performance experienced by users.

The two auxiliary metrics include followings.

- The CPU energy consumption in Joules.
- The overheads of a DVS scheme in time, in system energy consumption, and in CPU

operating point transition numbers.

These metrics are supplementary because the quantities measured in these metrics are already incorporated in the quantities measured in the major metrics. For example, CPU energy consumption and the overhead of a DVS scheme in system energy consumption are already included in a major metric, the system energy consumption. Likewise, a major metric, the QoS of time constrained applications is the measure that is already affected by the overhead of a DVS scheme in time and in the number of CPU operating point transitions. Therefore, if a DVS scheme yields a better result in the auxiliary metrics but a worse result in the major metrics than another DVS scheme, the desirable DVS scheme is the latter.

However, the performance of a DVS scheme in the auxiliary metrics will be still useful in the choice of the appropriate DVS scheme if the characteristics of the target system vary. For example, for a system in which the system energy consumption is mostly due to the CPU, the DVS scheme that achieves the largest amount of CPU energy savings will be the best choice. Also, if a system has such a CPU whose operating point transition is extremely costly in time, the DVS scheme that causes the least number of CPU operating point transition will be the best choice.

The following chapters define these metrics and their measurement method in the concrete. First, Section 6.2.4.1 describes the energy and power metrics. Then, Section 6.2.4.2 deals with the QoS metrics. Finally, the overhead metrics are treated in Section 6.2.4.3.

6.2.4.1 Energy and power consumptions

This section defines the three types of energy and power metrics – the system energy consumption, the CPU energy consumption, and the peak and average CPU power consumptions – and the measurement method for them in sections 6.2.4.1.1, 6.2.4.1.2, and 6.2.4.1.3, respectively.

6.2.4.1.1 System energy consumption

System energy consumption is the energy consumed by the entire experimental platform. The system energy consumption for an experiment equals the average system power consumption for the experiment multiplied by the duration of the experiment in seconds. Therefore, assuming that the average system power consumption was measured *n* times for an experiment, once for every interval, the system energy consumption during the experiment is calculated by Equation 6.2.

$$E_{sys} = \sum_{i=1}^{n} (\overline{P}_{sys,i} \times T_i), \qquad \text{(Equation 6.2)}$$

where E_{sys} is the system energy consumption during the experiment in Joules, $\overline{P}_{sys,i}$ is the average system power consumption for the i^{th} interval in Watts, and T_i is the length of the i^{th} interval in seconds.

The average system power consumption for each interval of time can be measured by using an electrical measurement instrument such as an oscilloscope or a multi-meter. An average system power consumption measurement setup with a HP 54602B oscilloscope [89] and a LEM PR30 current probe was shown in Figure 6.1 in Section 6.1. And, an example of the measurement setup with an Agilent 3458A digital multi-meter [92] can be found in [91].

In the measurement setup shown in Figure 6.1, the current probe detects the instantaneous current drawn by the experimental platform and outputs the measured value to the oscilloscope in voltage. The conversion ratio between the measured current and the voltage output of the current probe is 100 mV / 1 A. Then, the oscilloscope samples the input voltage from the current probe and calculates the maximum, minimum, and average values for every measurement interval. For example, Figure 6.6 shows the plot of the current measured by the oscilloscope for 5 seconds of a measurement interval while the experimental platform is playing the AVI movie described in Table 6.4 by using aviplay, the continuous media application defined in Table 6.3. The 151.68 mV of average voltage shown in Figure 6.6 actually means the 1.5168 A of average current for

the interval. Then, the average system power consumption for the interval can be calculated by multiplying 19.5 V, the experimental platform's power adaptor supply voltage, to the average current such that 1.5168 A × 19.5 V = 29.5776 W. Finally, the system energy consumption for the interval is calculated by multiplying the interval length to the average system power consumption such that 29.5776 W × 5 seconds = 147.888 J. The system energy consumption for the entire duration of an experiment can be obtained, first, by acquiring the average system power consumptions for the consecutive measurement intervals until the end of the experiment, then, summing up the individual system energy consumptions for each interval as described in Equation 6.2.

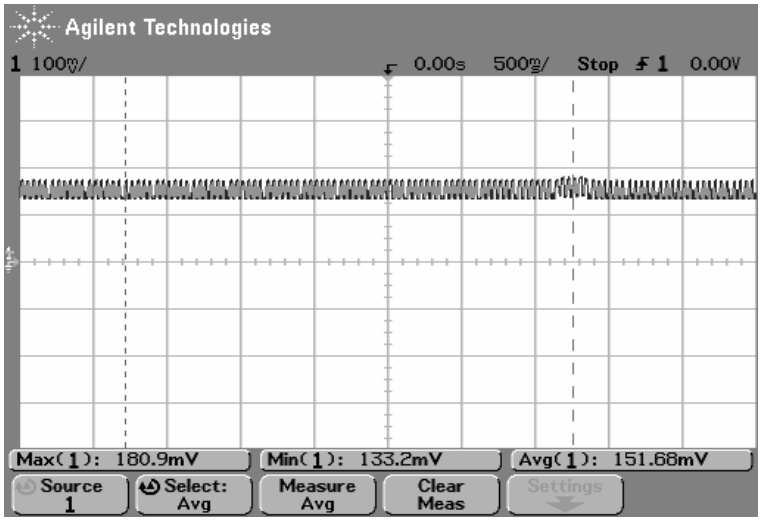


Figure 6.6 Current drawn by the experimental platform for 5 seconds of an interval while playing the AVI movie with the aviplay (The plot was captured from the HP 54602B oscilloscope)

The average system power consumptions for the consecutive measurement intervals and, ultimately, the system energy consumption for the entire duration of an experiment can be obtained in a similar way under the measurement setup described in [91]. The HP 54602B oscilloscope supports 20 nanoseconds to 50 seconds of measurement interval lengths and is able to store up to 4000 measured values, one for each interval. The Agilent 3458A digital multi-meter provides 10 nanoseconds to 100 minutes of measurement interval lengths and can store up to 65536 measured values, one for each interval.

However, accurately measuring the system energy consumption during an experiment in

this way is difficult, if not impossible, due to the practical limit in fitting the measurement interval into the exact duration of the experiment. Most experiments conducted in this research have a duration that does not exactly fit into a minute or a second resolution; actually, the time resolution that the DVS-suite uses is one microsecond. To measure the system energy consumption in a microsecond resolution with an electrical measurement instrument, a million values measured for each one microsecond measurement interval should be stored because the measurement interval length of the instrument cannot be changed while measuring the system energy consumption for an experiment. Therefore, if the duration of an experiment is longer than 5 minutes, for example, the total number of measurements required for the experiment becomes greater than 300 millions. This number of samples takes more than 300 Mbytes in the measurement setup with a HP 54602B oscilloscope [89] and 600 Mbytes in the setup with an Agilent 3458A digital multi-meter [92] because the digitizing resolutions of the instruments are 8 bits and 16 bits, respectively. This amount of computational requirement makes the experiments extremely laborious. In addition to this difficulty, it is impossible to acquire the overhead of a DVS scheme in system energy consumption in this way because the measurement intervals for the operations of the DVS scheme cannot be distinguished.

To make the acquisition of the system energy consumption easier and the acquisition of the overhead of a DVS scheme in system energy consumption possible, the system energy consumption was estimated in this research by using the energy model given by Equation 6.3.

$$E_{sys} = \sum_{op=0}^{n} (P_{sys\_busy,op} \times T_{busy,op} + P_{sys\_idle,op} \times T_{idle,op}) \quad \text{(Equation 6.3)}$$

where E_{sys} is the system energy consumption for an experiment in Joules, $P_{sys\_busy,op}$ is the average system power consumption measured while CPU is busy at CPU operating point *op*, $P_{sys\_idle,op}$ is the average system power consumption measured while CPU is idling at CPU operating point *op*, $T_{busy,op}$ is the total CPU busy time spent at CPU operating point *op*, and $T_{idle,op}$ is the total CPU idling time spent at CPU operating point *op* during the experiment.

$P_{sys\_busy,op}$ and $P_{sys\_idle,op}$ are the coefficients of the model which was measured beforehand using the average system power consumption measurement setup shown in Figure 6.1. Tweak-PM tool, which will be described in Section 6.2.5.1, was used to set the CPU speed for each system power consumption measurement. $T_{busy,op}$ and $T_{idle,op}$ are the independent variables that are reported in runtime by the DVS-suite for each experiment. Table 6.12 presents the measured system power consumption parameters for each CPU operating point.

Table 6.12 System power consumptions at each CPU operating point measured from the experiment platform

CPU operating point	Measured system power (W)	
	Busy state	Idle state
CPU_OP0	34.223	25.671
CPU_OP1	31.093	24.241
CPU_OP2	28.605	22.935
CPU_OP3	26.814	21.643
CPU_OP4	25.019	20.871
CPU_OP5	23.295	20.296

$P_{sys\_busy,op}$ and $P_{sys\_idle,op}$ were measured at each CPU operating point while keeping the CPU fully busy or fully idling by repeating an arithmetic operation that does not cause any memory or hard disk access. As pointed out in Section 6.2.1, the version of Linux used in this experiment does not perform any type of DPM unlike Microsoft Windows that perform the Advanced Configuration and Power Interface (ACPI), a dynamic power management (DPM) on the non-CPU components. Therefore, all non-CPU components such as monitor, wireless network card, mouse, hard drive, and memory system were kept turned-on consuming some amount of energy during the parameter measurement even though they are not performing any dynamic activity. This aspect makes the difference between the energy consumptions of non-CPU components when they are idling (i.e., the situation where the parameters were measured) and when they are busy (i.e., the situation for which the system energy consumption need to be estimated). This is the reason for why the estimated system energy consumption closely follows the actual system energy consumption as can be seen in Figure 6.7.

The verification was performed by comparing the system energy consumptions estimated

by the energy model and measured with the measurement setup shown in Figure 6.1 for 168 usage scenarios. Table 6.13 lists the usage scenarios selected for the verification. The verification of the model was performed both graphically and numerically. As a graphical verification, if the residuals between the estimated and measured system energy consumptions have any dependency on the model's independent variables, i.e., the time spent at each CPU operating point, were examined. Figure 6.8 shows the Scattergrams of the residuals along with the independent variable. It can be confirmed that the residuals are independent of the independent variables. This means that the model parameters are sufficiently accurate such that the only error between the measured and estimated values is random errors.

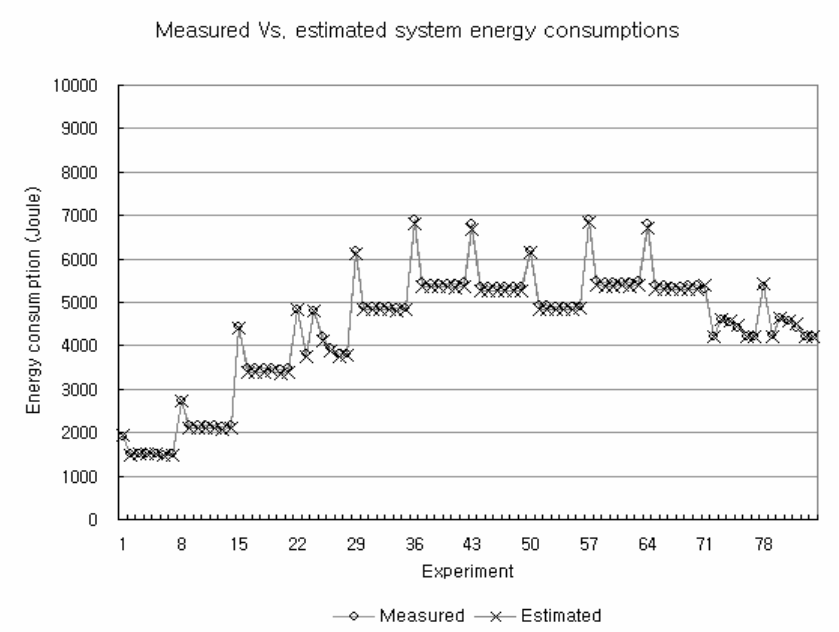


Figure 6.7 Accuracy of the system energy consumption estimated from energy model based on the measured system power consumption parameters

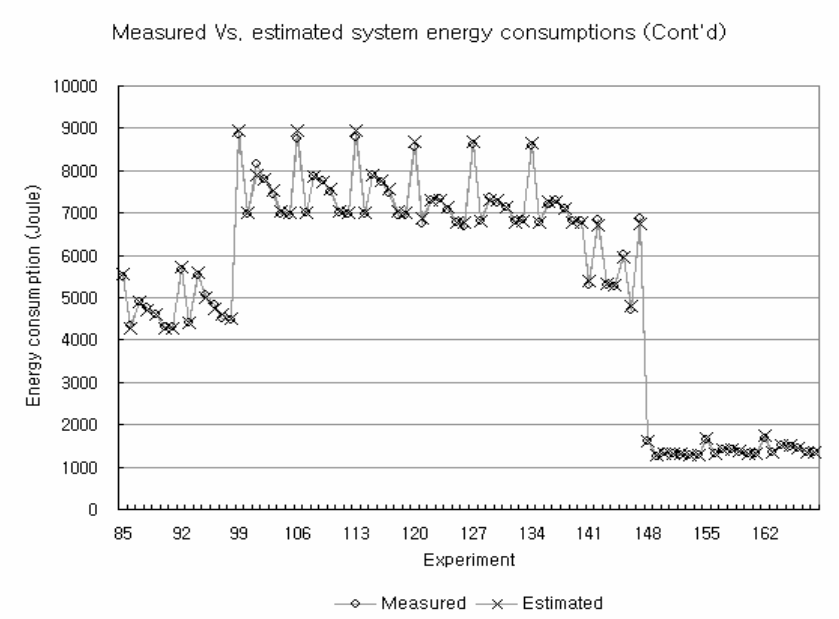


Figure 6.7 Accuracy of the system energy consumption estimated from energy model based on the measured system power consumption parameters (Cont'd)

Table 6.13 Selected usage scenarios to verify the system energy consumption model

Usage scenario	Description
mpeg1_m_u25	Playing a continuous media application alone. Unlike the usage scenarios described in Table 5.12 whose experimental duration is fixed,
mpeg2_m_u17	
avi_m_u20	
avi_a_u46	
mp31_m_u3	
mp32_m_u3	
wav_m_u2	
mp31_a_u4	
mp32_a_u3	
wav_a_u2	
mpeg1_m_gcc_u32	Playing a continuous media application while performing the gcc compilation in the background. These usage scenarios are that described in Table 5.12.
mpeg2_m_gcc_u34	
avi_m_gcc_u41	
avi_a_gcc_u52	
mp31_m_u38	
mp32_m_u38	
wav_m_u37	
mp31_a_u27	
mp32_a_u27	
wav_a_u27	
avgscan_u61	Performing avgscan anti-virus scanning alone.
gcc_u14	Performing gcc compilation alone.
gcc_u25	
gcc_u37	

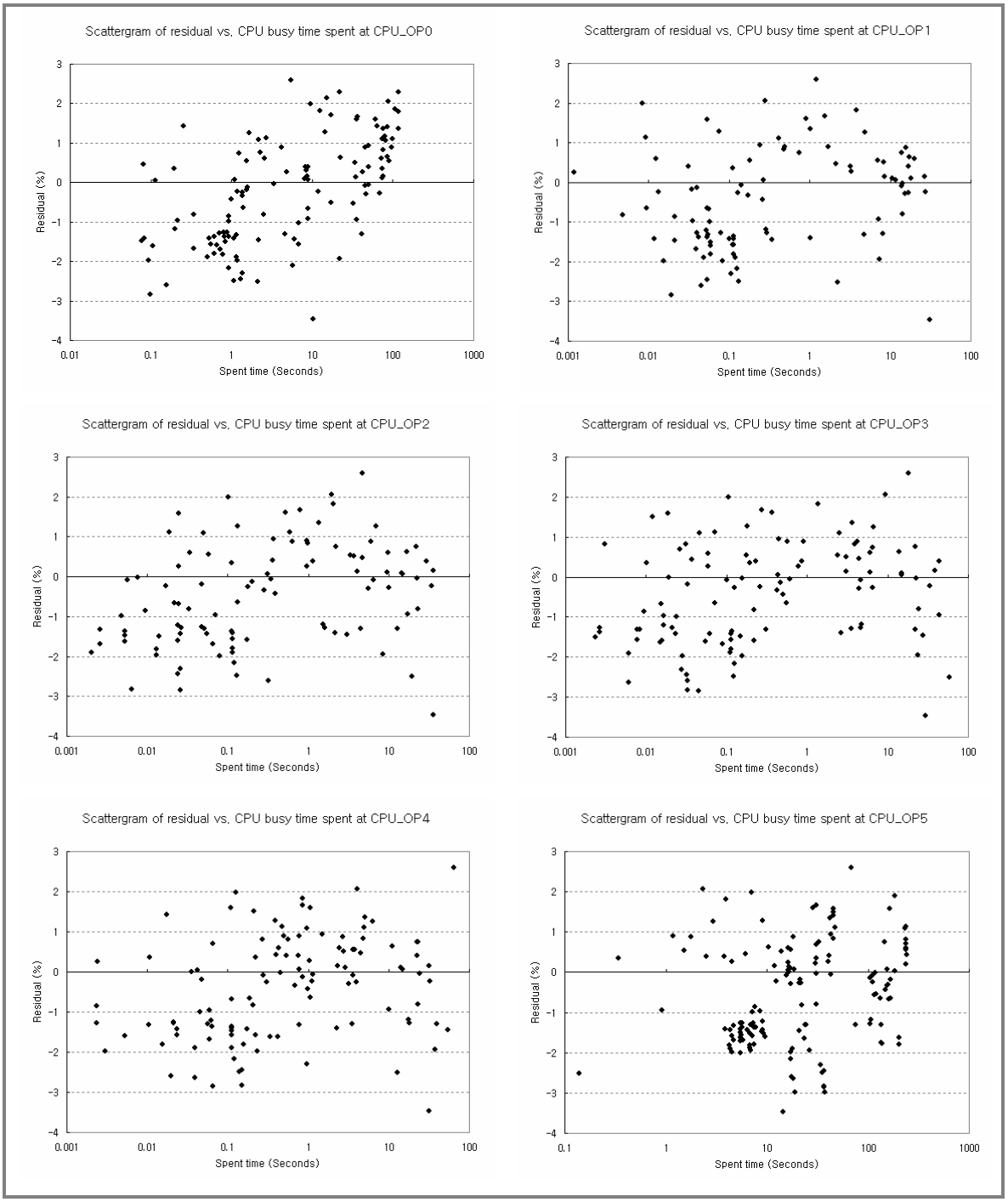


Figure 6.8 Scattergrams of residual vs. CPU times at each CPU operating point

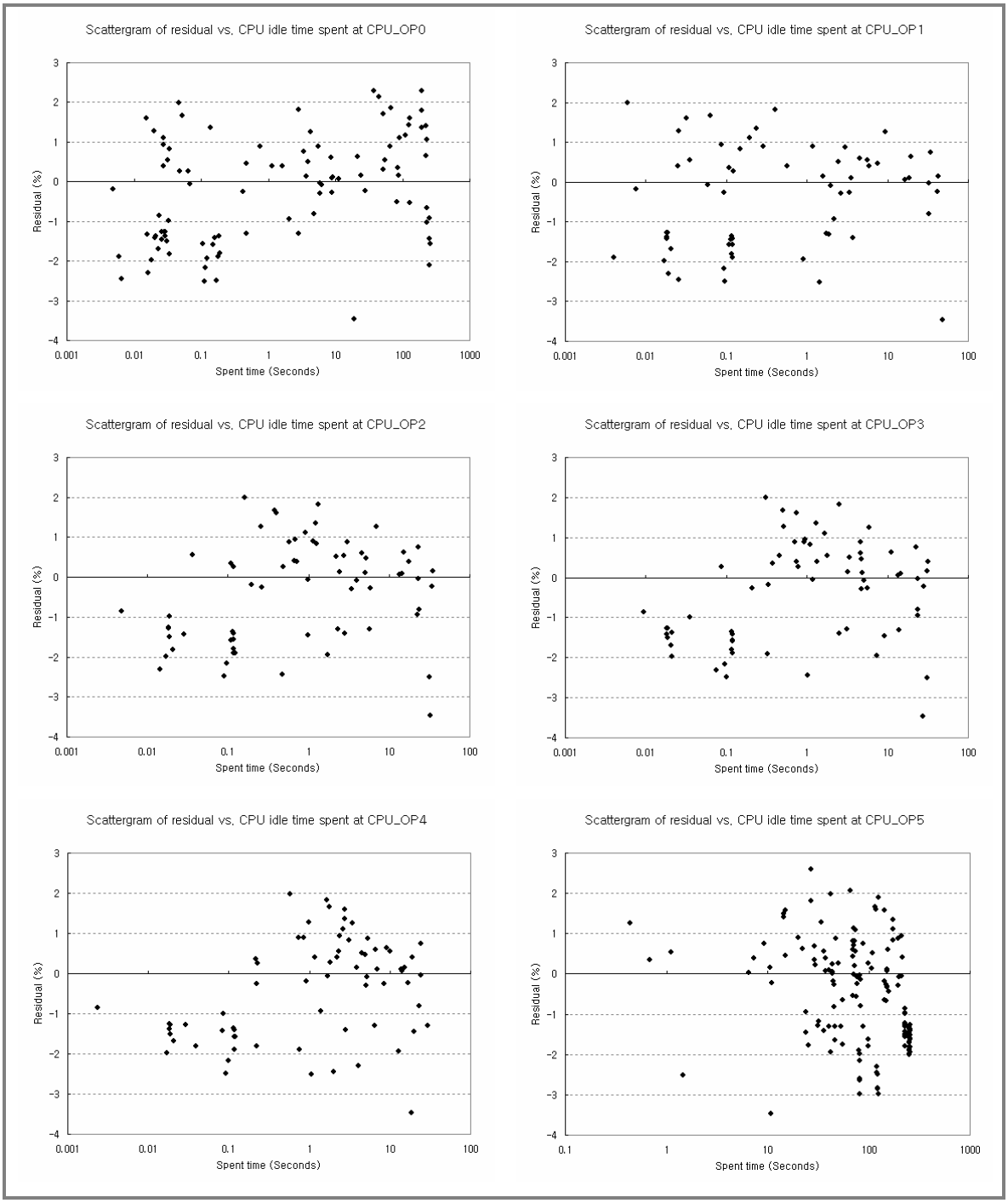


Figure 6.8 Scattergrams of residual vs. CPU times at each CPU operating point (Cont'd)

Then, as the numerical verification, the coefficient of determination that designates how the calculated system energy consumption closely follows the measured system energy consumption was calculated by Equation 6.4 for the 168 pairs of the system energy consumptions. The calculated coefficient of determination value was 0.999 and means

that the calculated system energy consumption follows the measured system energy consumption closely.

$$R^2 = 1 - \frac{SSE}{SST} = \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}, \qquad \text{(Equation 6.4)}$$

where R^2 is the coefficient of determination, *SSE* is the sum of squared error, *SST* is the sum of squared total, Y_i is the measured system energy consumption for the experiment *i*, $\hat{Y}_i$ is the calculated system energy consumption for the experiment *i*, and $\overline{Y}$ is the mean of the measured system energy consumption.

The confidence interval of the estimated system energy consumptions was calculated by Equation 6.5. For the calculation, first the estimated system energy consumptions were normalized to the measured system energy consumptions as Table 6.14 shows. Then, the normalized estimates were used to calculate the sample mean and the standard error required to calculate the confidence interval.

$$M \pm z \times \sigma_M, \qquad \text{(Equation 6.5)}$$

where M is the sample mean, z is the z-score, and σ_M is the standard error of the sample, i.e., the estimated standard error. When calculating the confidence interval with estimated standard error, t-table should be used instead of z-score. However, because the sample number exceeds 100 well following the normal distribution, z-score can be used in this case. Here, the statistic is the estimated system energy consumption normalized to the corresponding measured system energy consumptions. The confidence interval is for the mean of this statistics. And the population is all system energy consumption that will be estimated in the future. From the Table 6.14, M is calculated to be 99.630 and σ_M is 1.337. Thus, using z of 1.645 with 90% confidence level, the confidence intervals are 99.630 ± 2.199 with 90% confidence level, 2.621 with 95% confidence level, and 3.444 with 99% confidence level. These confidence intervals span 97.431~101.829, 97.009~102.251, and 96.186~103.074, respectively. These results mean that, for example, 95% of the future

estimations of energy consumption by the energy model are expected to be within 2.621% of the real energy consumption.

Table 6.14 Estimated system energy consumptions (%) normalized to the measured system energy consumptions

Compared usage scenarios	No DVS	Min.	OCG	AVG3	Vertigo	GPScheDVS	
						SE mode	CP mode
mpeg1_m_u25	101.717	98.350	99.742	100.554	99.812	99.180	98.697
mpeg2_m_u17	99.484	97.023	97.835	98.023	98.101	97.398	97.357
avi_m_u20	99.472	97.018	97.510	97.695	97.555	97.153	97.168
avi_a_u46	100.155	98.232	100.351	97.491	98.546	98.723	98.815
mp31_m_u3	98.973	98.489	98.435	98.497	98.729	98.575	98.697
mp32_m_u3	98.442	98.193	98.586	98.728	98.628	98.383	98.062
wav_m_u2	97.890	98.295	98.104	98.182	98.311	98.023	97.992
mp31_a_u4	99.348	98.576	98.196	99.010	99.141	99.035	98.789
mp32_a_u3	99.087	98.484	98.637	98.734	98.628	98.523	98.406
wav_a_u2	98.571	98.130	98.419	98.577	98.679	98.322	98.425
mpeg1_m_gcc_u32	101.590	99.441	100.099	100.510	101.604	99.869	99.921
mpeg2_m_gcc_u34	101.438	99.449	100.064	100.139	101.666	99.765	99.978
avi_m_gcc_u41	101.162	98.251	100.629	98.702	100.267	99.357	98.690
avi_a_gcc_u52	101.106	100.038	100.893	99.060	98.059	102.596	100.747
mp31_m_u38	101.364	100.708	96.537	100.111	101.106	101.123	100.814
mp32_m_u38	102.291	100.210	100.155	99.735	100.829	100.599	100.430
wav_m_u37	101.784	100.571	99.768	100.600	101.354	101.094	100.813
mp31_a_u27	101.406	101.901	99.962	99.704	100.942	100.063	101.587
mp32_a_u27	100.662	99.818	99.195	100.882	99.938	99.669	99.346
wav_a_u27	101.054	99.685	100.755	99.917	100.403	99.565	99.327
avgscan_u61	101.851	98.210	100.546	100.396	98.692	102.062	98.374
gcc_u14	100.310	100.233	98.594	100.887	100.259	100.044	100.001
gcc_u25	102.146	100.219	100.465	100.396	101.988	100.352	100.697
gcc_u37	102.287	101.577	101.256	101.820	101.272	101.419	101.500

6.2.4.1.2. CPU energy consumption

For the same reason as the actual measurement of system energy consumption, the CPU energy consumption was also estimated using a CPU energy model which has the CPU power consumptions at each CPU operating point as the parameters and the times spent at the corresponding CPU operating points as the explanatory variables.

Due to the difficulty in measuring CPU power consumption alone from a normal laptop, the CPU power consumptions were approximated from the measured system power consumptions. Because the system power consumptions were measured while running

only the CPU and keeping all non-CPU components standby as the previous section pointed out, the power consumed by non-CPU components can be regarded as a constant in the approximation. Then, the CPU power consumptions can be approximated by using a curve fitting function of Matlab. For example, the system power consumption while only CPU is busy can be expressed by Equation 6.5.

$$P_{sys,busy} = a1 + a2fV^2 + a3V, \qquad \text{(Equation 6.5)}$$

where $a1$ is the constant power of non-CPU components, $a2fV^2$ is dynamic power of CPU and $a3V$ is static power of CPU.

Because we know frequency, voltage, and the system power measured at each CPU operating point, the coefficients $a1$, $a2$, and $a3$ can be obtained using the following Matlab command.

```
[a, resnorm] = lsqcurvefit(func, a0, fv, p)
```

The CPU idle power can be estimated in the same way because some portion of the CPU such as clock path is still running dissipating dynamic power consumption. Table 6.15 shows the approximated CPU power consumptions. Figure 6.9 plots the measured system power consumption and the approximated CPU power consumptions. The approximated CPU power consumptions were at most 84.68% of the corresponding thermal design power (TDP) values available in Pentium-M datasheet [30].

Table 6.15 CPU power at each CPU operating point approximated from the measured system power

CPU operating point	Busy state CPU power consumption			Idle state CPU power consumption		
	Approximated (W)	TDP (W) (See note 1)	Ratio of the approximated to TDP (%)	Approximated (W)	TDP (W) (See note 1)	Ratio of the approximated to TDP (%)
CPU_OP0	17.337	21	82.557	8.261	10.9	75.789
CPU_OP1	14.594	N/A	-	6.817	N/A	-
CPU_OP2	12.173	N/A	-	5.551	N/A	-
CPU_OP3	9.838	N/A	-	4.346	N/A	-
CPU_OP4	8.038	N/A	-	3.419	N/A	-
CPU_OP5	6.351	7.5	84.680	2.569	3.3	77.848

Note—The thermal design power (TDP) values are quoted from [88].

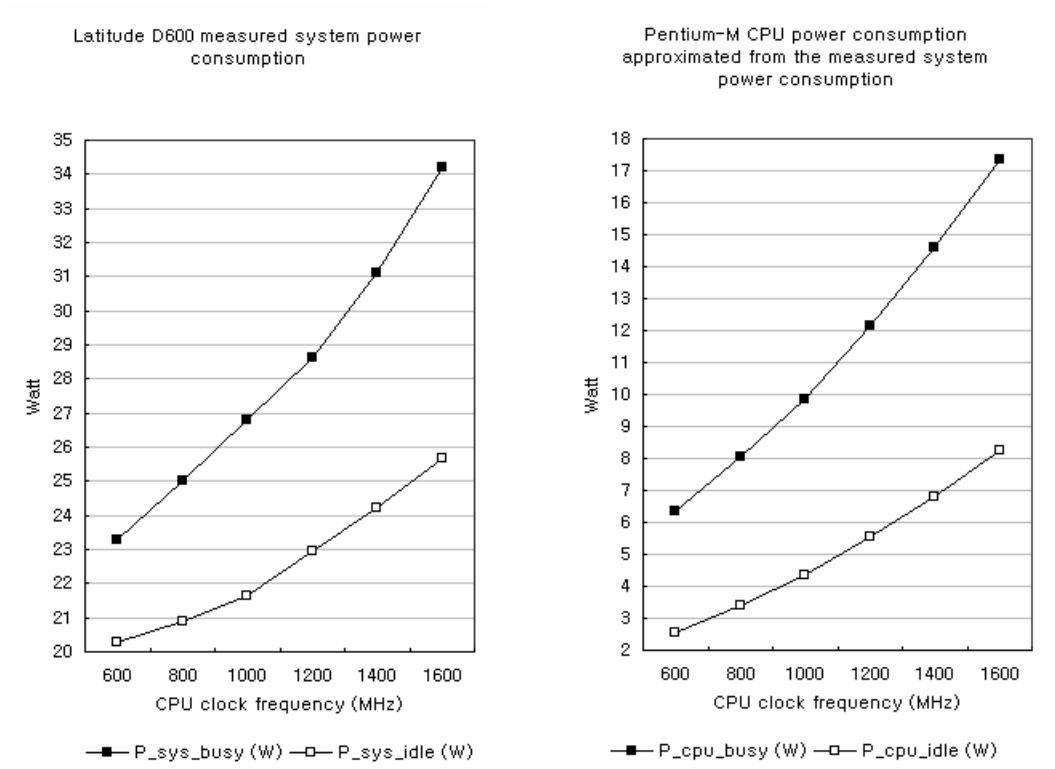


Figure 6.9 Pentium-M power consumptions approximated from measured system power consumptions

Finally, the CPU energy consumption for an experiment is defined by Equation 6.6.

$$E_{CPU} = \sum_{op=0}^{n} (P_{CPU\_busy,op} \times T_{busy,op} + P_{CPU\_idle,op} \times T_{idle,op}), \qquad \text{(Equation 6.6)}$$

where E_{CPU} is the CPU energy consumption for an experiment in Joules, $P_{CPU\_busy,op}$ is the average CPU power consumed while CPU is busy at CPU operating point *op*, $P_{CPU\_idle,op}$ is the average CPU power consumed while CPU is idling at CPU operating point *op*, $T_{busy,op}$ is the total CPU busy time spent at CPU operating point *op*, and $T_{idle,op}$ is the total CPU idling time spent at CPU operating point *op* during the experiment.

6.2.4.1.3. Average CPU power consumption

The average CPU power consumption, the pertinent metric to CPU heat dissipation, can be obtained simply by dividing the CPU energy consumption for an experiment by the duration of the experiment as given by Equation 6.7.

$$\overline{P}_{CPU} = \frac{E_{CPU}}{T}, \qquad \text{(Equation 6.7)}$$

where $\overline{P}_{CPU}$ is the average CPU power consumption for an experiment, E_{CPU} is the system energy consumption for the experiment, and T is the entire duration of the experiment in seconds.

6.2.4.2. QoS of SRT applications

The QoS of a SRT application is measured in three different ways according to the type of application and the type of input data: the audio deadline meet ratio, the video deadline meet ratio, and the interactive deadline meet ratio.

6.2.4.2.1. QoS of continuous video and audio applications

For a continuous media application which is playing music, the relevant metric is the audio deadline meet ratio defined by Equation 6.8.

$$DMR_{Audio} = \frac{m_{AudioOutput}}{n_{AudioOutput}}, \qquad \text{(Equation 6.8)}$$

where DMR_{Audio} is the audio deadline meet ratio in percent, $n_{AudioOutput}$ is the total number of the audio data output made by the continuous media application to either the audio driver or the sound server task, and $m_{AudioOutput}$ is the number of the audio data output that was made before the audio driver buffer becomes empty. If the continuous media application fails to supply a new audio data before the audio driver buffer runs out of the current audio data, users will feel the intermittence in the stream of audio.

For a continuous media application which is playing a movie, the pertinent metric is the video deadline meet ratio as well as the audio deadline meet ratio. The video deadline meet ratio is defined by Equation 6.9 in this research.

$$DMR_{Video} = \frac{m_{VideoOutput}}{n_{VideoOutput}}, \qquad \text{(Equation 6.9)}$$

where DMR_{Video} is the video deadline meet ratio in percent, $n_{VideoOutput}$ is the total number of the video frame output made by the continuous media application to the graphic user interface (GUI) server task (X in Linux), and $m_{VideoOutput}$ is the number of the video frame output that was made within the deadline described in the following.

In the strict sense, the deadline of every video frame output should be the reciprocal of the frame rate. For example, the frame rates and the corresponding deadlines of the AVI movie and the two MPEG movies described in Table 6.4 are 23.976 fps and 29.970 fps and 41.708 milliseconds and 33.367 milliseconds, respectively. However, as Figure 6.10 shows, the coarse resolution timer of Linux makes the video output period of video decoder applications irregular. Therefore, using the reciprocal of the frame rate as the deadline will mislead the QoS evaluation.

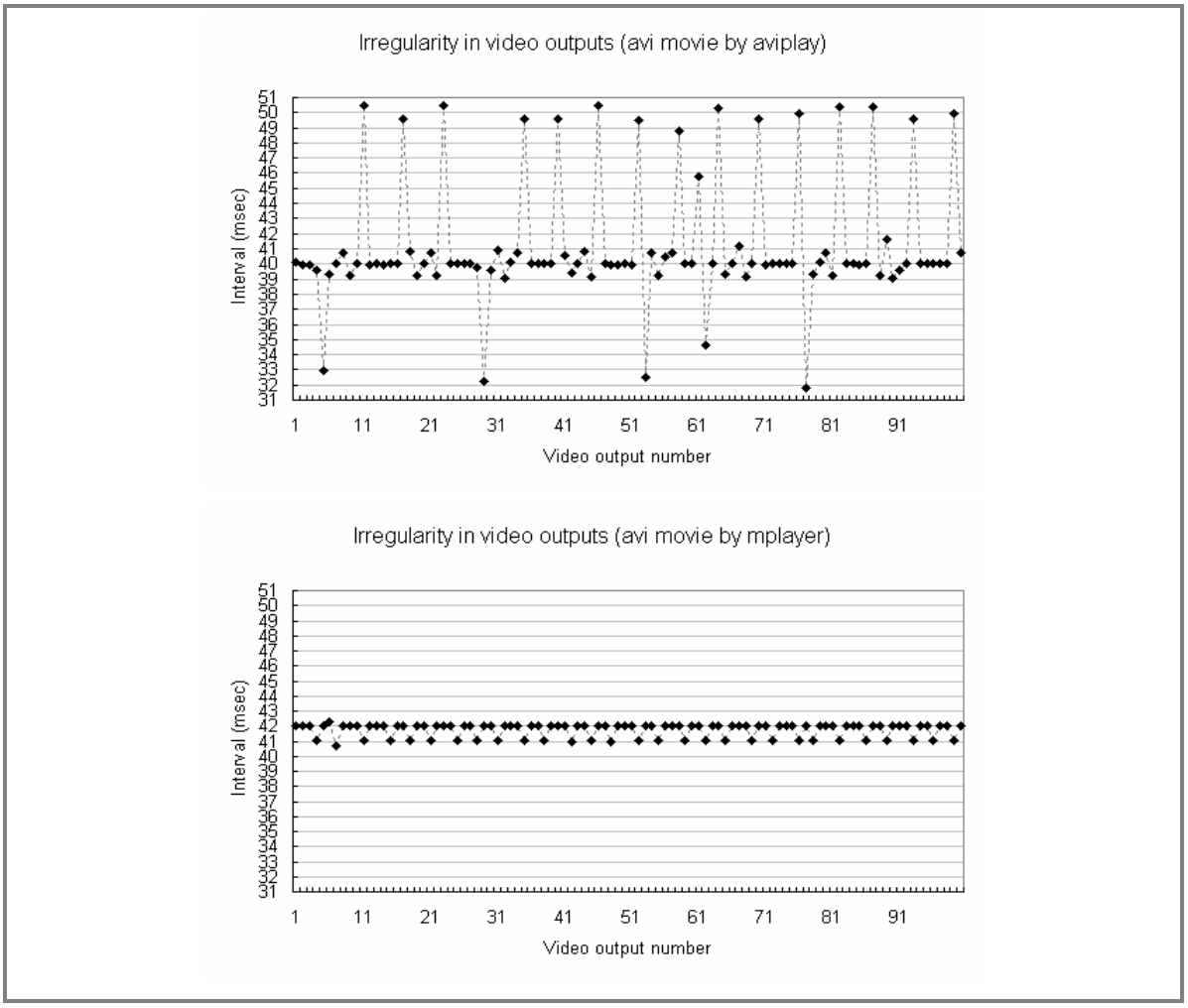


Figure 6.10 Irregularity in the video decoder output periods

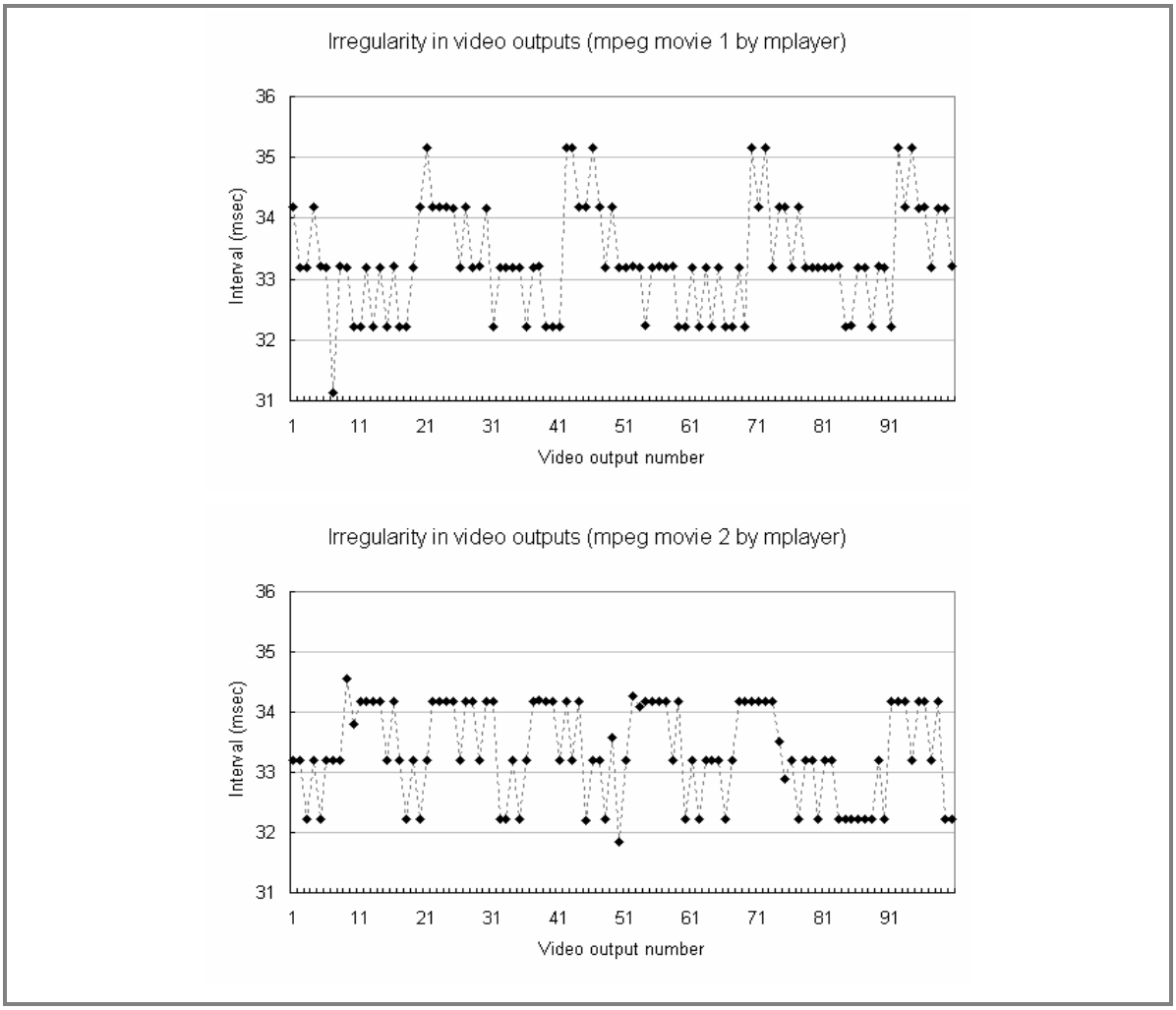


Figure 6.10 Irregularity in the video decoder output periods (Cont'd)

Considering this practical aspect in continuous video decoding process, the condition of video frame deadline meeting was defined as follows in this research.

- Condition 1—The average of the intervals between two subsequent video frame outputs should not exceed the reciprocal of the frame rate

- Condition 2—As long as the condition 1 is met, a video frame output made within the reciprocal of the frame rate plus 8.333 milliseconds since the last video frame output is considered met the deadline

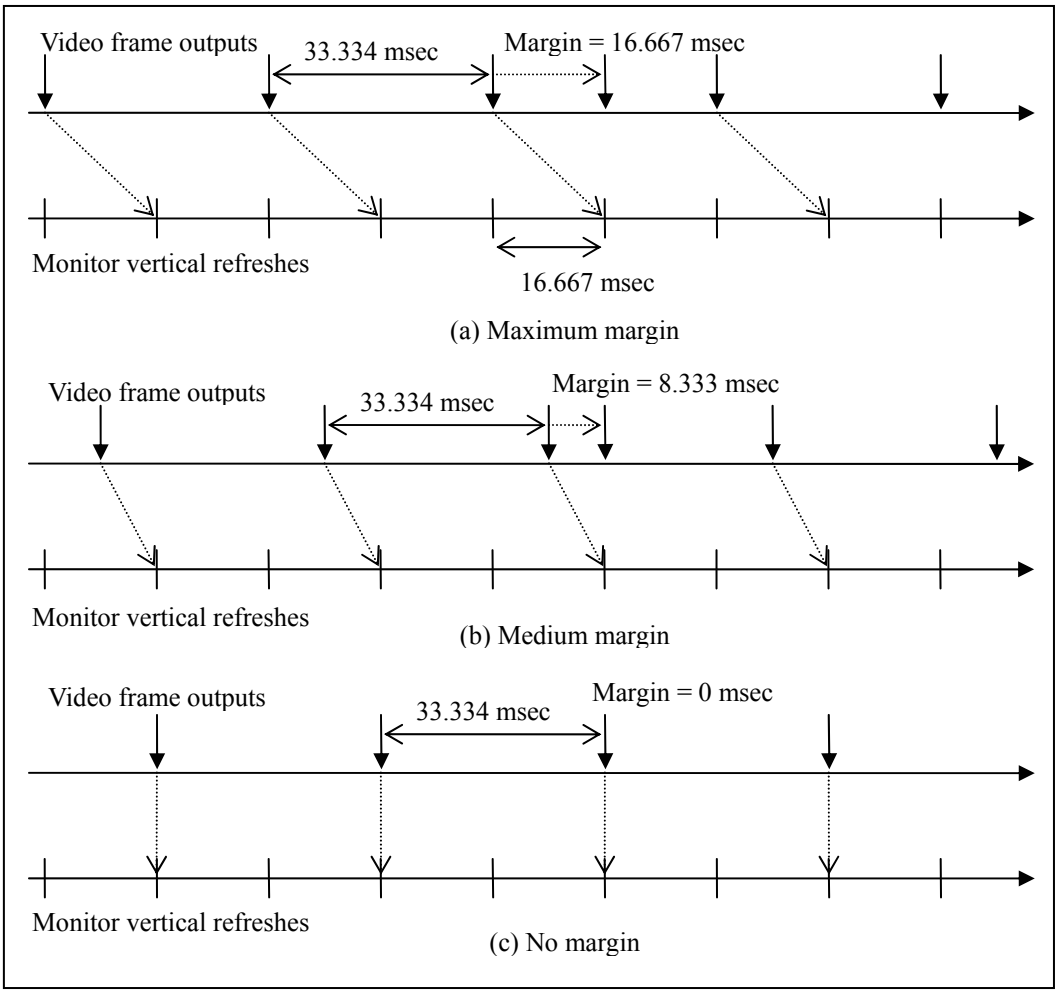


Figure 6.11 Margin in the deadline of video frame outputs

The 8.333 milliseconds of margin in deadline is the half of the monitor vertical refresh period (i.e., 16.667 milliseconds which is the reciprocal of 60 Hz) in the experimental platform. Figure 6.11 explains the reason for the selection of this margin. The graph (a), (b), and (c) in Figure 6.11 illustrate the three different cases from the view point of the synchronization between the video frame outputs and the monitor vertical refreshes. In case (a), the 16.667 milliseconds of delay in the next video frame output will not affect user experience because the video frame still can be displayed on the screen at the same monitor vertical refresh time point. Therefore, the margin in deadline is 16.667 milliseconds. Similarly, the margins for case (b) and case (c) are 8.333 and 0 milliseconds. Thus, assuming that the out-of-synch rate between the video frame outputs and the monitor vertical refreshes follows normal distribution, the half of the reciprocal of the monitor vertical refresh rate – 8.333 milliseconds – was selected as an appropriate margin

in the deadline of video frame outputs in this work.

6.2.4.2.2 QoS of interactive applications

The deadline meet ratio for interactive applications is defined as Equation 6.10.

$$DMR_{\text{int}eractive} = \frac{m_{\text{int}eractive}}{n_{\text{int}eractive}}, \qquad \text{(Equation 6.10)}$$

where $DMR_{interactive}$ is the interactive deadline meet ratio in percent, $n_{interactive}$ is the total number of the user input events such as keyboard strokes or mouse clicks, and $m_{interactive}$ is the number of user input events that got response in deadline.

It has been reported that human cannot detect intermissions within a sequence of visual events as long as the intervals between two consecutive events are shorter than 50 milliseconds [85][86]. This perceptional boundary is called human perception threshold. Human perception threshold is widely used as the deadline of the system response to user input events.

To determine whether a user input event got response in human perception threshold, the beginning and end of the response process to the user input event should be defined. In [Flautner02], the authors called this response process 'interactive episode'. According to the authors, an interactive episode begins when a user input event occurs. Upon the beginning of an interactive episode, the graphical user interface (GUI) server task and the recipient task of the user input are recognized as the involved tasks. If the involved tasks communicate with other tasks, then the tasks are also recognized as the involved tasks. The authors defined the end of an interactive episode as the time point when all the involved tasks have voluntarily relinquished CPU.

This definition of interactive episode is appropriate to preserve the system performance in responding to user input events. However, it is not in evaluating the interactive deadline meet ratio. For example, imagine the case where a user activates an application. The user clicks an icon and waits until the application is on the screen. This process typically takes

a much longer time than the human perception threshold. Because the application task will not relinquish CPU until it completes the whole activation process, if one evaluate whether the human perception threshold was exceeded or not upon the completion of the whole activation process, the result should be deadline miss. Keeping a high CPU speed until the completion of the whole activation process is desirable to improve the user experience. This is the rationale for the definition of interactive episode in [75]. However, evaluating deadline meeting at the end of the whole activation process does not make sense. Therefore, for the purpose of evaluation, a new definition of the response process is necessary.

In this work, the beginning and end of the response process to a user input event is defined as follows. The beginning of a response process is defined the same as the definition in [75]. Once a task receives the user input event from the GUI server task, the recipient task will start generating the requested outputs and send them back to the GUI server task. Note that, if a user input event does not stimulate the target task to generate any recognizable output on the screen, the user input event is not the subject to evaluate. Thus, it can be assumed that every user input event recipient task will reply back to the GUI server task with some data to be displayed on the screen without loss of generality. Upon the reception of the outputs from the user input event recipient task, the GUI server task will begin to update the screen. In this research, the end of a response process to a user input event is defined as the time point when the GUI server task updated the screen with the outputs from the user input event recipient task for the first time since the corresponding user input event. The GUI server task may repeat the screen update until the completion of the response process. But, a user will feel that the system is responding immediately as long as the screen update starts in the human perception threshold.

Figure 6.12 shows this definition of the response process to user input events. DVS-suite maintains a data structure to trace the phase transitions for every task that is currently transacting a response process to the corresponding user input event. One thing to note is that the phase 2 transitions to the phase 1 upon a new user input event to the same task. In this case, the current response process is stopped and evaluated and a new response process is started. As for the stopped response process, if the elapsed time exceeds human

perception threshold, DVS-suite increments the deadline miss number by one. Otherwise, the stopped response process is ignored.

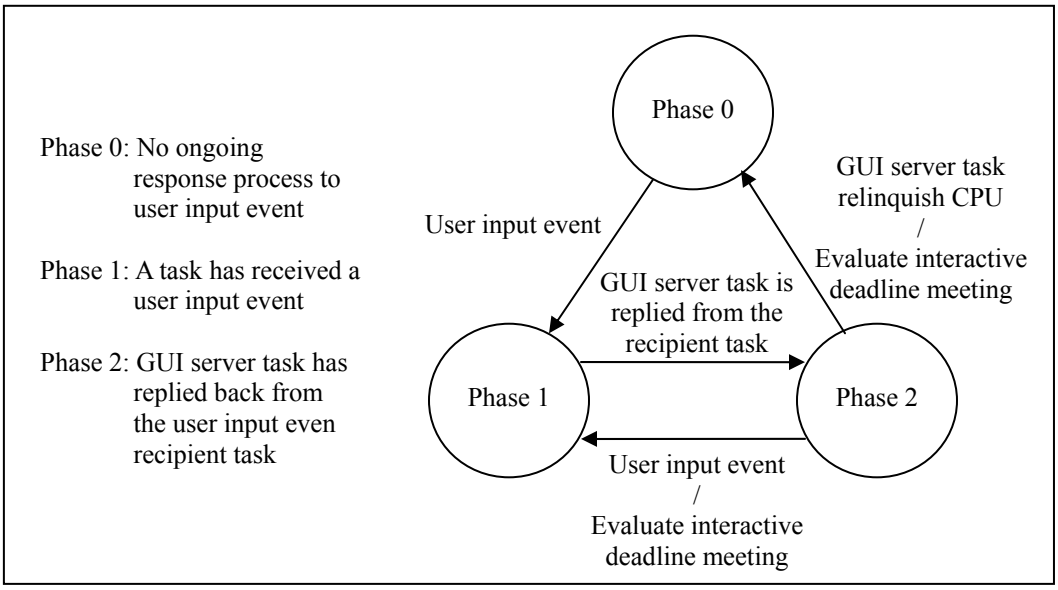


Figure 6.12 State diagram for the response process by a task to a user input event

6.2.4.3 Overheads

The overhead of a DVS scheme is evaluated in the following three ways.

- The time taken by the DVS scheme
- The system energy consumption consumed by the DVS scheme
- The number CPU operating point transitions caused by a DVS scheme

As mentioned at the beginning of this section, these quantities are already incorporated in the major metrics. The reason to measure the overheads is to get an insight into the choice of an appropriate DVS scheme to the system in which a property such as the CPU energy consumption or CPU operating point transition cost is especially critical. For example, if a system is equipped with a DVS-enabled CPU whose CPU operating point transition cost is extremely high, then the DVS scheme that causes the least number of CPU operating point transitions will be the best choice. To measure the overhead in time,

DVS-suite keeps tracing the spent time by DVS scheme separately. Because the time spent by DVS scheme is available, the overhead in system energy consumption can be calculated easily by using Equation 6.2 described in Section 6.2.4.1.1. The overhead in CPU operating point transition number is measured by distinguishing the transitions caused by a DVS scheme from the transitions caused by TM2 of Pentium-M processor. For the distinction, DVS-suite maintains a dirty bit that is set upon writing Pentium-M IA32_PERF_CTL MSR via WRMSR instruction. The dirty bit is kept 0 as long as the MSR is updated by the DVS scheme. If TM2 updates the MSR, the dirty bit is set to 1 and DVS-suite can detect the CPU operating point transition due to TM2.

6.2.5 Experimental tools

The experiments were performed by using two Linux-based tools, Tweak-PM and DVS-suite, developed for this research. This section describes the functions, implementations, and usages of Tweak-PM and DVS-suite.

6.2.5.1 Tweak-PM

Tweak-PM is a Linux loadable kernel module (LKM) to probe and set some features of Intel Pentium-M processors involved with the CPU speed and voltage scaling. As a LKM, it can be loaded and unloaded dynamically using Linux commands ‘insmod’ and ‘rmmod’. Tweak-PM takes advantage of Linux ‘proc’ file system to allow a user to access the kernel. Upon loading, Tweak-PM creates the ‘tweak_pm’ directory under */proc* and sets five files ‘cpuinfo’, ‘status’, ‘tm2_ctl’, ‘eist_ctl’, and ‘op’, which are the gateways through which the user can handle the processor features, in the tweak_pm directory. The functions that Tweak-PM provides and their usages are as follows.

- Probing the processor signature of a given Pentium-M processor and checking if the Pentium-M supports RDMSR and WRMSR instructions, thermal monitor 2 (TM2), Enhanced Intel SpeedStep (EIST), and on-demand clock modulation (ODCM) and if it has a digital thermal sensor (DTS). Following command activates this function.

```
% cat /proc/tweak_pm/cpuinfo
```

- Probing the current status of the Pentium-M including the current CPU operating point and the current CPU core temperature. Following command activates this function.

```
% cat /proc/tweak_pm/status
```

- Enabling and disabling TM2 and EIST. To switch from TM1 to TM2 input the following commands.

```
% echo on > /proc/tweak_pm/tm2_ctl
% cat /proc/tweak_pm/tm2_ctl
```

To switch back from TM2 to TM1, input the following commands.

```
% echo off > /proc/tweak_pm/tm2_ctl
% cat /proc/tweak_pm/tm2_ctl
```

To enable EIST,

```
% echo on > /proc/tweak_pm/eist_ctl
% cat /proc/tweak_pm/eist_ctl
```

And, to disable EIST,

```
% echo off > /proc/tweak_pm/eist_ctl
% cat /proc/tweak_pm/eist_ctl
```

- Enforcing a new CPU operating point. For example, to set the CPU operating point to ‘p4’ which means (800MHz, 1.068V), input the following commands.

```
% echo p4 > /proc/tweak_pm/op
% cat /proc/tweak_pm/op
```

To probe the Pentium-M processor’s CPU signature and check if it supports RDMSR and WRMSR instructions, TM2, EIST, and ODCM and if it has DTS, Tweak-PM calls the CPUID instruction (0x0FA2 in op-code) that returns various pieces of information about the CPU. The type of information that CPUID to return can be chosen by setting EAX register value before calling CPUID. Setting EAX to 0, 1, 2, or 3, respectively, makes CPUID return the vender ID, processor information and feature bits, cache and TLB

information, or processor serial number. Processor information and feature bits are the pieces of information required to check if the CPU supports ODCM, EIST, TM2, and the MSR access via RDMSR and WRMSR.

Upon calling with EAX of 1, CPUID returns the CPU's stepping, model, and family information (i.e., the signature of a CPU) into EAX, feature flags into EDX and ECX, and additional feature information into EBX. Whether the CPU supports EIST, TM2, and MSR access can be checked by observing the particular bits of the return values in EDX and ECX as follows.

- EIST bit (the bit 7 of ECX): If this feature bit is set, the CPU supports EIST.

- TM2 bit (the bit 8 of ECX): If this feature bit is set, the CPU supports TM2.

- MSR bit (the bit 5 of EDX): If this feature bit is set, the CPU supports the MSR accesses via RDMSR and WRMSR.

The first step to check if the Pentium-M has a DTS is calling CPUID with EAX of 0. This returns the maximum supported standard level of CPUID into EAX. If the level is same as or higher than 6, the Pentium-M has a DTS and the current core temperature in Celsius can be probed by reading the bits 22~16 of Pentium-M MSR IA32_THERM_STATUS, which contains the difference between T_JUNCTION, the thermal threshold particular to Pentium-M (100 degrees Celsius), and the current core temperature. For example, if the bits 22~16 value is 30, then the current core temperature is 100 – 30 = 70 degrees in Celsius.

DTS does not affect the operation of TM2. Regardless of the existence of DTS, TM2 activates upon the core temperature exceeding T_JUNCTION. The role of DTS is to provide the current core temperature in number. Whether TM2 is in active or not and had been active or not can be detected by observing the 'thermal status' (bit 0) and 'thermal status log' (bit 1) bits of IA32_THERM_STATUS MSR (MSR address 0x19C). Thermal status bit of 1 indicates TM2 is in active to reduce the core temperature. Thermal status log bit of 1, on the other hand, shows that TM2 had been active since the system was

booted.

Recent versions of Linux kernel provide cpu_has() macro, a wrapper of CPUID, which eases this checking. A usage example of cpu_has() is as follows.

```
struct cpuinfo_x86 *cpu = cpu_data;
cpu_has(cpu, X86_FEATURE_EST);
```

where, cpuinfo_x86 structure and cpu_data cpuinfo_x86 type array are defined in linux/include/asm-i386/processor.h, and cpu_has() macro and X86_FEATURE_EST are defined in linux/include/asm-i386/cpufeature.h.

However, the version of Linux used in this research, the Red Hat 9 based on Linux kernel version 2.4.8-20, does not define X86_FEATURE_EST. Although a user can define it himself and then recompile the kernel, an old-fashioned, a bit inconvenient, existing function cpuid(int op, int *eax, int *ebx, int *ecx, int *edx), which is defined in linux/include/asm-i386/processor.h, was used in Tweak-PM for backward compatibility.

An important reason to probe the processor signature is that even same Pentium-M CPUs have different ways of TM2 switching according to their processor signature as explained in Section 6.2.1.

EIST can be enabled or disabled by setting or resetting the bit 16 of Pentium-M MSR IA32_MISC_ENABLE (MSR index 0x1A0). The detailed methods Tweak-PM switches between TM1 and TM2, enforces a new CPU operating point, and configures the ODCM are described in Section 6.2.1.

Tweak-PM was originally developed as the pilot tool for DVS-suite. But, as mentioned in Section 6.2.4.1.1, it was used to manually adjust the CPU operating point when the average system power consumptions at each CPU operating point were measured to build the energy model.

6.2.5.2 DVS-suite

DVS-suite is the main tool developed to conduct the experiment on GPScheDVS and the

existing autonomous DVS schemes on a laptop which runs Red Hat Linux 9 OS. Unlike Tweak-PM which is solely a Linux LKM and does not require any modification of Linux kernel source, DVS-suite consists of (1) a collection of codes which is built-in Linux kernel source and implements GPSched, GPScheDVS, the existing autonomous DVS schemes, DVS performance metric tracing functions, and a job sequence tracing function and (2) DVS-suite module, a Linux LKM which allows the user to configure the DVS-suite functions implemented in the kernel source. The functions that DVS-suite provides include the followings. All the functions can be performed dynamically while the system is running on demand of users.

- Switching between GPSched and the native Linux task scheduling.

- Switching among the implemented DVS schemes.

- Tracing, storing, and reporting DVS performance metrics.

- Tracing, storing, and reporting the job execution sequence. The trace can be used to further analyze the properties in the execution of tasks, which are not available in the report file of the DVS performance metrics. The maximum number of jobs that can be reported is one million.

Table 6.16 lists the DVS schemes that DVS-suite supports. Table 6.17 and 6.18 describe the pieces of information that the DVS performance tracing function and the job sequence tracing function of DVS-suite report, respectively.

Table 6.16 DVS schemes that DVS-suite supports

DVS scheme name	Description
No-DVS	Keeping the maximum CPU operating point.
Const-Min	Keeping the minimum CPU operating point.
Manual	Keeping the current CPU operating point. This option should be chosen to adjust the CPU operating point manually with the tweak-pm LKM described in Section 6.2.5.1.
OnDemand cpufreq governor (OCG) for EIST	An interval-based autonomous DVS scheme proposed in [38].
AVG3	An interval-based autonomous DVS scheme proposed in [39].
Vertigo	A task-based DVS autonomous scheme proposed in [43].
GPScheDVS	The proposed DVS scheme that follows the suggested autonomous CPU speed control paradigm for commodity-OS-based GP PCs.

Table 6.17 Pieces of information the metric tracing function of DVS-suite reports

Category	Description
System statistics and overhead measures	Duration of the experiment (usec)
	The last time when TM2 was activated (Kept even after the TM2 deactivation)
	The elapsed time since TM2 has been activated (Reset upon TM2 deactivation)
	Number of context switches
	Number of CPU operating point transition due to TM2/DVS schemes
	Number of *ret_from_system_call*, i.e., total number of system calls and interrupts
	CPU_CLK_UNHALTED, i.e., the number of clock cycles seen when CPU is in active
	Fractions of time spent during CPU was idling, busy (total), busy executing tasks, busy scheduling and switching tasks, and busy executing DVS scheme codes
QoS measures	Total number of user input events and the numbers completed within 1, 5, 10, 50, 100, 500, and 1000 msec, respectively
	Total number of video outputs and the number met deadline
	Total number of audio outputs and the number met deadline
Energy and power measures	System energy consumption during the experiment, CPU was idling, CPU was busy (total), busy executing tasks, busy scheduling and switching tasks, and busy executing DVS scheme codes
	Average system power consumption during the experiment, CPU was idling, CPU was busy (total), busy executing tasks, busy scheduling and switching tasks, and busy executing DVS scheme codes
	CPU energy consumption during the experiment, CPU was idling, CPU was busy (total), busy executing tasks, busy scheduling and switching tasks, and busy executing DVS scheme codes
	Average CPU power consumption during the experiment, CPU was idling, CPU was busy (total), busy executing tasks, busy scheduling and switching tasks, and busy executing DVS scheme codes

Table 6.18 Pieces of information the job tracing function of DVS-suite reports

Information name	Description
comm	The command line name of tasks
pid	Process ID
tgid	Thread group ID
prio	Dynamic priority of tasks
state	State of tasks.
io_type	Whether the job communicated with either X, ESD, or the audio driver
io_bytes	The amounts of data transferred to X, ESD, or the audio driver in Bytes
io_sec/io_usec	The time point when the communication was performed
exe_begin_sec/ exe_begin_usec	The beginning time points of each job
exe_end_sec/ exe_end_usec	The end time points of each job
exe_length_usec	The length of each job's execution in wall clock time
exe_length_usec_fse	The length of each job's execution in full speed equivalent time
cpu_op	The CPU operating point during the execution of each job

The commands to activate or deactivate DVS-suite are as follows.

Activate DVS-suite:

```
% echo on > /proc/dvs_suite_mod/on_dvs
% cat /proc/dvs_suite_mod/on_dvs
```

Deactivate DVS-suite:

```
% echo off > /proc/dvs_suite_mod/on_dvs
% cat /proc/dvs_suite_mod/on_dvs
```

The commands to switch task scheduling mechanism between the native Linux task scheduling and GPSched are as follows.

Switch from the native Linux task scheduling to GPSched:

```
% echo gpsched > /proc/dvs_suite_mod/sched_scheme
% cat /proc/dvs_suite_mod/sched_scheme
```

Switch back from GPSched to the native Linux task scheduling:

```
% echo native > /proc/dvs_suite_mod/sched_scheme
% cat /proc/dvs_suite_mod/sched_scheme
```

Followings are the commands to switch among the implemented DVS schemes.

Apply the DVS scheme No-DVS:

```
% echo no_dvs > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Apply the DVS scheme Const-min:

```
% echo min > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Apply the DVS-scheme Manual:

```
% echo manual > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Apply the DVS-scheme OCG:

```
% echo ocg > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Apply the DVS-scheme AVG3:

```
% echo avg3 > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Apply the DVS-scheme Vertigo:

```
% echo vertigo > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Apply the DVS-scheme GPScheDVS:

```
% echo gpschedvs > /proc/dvs_suite_mod/dvs_scheme
% cat /proc/dvs_suite_mod/dvs_scheme
```

Run GPScheDVS in SE mode:

```
% echo 0 > /proc/dvs_suite_mod/set_gpschedvs_strategy
% cat /proc/dvs_suite_mod/set_gpschedvs_strategy
```

Run GPScheDVS in CP mode:

```
% echo 1 > /proc/dvs_suite_mod/set_gpschedvs_strategy
```

```
% cat /proc/dvs_suite_mod/set_gpschedvs_strategy
```

The video frame period and uncompressed audio bit rate of a continuous media data should be input by the user to guide DVS-suite in evaluating the QoS of continuous media applications before the experiment is conducted. The video frame periods of the continuous media data experimented in this research were 33.367 and 41.708 milliseconds as Table 6.4 in Section 6.2.3 showed. And, the uncompressed audio bit rates were 1536, 1411.2, and 705.6 Kbits per seconds and are 0.1920 (= $1536/8/10^6$), 0.1764 (=$1411.2/8/10^6$), and 0.0882 (=$705.6/8/10^6$) Bytes per microsecond, as Table 6.4 and 6.5 showed. No user guidance is required for the evaluation of the QoS of interactive applications. Followings are the commands to guide, enable, and disable the DVS performance metric tracing functions of DVS-suite.

Set video frame period to 33.367 milliseconds:

```
% echo 33367 > /proc/dvs_suite_mod/set_v_period
% cat /proc/dvs_suite_mod/set_v_period
```

Set video frame period to 33.367 milliseconds:

```
% echo 41708 > /proc/dvs_suite_mod/set_v_period
% cat /proc/dvs_suite_mod/set_v_period
```

Set uncompressed audio bit rate to 0.0882 Bytes per microsecond:

```
% echo "0.0882" > /proc/dvs_suite_mod/set_a_rate
% cat /proc/dvs_suite_mod/set_a_rate
```

Set uncompressed audio bit rate to 0.1764 Bytes per microsecond:

```
% echo "0.1764" > /proc/dvs_suite_mod/set_a_rate
% cat /proc/dvs_suite_mod/set_a_rate
```

Set uncompressed audio bit rate to 0.1920 Bytes per microsecond:

```
% echo "0.1920" > /proc/dvs_suite_mod/set_a_rate
% cat /proc/dvs_suite_mod/set_a_rate
```

Activate the DVS-suite function that traces the QoS of continuous media applications:

```
% echo on > /proc/dvs_suite_mod/on_cmqos_trace
% cat /proc/dvs_suite_mod/on_cmqos_trace
```

Disable the DVS-suite function that traces the QoS of continuous media applications:

```
% echo off > /proc/dvs_suite_mod/on_cmqos_trace
% cat /proc/dvs_suite_mod/on_cmqos_trace
```

Activate the DVS-suite function that traces the QoS of interactive applications:

```
% echo on > /proc/dvs_suite_mod/on_iaqos_trace
% cat /proc/dvs_suite_mod/on_iaqos_trace
```

Disable the DVS-suite function that traces the QoS of interactive applications:

```
% echo off > /proc/dvs_suite_mod/on_iaqos_trace
% cat /proc/dvs_suite_mod/on_iaqos_trace
```

Finally, the job sequence tracing function of DVS-suite is activated and deactivated using the following commands.

Activate the DVS-suite function that traces job sequence:

```
% echo on > /proc/dvs_suite_mod/on_job_trace
% cat /proc/dvs_suite_mod/on_job_trace
```

Disable the DVS-suite function that traces job sequence:

```
% echo off > /proc/dvs_suite_mod/on_job_trace
% cat /proc/dvs_suite_mod/on_job_trace
```

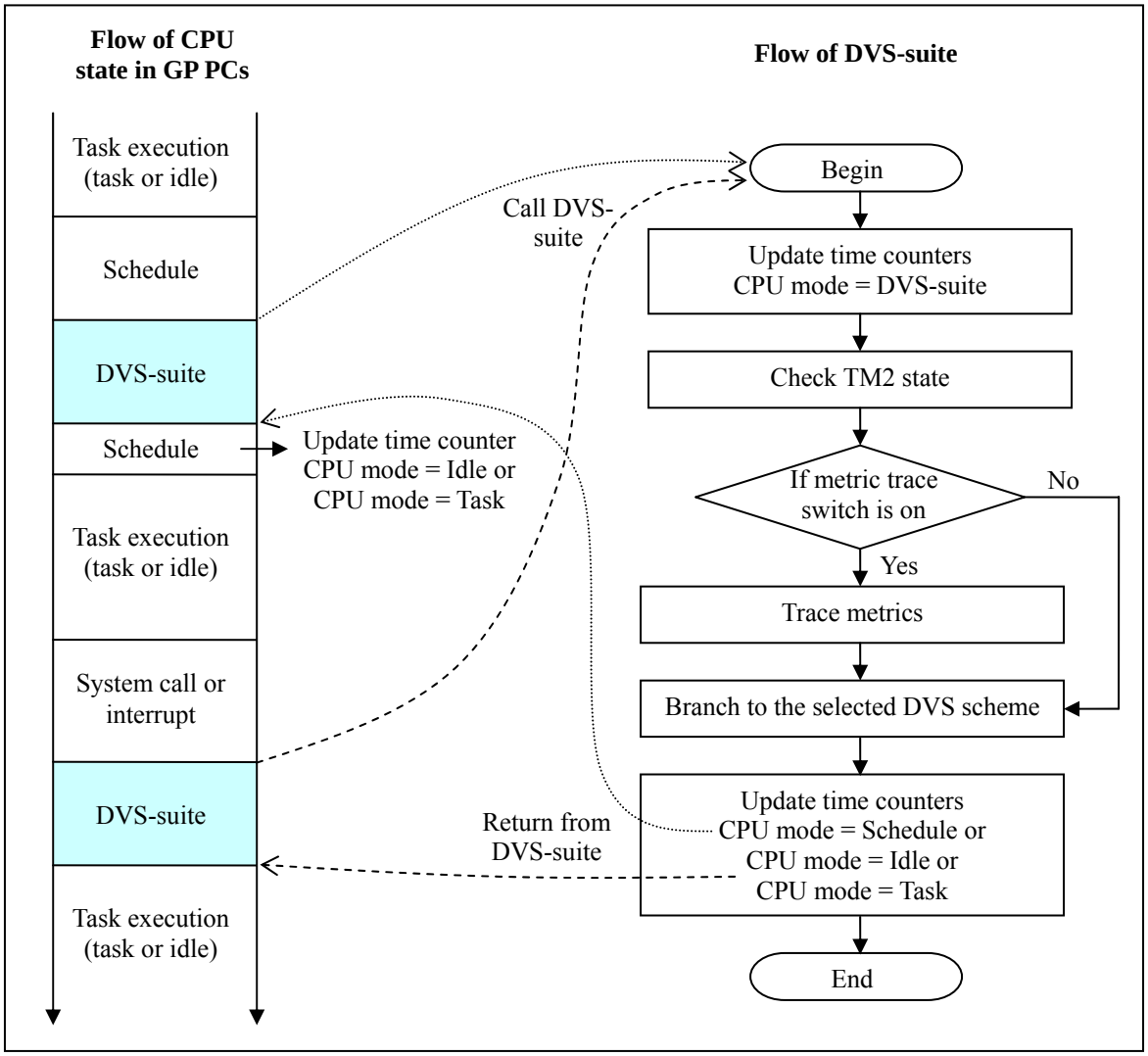


Figure 6.13 Flow of DVS-suite operation

Figure 6.13 illustrates the flow of DVS-suite operations. DVS-suite is called either from linux/arch/i386/kernel/process.c: __switch_to() when a context switch occurs or from entry.S: ENTRY(ret_from_system_call) when a system call or interrupt occurs. Once called, it updates time counters and the current CPU mode, check TM2 states, performs the metric trace functions if it is configured to do by the user, and gives the DVS scheme selected by the user a chance to run. After the DVS scheme executes, DVS-suite updates the CPU mode and the time counters again with the time spent since it was called this time.

Given a chance to run, a DVS scheme updates its own timer, estimates tasks' CPU speed

requirements, and adjusts the CPU operating point if needed. DVS-suite provides the tools required by a DVS scheme in doing these. Figure 6.14 shows the four tools that DVS-suite provides to a DVS scheme.

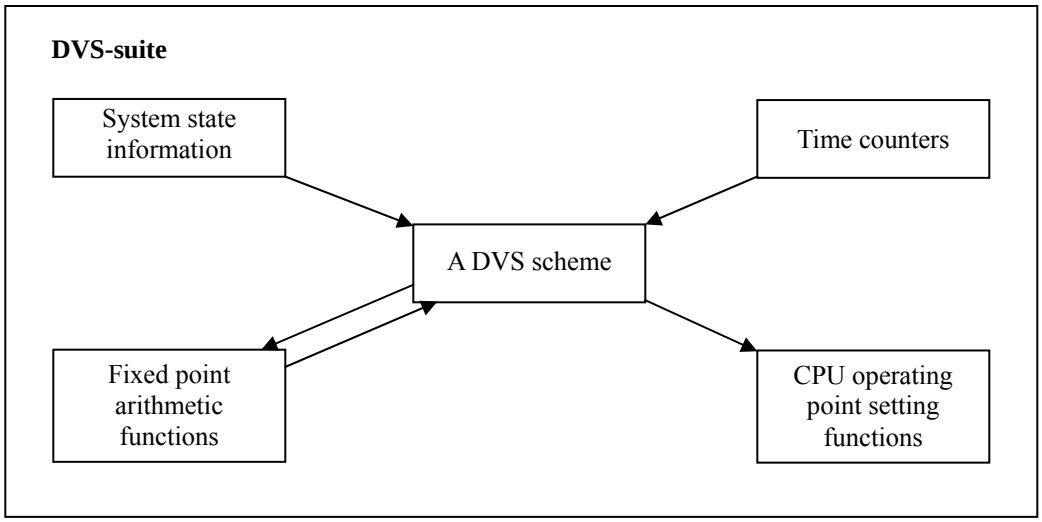


Figure 6.14 Four tools that DVS-suite provides to a DVS scheme

A DVS scheme can refer the system state information that DVS-suite maintains. Table 6.19 lists some important pieces of system state information. For example, a DVS scheme can easily detect a user input event and whether it is a keyboard stroke or a mouse click simply by checking the just_received_user_event_x_opcode variable of DVS-suite. The DVS scheme then can use the DVS-suite time counters to obtain the timing information required to make the decisions on CPU operating point. Table 6.20 lists the type of DVS-suite time counters. In many cases, tasks' CPU speed requirement takes a form of fractional number. However, Linux does not provide the arithmetic functions to calculate such fractional numbers in the kernel space. DVS-suite implements a set of fixed-point arithmetic functions that can be used by a DVS scheme. Finally, the CPU operating point setting function performs the CPU operating point transition on behalf of a DVS scheme. The DVS scheme just needs to call the function with the level of CPU operating point to set.

Table 6.19 Important fields of DVS-suite system state data structure

Category	Field	Description
System state	tm2_active	TM2 is in active
	cpu_op_index	The current CPU operating point
	cpu_mode	The current CPU mode (IDLE, TASK, SCHEDULE, or DVS_SUITE)
Event information	just_read_ipc_writer_pid	The PID of the task that sent an IPC which the current job received
	flag_just_wrote_an_ipc	The current job sent an IPC
	just_received_user_event_x_opcode	The X protocol op-code of the user input event that the current job received
	just_written_ui_server_type	The type of UI server (X, ESD, or audio driver) to which the current job made an output

Table 6.20 DVS-suite time counters

Field	Description
elapsed_sec/usec	The wall time counter
idle_sec/usec [op]	The idle times spent at each CPU operating point
idle_total_sec/usec	The total idle time spent so far
busy_sec/usec [op]	The busy time spent at each CPU operating point
busy_task_sec/usec [op]	The busy time spent executing task at each CPU operating point
busy_schedule_sec/usec [op]	The busy time spent executing task scheduling and context switching at each CPU operating point
busy_dvs_suite_sec/usec [op]	The busy time spent executing DVS schemes at each CPU operating point
busy_total_sec/usec	The busy time spent so far (Wall time)
busy_fse_sec/usec	The full speed equivalent busy time spent so far
tm2_activated_sec/usec	The last time when TM2 was activated
tm2_duration_sec/usec	The elapsed time since the last TM2 activation (Reset upon TM2 deactivation)

6.3 Experimental results

This section presents and analyzes the experimental results on the performances of GPScheDVS and the existing autonomous DVS schemes under the various usage scenarios described in Section 6.2.3. Sections 6.3.1 treats the experimental results obtained under the typical real-life usage scenarios of a commodity-OS-based GP PCs defined in Table 6.6 and 6.7, respectively. In these usage scenarios, a SRT application is executed either alone or with various amounts of NRT applications running in the background. Section 6.3.2 focuses on the DVS schemes' performances under the usage scenarios described in Table 6.8 and 6.9 in which a SRT application is executed while running a HINRT application which keeps causing the hard CPU idle time. Finally, Section 6.3.3 deals with the usage scenarios described in Table 6.10 and 6.11. These usage scenarios are to investigate if the effectiveness of GPSched as a DVS-friendly GP task scheduling holds when only SRT applications are executed without running any NRT application in the background.

The experimental results are presented in the five DVS performance metrics described in Section 6.2.4: system energy consumption, CPU energy consumption, the average CPU power consumption, the QoS of time constrained applications, and the overhead of DVS in time, system energy, and the number of CPU operating point transitions. Then, how the following three main features of GPScheDVS make it outperform over the existing autonomous DVS schemes is discussed: (1) GPSched, a DVS-friendly GP BE task scheduler, (2) the SE mode operation in which HINRT tasks are specially cared to avoid the increased system energy consumption due to the misuse of the hard CPU idle time, and (3) the CP mode operation to restrain the average CPU power consumption that determines the amount of CPU heat.

6.3.1 Typical usage scenarios

6.3.1.1 Continuous media applications

This section presents the experimental results obtained under one of the ordinary type of usage scenarios in which a continuous media application is executed to play a movie or

music with or without the NRT applications running in the background. A single-threaded continuous media application mplayer and a multi-threaded continuous media application aviplay were experienced. Various CPU utilizations of the gcc compilations were used to emulate various types and workloads of NRT applications taking advantage of the nature of gcc compilation shown in Section 6.2.3. This section focuses on the two usage scenarios avi_m_gcc_u[n] and avi_a_gcc_u[n] that typify the ten usage scenarios listed in Table 6.6.

Figure 6.15 shows the experimental results obtained under the usage scenario where mplayer plays the AVI movie described in Table 6.4 with or without the gcc compilation running in the background. Figure 6.15 (a), (b), (c), (d), (e), and (f) show the total completion time of the imposed workload, system energy consumption, CPU energy consumption, average CPU power consumption, video deadline meet ratio, and audio deadline meet ratio achieved by each DVS scheme, respectively.

The point of Figure 6.15 (a) is that only the Const-min and the GPScheDVS-CP delay the total completion time. This is because the Const-min always keeps the lowest CPU operating point and the GPScheDVS-CP also executes all NRT tasks always at the lowest CPU operating point. As can be seen in Figure 6.15 (c) and (d), the extended execution time reduces CPU energy consumption and, especially, average CPU power consumption. This is a natural result because, given a DVS-enabled CPU, the condition to minimize CPU energy consumption is to run the CPU at the lowest CPU operating point as the previous work on energy-optimal CPU control showed, and average CPU power consumption is the CPU energy consumption divided by the total execution time (Note that the total execution time by the Const-min and the GPScheDVS-CP is much longer than that by other DVS schemes in this case).

The difference between the results with the Const-min and the GPScheDVS-CP is that the GPScheDVS-CP achieved this without degrading the video/audio deadline meet ratios whereas Const-min severely degraded the deadline meet ratios, as Figure 6.15 (e) and (f) show. The primary reason for this difference is that the DVS-friendly GP task scheduler GPSched assigns a higher priority to SRT tasks than NRT tasks such that SRT

tasks meet their deadlines even under overload situations.

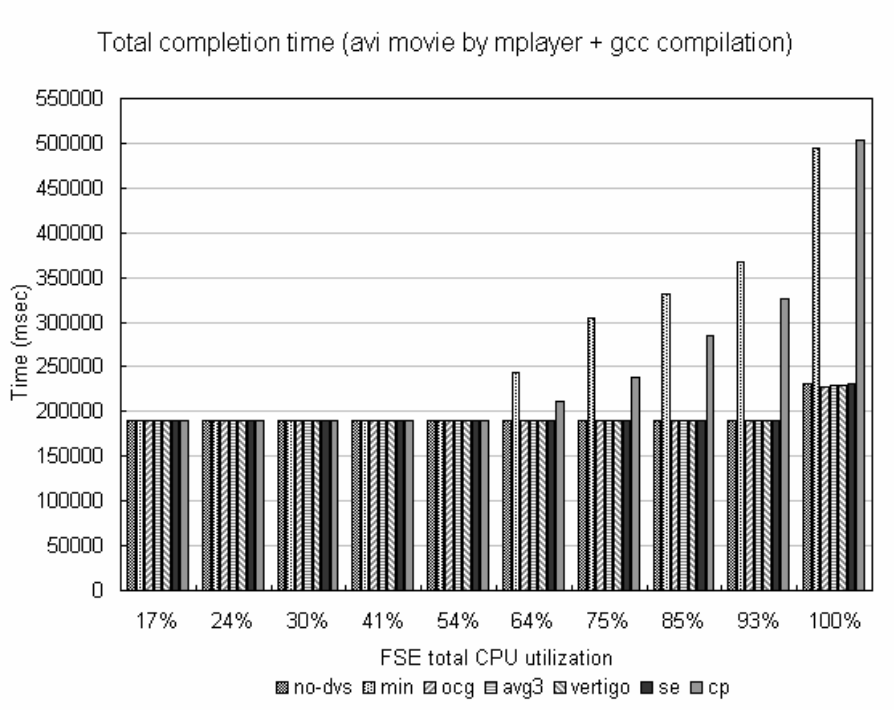
Figure 6.15 (b) shows, however, the longer execution time causes non-CPU components to consume more energy enough to overwhelm the CPU energy savings and, eventually, increase the total system energy consumption. To avoid this, all other DVS schemes raise the CPU speed along with the increase of workload such that the execution time is not extended. However, the increased system energy consumption is the correct result that meets the design goal of the GPScheDVS-CP – Reducing CPU heat dissipation by decreasing average CPU power consumption at the cost of the increased system energy consumption.

On the other hand, Figure 6.15 (a) and (c) show that the GPScheDVS-SE further reduces the CPU energy consumption over the existing DVS schemes without extending the total execution time. Accordingly, the system energy consumption reduces as much as the CPU energy savings and also does the average CPU power consumption as Figure 6.15 (b) and (d) show. This is because GPSched makes tasks' CMFSs uniform and close to the ICMFS than the native Linux task scheduling as Figure 6.16 (c) shows. Figure 6.16 (a), (b), and (c) show the IMFS of mplayer in playing the AVI movie, the CMFS of mplayer affected by the background NRT workload under the native Linux task scheduling, and the CMFS of mplayer generated by GPSched, respectively, for the usage scenarios avi_m_gcc_u41. In the figure, the CMFSs exceeding 1 means that transacting the corresponding video frames within deadline is impossible with the given CPU.

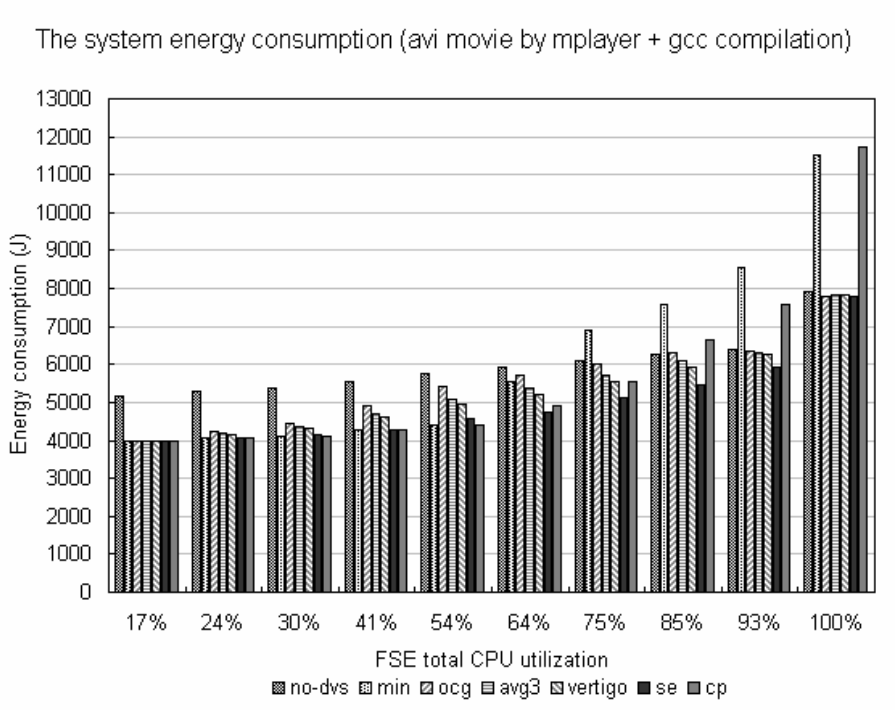
Figure 6.17 shows the CPU speed pattern with OCG, AVG3, Vertigo, and GPScheDVS for 30 seconds under the same usage scenario. The figure shows that GPScheDVS-SE keeps the CPU speed at the minimum for most of time based on the uniform and low CMFSs by GPSched, whereas other DVS schemes often have to run the CPU at a high speed due to the high CMFSs by the native Linux task scheduler.

Figure 6.18 confirm this superiority of GPScheDVS by showing the fraction of the entire execution time spent at each CPU operating point with each DVS scheme under the same usage scenario as Figure 6.16 and 6.17. In Figure 6.18, B0~B5 mean the fraction of busy

time spent at CPU operating point CPU-OP0 (the highest) ~ CPU-OP5 (the lowest). Similarly, I0~I5 mean the fraction of idle time spent at CPU operating point CPU-OP0 ~ CPU-OP5. Figure 6.19 also confirms the superiority of GPScheDVS by showing the patterns of the current (i.e., power) drawn by the experimental platform for a 5 seconds interval under the usage scenario avi_m_gcc_u41 with each DVS scheme.

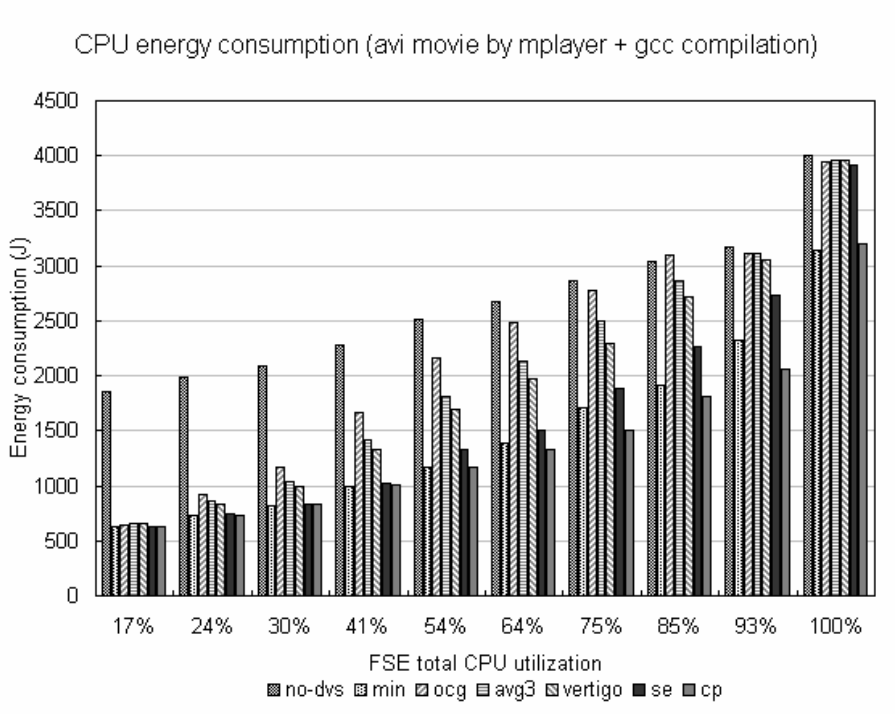


(a)

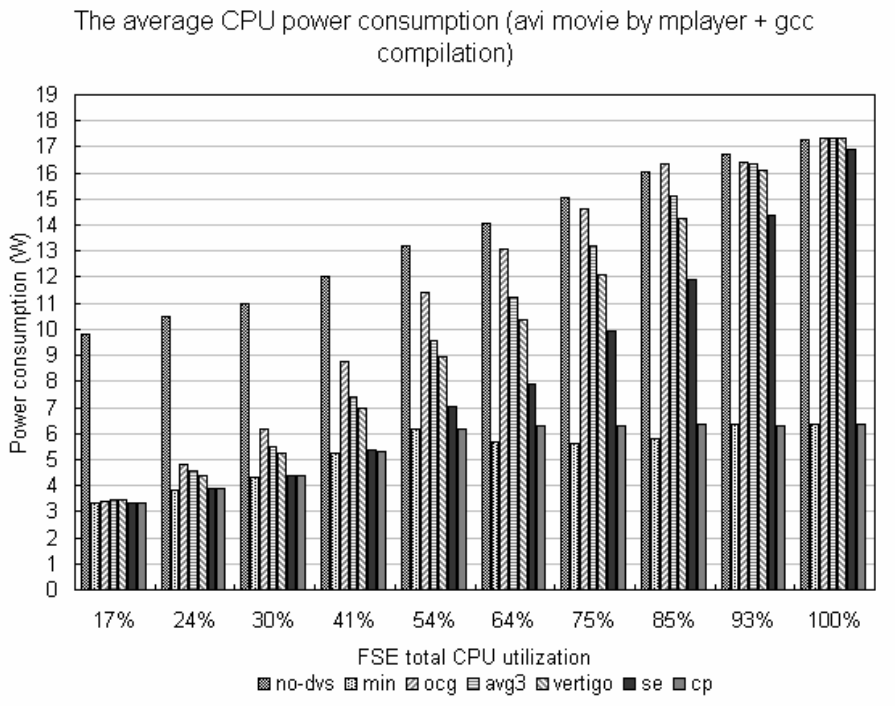


(b)

Figure 6.15 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios avi_m_gcc_u[n]

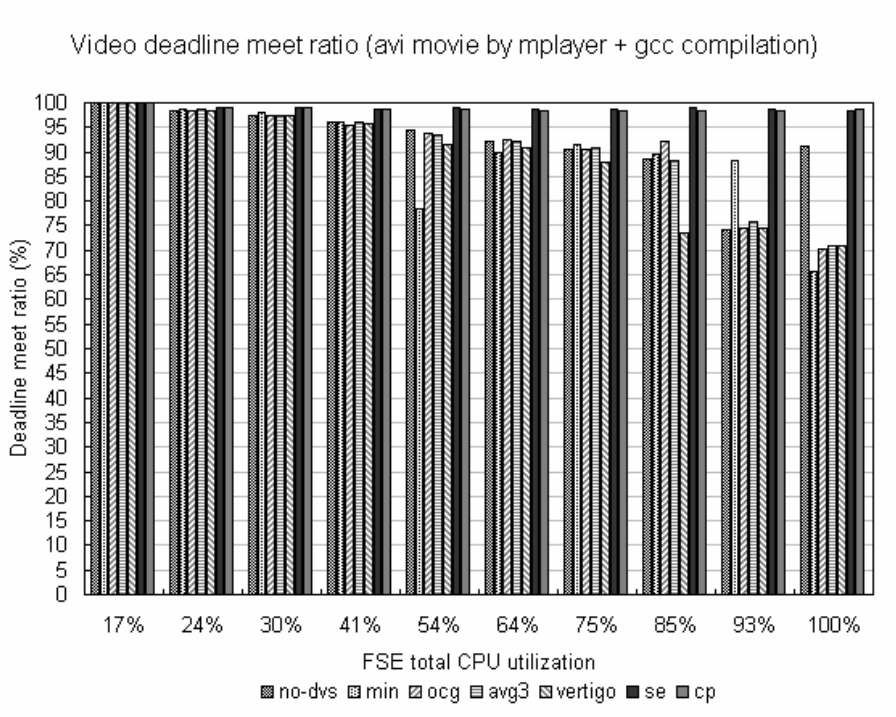


(c)

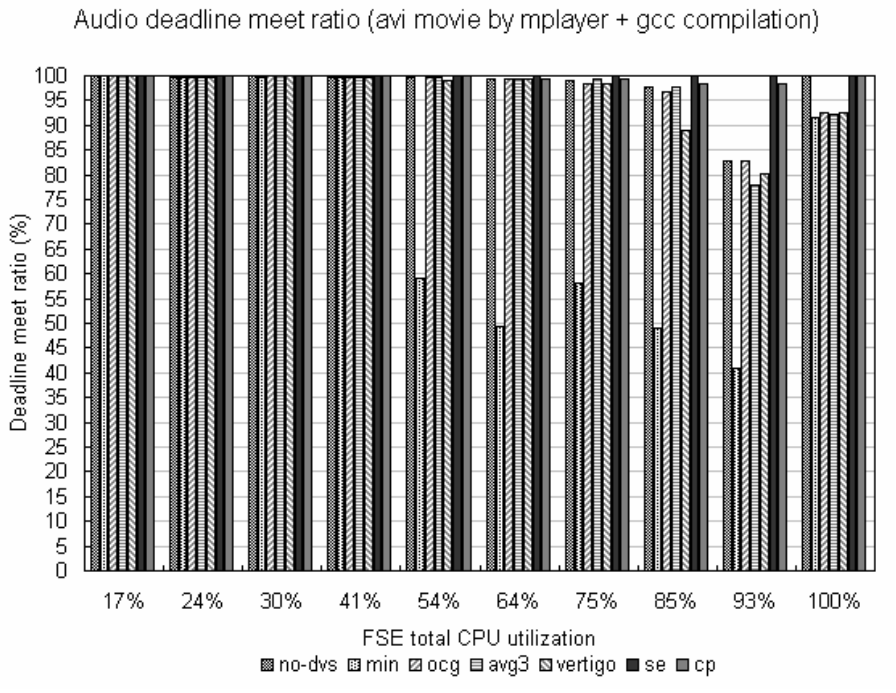


(d)

Figure 6.15 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios avi_m_gcc_u[n] (Cont'd)

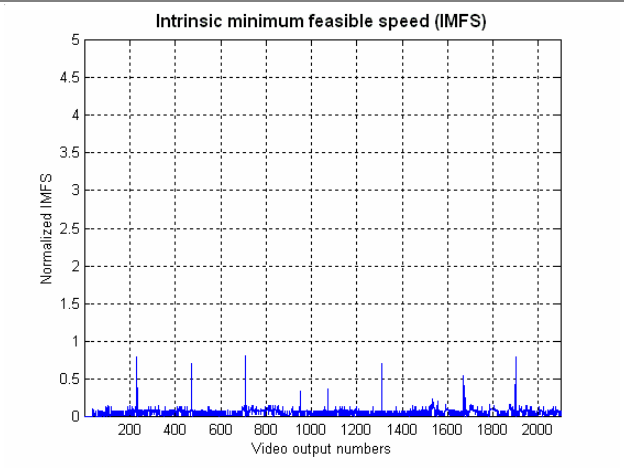


(e)

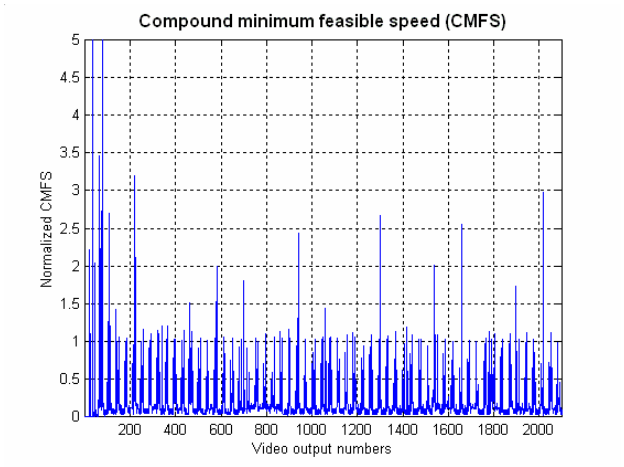


(f)

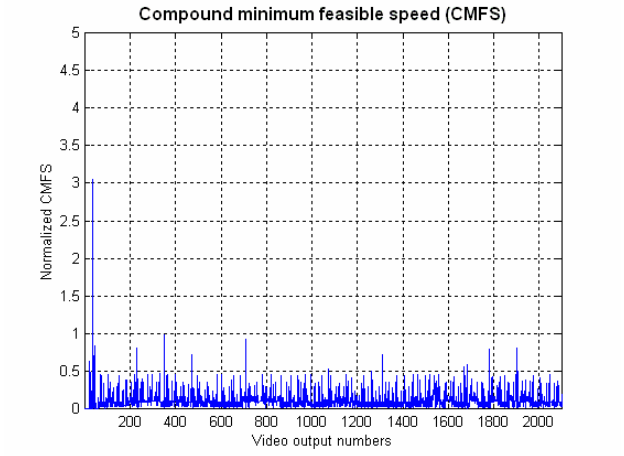
Figure 6.15 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios avi_m_gcc_u[n] (Cont'd)



(a) IMFS of mplayer



(b) CMFS of mplayer under the native Linux task scheduling



(c) CMFS of mplayer by GPSched

Figure 6.16 mplayer IMFS and the CMFS by the native Linux scheduler and GPSched under the usage scenarios avi_m_gcc_u41

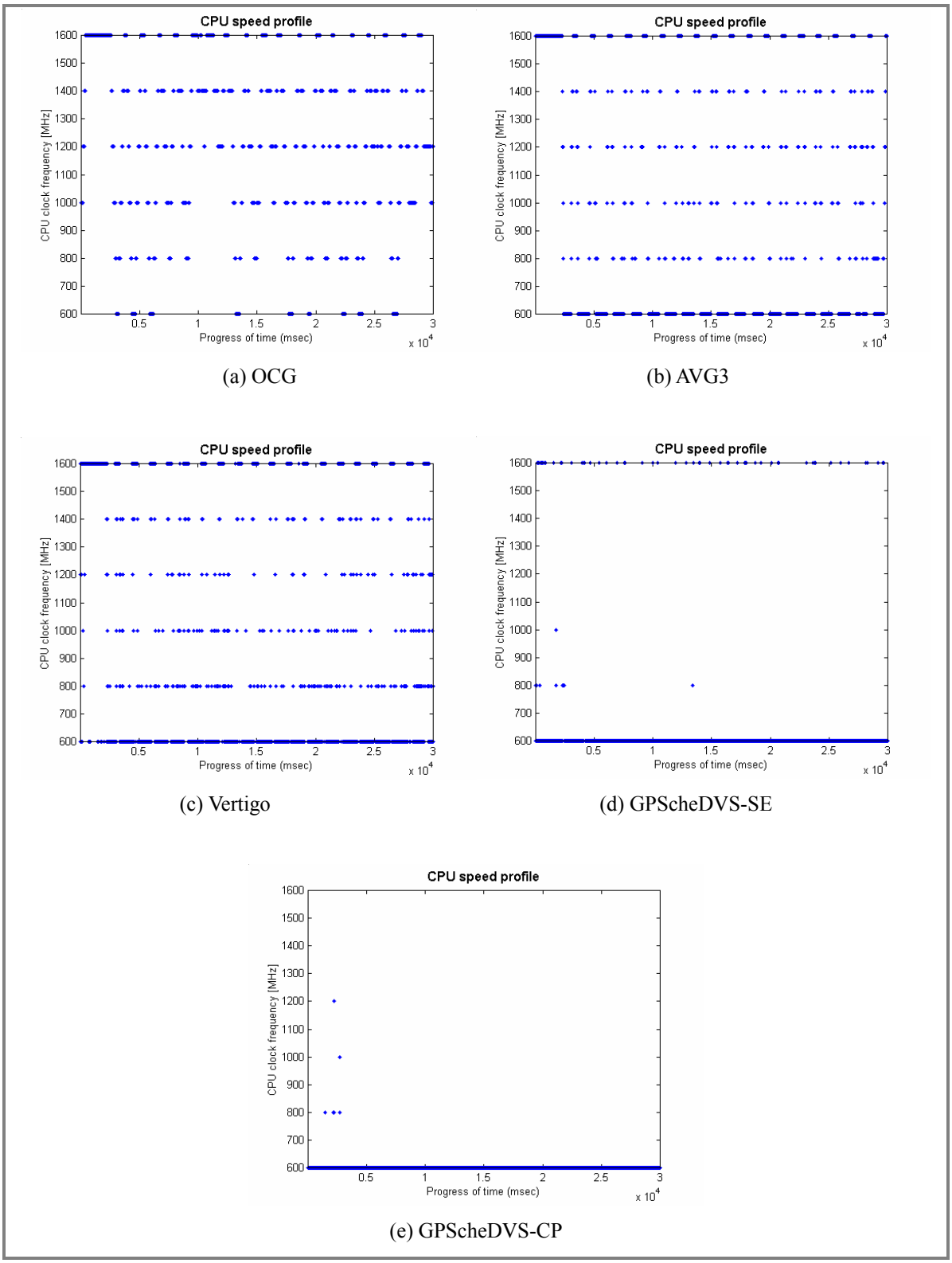


Figure 6.17 CPU speed pattern with OCG, AVG3, Vertigo, and GPScheDVS for 30 seconds under the usage scenarios avi_m_gcc_u41

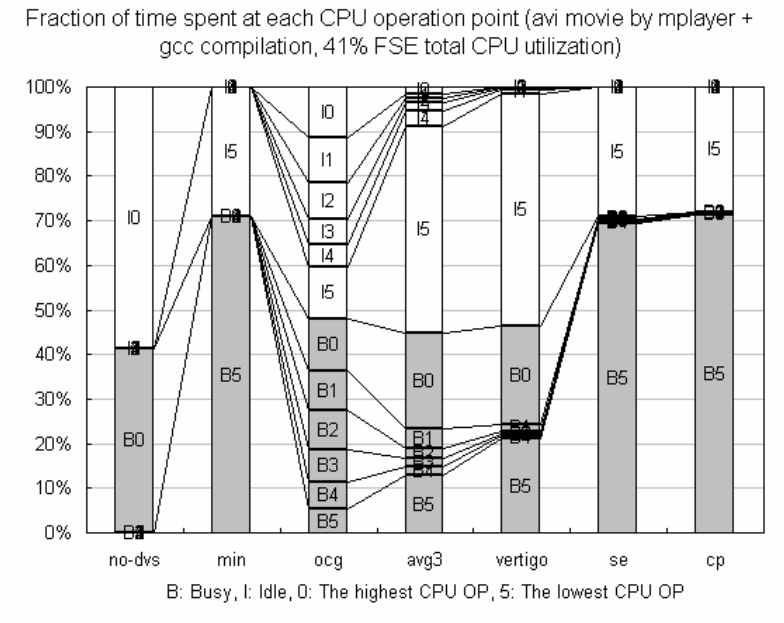
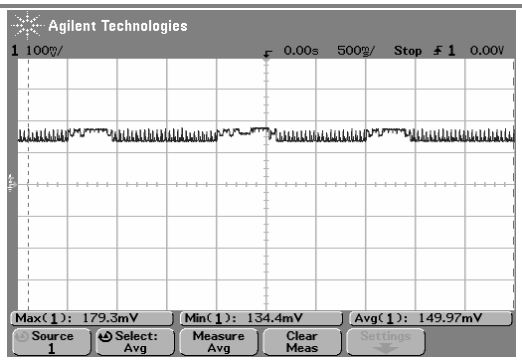
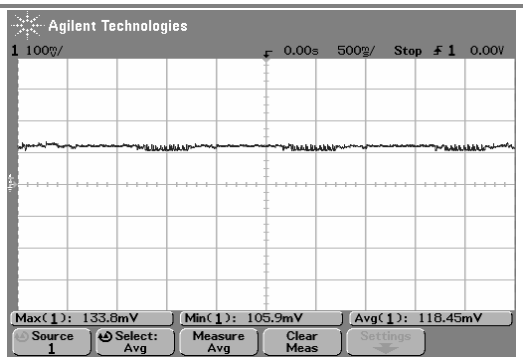


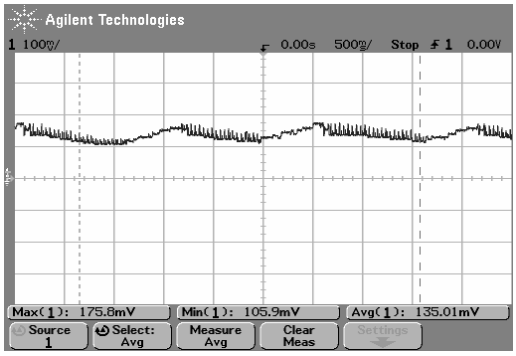
Figure 6.18 Fractions of time spent at each CPU operating point with each DVS scheme under the usage scenarios avi_m_gcc_u41



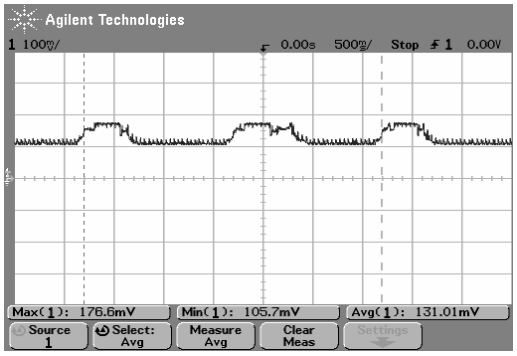
(a) No-DVS



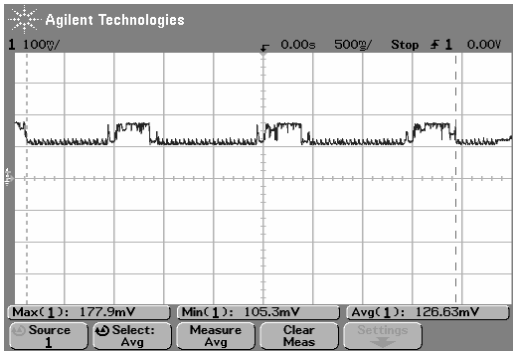
(b) Const-min



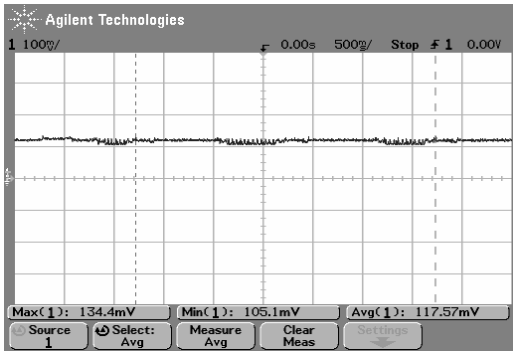
(c) OCG



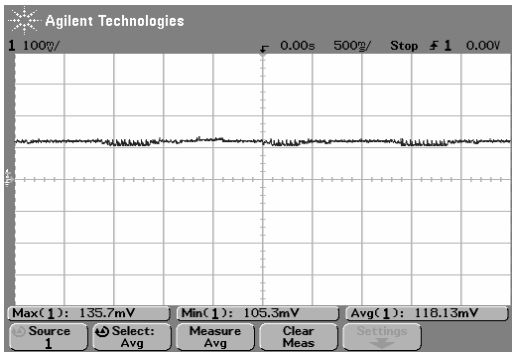
(d) AVG3



(e) Vertigo



(f) GPScheDVS-SE



(g) GPScheDVS-CP

Figure 6.19 System power consumption patterns with each DVS scheme under the usage scenario avi_m_gcc_u41

Figure 6.20 and 6.21 show the improvements in each DVS performance metric achieved by the GPScheDVS-SE and GPScheDVS-CP, respectively, over the existing DVS schemes. The improvements in energy and power were calculated by Equation 6.11.

$$(1-\frac{energy\ or\ power\ by\ an\ existing\ DVS\ scheme}{energy\ or\ power\ by\ GPScheDVS})\times 100\ \ (\%) \qquad \text{(Equation 6.11)}$$

On the other hand, the improvement in deadline meet ratios is the difference between the deadline meet ratios achieved by GPScheDVSs and existing DVS schemes.

Figure 6.20 shows that, under the usage scenario avi_m_gcc_u17 where the mplayer runs alone, all DVS schemes did not make a noticeable difference because all of them keep the CPU operating point to the lowest for most of time. This was because the lowest CPU operating point is enough to meet most deadlines in transacting video frames and audio data.

Under the heavy load situations where the gcc compilation workload tasks the CPU time left by the mplayer, all the schemes kept CPU operating point to the highest for most of time as there is insufficient CPU idle time to reduce the CPU operating point. Thus, also under the heavy load situations, there was no noticeable difference in system energy consumption, CPU energy consumption, and the average power consumption. However, in the video and audio deadline meet ratios, the time constraint-based task scheduling nature of the GPSched gives GPScheDVS a great improvement over the autonomous DVS schemes.

The advantage of GPSched in making CMFSs uniform and close to the ICMFS so as to allow GPScheDVS to keep the CPU speed at a low level for a longer time is observed most clearly under moderate load situations. Although there still is a sufficient CPU idle time under moderate load situations, the existing DVS schemes fail to utilize the idle time because of the unnecessarily high CMFSs of tasks. Unlike the existing DVS schemes, GPScheDVS better utilizes the CPU idle time as a result of GPSched.

The GPScheDVS-CP showed a similar pattern to the GPScheDVS-SE for light load and

medium load situations as can be seen in Figure 6.21. Under heavy load situations, however, the GPScheDVS-CP achieves even a larger improvement in the average CPU power consumption over the autonomous DVS schemes as it keeps running all NRT tasks at the lowest CPU operating point unlike the autonomous DVS schemes. For the same reason, the GPScheDVS-CP yields a better improvement in CPU energy consumption than the GPScheDVS-SE. But the system energy consumption increases as the GPScheDVS-CP keeps running all NRT tasks at the lowest CPU operating point even though there is no more CPU idle time under the heavy load situations; the system running time is lengthened and the energy consumptions of the non-CPU components increase overwhelming the CPU energy savings. However, as pointed out earlier, these results exactly meet the purpose of the CP mode operation of GPScheDVS.

Figure 6.22 shows the overhead of each DVS scheme in time, system energy, and CPU operating point transition numbers. As GPScheDVS performs not only the CPU speed control but task scheduling, it has a larger overhead than other DVS schemes in time and system energy consumption. Note that, however, the improvements in system energy consumption and video/audio deadline meet ratios shown in Figure 6.15, 6.20, and 6.21 already included this overhead in them. As for CPU operating point transition numbers, GPScheDVS showed a substantial improvement, i.e., a less number of CPU operating point transitions, over the existing task-based autonomous DVS scheme. Unlike interval-based schemes, task-based schemes tend to cause a frequent CPU operating point transitions because they treat tasks individually. Although GPScheDVS also uses a task-based DVS scheme called GPSDVS, the CPU operating point transition number reduced. This is the natural result of GPSched which makes tasks' CMFSs more uniform reducing the need for CPU operating point transitions.

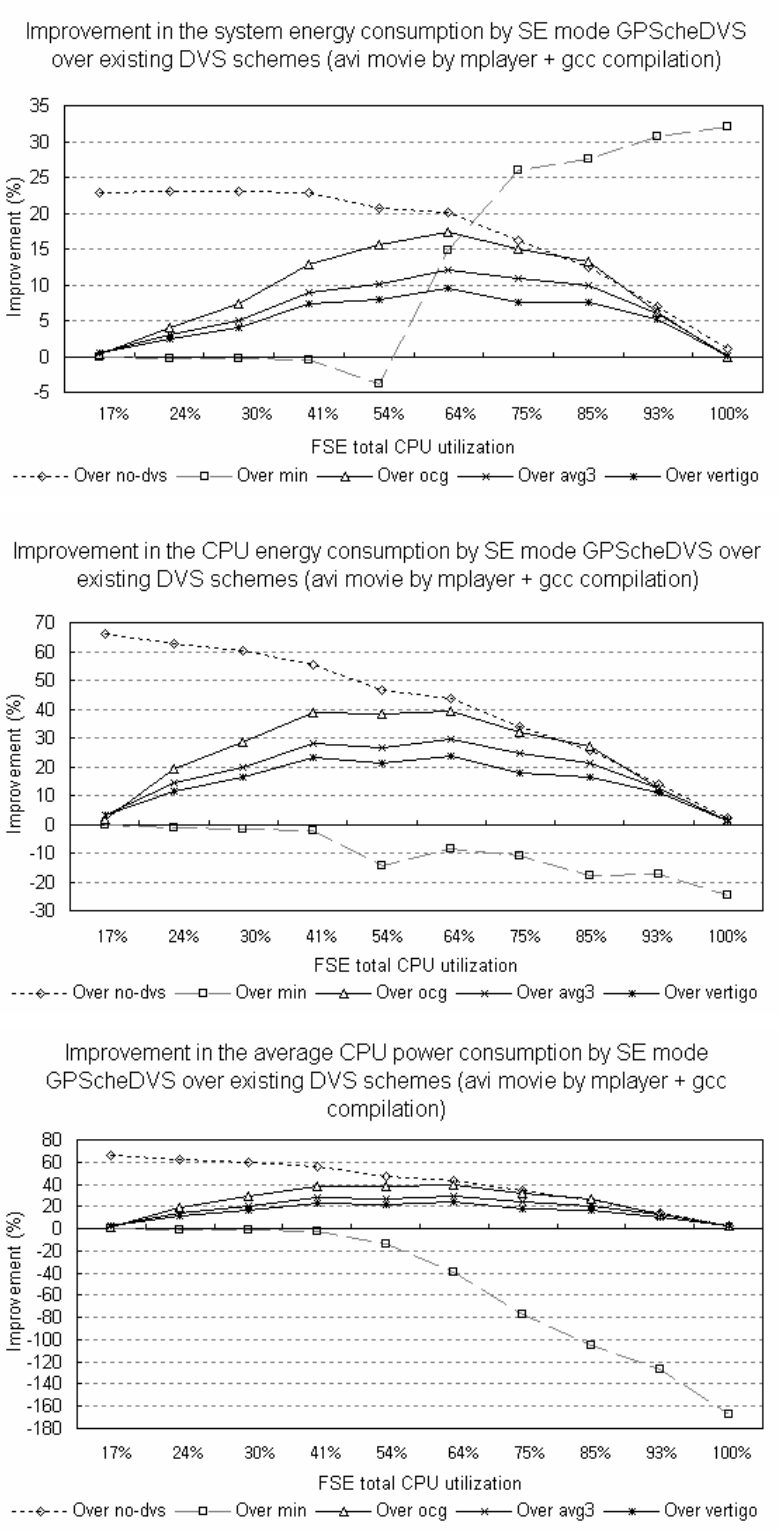


Figure 6.20 Improvements in energy, power, and QoS by the GPScheDVS-SE over the existing DVS schemes under the usage scenarios avi_m_gcc_u[n]

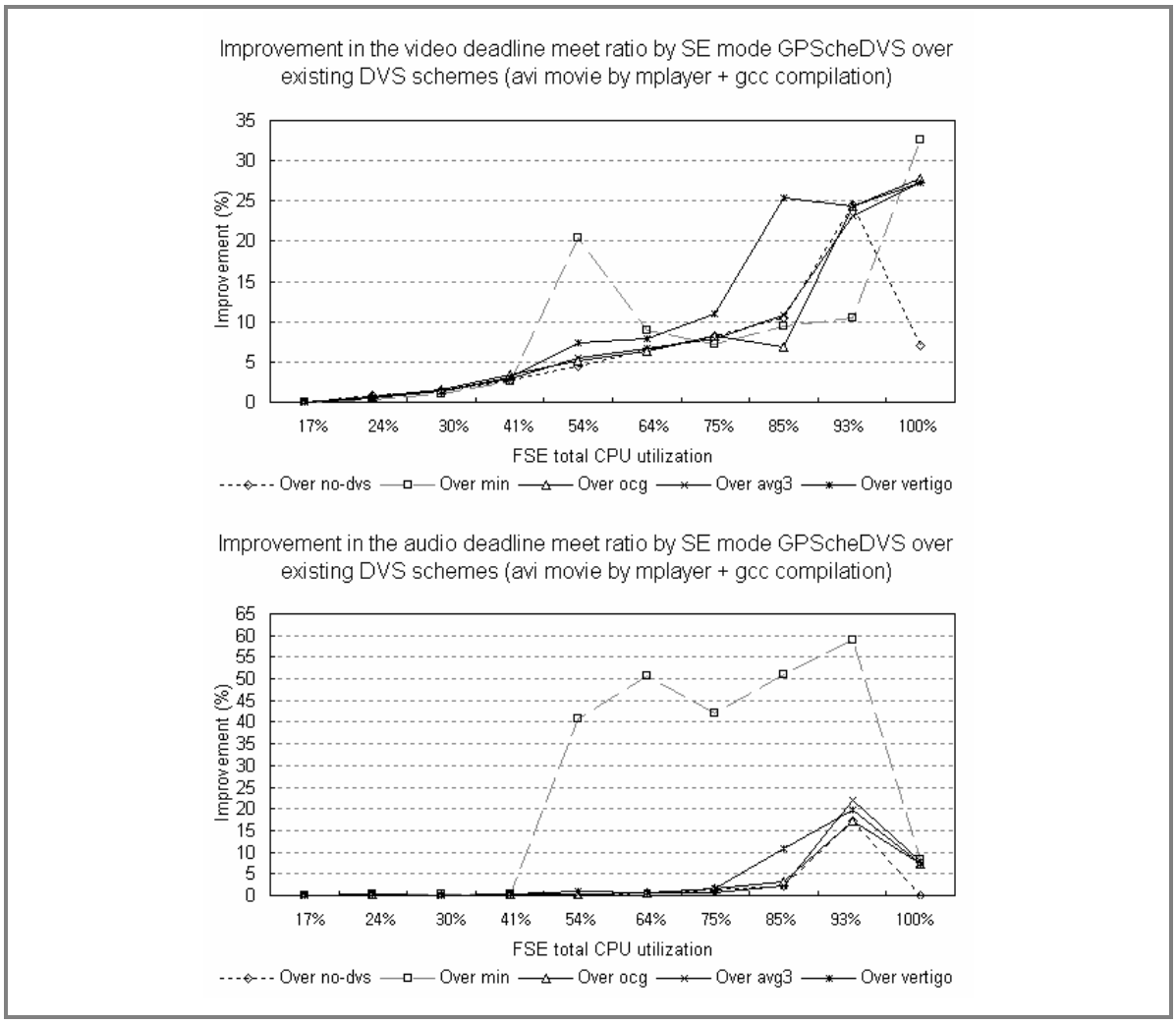


Figure 6.20 Improvements in energy, power, and QoS by the GPScheDVS-SE over the existing DVS schemes under the usage scenarios avi_m_gcc_u[n] (Cont'd)

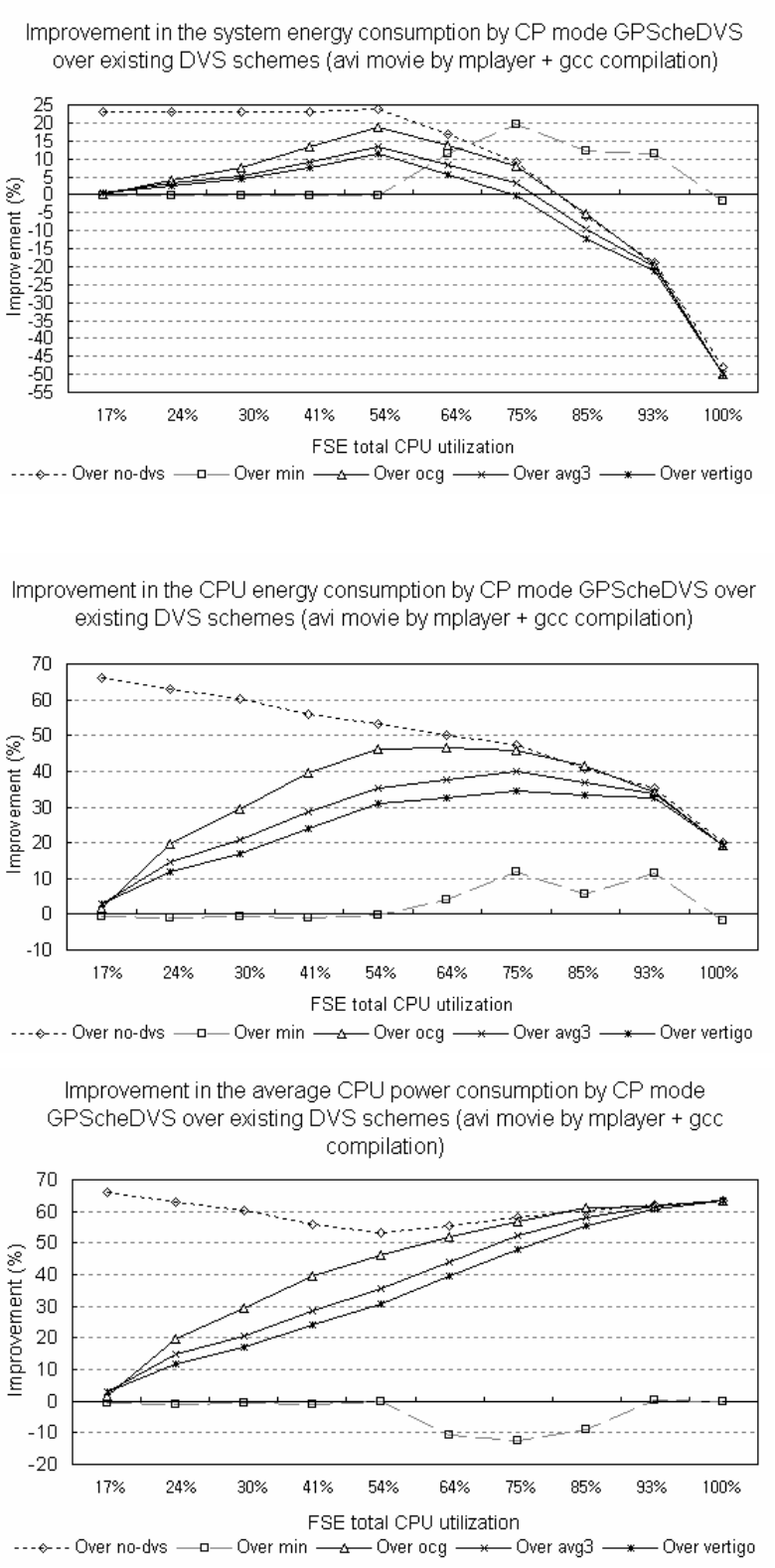
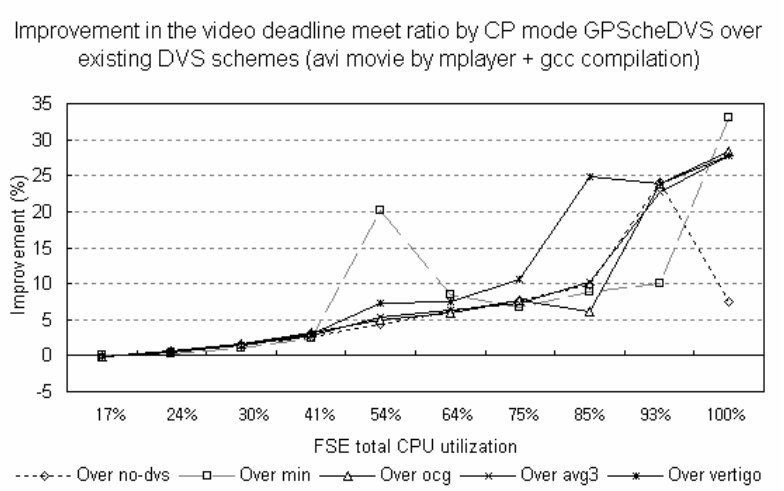
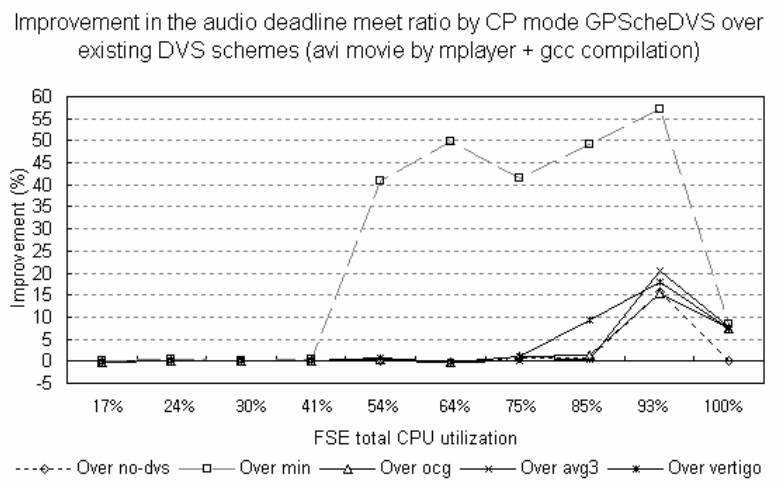


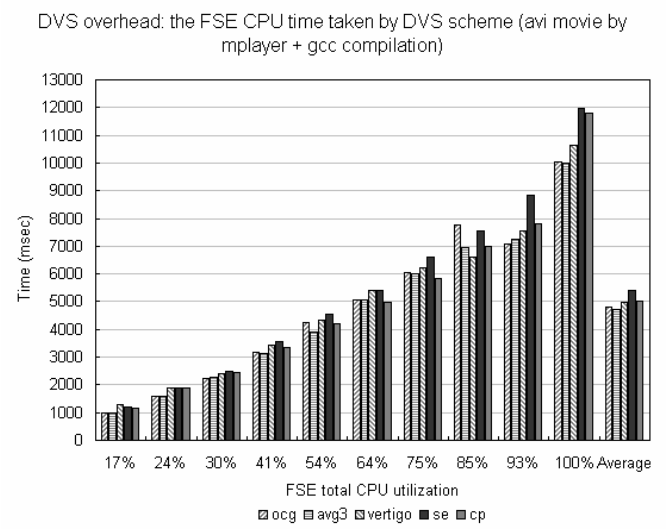
Figure 6.21 Improvements in energy, power, and QoS by the GPScheDVS-CP over the existing DVS schemes under the usage scenarios avi_m_gcc_u[n]

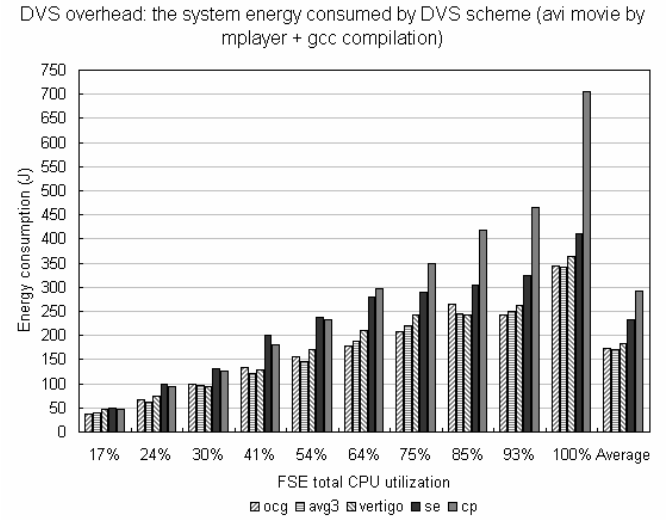




(e) Improvement in audio deadline meet ratio

Figure 6.21 Improvements in energy, power, and QoS by the GPScheDVS-CP over the existing DVS schemes under the usage scenarios avi_m_gcc_u[n] (Cont'd)





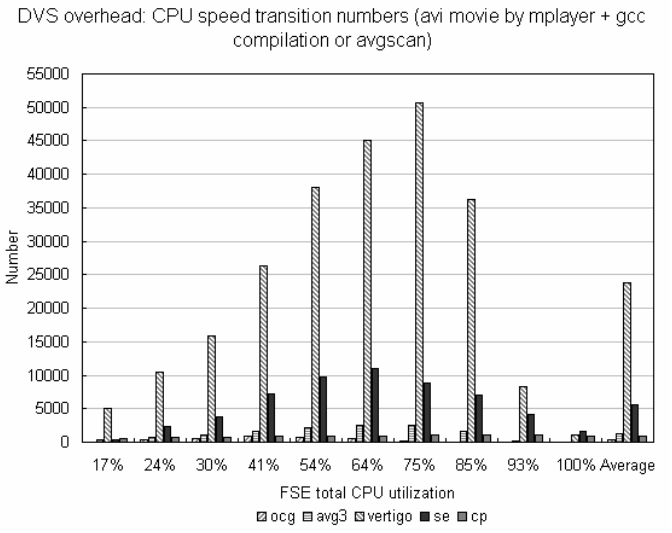


Figure 6.22 Overheads of each DVS scheme under the usage scenarios avi_m_gcc_u[n]

As pointed out earlier, both single-threaded applications and multi-threaded applications were experimented in this research. Figure 6.23 and 6.24 show the improvement achieved by the GPScheDVS-SE and the GPScheDVS-CP, respectively, over the existing DVS schemes under the usage scenarios where a multi-threaded continuous media application aviplay is used to play an AVI movie with or without having NRT workload in the background. As can be seen in the x-axis of Figure 6.23 and 6.24, aviplay imposes a much larger amount of workload than mplayer because it consists of many threads unlike the mplayer that has only one thread. This property of aviplay makes it create a moderate (not light) load situation even when it runs alone.

An interesting result is that GPScheDVS improves energy and power over the existing DVS schemes even under the usage scenario avi_a_gcc_u40 where aviplay is executed alone without having any background NRT workload. The trends of the improvements achieved by GPScheDVS in energy and power consumptions shown in Figure 6.23 and 6.24 look like the right-hand-side half of the trends shown in Figure 6.20 and 6.21. Note that, however, the moderate workloads shown in Figure 6.20 and 6.21 are made up of both SRT tasks and NRT tasks whereas the moderate workload imposed under the usage scenario avi_a_gcc_u40 is made up solely of SRT tasks – the aviplay tasks. As there is no NRT task affecting the CMFSs of aviplay tasks in this case, both the native Linux task scheduler and GPSched yield similar CMFSs which are uniform and low as Figure 6.25 shows.

Nevertheless, the existing autonomous DVS schemes run CPU at higher speeds than GPScheDVS as Figure 6.26 shows. Figure 6.27 shows the correspondingly high system power consumption of the experimental platform measured by the oscilloscope. This is because the existing autonomous DVS schemes do not share the information about tasks with GPSched. The existing schemes are unable to distinguish the aviplay tasks from other NRT tasks. On the other hand, the workload arrival pattern of a multi-thread application is busty. As a consequence, the existing schemes raise up the CPU speed upon the busty workload irrespective of the task type and the corresponding timeliness requirements. Unlike the existing DVS schemes, GPScheDVS shares the information about task types and, thus, tasks' timeliness requirements. Based on this information,

GPScheDVS keeps the CPU speed at the ICMFS unless a group of aviplay threads executes beyond the timeout threshold of GPScheDVS for DBSRT tasks.

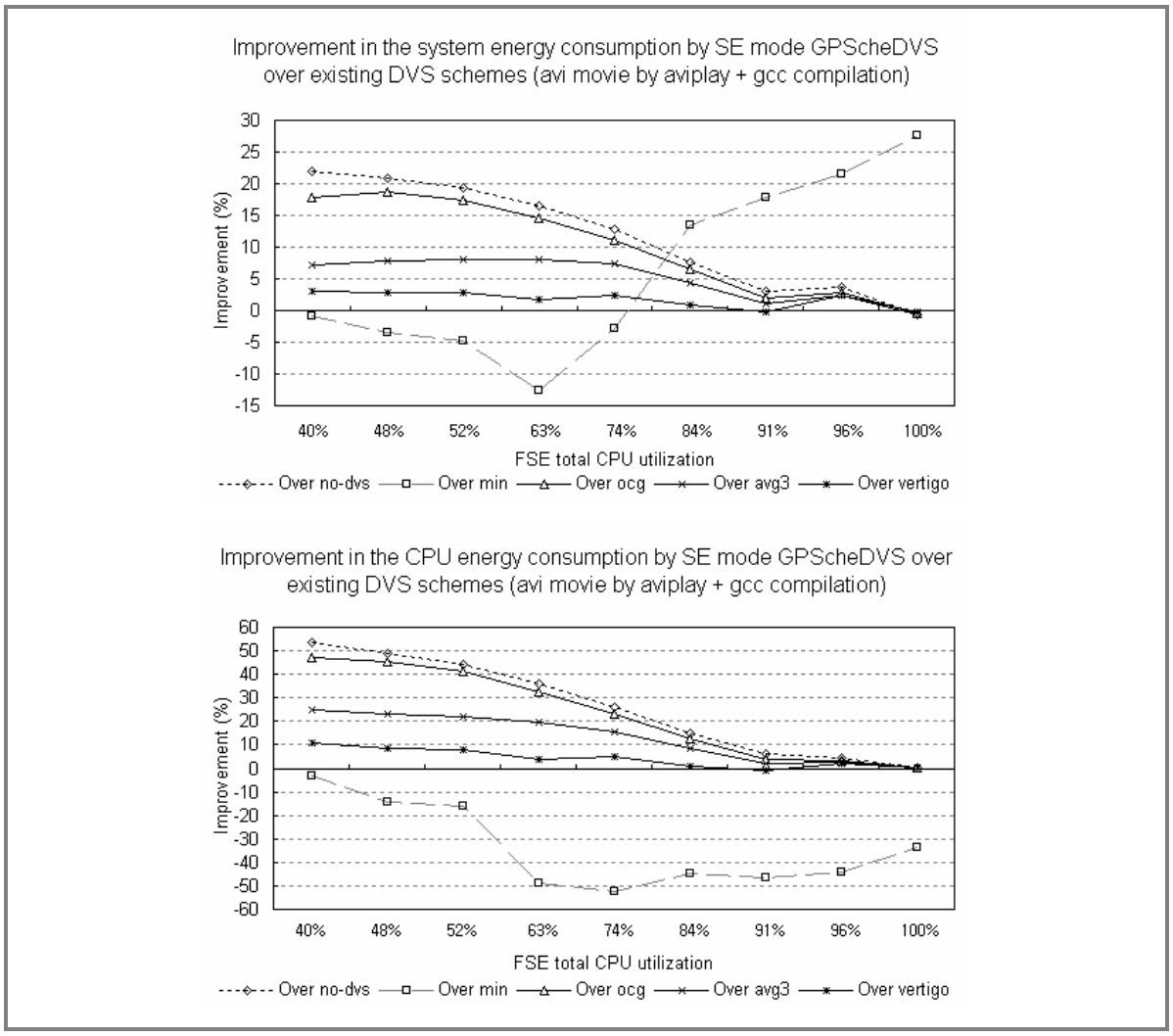


Figure 6.23 Improvements in energy, power, and QoS by the GPScheDVS-SE over the existing DVS schemes under the usage scenarios avi_a_gcc_u[n]

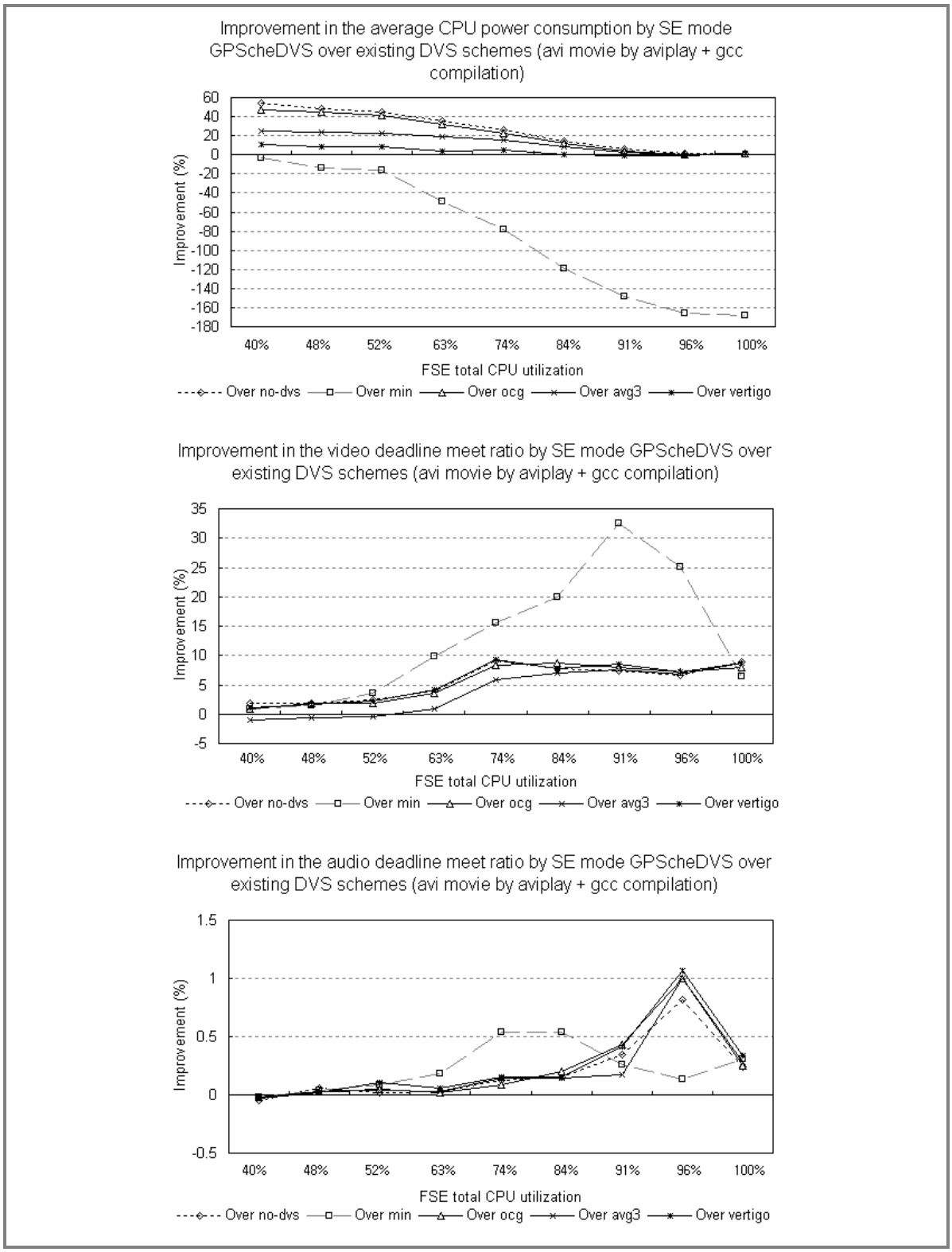


Figure 6.23 Improvements in energy, power, and QoS by the GPScheDVS-SE over the existing DVS schemes under the usage scenarios avi_a_gcc_u[n] (Cont'd)

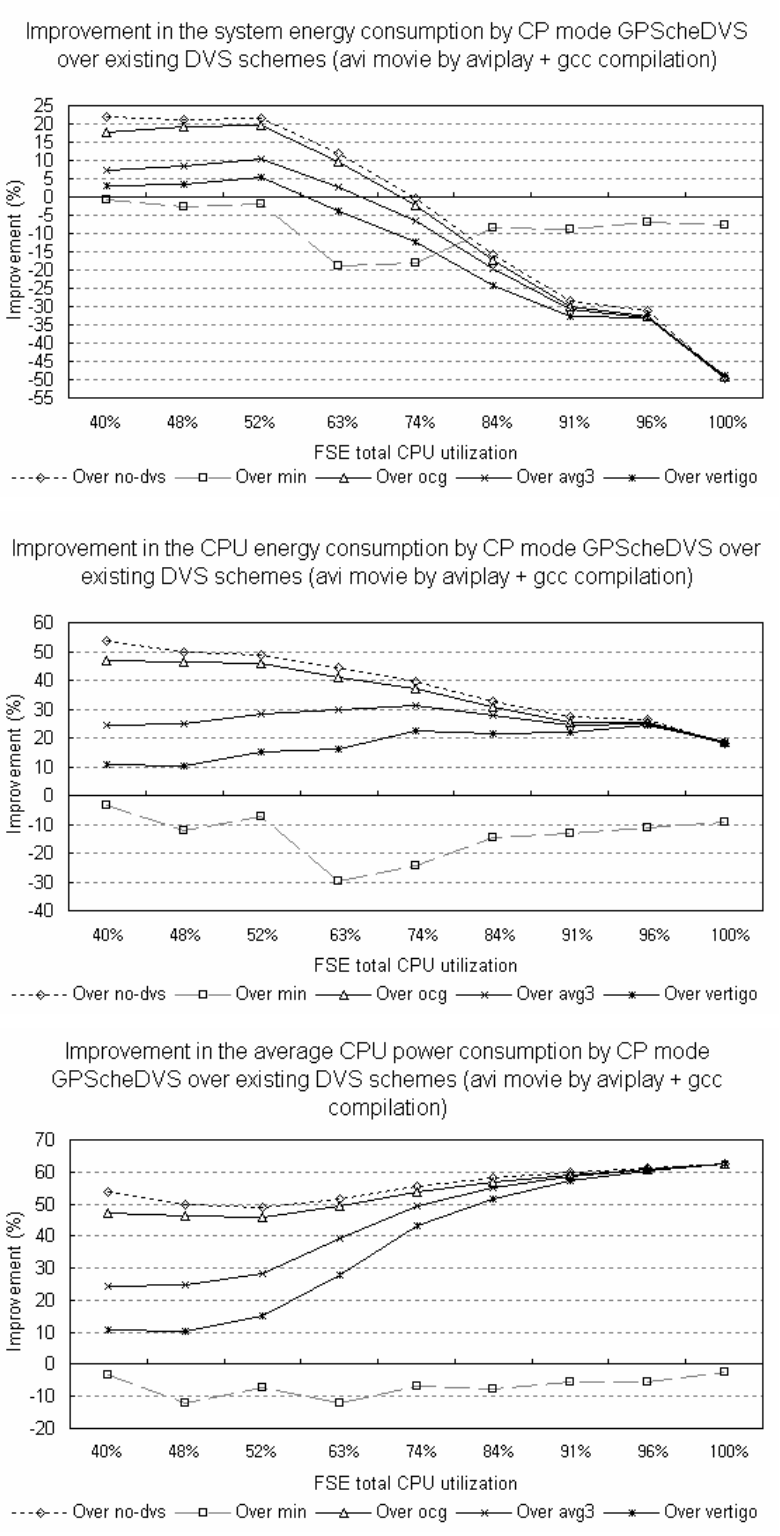
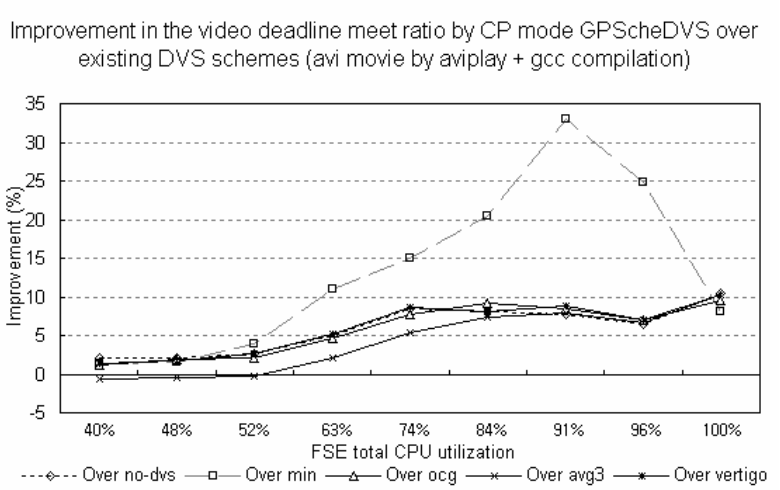


Figure 6.24 Improvements in energy, power, and QoS by the GPScheDVS-CP over the existing DVS schemes under the usage scenarios avi_a_gcc_u[n]



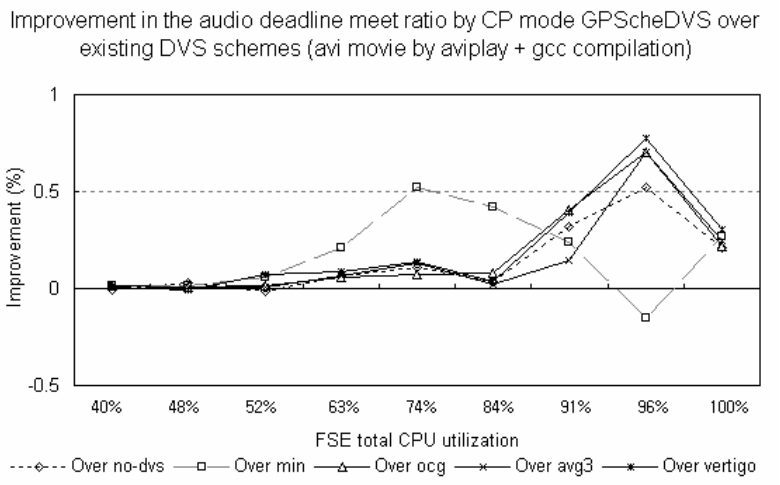
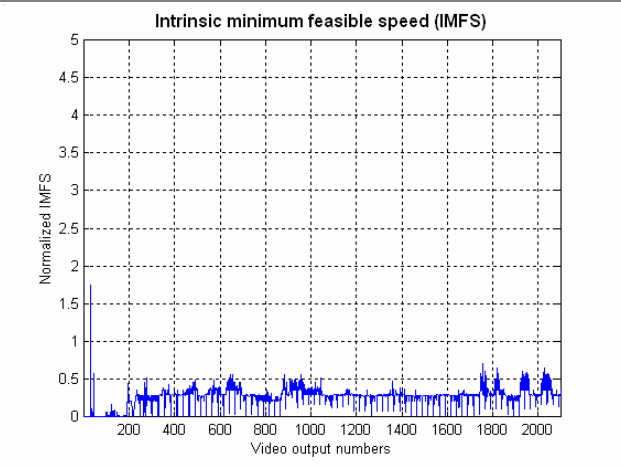
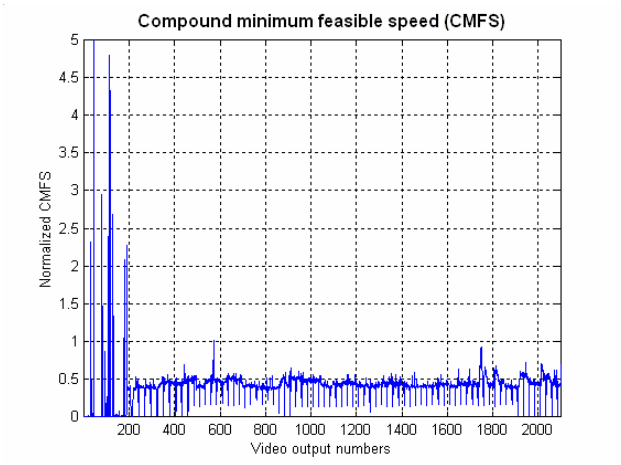


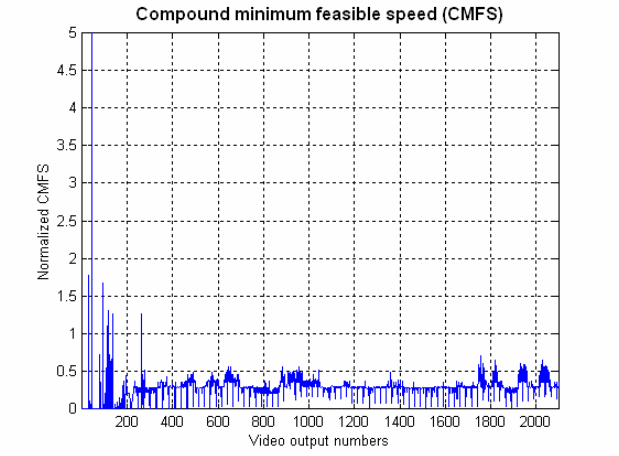
Figure 6.24 Improvements in energy, power, and QoS by the GPScheDVS-CP over the existing DVS schemes under the usage scenarios avi_a_gcc_u[n] (Cont'd)



(a) IMFS of aviplay



(b) CMFS of aviplay under the native Linux task scheduling



(c) CMFS of aviplay by GPSched

Figure 6.25 aviplay IMFS and the CMFS by the native Linux scheduler and GPSched under the usage scenarios avi_a_gcc_u40 (aviplay alone)

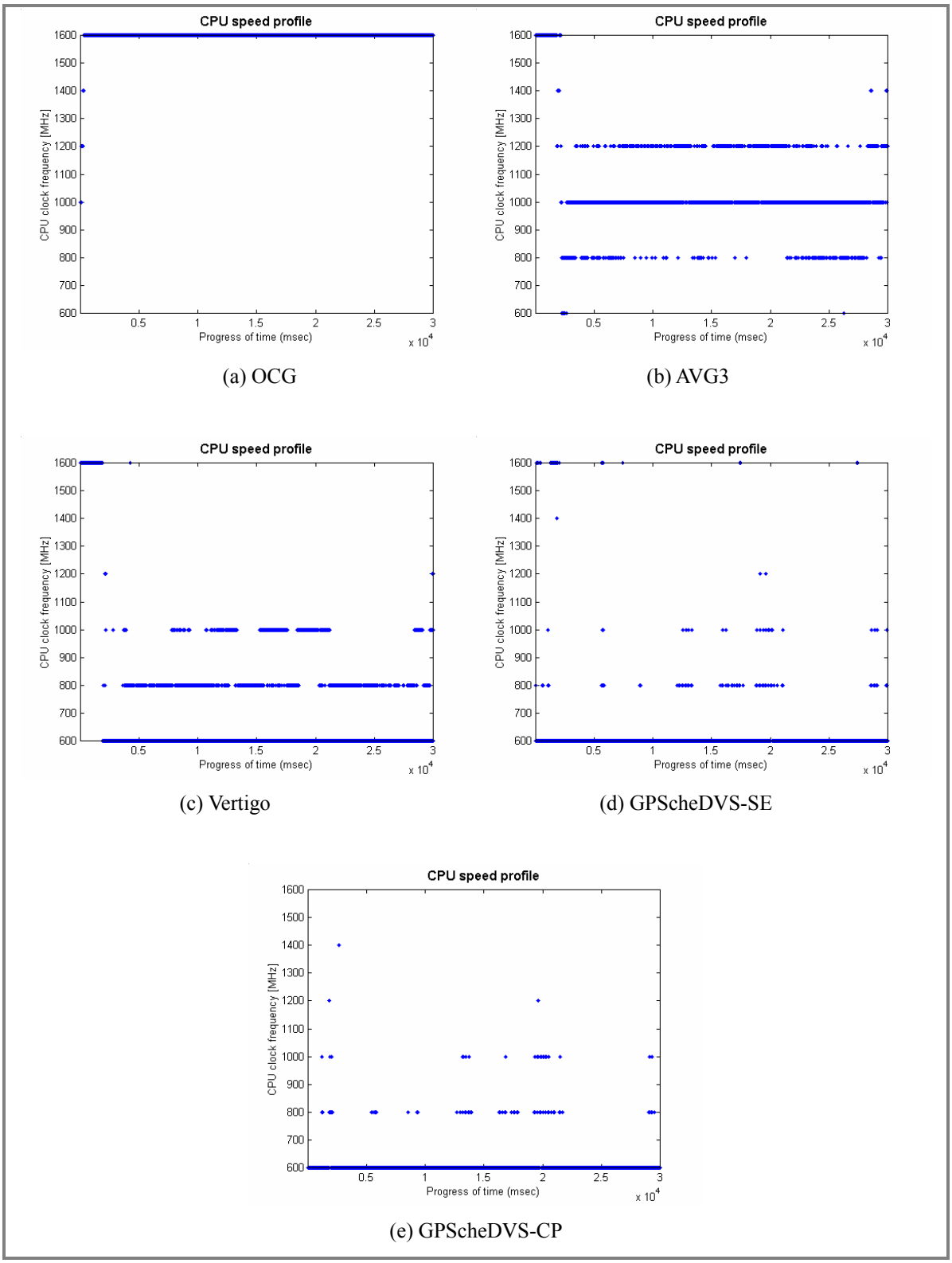
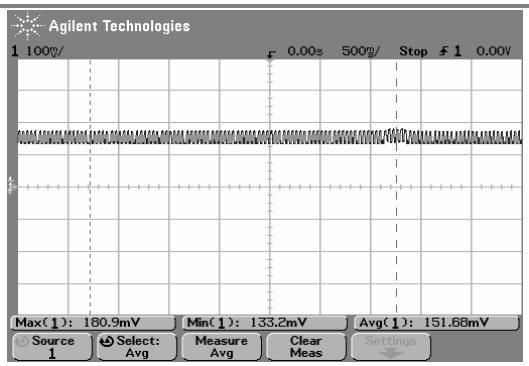
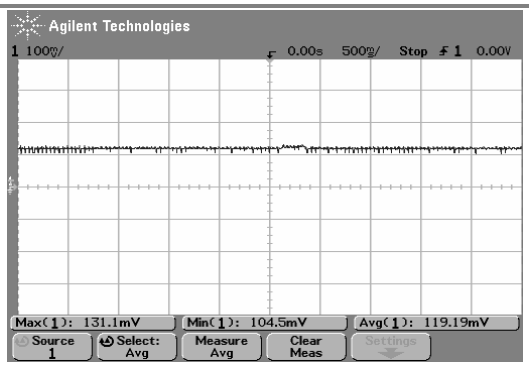


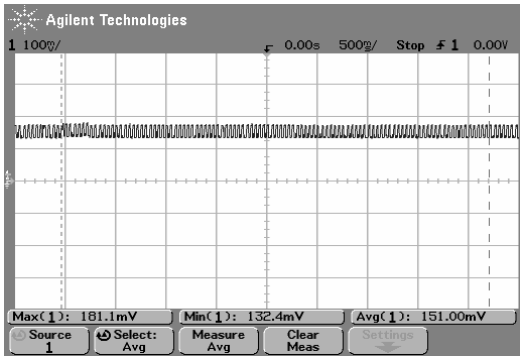
Figure 6.26 CPU speed pattern with OCG, AVG3, Vertigo, and GPScheDVS for 30 seconds under the usage scenarios avi_a_gcc_u40 (aviplay alone)



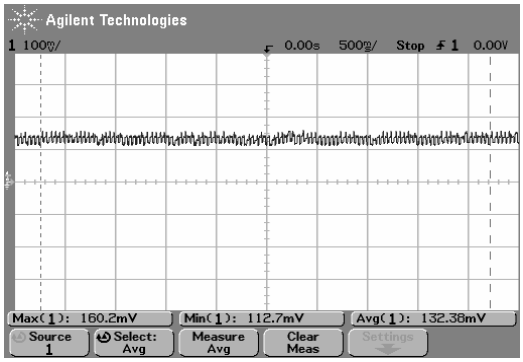
(a) No-DVS



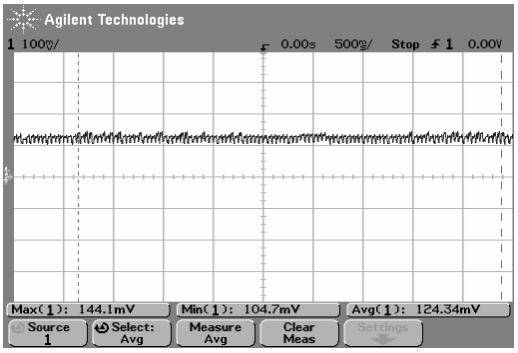
(b) Const-min



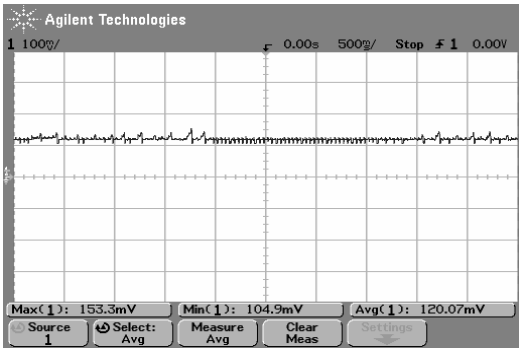
(c) OCG



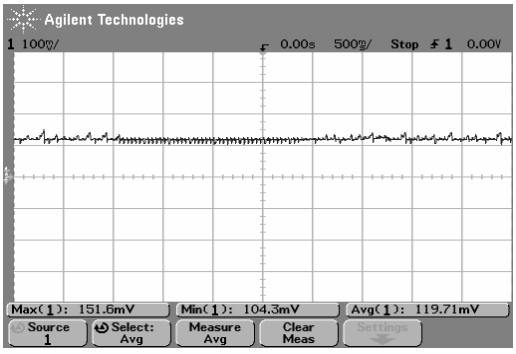
(d) AVG3



(e) Vertigo



(f) GPScheDVS-SE



(g) GPScheDVS-CP

Figure 6.27 System power consumption pattern by each DVS scheme under the usage scenario avi_a_gcc_u40 (aviplay alone)

6.3.1.2 Interactive applications

This section presents the experimental results obtained under another typical type of usage scenarios in which an interactive application is used with or without having NRT workload in the background. The three usage scenarios belonging to this type were described in Table 6.7. Among the three usage scenarios, this section focuses on the usage scenario mozilla_gcc_u[n] in which the mozilla web browser is run with or without the background NRT workload imposed by gcc compilation.

Figure 6.28 (a), (b), (c), (d), and (e) shows the total completion time, system energy consumption, CPU energy consumption, average CPU power consumption, and the interactive deadline meet ration which means the ratio of user input events that got response from the system within the 50 milliseconds of human perception threshold. And Figure 6.30 shows the improvements in energy and power achieved by GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes. The general trends of the results shown in Figure 6.28 and 6.30 are similar to the trends shown in Figure 6.15, 6.20, and 6.21 observed under the typical usage scenarios with a continuous media application and background workload. The only difference in the trends is that both the existing DVS schemes and GPScheDVS result in a similar level of interactive deadline meet ratios. This is because an interactive application is assigned with a high priority under the native Linux task scheduling and GPSched inherits this property as explained in Chapter 4.

Figure 6.29 and 6.31 show the fraction of total execution time spent at each CPU operating point and the overhead in time, system energy consumption, and the CPU operating point transition numbers, respectively, in the same way as Figure 6.18 and 6.22. The results shown in Figure 6.29 and 6.31 are also similar to Figure 6.18 and 6.22; GPScheDVS is able to keep the CPU speed at a lower level for a longer time than the existing DVS schemes but has a bit larger overhead because of the burden of task scheduling.

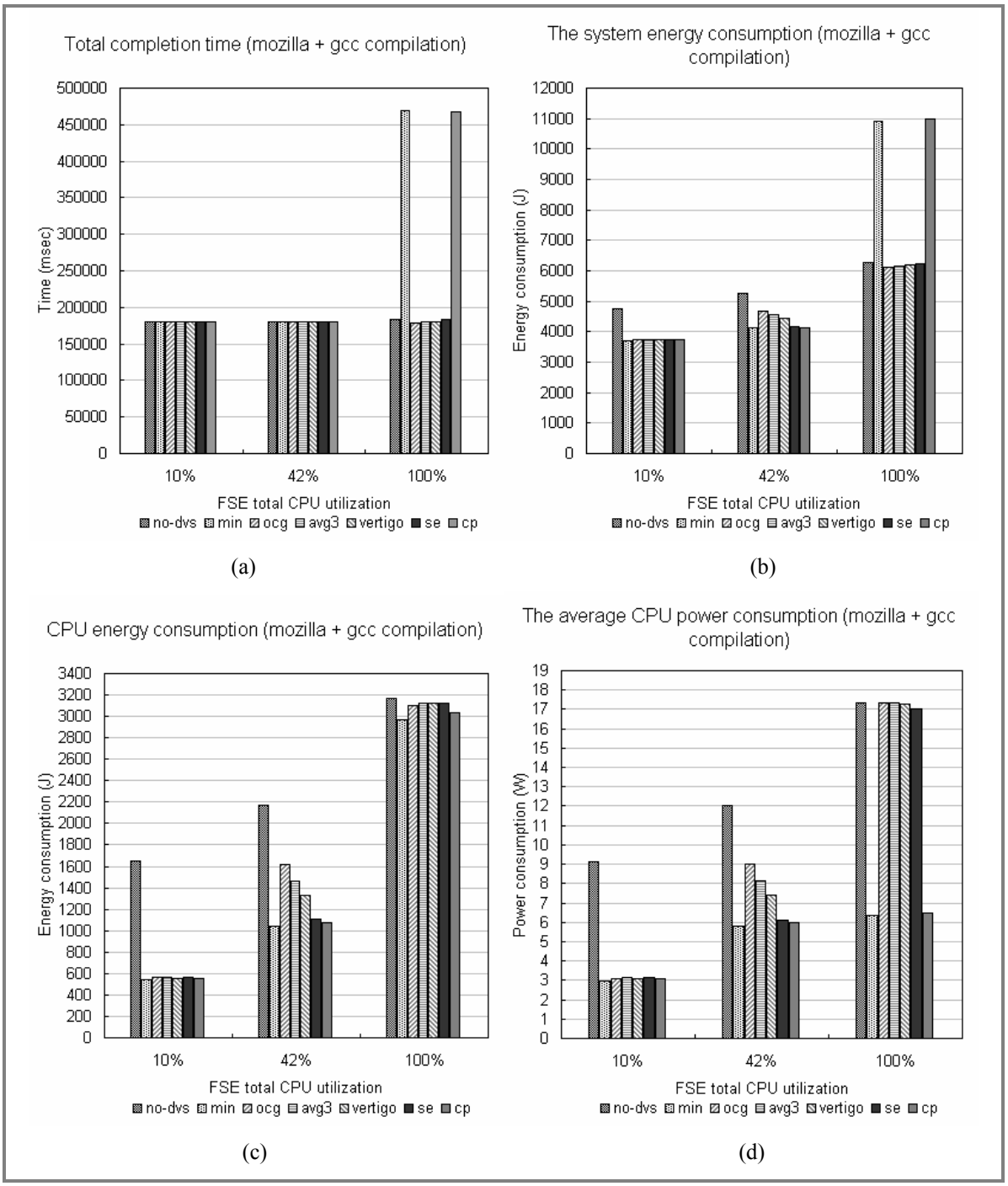
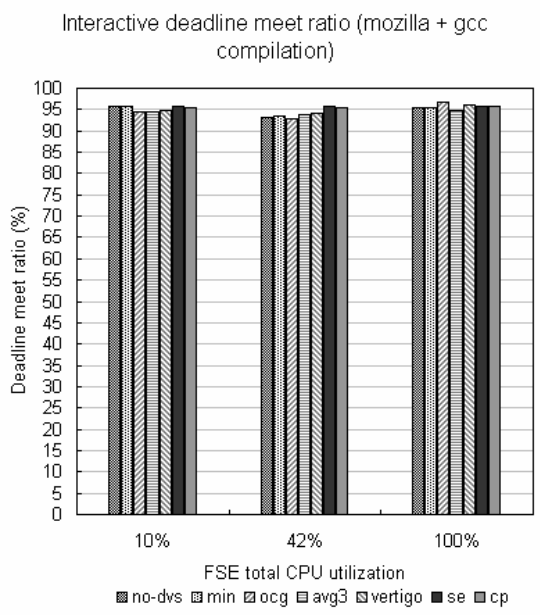


Figure 6.28 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios mozilla_gcc_u[n]. No gcc compilation was executed under the usage scenario mozilla_gcc_u10



(e)

Figure 6.28 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios mozilla_gcc_u[n]. No gcc compilation was executed under the usage scenario mozilla_gcc_u10 (Cont'd)

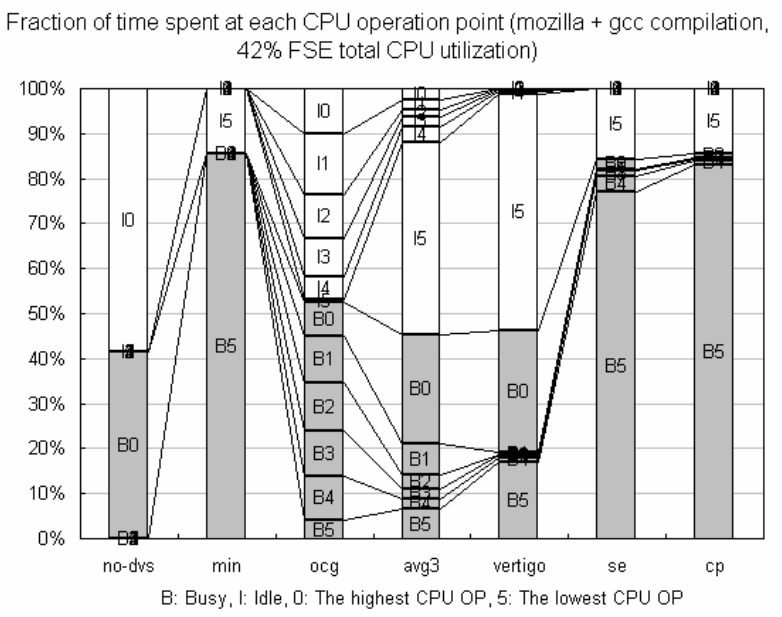
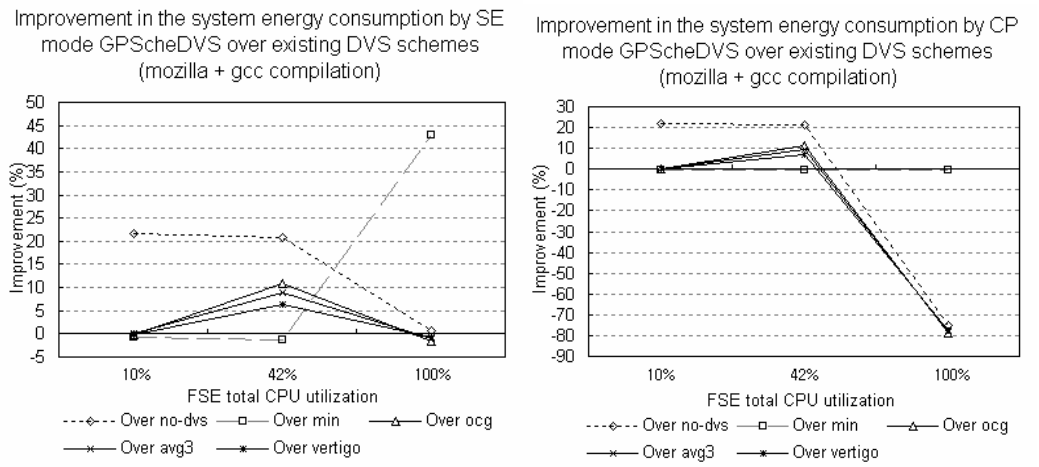
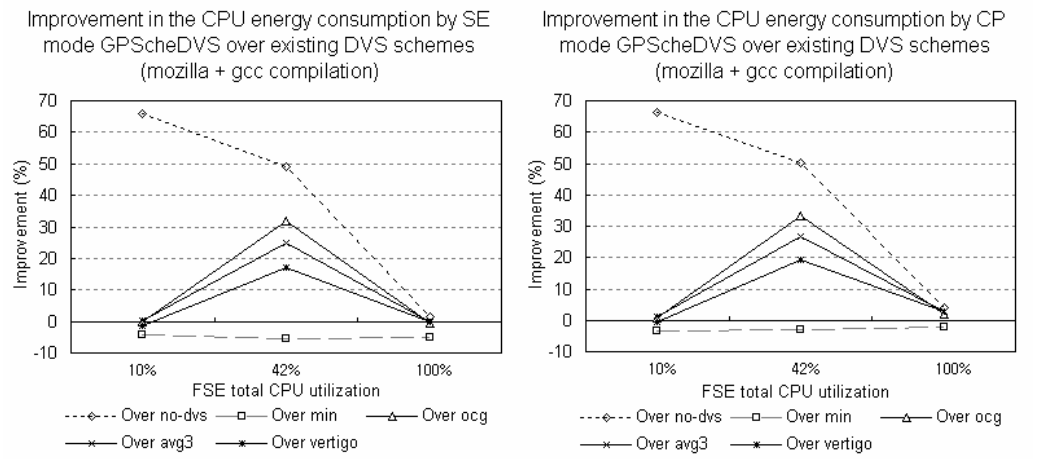


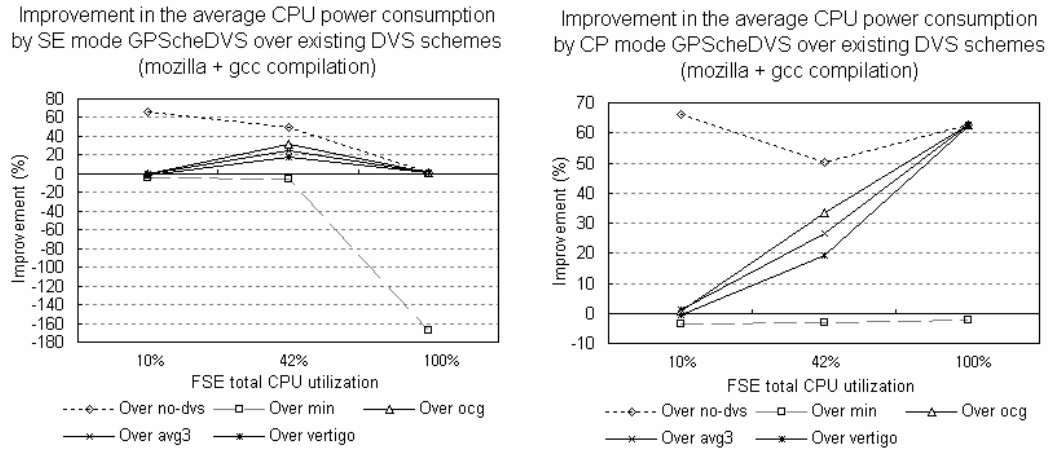
Figure 6.29 Fractions of time spent at each CPU operating point with each DVS scheme under the usage scenarios mozilla_gcc_u42



(a) Improvement in system energy consumption



(a) Improvement in CPU energy consumption



(a) Improvement in average CPU power consumption

Figure 6.30 Improvements in energy, power, and QoS by the GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes under the usage scenarios mozilla_gcc_u[n]

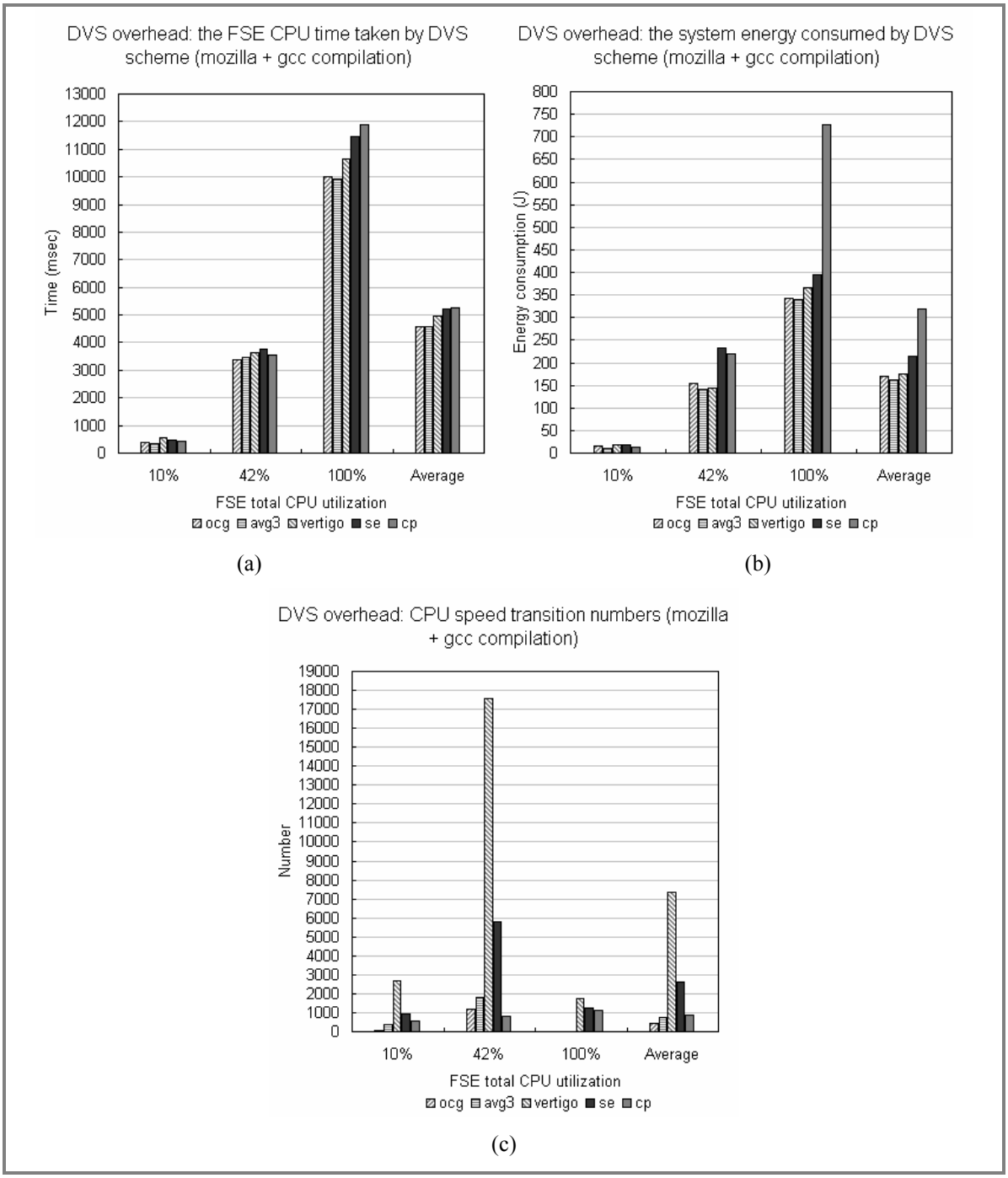


Figure 6.31 Overheads of each DVS scheme under the usage scenarios mozilla_gcc_u[n]

6.3.1.3 Summary

This section provides the summary of all experimental results obtained under the typical usage scenarios. Table 6.21 and 6.22 provide the best, worst, and average improvements in each metric achieved by GPScheDVS-SE and GPScheDVS-CP, respectively, over the existing DVS schemes. In the tables, an improvement in the QoS of time constrained applications simply means the difference between the deadline meet ratios achieved by GPScheDVS and an existing DVS scheme; for example, if the respective deadline meet ratios achieved with GPScheDVS and the existing DVS scheme are 100% and 90%, the improvement by GPScheDVS is reported as 10%. The improvements in energy and power metrics are calculated in a different way; for instance, an improvement of 30% means that GPScheDVS consumes only the 70% of the energy or power that was consumed by the existing DVS scheme in completing a same given amount of workload. The target metrics of each mode of GPScheDVS are expressed with bald italic characters.

A notable point in Table 6.22 is that GPScheDVS-CP results in rather increased system energy consumption than other DVS schemes. As explained earlier, this is the correct result meeting the design goal of GPScheDVS which is focusing on the reduction of average CPU power consumption at the cost of system energy but while preserving the QoS of SRT applications.

Table 6.21 Improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes under the typical usage scenarios

Usage scenario	Metric (See note 1)	Case (See note 2)	Improvement (%) over. (See note 3)			
			No DVS	OCG	AVG3	Vertigo
A continuous media application + NRT applications	***System energy consumption***	Best	23.0	20.6	14.8	11.0
		Worst	-0.9	-0.8	-0.8	-0.3
		Avg.	***17.0***	***9.2***	***7.0***	***4.9***
	CPU energy consumption	Best	68.7	47.1	37.4	26.5
		Worst	0.1	0.4	0.2	-0.7
		Avg.	***42.3***	***23.9***	***19.1***	***13.5***
	Average CPU power consumption	Best	68.7	47.1	37.4	26.5
		Worst	1.4	0.4	0.6	-0.7
		Avg.	***42.3***	***23.9***	***19.2***	***13.5***
	Video deadline meet ratio	Best	26.7	27.8	27.3	27.2
		Worst	-1.7	0.0	-1.0	-0.1
		Avg.	***6.9***	***7.8***	***7.6***	***9.5***
	Audio deadline meet ratio	Best	17.3	17.0	22.0	19.8
		Worst	-0.1	-0.1	-0.1	-0.1
		Avg.	***0.8***	***1.0***	***1.1***	***1.3***
An interactive application + NRT applications	***System energy consumption***	Best	21.7	11.0	8.8	6.9
		Worst	-1.5	-1.6	-1.4	-0.7
		Avg.	***13.8***	***2.8***	***2.4***	***2.1***
	CPU energy consumption	Best	68.5	31.8	26.0	20.1
		Worst	-0.4	-1.8	-0.3	-1.3
		Avg.	***39.3***	***9.9***	***8.5***	***6.6***
	Average CPU power consumption	Best	68.5	31.8	25.9	20.1
		Worst	1.8	-1.8	0.0	-1.3
		Avg.	***40.0***	***10.7***	***9.2***	***7.0***
	Interactive deadline meet ratio	Best	2.4	2.8	1.8	1.5
		Worst	-0.6	-1.1	-0.5	-0.3
		Avg.	***0.1***	***0.3***	***0.4***	***0.3***

Note 1—The relevant metrics to the GPScheDVS-SE are the system energy consumption and the QoS of time constrained applications. The GPScheDVS-SE aims at a longer the battery life-time without hurting user experiences on the time constrained applications.
Note 2— As for the QoS of time constrained applications, the best case improvements were achieved under the heavy load situations, whereas the worst case improvements were yielded under the light load situations. On the other hand, the best case improvements in the energy and power metrics were achieved under the moderate load situations, and the worst case improvements were yielded under either the lightest or the full load situations. The improvements were calculated over all the experimental results obtained under the light, moderate, and heavy load situations.
Note 3—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

Table 6.22 Improvements achieved by the GPScheDVS-CP over the existing autonomous DVS schemes under the typical usage scenarios

Usage scenario	Metric (See note 1)	Case (See note 2)	Improvement (%) over. (See note 3)			
			No DVS	OCG	AVG3	Vertigo
A continuous media application + NRT applications	System energy consumption	Best	23.7	23.1	14.9	12.2
		Worst	-57.7	-57.3	-57.2	-56.9
		Avg.	***4.7***	***-3.1***	***-5.4***	***-7.8***
	CPU energy consumption	Best	68.7	52.5	40.0	34.5
		Worst	15.0	0.8	0.8	0.8
		Avg.	***47.8***	***29.4***	***25.0***	***19.7***
	Average CPU power consumption	Best	68.7	63.3	63.4	63.3
		Worst	48.8	-1.4	0.2	-0.4
		Avg.	***59.5***	***38.7***	***35.3***	***31.2***
	Video deadline meet ratio	Best	27.5	28.4	27.8	27.7
		Worst	-1.5	-0.1	-0.7	0.0
		Avg.	***6.9***	***7.8***	***7.6***	***9.5***
	Audio deadline meet ratio	Best	15.6	15.3	20.3	18.0
		Worst	-0.4	-0.4	-1.1	-0.4
		Avg.	***0.6***	***0.8***	***0.8***	***1.1***
An interactive application + NRT applications	System energy consumption	Best	21.8	11.6	9.4	7.2
		Worst	-75.3	-79.0	-77.9	-77.6
		Avg.	***-8.4***	***-19.5***	***-19.8***	***-19.9***
	CPU energy consumption	Best	68.6	33.5	27.0	21.4
		Worst	4.2	-1.4	0.2	-0.4
		Avg.	***42.6***	***13.4***	***12.0***	***10.2***
	Average CPU power consumption	Best	68.6	63.3	63.3	63.3
		Worst	50.4	-1.4	0.2	-0.4
		Avg.	***60.6***	***31.6***	***30.1***	***28.0***
	Interactive deadline meet ratio	Best	2.1	2.4	1.4	1.1
		Worst	-0.6	-1.0	-0.3	-0.3
		Avg.	***0.2***	***0.3***	***0.4***	***0.4***

Note 1—The relevant metrics to the GPScheDVS-CP are the average CPU power consumption and the QoS of time constrained applications. The GPScheDVS-CP aims at the reduced CPU temperature without hurting user experiences on the time constrained applications.
Note 2— As for the QoS of time constrained applications, the best case improvements were achieved under the heavy load situations, whereas the worst case improvements were yielded under the light load situations. On the other hand, the best case improvements in the energy and power metrics were achieved under the moderate load situations, and the worst case improvements were yielded under either the lightest or the full load situations. The improvements were calculated over all the experimental results obtained under the light, moderate, and heavy load situations.
Note 3—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

6.3.2 Special usage scenarios with a hard-idling NRT application

This section presents the experimental results achieved under a special type of usage scenarios where a SRT application is used while running a hard-idling NRT (HINRT)

application in the background. The focus in these experiments is how the HINRT treatment technique of GPScheDVS-SE is effective in preserving the total execution time such that the energy consumption of non-CPU components are prevented from increasing and, accordingly, the system energy consumption is NOT increased due to DVS. As explained in Section 6.2.3, an anti-virus scanner avgscan was used as the HINRT application.

Figure 6.32 shows the total completion time, system energy consumption, CPU energy consumption, average CPU power consumption, and the video and audio deadline meet ratios achieved by each DVS scheme under the usage scenario avi_m_avgscan described in Table 6.8 in Section 6.2.3. Figure 6.32 (a) shows that GPScheDVS-SE and the two interval-based DVS schemes, OCG and AVG3, are good in preserving the total execution time while the task-based DVS scheme, Vertigo, is not. The existing interval-based schemes are typically overcautious to the temporary increase of workload because they cannot accurately predict the individual tasks' timeliness requirements. As can be seen in Figure 6.32, the total CPU utilization under this usage scenario is up to 71%. As a result, OCG and AVG3 keep the CPU speed at the highest level throughout the experiment and results in almost the same amount of system energy consumption, CPU energy consumption, and average CPU power consumption as Figure 6.32 (b), (c), and (d) show.

Unlike these interval-based DVS schemes, GPScheDVS-SE is able to detect the HINRT tasks by observing which type of schedule function each task calls. Once GPScheDVS-SE detects that any HINRT task is in the run queue (i.e., either running or waiting the completion of the currently running task), it sets the CPU speed to the highest such that the total execution time and, accordingly, the energy consumption of non-CPU components does not increase. This is the reason for the higher CPU energy consumption and average CPU power consumption of GPScheDVS-SE than that of Vertigo as Figure 6.32 (c) and (d) show. However, as can be seen in Figure 6.32 (b), GPScheDVS-SE results in a less system energy consumption than Vertigo – the correct result that meets the design goal of GPScheDVS-SE. Moreover, the system energy consumption with GPScheDVS-SE is even less than the No-DVS policy and the existing interval-based DVS schemes. This is because GPScheDVS-SE performs the normal operations, i.e.,

setting the CPU speed according to tasks' CPU speed requirements, if there is no HINRT task in the run queue reducing the CPU energy consumption as Figure 6.32 (c) shows.

Figure 6.32 (e) shows that the video deadline meet ratio with GPScheDVS-SE is a bit worse than other DVS schemes. This is because GPScheDVS-SE additionally increases the priority of HINRT tasks up to the DBSRT priority range such that other tasks utilize the hard idle times caused by HINRT tasks. As the HINRT tasks' priorities are raised up, SRT tasks become to starve more frequently. However, GPScheDVS-SE still provides a high degree of video deadline meet ratio because it performs the fair scheduling among all tasks within a same priority range as Chapter 4 explained. In other words, the HINRT tasks' priority is lowered as their CPU occupancy increases giving SRT tasks a chance to take the CPU. In case of GPScheDVS-CP, HINRT tasks are treated in the exactly same way as other NRT tasks. Because GPScheDVS-CP does not raise the priority of HINRT tasks, the video deadline meet ratio achieved by GPScheDVS-CP is rather better than the existing DVS schemes due to the advantage of GPSched.

Figure 6.33 shows the fraction of time spent at each CPU operating points for the experienced DVS schemes. The large portion of time fraction spent at B0 is the result of the HINRT tasks executed at the highest CPU operating point. The overhead in time, system energy consumption, and the CPU operating point transition numbers presented in Figure 6.34 are similar to the cases under the typical usage scenarios presented in the previous section.

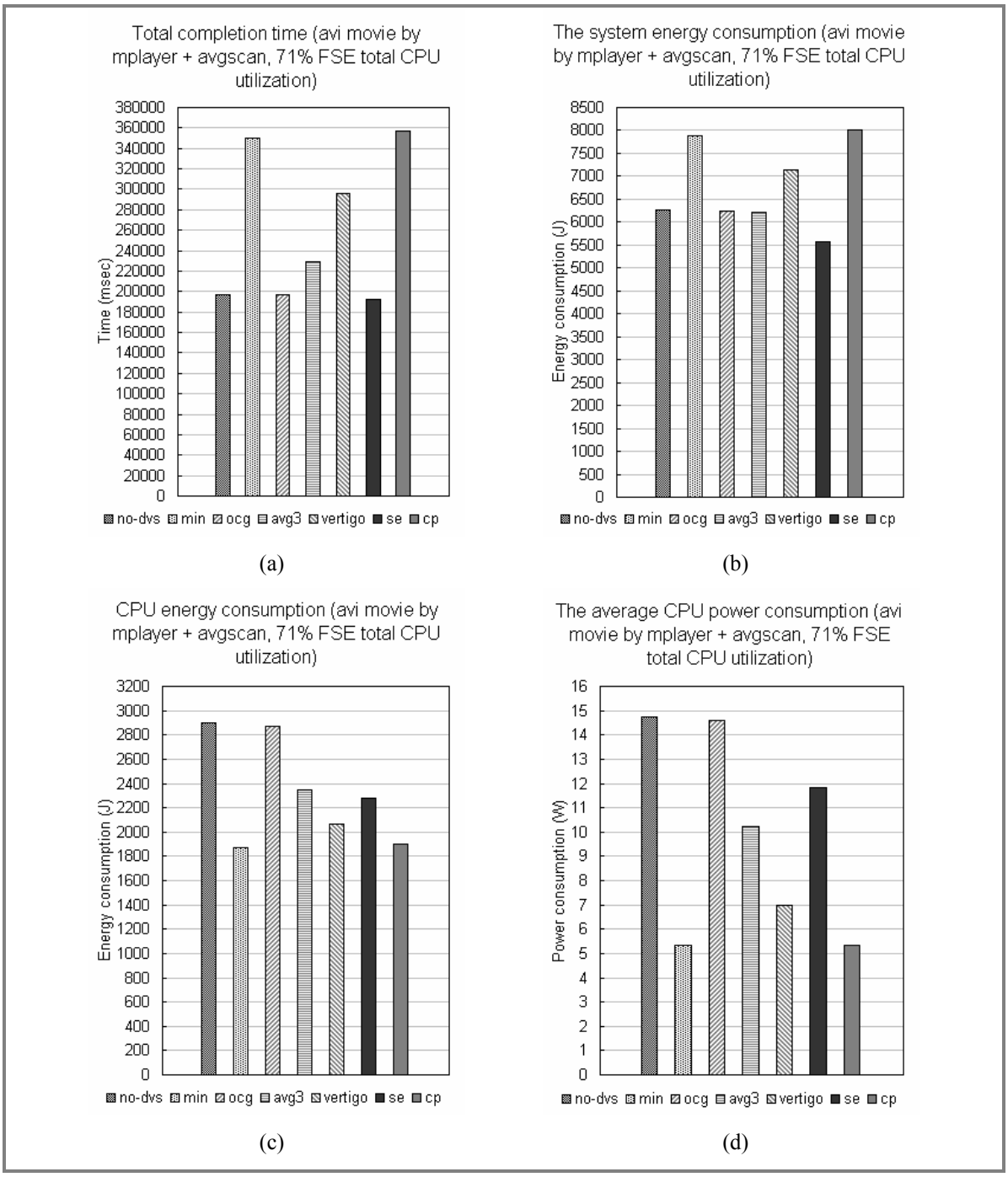


Figure 6.32 Performances of each DVS scheme in energy, power, and QoS under the usage scenario avi_m_avgscan

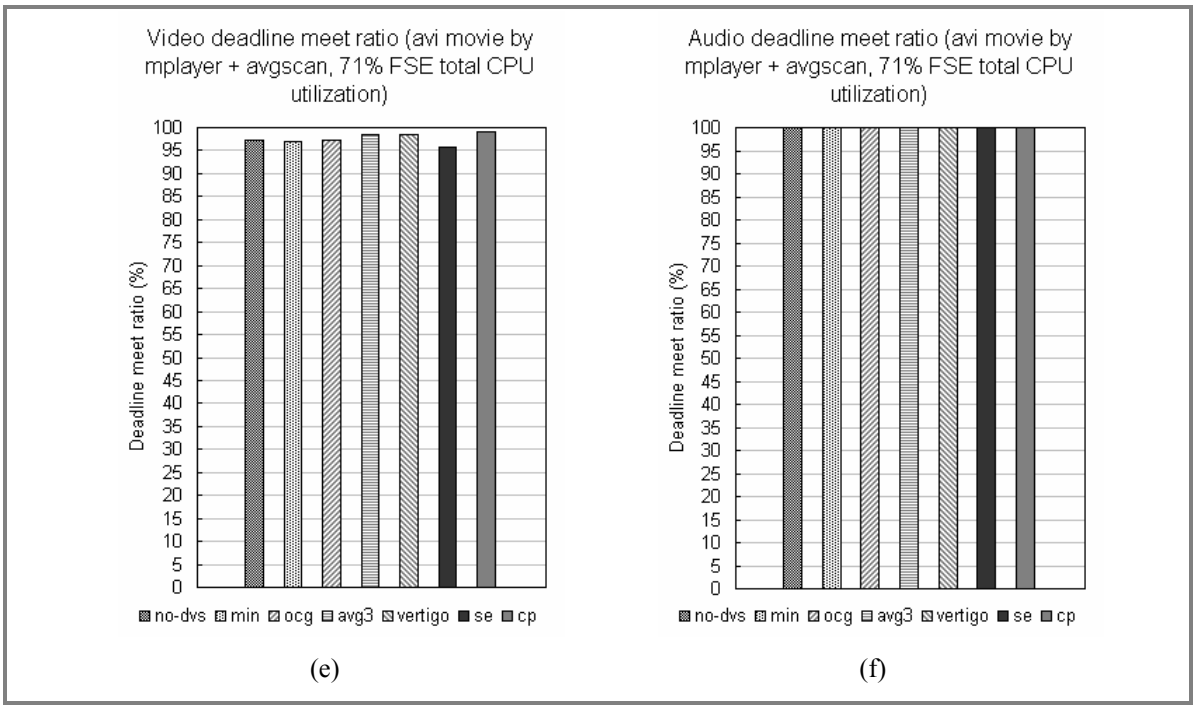


Figure 6.32 Performances of each DVS scheme in energy, power, and QoS under the usage scenario avi_m_avgscan (Cont'd)

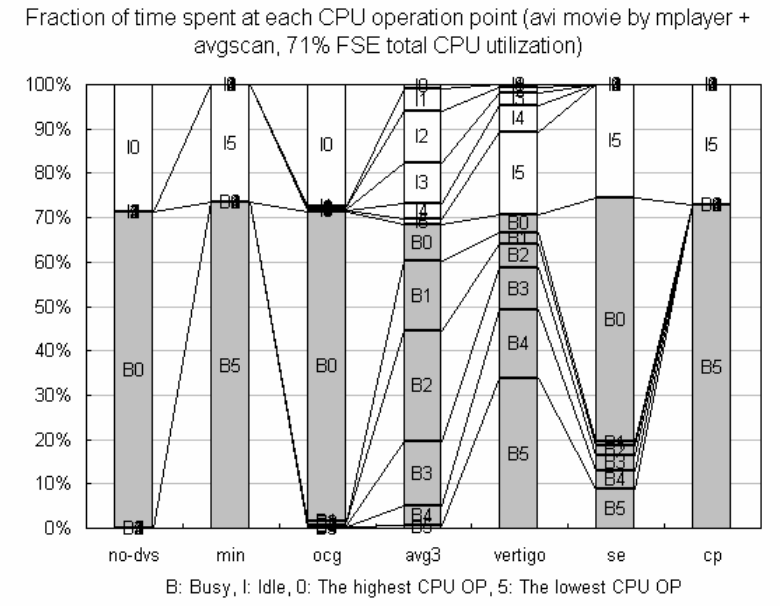


Figure 6.33 Fractions of time spent at each CPU operating point with each DVS scheme under the usage scenario avi_m_avgscan

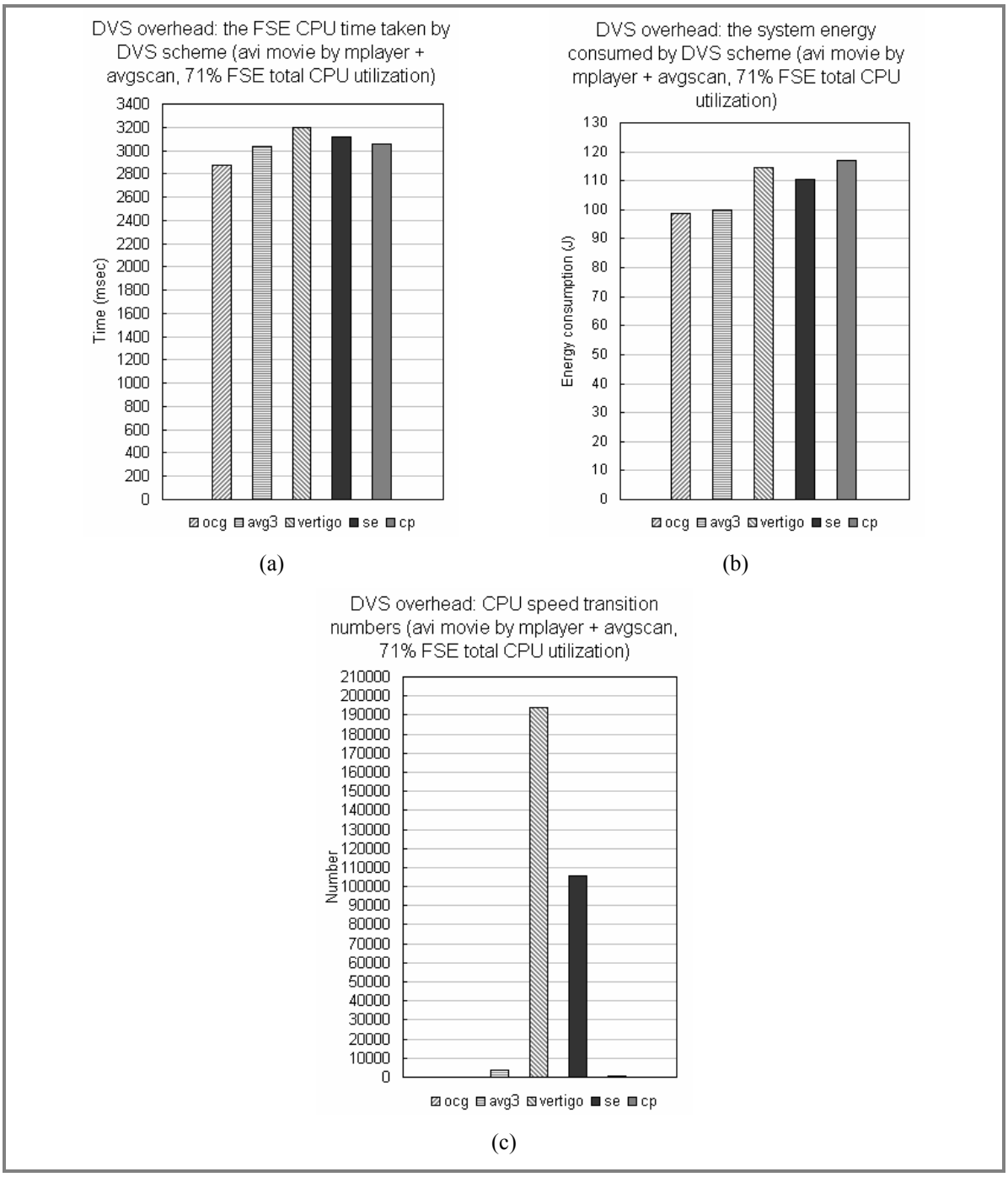


Figure 6.34 Overheads of each DVS scheme under the usage scenario avi_m_avgscan

Figure 6.35, 6.36, and 6.37 show the experimental results obtained under the usage scenario mozilla_avgscan described in Table 6.9 in Section 6.2.3. The results repeat the same story as the results shown in Figure 6.32, 6.33, and 6.34 except one thing; the interactive deadline meet ratio by GPScheDVS-SE is better than the existing DVS schemes. This is because the interactive tasks are assigned with the highest priority under GPSched scheduling as they are under the native Linux task scheduling; GPSched prioritizes the tasks within a same priority range according to their CPU occupancies. Because interactive tasks spend idling most of the time, they typically have the highest priority under GPSched.

Table 6.23 and 6.24 summarize the improvements obtained by GPScheDVS over the existing DVS schemes under all this type of usage scenarios described in Table 6.9 in Section 6.2.3. As is expected, Table 6.23 shows that GPScheDVS-SE reduces system energy consumption than the no-DVS policy and Vertigo at the cost of increased CPU energy consumption and average CPU power consumption. Table 6.24 shows the improvements by GPScheDVS-CP under this type of usage scenarios. Unlike the case of GPScheDVS-SE, the general trend of the improvements by GPScheDVS-CP remains the same as the case of typical usage scenarios; GPScheDVS-CP reduces the CPU energy consumption and average CPU power consumption at the cost of system energy consumption but without degrading the QoS of SRT applications.

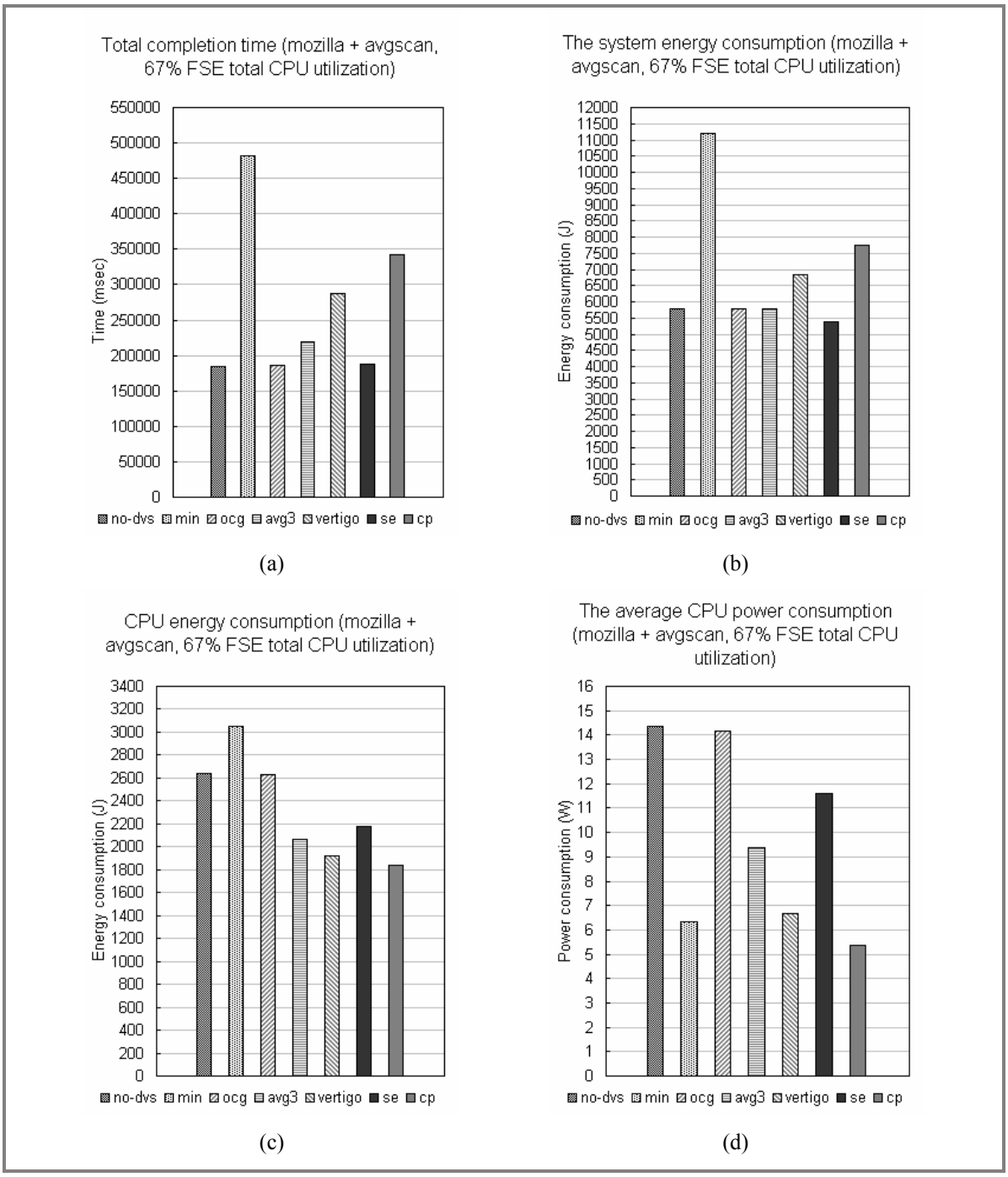
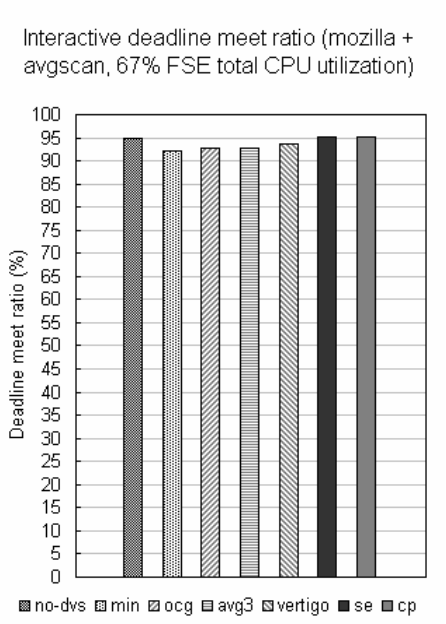


Figure 6.35 Performances of each DVS scheme in energy, power, and QoS under the usage scenario mozilla_avgscan



(e)

Figure 6.35 Performances of each DVS scheme in energy, power, and QoS under the usage scenario mozilla_avgscan (Cont'd)

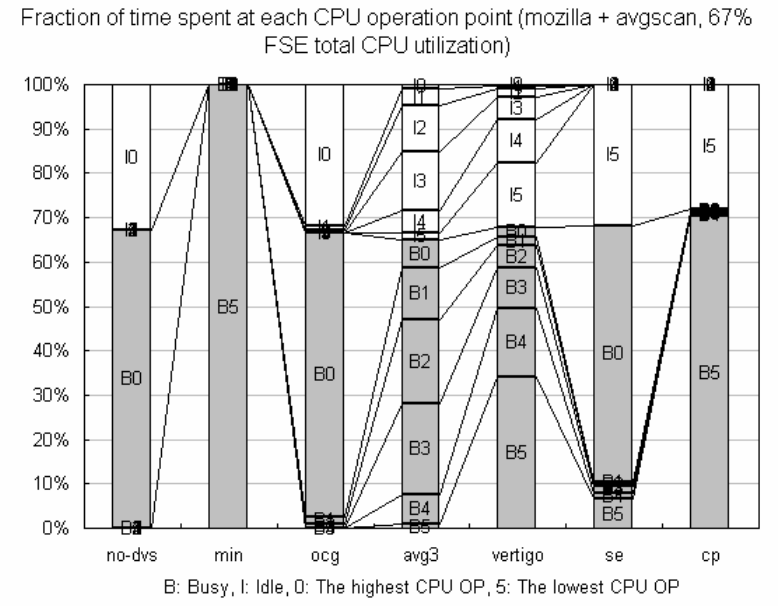


Figure 6.36 Fractions of time spent at each CPU operating point with each DVS scheme under the usage scenario mozilla_avgscan

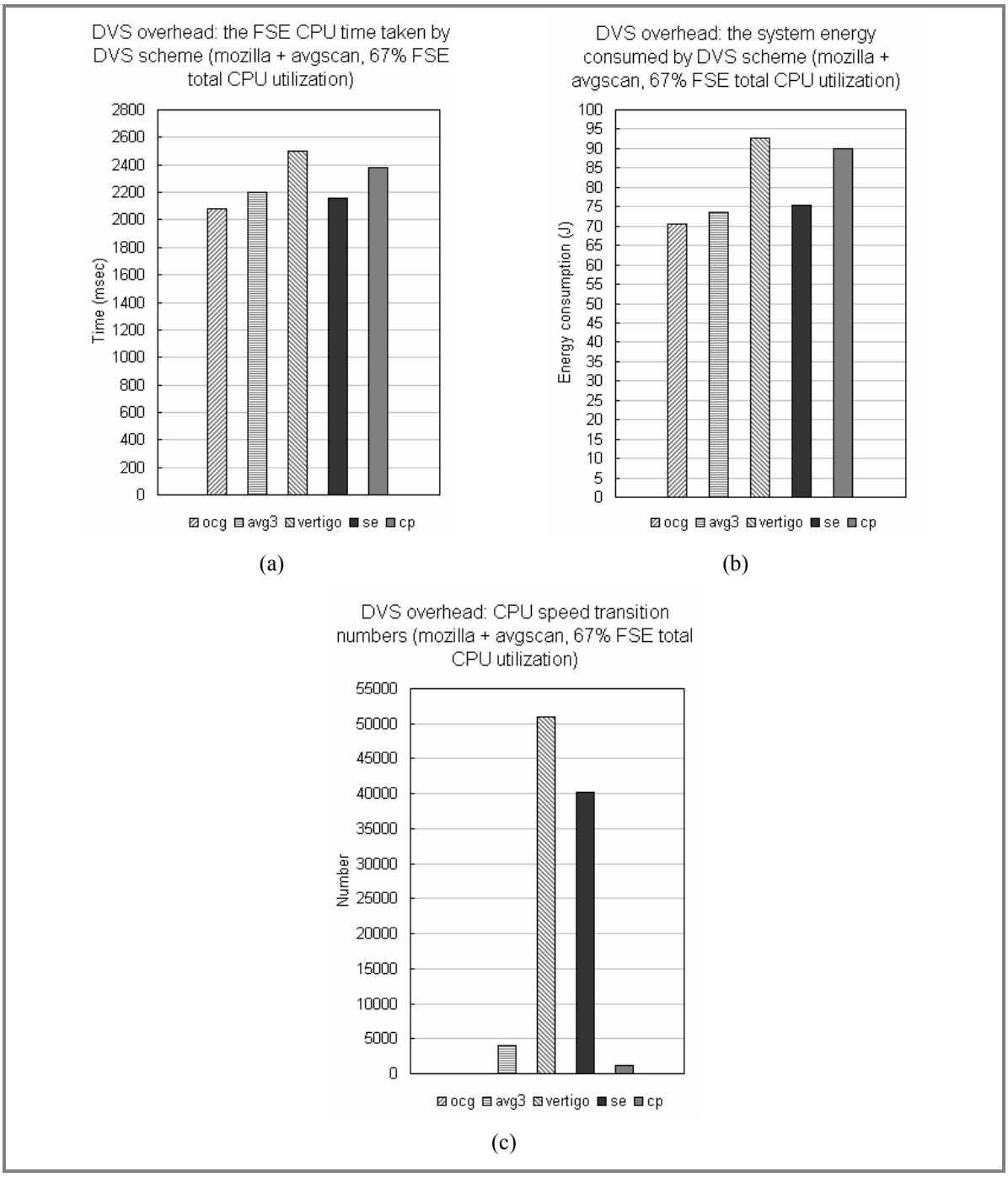


Figure 6.37 Overheads of each DVS scheme under the usage scenario mozilla_avgscan

Table 6.23 Improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes under the special usage scenarios with a HINRT application

Usage scenario	Metric (See note 1)	Case	Improvement (%) over. (See note 2)			
			No DVS	OCG	AVG3	Vertigo
A continuous media application + NRT applications	***System energy consumption***	Best	11.3	10.7	10.4	22.0
		Worst	5.4	5.4	-10.4	-3.3
		Avg.	***8.9***	***9.2***	***6.0***	***16.9***
	CPU energy consumption	Best	23.3	21.3	2.9	-9.2
		Worst	13.6	12.9	-12.2	-20.2
		Avg.	***20.6***	***18.1***	***-4.3***	***-12.9***
	Average CPU power consumption	Best	22.9	19.4	10.4	-11.1
		Worst	14.5	8.7	-32.9	-74.6
		Avg.	***20.8***	***15.8***	***-19.7***	***-60.7***
	Video deadline meet ratio	Best	3.9	0.2	3.1	2.3
		Worst	-2.5	-2.5	-3.0	-2.9
		Avg.	***-0.3***	***-1.2***	***-1.2***	***-1.4***
	Audio deadline meet ratio	Best	0.4	0.4	0.1	0.1
		Worst	-0.1	-0.1	-0.1	-0.1
		Avg.	***0.1***	***0.1***	***0.0***	***0.0***
An interactive application + NRT applications	***System energy consumption***	Best	9.1	9.5	7.5	21.3
		Worst	6.6	6.9	6.8	18.8
		Avg.	***8.0***	***8.4***	***7.2***	***19.9***
	CPU energy consumption	Best	21.8	19.3	-5.0	-11.7
		Worst	17.5	17.2	-6.2	-13.1
		Avg.	***20.2***	***18.3***	***-5.6***	***-12.5***
	Average CPU power consumption	Best	22.6	18.1	-23.4	-63.6
		Worst	19.1	16.6	-24.8	-73.7
		Avg.	***21.3***	***17.3***	***-24.0***	***-67.2***
	Interactive deadline meet ratio	Best	0.3	2.6	2.7	1.8
		Worst	0.0	-0.4	-0.3	-0.1
		Avg.	***0.2***	***0.7***	***0.8***	***0.6***

Note 1—The relevant metrics to the GPScheDVS-SE are the system energy consumption and the QoS of time constrained applications. The GPScheDVS-SE aims at a longer battery life-time without hurting user experiences on the time constrained applications.
Note 2—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

Table 6.24 Improvements achieved by the GPScheDVS-CP over the existing autonomous DVS schemes under the special usage scenarios with a HINRT application

Usage scenario	Metric (See note 1)	Case	Improvement (%) over. (See note 2)			
			No DVS	OCG	AVG3	Vertigo
A continuous media application + NRT applications	System energy consumption	Best	-23.8	-25.9	-28.0	-12.2
		Worst	-33.5	-28.7	-56.5	-46.4
		Avg.	***-27.2***	***-26.8***	***-31.3***	***-16.2***
	CPU energy consumption	Best	36.7	33.9	18.9	12.8
		Worst	31.0	32.0	12.2	5.5
		Avg.	***34.7***	***32.7***	***14.2***	***7.2***
	Average CPU power consumption	Best	64.1	63.5	62.3	53.3
		Worst	63.0	59.4	40.3	21.3
		Avg.	***63.3***	***60.9***	***44.3***	***25.2***
	Video deadline meet ratio	Best	5.8	2.1	4.9	4.2
		Worst	1.2	1.3	0.5	0.4
		Avg.	***2.6***	***1.7***	***1.7***	***1.5***
	Audio deadline meet ratio	Best	0.4	0.4	0.1	0.1
		Worst	-0.1	-0.1	-0.1	-0.1
		Avg.	***0.1***	***0.1***	***0.0***	***0.0***
An interactive application + NRT applications	System energy consumption	Best	-27.0	-26.4	-28.6	-12.6
		Worst	-34.0	-33.6	-33.7	-13.4
		Avg.	***-29.7***	***-29.2***	***-30.9***	***-12.9***
	CPU energy consumption	Best	34.8	32.4	11.9	6.6
		Worst	30.1	29.8	10.6	4.2
		Avg.	***32.8***	***31.2***	***11.1***	***5.3***
	Average CPU power consumption	Best	63.0	62.1	42.8	21.8
		Worst	62.5	60.1	40.3	19.6
		Avg.	***62.7***	***60.8***	***41.2***	***20.8***
	Interactive deadline meet ratio	Best	2.5	2.6	2.6	2.1
		Worst	0.1	0.0	0.1	0.0
		Avg.	***0.9***	***1.4***	***1.5***	***1.3***

Note 1—The relevant metrics to the GPScheDVS-CP are the average CPU power consumption and the QoS of time constrained applications. The GPScheDVS-CP aims at the reduced CPU temperature without hurting user experiences on the time constrained applications.
Note 2—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

6.3.3. Special usage scenarios with fully SRT workloads

This section shows the experimental results achieved under another special type of usage scenarios where only SRT applications are executed without any explicit NRT application in the background. The point of interest in this experiment is whether GPScheDVS will still be effective in further reducing the energy and power without degrading system performance over the existing autonomous DVS schemes. As described in Chapter 4,

GPSched provides the deadline-based task prioritization only among HRT, DBSRT, RBSRT, and NRT tasks. Under this type of usage scenarios, the imposed workload is made up mostly of DBSRT tasks. This situation naturally raises the question that what is the difference between GPSched and the native Linux task scheduler?

Answering in advance, GPScheDVS still achieves considerable improvements in energy, power, and even the QoS of DBSRT applications over the existing autonomous DVS schemes. The reason is two folds:

- Still there is RBSRT tasks such as kernel threads and daemon processes in the system under these extreme usage scenarios. This means that the advantage of GPSched in making tasks' CMFSs uniform and close to the ICMFS works between the DBSRT tasks and RBSRT tasks. This allows GPScheDVS to keep a still lower CPU speed than the existing DVS schemes. Also, this property affords a better QoS to GPScheDVS than the existing DVS schemes.

- GPSDVS, the task-based DVS scheme of GPScheDVS customized GPSched, predicts tasks' CMFSs more accurately than the existing DVS schemes. The higher accuracy of GPSDVS is, in turn, due to the following two reasons. (1) GPSDVS shares the task type information with GPSched. The existing interval-based DVS schemes are typically overcautious as pointed out earlier in order to prevent the deadline missing of SRT tasks. Even a SRT task has a low CPU speed requirement, these schemes quickly raise the CPU speed upon the arrival of tasks because they cannot distinguish SRT tasks from NRT tasks. Unlike these existing interval-based DVS schemes, GPSDVS knows the type of tasks by the help of GPSched. Therefore, it does not raise the CPU speed prematurely upon the arrival of SRT workload. (2) AIDVS, the adaptive interval-length interval-based DVS algorithm described in Chapter 5, more accurately predicts the ICMFS, i.e., the total CPU utilization.

Figure 6.38 and 6.39 shows the experimental results obtained under mozilla_avi_m and mozilla_mp31_m described in Table 6.10 in Section 6.2.3. In these usage scenarios, an interactive application mozilla is executed while a continuous media application is

running. Figure 6.38 shows the total completion time, system energy consumption, CPU energy consumption, average CPU power consumption, and interactive, video, and audio deadline meet ratios achieved by each DVS scheme. Figure 6.38 shows that all DVS schemes yield similar results. However, GPScheDVS earned slightly better energy, power, and QoS than other schemes. This is because GPSched still works due to the existence of RBSRT workload in the system.

Figure 6.39 shows the overhead in time, system energy consumption, and the CPU operating point transition numbers. An interesting result about the overhead is that GPScheDVS had a less overhead in time and system energy consumption than other DVS schemes. This is the opposite results to other types of usage scenarios. The reason is the reduced task scheduling burden. Unlike the previous usage scenarios where gcc caused frequent task creation and removal, this type of usage scenarios have only the SRT tasks that executes for a long time. Therefore, the need to allocate and deallocate the data structures required to manage tasks' information decreases. This is a motivation of the future refinement on GPScheDVS; if GPScheDVS shares the data structure required to schedule tasks with the original Linux kernel, then the overhead of GPScheDVS in time and system energy will be reduced.

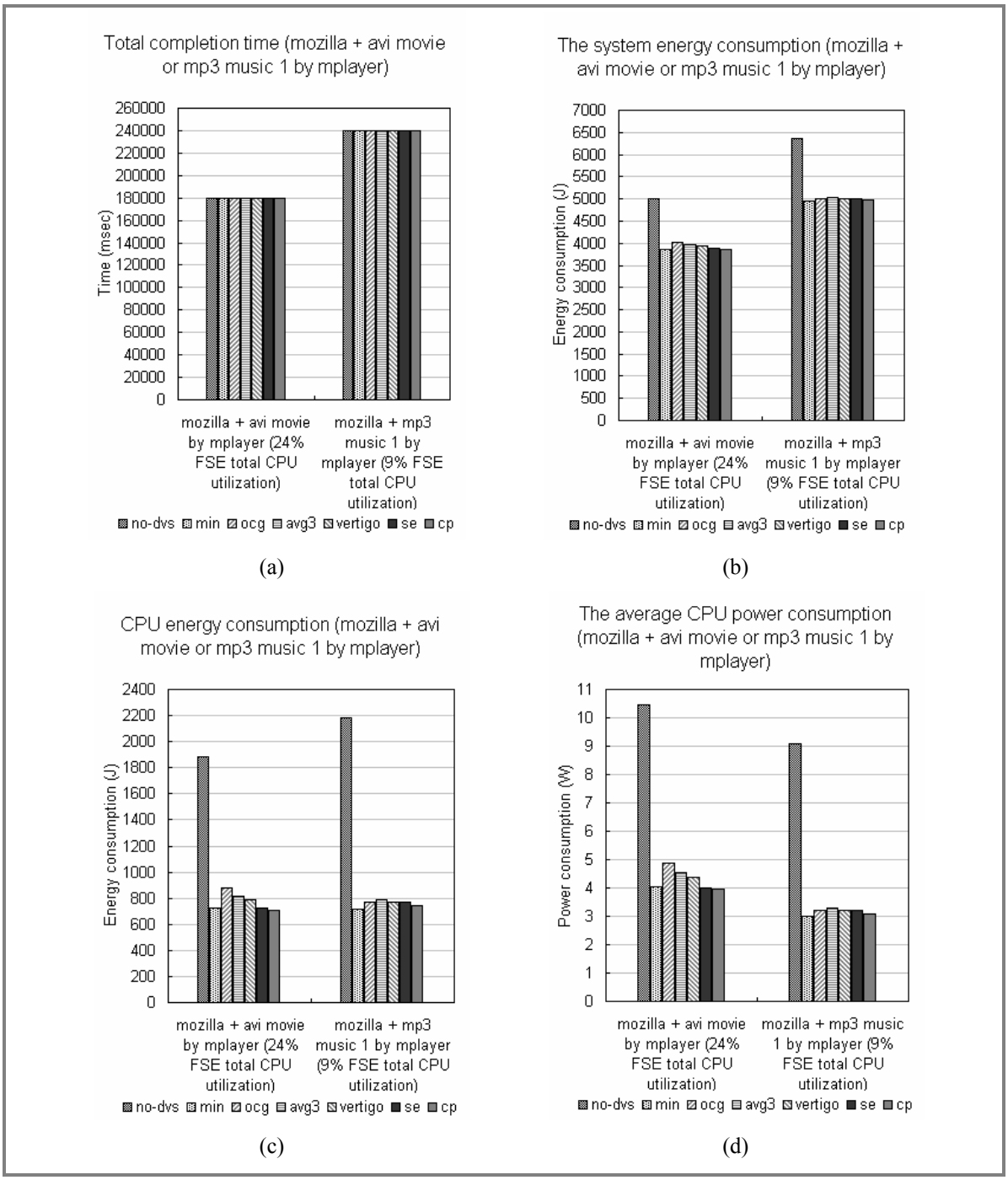


Figure 6.38 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios mozilla_avi_m and mozilla_mp31_m

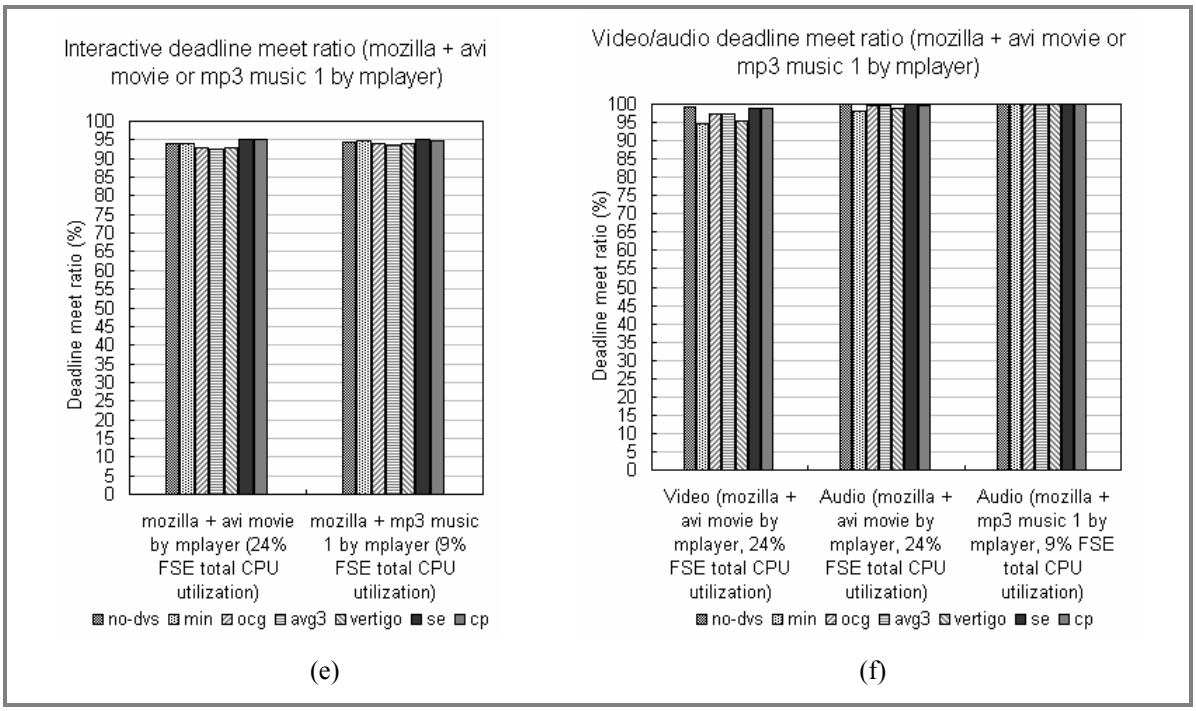


Figure 6.38 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios mozilla_avi_m and mozilla_mp31_m (Cont'd)

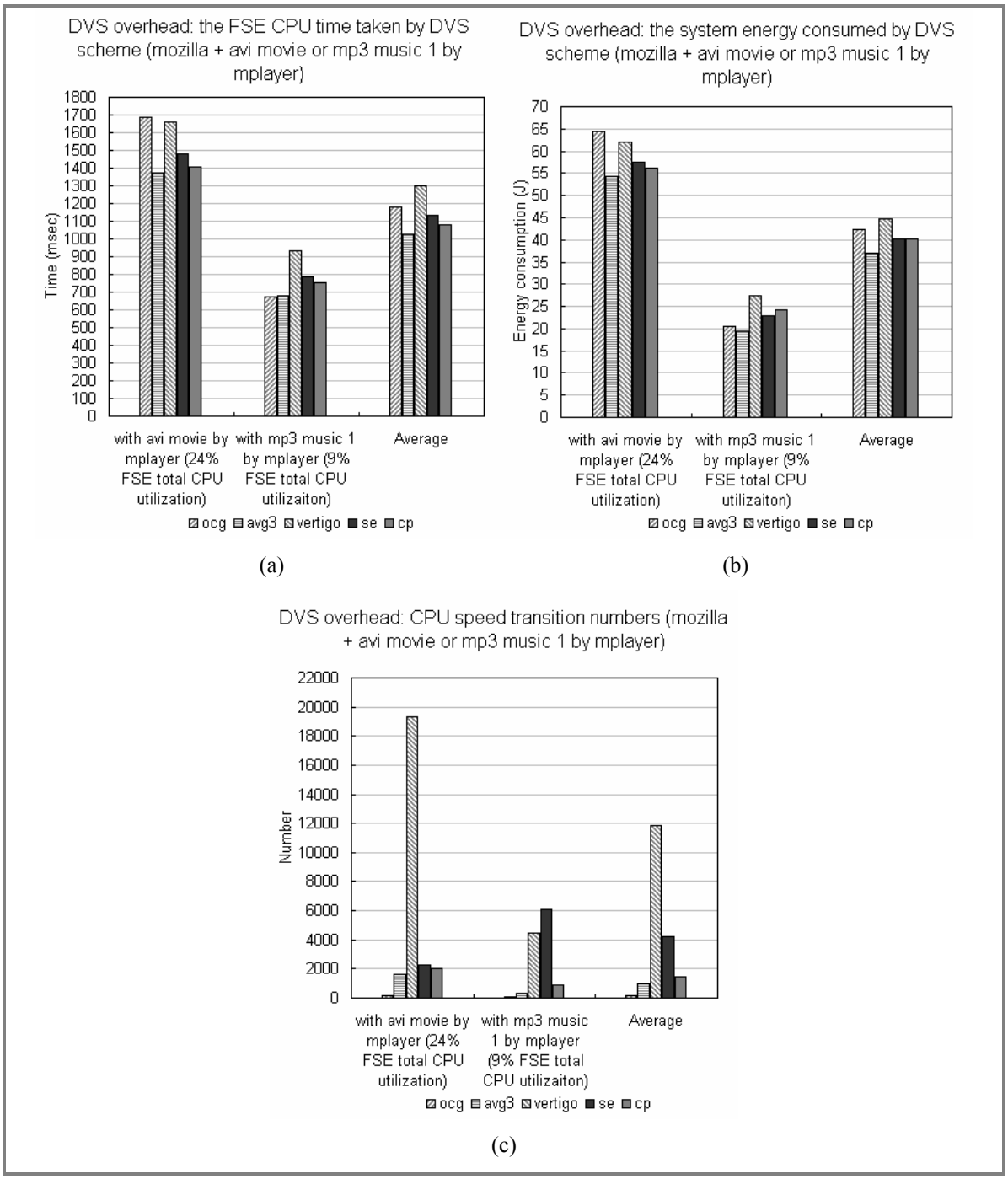


Figure 6.39 Overheads of each DVS scheme under the usage scenarios mozilla_avi_m and mozilla_mp31_m

Figure 6.40, 6.41, 6.42, and 6.43 present a more severe case of the special type of usage scenarios. In this usage scenario, a multiple number of continuous media applications were executed simultaneously. The detailed description of this scenario is available in Table 6.11 of Section 6.2.3. This usage scenario especially mimics the video conference

usage scenarios where a multiple number of video and audio applications have to run simultaneously. Unlike the previous usage scenario where a continuous media application and an interactive application were used simultaneously but the amount of workload were at most 24% of the CPU capacity, this usage scenario imposes a heavier workload that reaches up to 87% of CPU capacity.

Figure 6.40 shows the total completion time, system energy consumption, CPU energy consumption, average CPU power consumption, and the video and audio deadline meet ratios for cases of 1, 2, 4, and 6 instances of mplayer. As pointed out earlier, Figure 6.40 shows that GPScheDVS improves the energy, power, and the QoS of SRT applications over the existing DVS schemes. For 1, 2, and 3 mplayer instances cases, the two main reason for the better results by GPScheDVS than other DVS schemes are that (1) GPSDVS shares the task type information with GPSched and that (2) AIDVS more accurately predicts the ICMFS, i.e., the current total CPU utilization. For these cases, GPScheDVS outperforms especially the existing interval-based DVS schemes. The interval-based DVS schemes overestimate the current CPU speed requirement and waste energy while GPScheDVS keeps the predicted ICMFS. However, this advantage reduces because the disadvantage of the overestimation of total CPU utilization by the existing interval-based DVS schemes decreases naturally as the ICMFS increases. Figure 6.41 (a), (b), and (c) illustrate this trend.

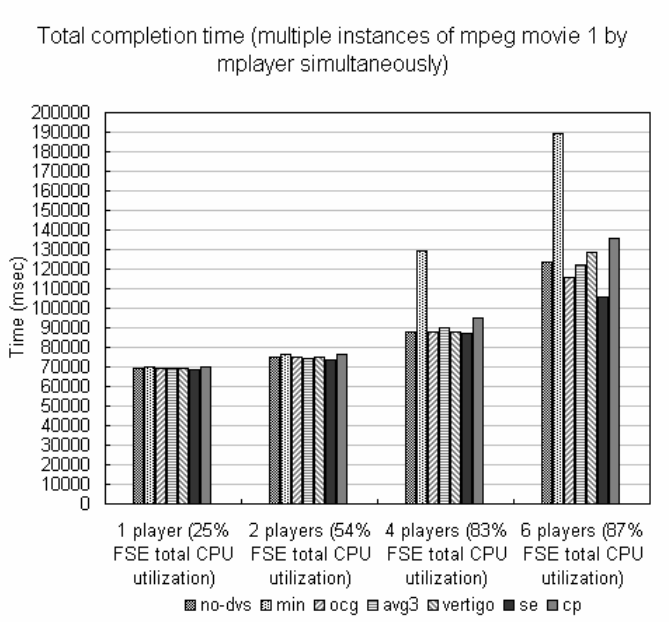
For the 6 instances case, the reason for the better results by GPScheDVS is the accurate prediction of ICMFS with AIDVS. As can be seen in Figure 6.40 (a), the existing DVS schemes cause a considerable delay in the final completion time of the given task set. The problem is especially severe with Vertigo for the following reason. Vertigo has two different CPU speed control algorithms; one for all tasks (the perspectives-based algorithm) and one for interactive tasks. Because there is no interactive task under this scenario, the perspectives-based algorithm thoroughly determines the CPU speed to apply. As explained in Section 6.2.2, the perspectives-based algorithm traces the ratio of job execution time to job inter-arrival time on a per-task basis. Because the total CPU utilization of the task set imposed under the 6 mplayer instances case is not 100%, there is such task that includes some CPU idle time between its consecutive arrivals. The

perspectives-based algorithm, therefore, applies a low CPU speed to the task. However, under the native Linux task scheduling, such tasks are typically assigned with a high priority because their CPU occupancies tend to low. The result is that the high priority but slowly executing task blocks other tasks delaying the final completion time of the given task set.

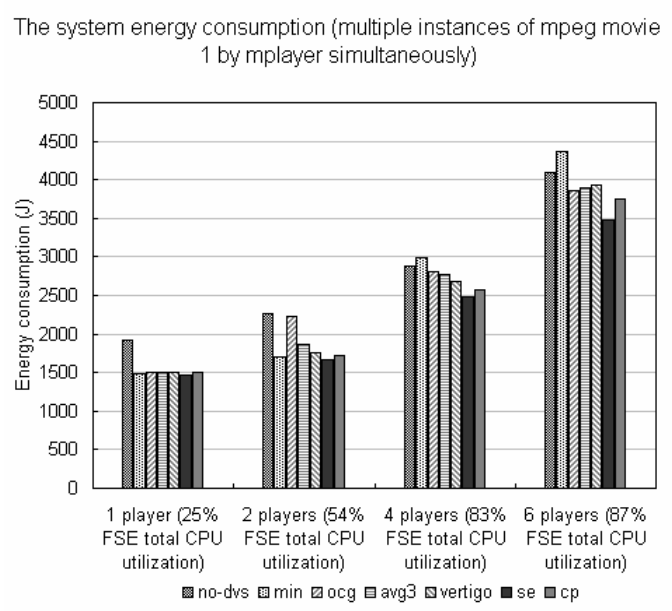
The low system energy consumption of GPScheDVS-SE shown in Figure 6.40 (b) and its high average CPU power consumption shown in Figure 6.40 (d) explain that GPScheDVS-SE keeps the highest CPU speed for a longer time than other DVS schemes and prevents the delay in the final completion time. Figure 6.42 (d) confirm this mechanism by showing the large fraction of time spent by GPScheDVS-SE at the highest CPU operating point for the 6 mplayer instances case.

As Figure 6.40 (e) and Figure 6.41 (d) and (e) show, the delay in the final completion time of a fully continuous media application task set degrades the video and audio deadline meet ratios.

Figure 6.42 shows the similar trend in the overhead. As most of the tasks are long lasting SRT application tasks, the burden of task scheduling reduces and so does the overhead of GPScheDVS in time and system energy consumption.

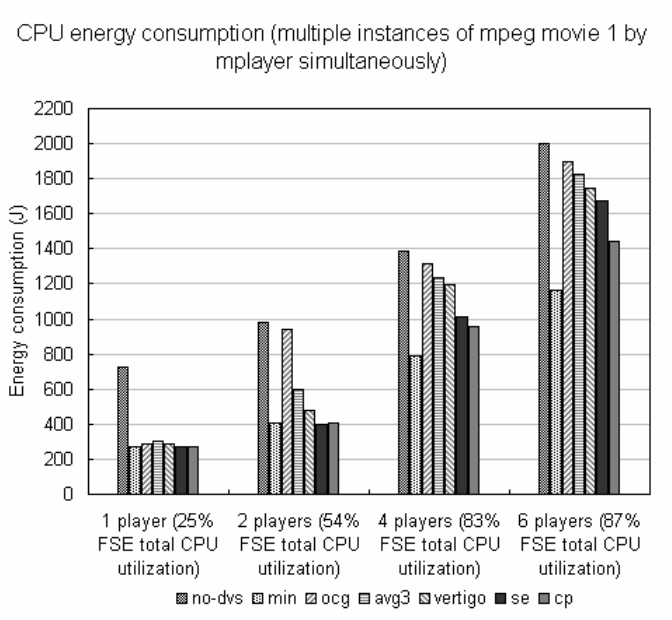


(a)

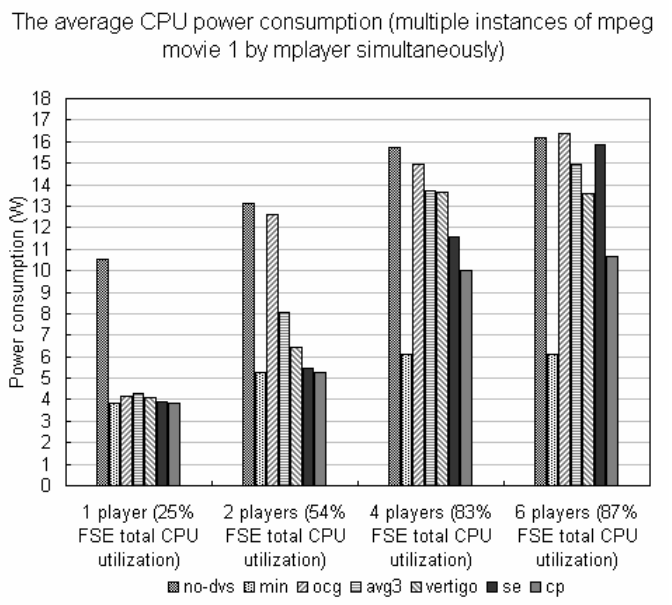


(b)

Figure 6.40 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios [1, 2, 4, and 6]_mpeg1_m

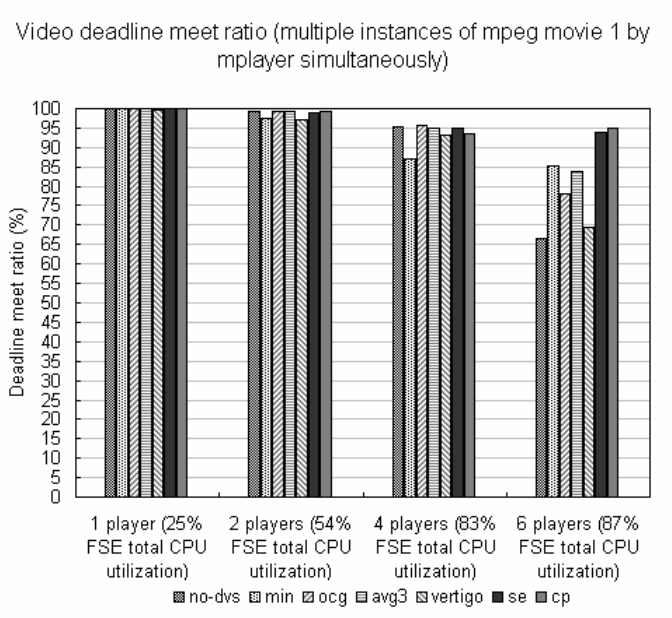


(c)

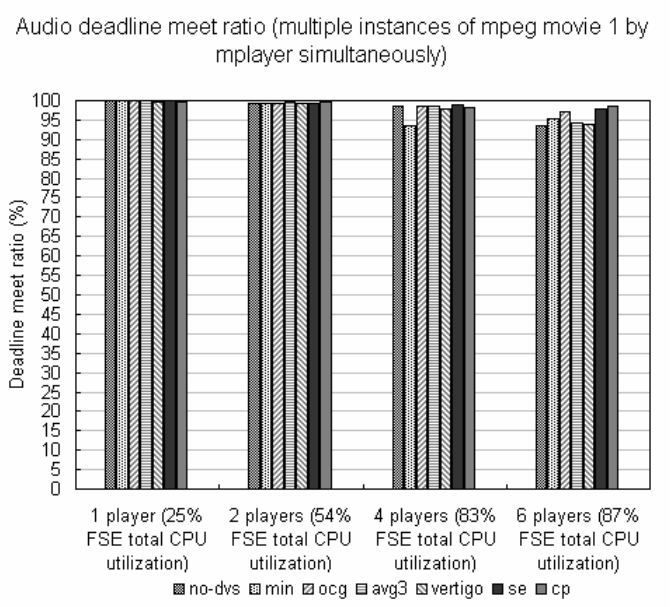


(d)

Figure 6.40 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)

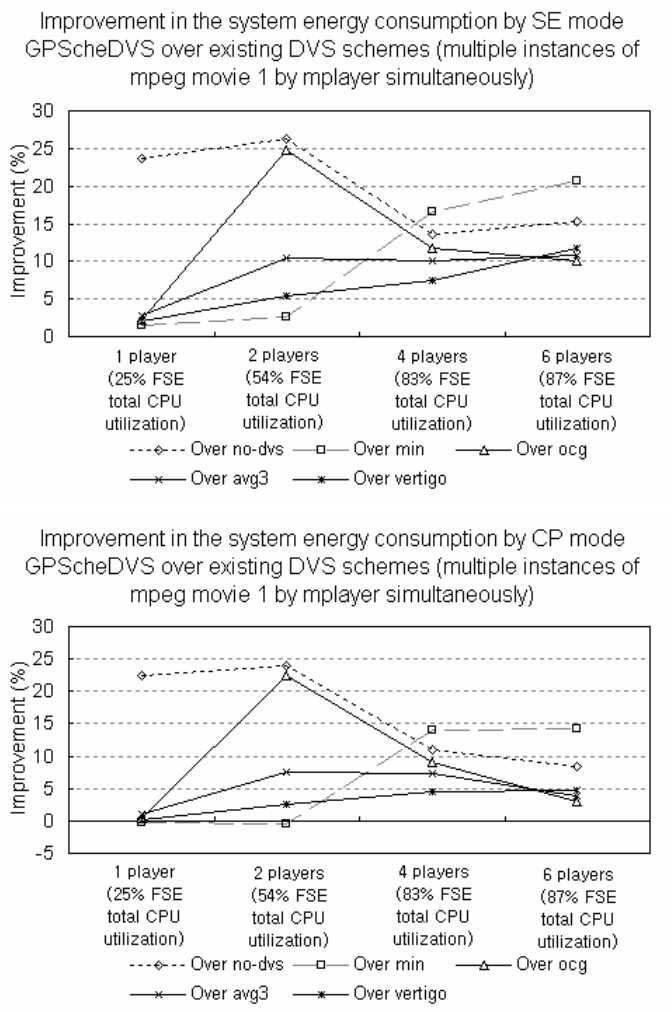


(e)



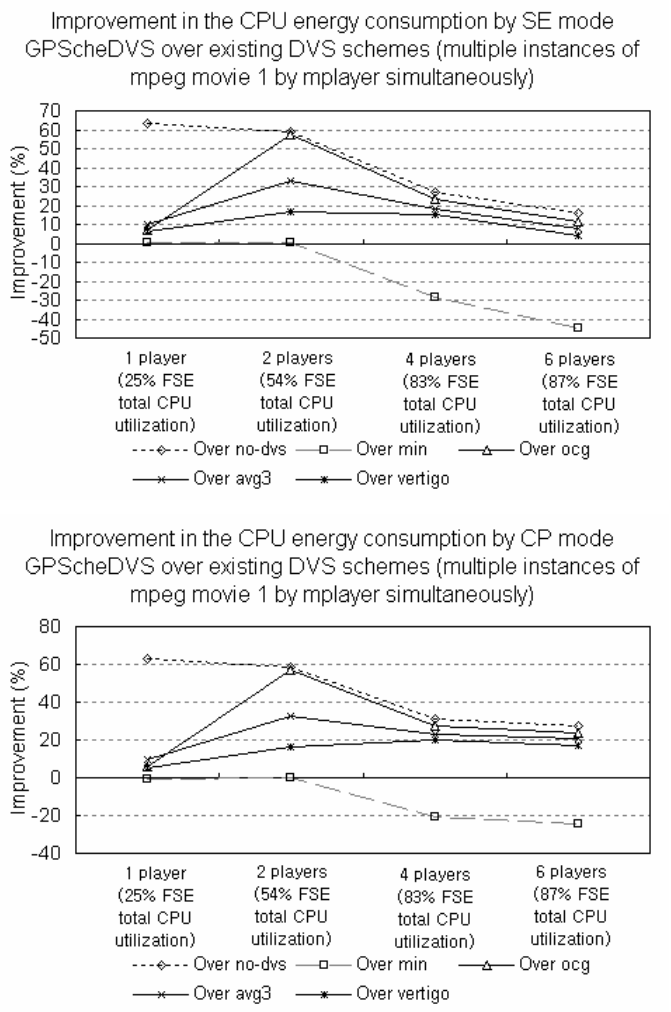
(f)

Figure 6.40 Performances of each DVS scheme in energy, power, and QoS under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)



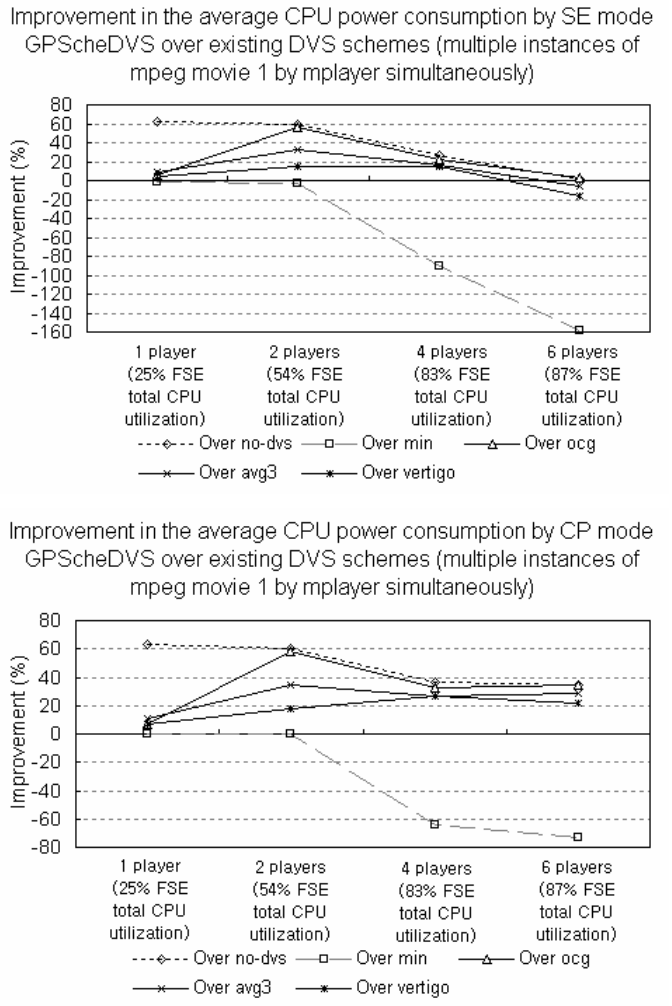
(a) Improvement in system energy consumption

Figure 6.41 Improvements in energy, power, and QoS by the GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes under the usage scenarios [1, 2, 4, and 6]_mpeg1_m



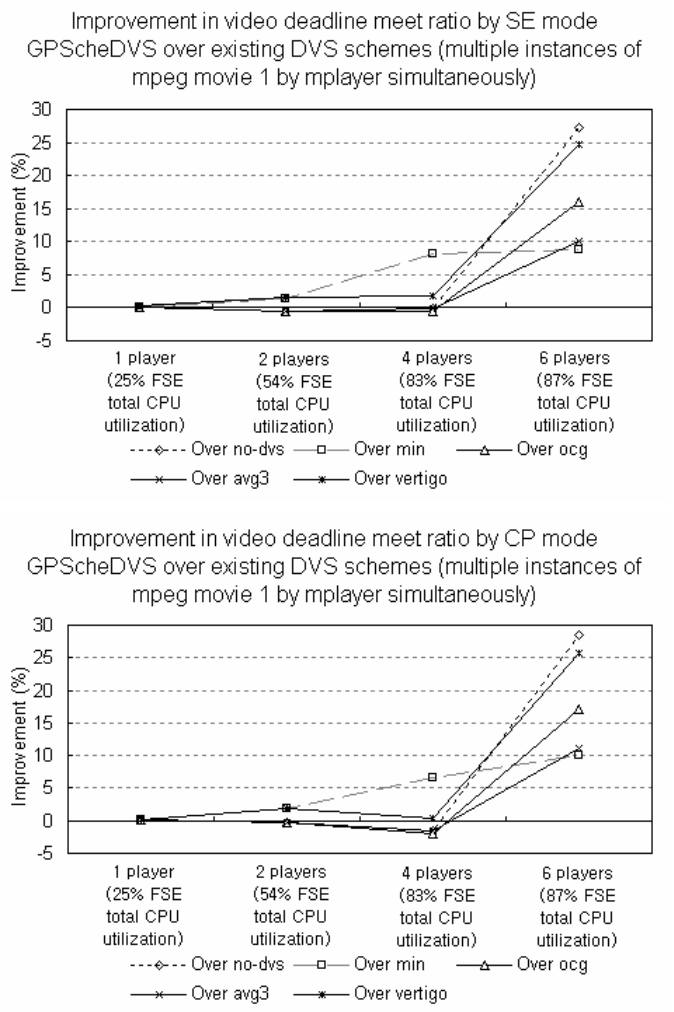
(b) Improvement in CPU energy consumption

Figure 6.41 Improvements in energy, power, and QoS by the GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)



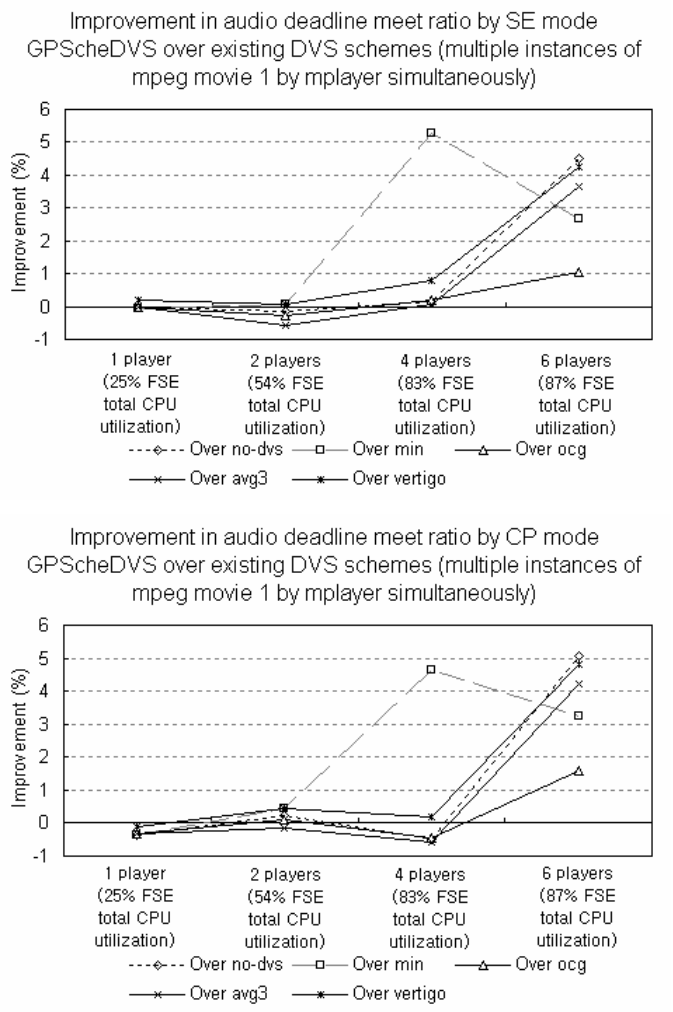
(c) Improvement in average CPU power consumption

Figure 6.41 Improvements in energy, power, and QoS by the GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)



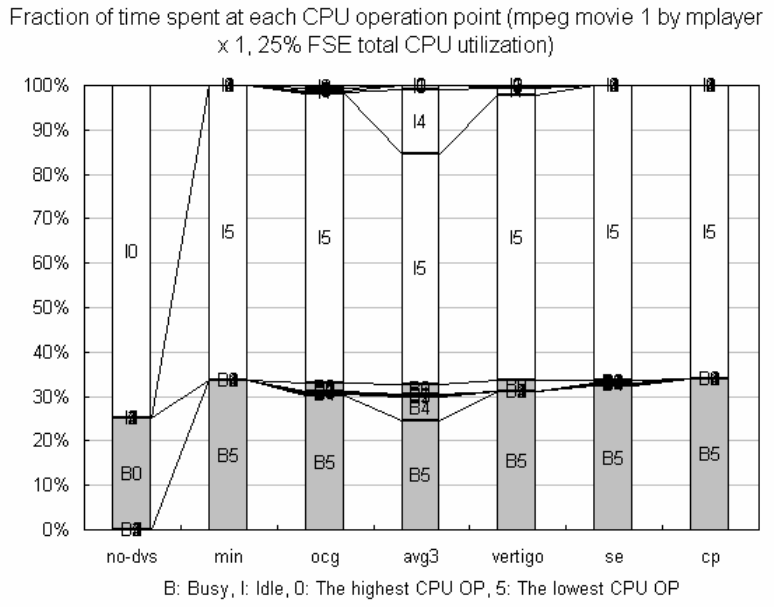
(d) Improvement in video deadline meet ratio

Figure 6.41 Improvements in energy, power, and QoS by the GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)

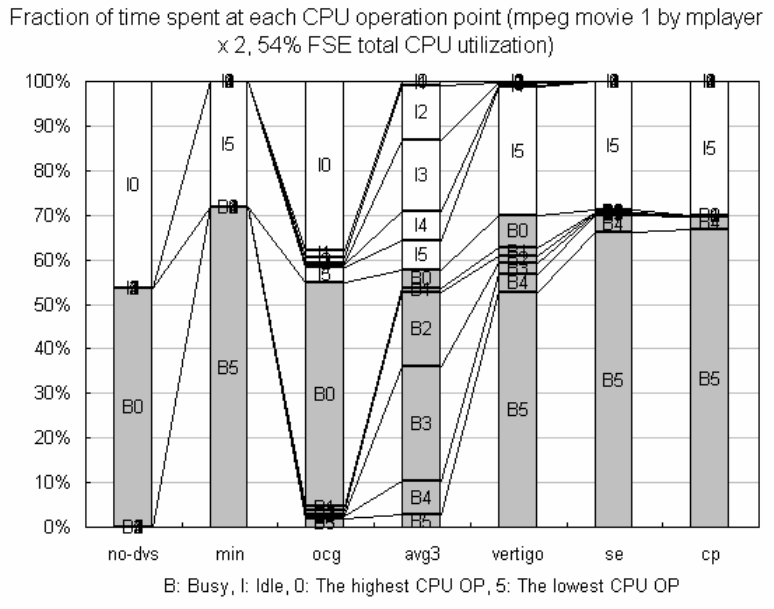


(e) Improvement in audio deadline meet ratio

Figure 6.41 Improvements in energy, power, and QoS by the GPScheDVS-SE and GPScheDVS-CP over the existing DVS schemes under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)

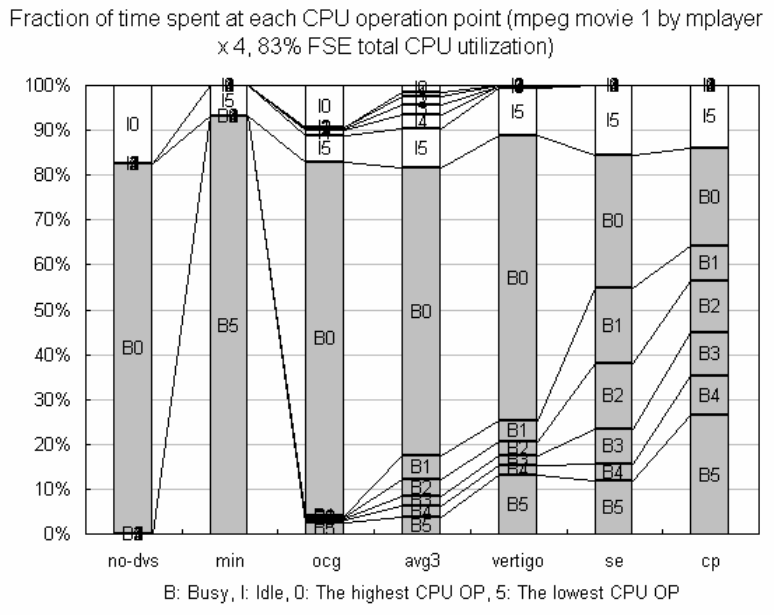


(a)

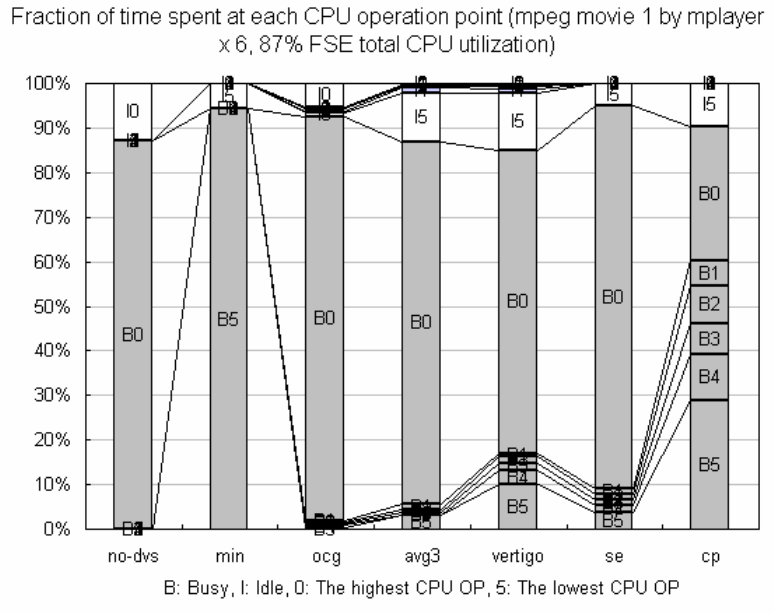


(b)

Figure 6.42 Fractions of time spent at each CPU operating point with each DVS scheme under the usage scenarios [1, 2, 4, and 6]_mpeg1_m

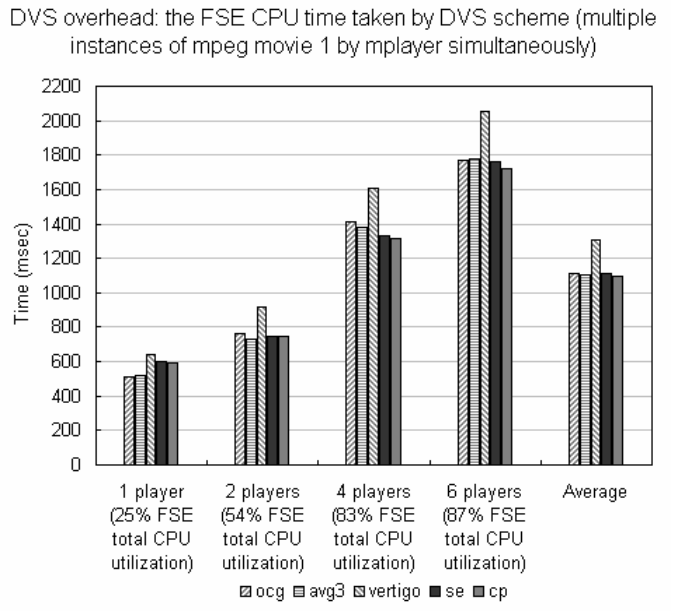


(c)

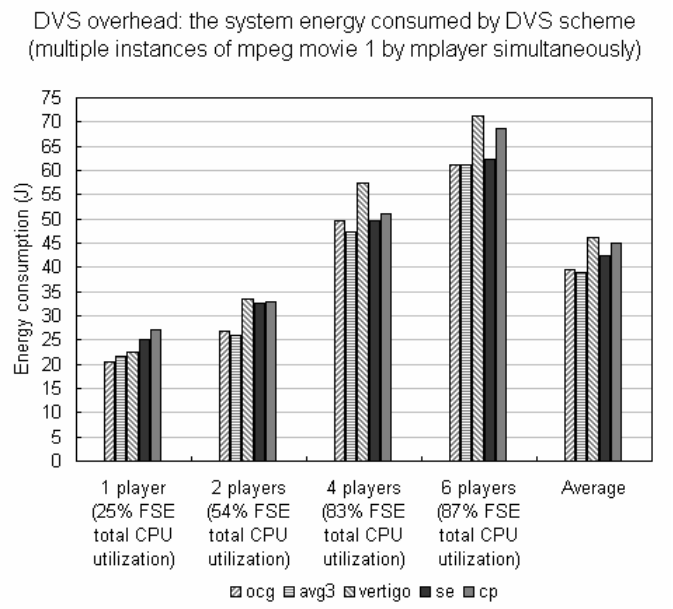


(d)

Figure 6.42 Fractions of time spent at each CPU operating point with each DVS scheme under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)

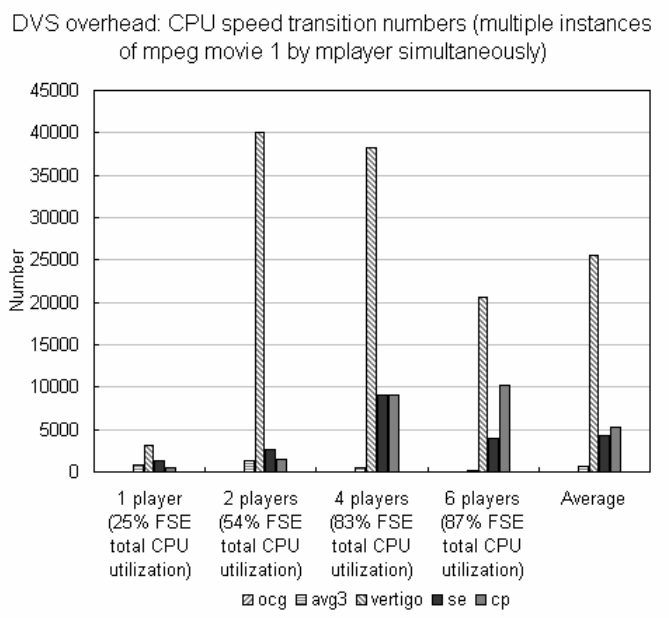


(a)



(b)

Figure 6.43 Overheads of each DVS scheme under the usage scenarios [1, 2, 4, and 6]_mpeg1_m



(c)

Figure 6.43 Overheads of each DVS scheme under the usage scenarios [1, 2, 4, and 6]_mpeg1_m (Cont'd)

Table 6.25 and 6.26 provide the best, worst, and average improvements in each metric achieved by GPScheDVS-SE and GPScheDVS-CP, respectively, over the existing DVS schemes.

Table 6.25 Improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes under special usage scenarios with fully SRT workloads

Usage scenario	Metric (See note 1)	Case (See note 2)	Improvement (%) over. (See note 4)			
			No DVS	OCG	AVG3	Vertigo
A continuous media application + An interactive application (See note 3)	***System energy consumption***	Best	23.7	3.6	2.2	1.4
		Worst	21.2	-0.1	0.1	-0.1
		Avg.	***22.1***	***0.6***	***0.6***	***0.3***
	CPU energy consumption	Best	68.0	17.7	11.7	8.2
		Worst	61.6	-0.8	0.3	-0.8
		Avg.	***65.0***	***3.0***	***3.2***	***2.0***
	Average CPU power consumption	Best	68.0	17.7	11.7	8.2
		Worst	61.6	-0.7	0.3	-0.8
		Avg.	***65.0***	***3.0***	***3.2***	***2.1***
	Video deadline meet ratio	Best	-0.1	1.4	1.2	3.4
		Worst	-0.8	0.0	-0.1	0.0
		Avg.	***-0.5***	***0.7***	***0.4***	***1.3***
	Audio deadline meet ratio	Best	0.1	0.5	0.3	1.1
		Worst	0.0	0.0	0.0	0.0
		Avg.	***0.0***	***0.1***	***0.0***	***0.3***
	Interactive deadline meet ratio	Best	1.2	2.2	2.5	2.2
		Worst	0.0	-0.3	-0.2	0.0
		Avg.	***0.5***	***0.7***	***0.8***	***0.7***

Note 1—The relevant metrics to the GPScheDVS-SE are the system energy consumption and the QoS of time constrained applications. The GPScheDVS-SE aims at a longer battery life-time without hurting user experiences on the time constrained applications.

Note 2— As for the QoS of time constrained applications, the best case improvements were achieved under the heavy load situations, whereas the worst case improvements were yielded under the light load situations. On the other hand, the best case improvements in the energy and power metrics were achieved under the moderate load situations, and the worst case improvements were yielded under either the lightest or the full load situations. The improvements were calculated over all the experimental results obtained under the light, moderate, and heavy load situations.

Note 3—The total CPU utilizations of the usage scenarios of a continuous media application and an interactive application were at most 24% while the ratio of the lowest CPU operating point to the maximum CPU operating point is 37.5% for the DVS-enabled CPU used in the experiments. This means that these usage scenarios cause only the light load situations. Therefore, the improvements achieved by GPScheDVS are small.

Note 4—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

Table 6.25 Improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes under special usage scenarios with fully SRT workloads (Cont'd)

Usage scenario	Metric (See note 1)	Case (See note 2)	Improvement (%) over. (See note 4)			
			No DVS	OCG	AVG3	Vertigo
A multiple number of continuous media applications	***System energy consumption***	Best	26.2	24.9	10.9	11.8
		Worst	13.7	2.3	2.8	2.0
		Avg.	***19.7***	***12.2***	***8.5***	***6.6***
	CPU energy consumption	Best	63.1	57.4	32.7	16.6
		Worst	16.0	6.7	8.0	3.8
		Avg.	***41.4***	***24.8***	***17.3***	***10.5***
	Average CPU power consumption	Best	62.8	56.8	32.4	15.5
		Worst	2.0	3.4	-5.7	-16.8
		Avg.	***37.5***	***22.1***	***12.9***	***4.7***
	Video deadline meet ratio	Best	27.3	16.0	10.0	24.7
		Worst	-0.5	-0.6	-0.5	0.2
		Avg.	***6.7***	***3.8***	***2.4***	***7.1***
	Audio deadline meet ratio	Best	4.5	1.0	3.7	4.2
		Worst	-0.2	-0.3	-0.6	0.1
		Avg.	***1.1***	***0.2***	***0.8***	***1.3***

Note 1—The relevant metrics to the GPScheDVS-SE are the system energy consumption and the QoS of time constrained applications. The GPScheDVS-SE aims at a longer battery life-time without hurting user experiences on the time constrained applications.

Note 2— As for the QoS of time constrained applications, the best case improvements were achieved under the heavy load situations, whereas the worst case improvements were yielded under the light load situations. On the other hand, the best case improvements in the energy and power metrics were achieved under the moderate load situations, and the worst case improvements were yielded under either the lightest or the full load situations. The improvements were calculated over all the experimental results obtained under the light, moderate, and heavy load situations.

Note 3—The total CPU utilizations of the usage scenarios of a continuous media application and an interactive application were at most 24% while the ratio of the lowest CPU operating point to the maximum CPU operating point is 37.5% for the DVS-enabled CPU used in the experiments. This means that these usage scenarios cause only the light load situations. Therefore, the improvements achieved by GPScheDVS are small.

Note 4—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

Table 6.26 Improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes under special usage scenarios with fully SRT workloads

Usage scenario	Metric (See note 1)	Case (See note 2)	Improvement (%) over. (See note 4)			
			No DVS	OCG	AVG3	Vertigo
A continuous media application + An interactive application (See note 3)	System energy consumption	Best	23.8	3.9	2.5	1.8
		Worst	21.4	0.1	0.2	0.1
		Avg.	***22.3***	***0.9***	***0.8***	***0.6***
	CPU energy consumption	Best	68.3	19.4	13.4	10.1
		Worst	62.4	0.7	1.3	0.6
		Avg.	***65.7***	***4.7***	***4.9***	***3.8***
	Average CPU power consumption	Best	68.3	19.4	13.4	10.0
		Worst	62.4	0.7	1.3	0.6
		Avg.	***65.7***	***4.7***	***4.9***	***3.8***
	Video deadline meet ratio	Best	0.0	1.5	1.2	3.4
		Worst	-0.6	0.0	0.0	0.0
		Avg.	***-0.2***	***0.9***	***0.7***	***1.6***
	Audio deadline meet ratio	Best	0.1	0.5	0.0	0.8
		Worst	-0.4	-0.1	-0.1	-0.1
		Avg.	***-0.1***	***0.1***	***0.0***	***0.2***
	Interactive deadline meet ratio	Best	1.1	2.1	2.4	2.2
		Worst	0.0	-0.2	-0.1	0.0
		Avg.	***0.4***	***0.6***	***0.7***	***0.6***

Note 1—The relevant metrics to the GPScheDVS-CP are the average CPU power consumption and the QoS of time constrained applications. The GPScheDVS-CP aims at the reduced CPU temperature without hurting user experiences on the time constrained applications.
Note 2— As for the QoS of time constrained applications, the best case improvements were achieved under the heavy load situations, whereas the worst case improvements were yielded under the light load situations. On the other hand, the best case improvements in the energy and power metrics were achieved under the moderate load situations, and the worst case improvements were yielded under either the lightest or the full load situations. The improvements were calculated over all the experimental results obtained under the light, moderate, and heavy load situations.
Note 3—The total CPU utilizations of the usage scenarios of a continuous media application and an interactive application were at most 24% while the ratio of the lowest CPU speed to the maximum CPU speed is 37.5% for the DVS-enabled CPU used in the experiments. This means that these usage scenarios cause only the light load situations. Therefore, the improvements achieved by GPScheDVS are small.
Note 4—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

Table 6.26 Improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes under special usage scenarios with fully SRT workloads (Cont'd)

Usage scenario	Metric (See note 1)	Case (See note 2)	Improvement (%) over. (See note 4)			
			No DVS	OCG	AVG3	Vertigo
A multiple number of continuous media applications	System energy consumption	Best	23.9	22.5	7.6	4.8
		Worst	8.5	0.5	1.0	0.2
		Avg.	***16.5***	***8.7***	***4.9***	***3.0***
	CPU energy consumption	Best	62.9	57.1	32.2	20.3
		Worst	27.6	6.0	9.6	5.5
		Avg.	***45.2***	***28.7***	***21.4***	***14.7***
	Average CPU power consumption	Best	63.4	58.1	34.4	26.4
		Worst	34.1	6.8	10.5	6.4
		Avg.	***48.4***	***33.2***	***25.2***	***18.0***
	Video deadline meet ratio	Best	28.4	17.2	11.2	25.8
		Worst	-1.6	-2.0	-1.5	0.2
		Avg.	***6.7***	***3.8***	***2.4***	***7.1***
	Audio deadline meet ratio	Best	5.1	1.6	4.2	4.8
		Worst	-0.5	-0.4	-0.6	-0.1
		Avg.	***1.1***	***0.2***	***0.8***	***1.3***

Note 1—The relevant metrics to the GPScheDVS-CP are the average CPU power consumption and the QoS of time constrained applications. The GPScheDVS-CP aims at the reduced CPU temperature without hurting user experiences on the time constrained applications.
Note 2— As for the QoS of time constrained applications, the best case improvements were achieved under the heavy load situations, whereas the worst case improvements were yielded under the light load situations. On the other hand, the best case improvements in the energy and power metrics were achieved under the moderate load situations, and the worst case improvements were yielded under either the lightest or the full load situations. The improvements were calculated over all the experimental results obtained under the light, moderate, and heavy load situations.
Note 3—The total CPU utilizations of the usage scenarios of a continuous media application and an interactive application were at most 24% while the ratio of the lowest CPU speed to the maximum CPU speed is 37.5% for the DVS-enabled CPU used in the experiments. This means that these usage scenarios cause only the light load situations. Therefore, the improvements achieved by GPScheDVS are small.
Note 4—The confidence interval of the energy and power values is ± 2.621% at 95% confidence level.

6.4 Summary

This chapter presented the experimental results on GPScheDVS in comparison with the existing autonomous DVS schemes. The fundamental goal of the experiments was to verify the effectiveness of the suggested autonomous CPU speed control paradigm in which a DVS scheme predicts and enforces the appropriate CPU speed under a DVS-friendly task scheduling through GPScheDVS that was designed following the suggested paradigm. The specific questions were how much system energy consumption the GPScheDVS-SE will reduce and how much CPU power consumption the GPScheDVS-CP will reduce both while improving the QoS of SRT applications. This question was investigated for typical usage scenarios of a commodity-OS-based GP PC and two types of special usage scenarios. In a type of special usage scenario, a SRT application was run with a NRT application in the background which generates hard-idle times. In the other type of special usage scenario, only SRT tasks were executed to impose workload on the CPU.

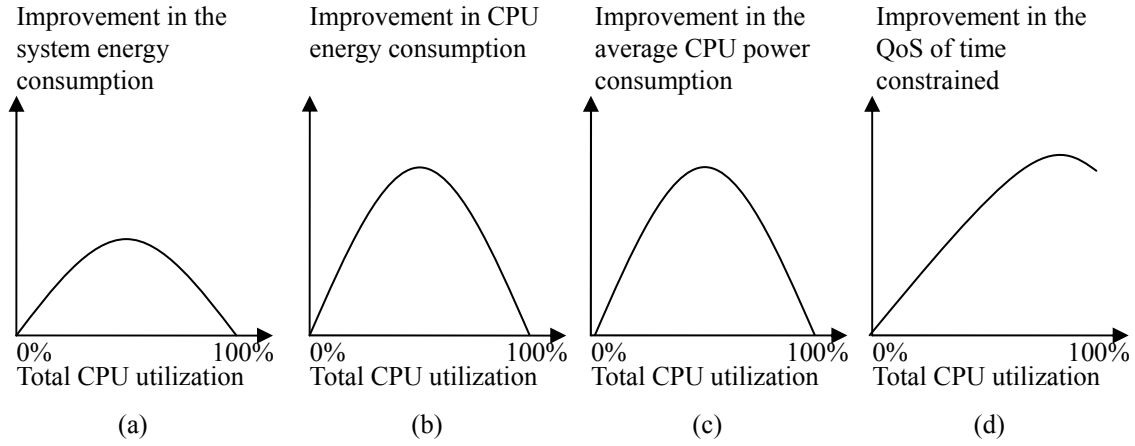


Figure 6.44 General trends in the improvements achieved by the suggested DVS paradigm over the current DVS paradigm.

Figure 6.44 summarizes the improvements achieved by the GPScheDVS-SE over the existing autonomous DVS schemes that follow the current paradigm of autonomous CPU speed control. Plot (a), (b), (c), and (d) depict the trends of the improvements in the system energy consumption, the CPU energy consumption, the average CPU power consumption, and the QoS of time constrained applications, respectively. The trends in

the improvements and the reasons can be briefed as follows:

- Under the light load situation where a time constrained application runs alone, all the schemes keep the CPU operating point to the lowest for most of time as the lowest CPU operating point is enough to meet most timeliness requirements and, therefore, do not make a discernable difference in any metric.

- Under the heavy load situations where a NRT background workload takes the CPU time left by the time constrained application, all the schemes keep CPU operating point to the highest for most of time because there is little room, i.e., the CPU idle time for the reduction of the CPU operating point. Thus, there is no noticeable difference in the system energy consumption, the CPU energy consumption, and the average CPU power consumption. In the QoS of time constrained applications, however, the time constraint-based task scheduling nature of the GPSched gives GPScheDVS a big improvement over the autonomous DVS schemes as the plot (d) of Figure 6.44 suggests. The improvements are prominent in the video and audio deadline meet ratios. As the GPOS's native task scheduling mechanism, i.e., the CPU-occupancy-based task scheduling favors interactive applications, the improvement in the interactive deadline meet ratio achieved under the heavy load situation was small. The improvement is a bit reduced under the full load situation as all the DVS schemes never decrease the CPU operating point. The video, audio, and interactive deadline meet ratios with GPScheDVS were at least 91.2%, 99.7%, and 95.0%, respectively, under any load situation throughout the experiments.

- Under the moderate load situations, the effectiveness of the suggested DVS paradigm in reducing the system energy consumption without degrading the QoS of time constrained applications over the current DVS paradigm is maximized. Unlike the native task scheduler of GPOSs under which the existing DVS schemes work, the DVS-friendly GP task scheduler keeps the required CPU speeds by the individual tasks to meet their timeliness requirements, i.e., the CMFS close to the CPU speed corresponding to the total CPU utilization, i.e., the ICMFS. This feature allows GPScheDVS keep the CPU operating point at the lower level than the existing DVS

schemes without either extending the system running time or degrading the QoS of time constrained applications. As the system running time and, therefore, the energy consumptions of non-CPU components are kept the same, the system energy consumption is reduced as much as the achieved CPU energy savings.

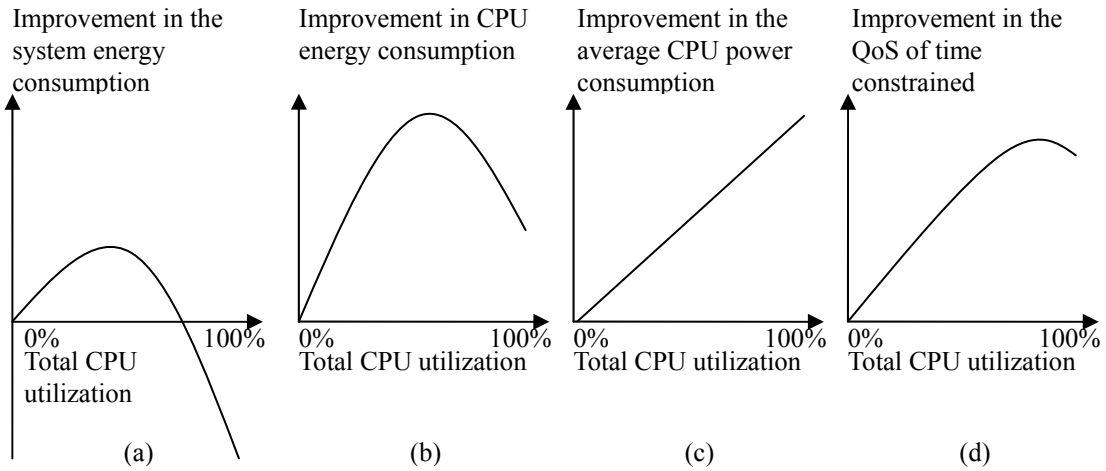


Figure 6.45 The general trend in the improvements by the GPScheDVS-CP over the existing autonomous DVS schemes.

Figure 6.45 then summarizes the experimental results on the GPScheDVS-CP in the same way that Figure 6.44 summarized the results on the GPScheDVS-SE. Because GPScheDVS deals with all but the NRT tasks in a same way in both modes, the general trend of the improvement in the QoS of time constrained applications over the existing autonomous DVS schemes remains same between the two modes of GPScheDVS. The improvements in the system energy consumption, CPU energy consumption, and the average CPU power consumption by the GPScheDVS-CP follow the similar trends to their counterparts shown in Figure 6.44 until the moderate level of CPU utilization.

Under heavy load situations, however, the GPScheDVS-CP achieves even a larger improvement in the average CPU power consumption over the autonomous DVS schemes as it keeps running all NRT tasks at the lowest CPU speed unlike the autonomous DVS schemes. Under the usage scenarios with the continuous media applications, the improvements were up to 63.337%, 63.350%, and 63.297% over EIST, AVG3, and Vertigo, respectively. And, the improvements were up to 63.324%, 63.341%,

and 63.288% over EIST, AVG3, and Vertigo, respectively, under the interactive application usage scenarios. For the same reason, the GPScheDVS-CP yields a better improvement in CPU energy consumption than the GPScheDVS-SE. But, the system energy consumption increases as the GPScheDVS-CP keeps running all NRT tasks at the lowest CPU speed even though there is no more CPU idle time under the heavy load situations; the system running time is lengthened and the energy consumptions of the non-CPU components increase over the CPU energy savings. However, these results exactly meet the purpose of the CP mode operation of GPScheDVS: reducing the average CPU power consumption so as to alleviate the CPU heating problem without degrading system performances especially for the systems in which the primary concern of users is not the battery life-time but the CPU heating problem.

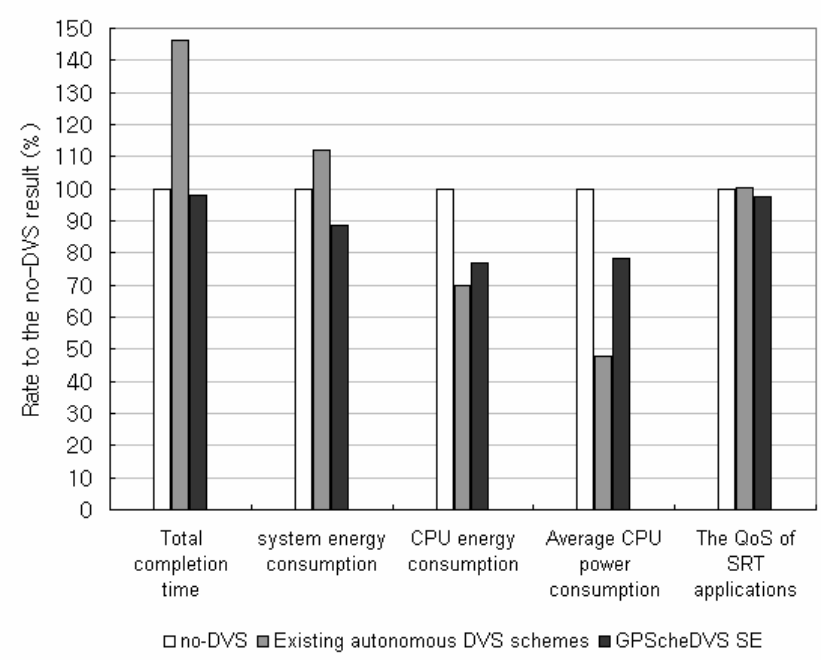


Figure 6.46 Benefit of the GPScheDVS-SE's HINRT task treatment

Figure 6.46 summarizes the experimental results on the performances of GPScheDVS-SE and the existing DVS schemes under the special usage scenarios with a HINRT application. The gray colored bars in Figure 6.46 represent the results of an autonomous DVS scheme normalized to the corresponding results of the no-DVS policy, i.e., the white colored bars. And, the black colored bars represent the normalized results of GPScheDVS.

As shown in Figure 6.46, GPScheDVS-SE prevents the delay in the given workload's total completion time by distinguishing the HINRT tasks and appropriately treating them. As a consequence, the system running time is preserved and, eventually, the increase of the non-CPU components' energy consumptions is avoided. GPScheDVS sets the CPU speed to the highest when a HINRT task is in the run queue. But, it runs tasks at a low CPU speed when the HINRT task is sleeping in a waiting queue and achieves a considerable amount of CPU energy savings and the reduction of the average CPU power consumption over the no-DVS policy. As the energy consumptions of the non-CPU components remain the same, the system energy consumption reduces as much as the achieved CPU energy savings.

Chapter 7

Conclusion

This chapter summarizes the dissertation, lists the contributions to the area of the power-performance trade-offs for commodity-OS-based GP PCs, and describes future extensions.

7.1 Summary

This work proposed a new paradigm of the autonomous CPU speed control for commodity-OS-based GP PCs. Unlike the existing paradigm in which focus has been on the accurate prediction of tasks' CPU speed requirement, the new paradigm focuses on making the tasks' CPU speed requirement low, uniform, and close to the total CPU utilization. The motivating observation is that most contemporary GP PCs continue using the native GPOS task schedulers that prioritizes tasks regardless of their deadlines. This research showed that prioritizing tasks according to their deadlines makes the tasks' CPU speed requirement uniform and close to the total CPU utilization under multi-program environments.

Chapter 2 first surveyed the two existing OS-level DVS approaches: the power-aware RT task scheduling, which is the DVS extension of the conventional RT task scheduling, and the other DVS approach that does not meddle in task scheduling and works irrespective of the particular task scheduling mechanism and, therefore, called *autonomous DVS* in this dissertation. The wide-spread use of the power-aware RT task scheduling in *commodity-OS-based GP systems*, the target GP systems of this research that run the existing applications designed for commodity GPOSs such as Microsoft Windows or Linux without any modification, is still a long way off due to the fundamental incongruity of the underlying task model of the conventional RT task scheduling (which assumes that tasks have regular inter-arrival times and execution times and are independent from each other) with the actual tasks running in commodity-OS-based GP systems. As Chapter 3 pointed, the task model demands the behavior of actual tasks to be tailored causing the

dilemma of choice between meeting the aperiodic timeliness requirements (especially that of interactive applications) and preserving system utilization (to prevent non-CPU components and, ultimately, the entire system from consuming more energy). Moreover, the task model cannot properly cope with the precedence constraints among tasks (e.g., the dependencies between a UI (User Interface) server task and client tasks or among collaborating threads). For this reason, the current paradigm of DVS for commodity-OS-based GP systems is to use the autonomous DVS under the native task scheduling mechanism of commodity GPOSs – the traditional CPU-occupancy-based task scheduling.

Then Chapter 2 summarized the previous work on the energy-optimal CPU speed control that provides a basis for this research. The goal of the energy-optimal CPU speed control is to find the speed profile that minimizes the CPU energy consumption among all the possible speed profiles that complete a given job within the deadline. Given a CPU which has a discrete speed set and whose active power consumption is a (discrete version of) convex function of speed, the previous work showed that the energy-optimal CPU speed control policy is to apply (1) consistently the *MFS* (*Minimum Feasible Speed*, i.e., the lowest constant CPU speed that completes a job within its deadline) if the MFS is higher than the *critical speed* of the CPU (the supported speed that minimizes the CPU energy consumption when a given job is executed consistently at the speed until the completion regardless of the deadline) and supported in the speed set, (2) a combination of only the two immediately neighboring supported speeds of the MFS if the MFS is higher than the critical speed but not supported, or (3) consistently the critical speed if the MFS is lower than the critical speed. This result can be expressed intuitively as follows: Given a well designed CPU (such that its critical speed is the lowest supported speed and its active power consumption is a convex function of speed), the energy-optimal CPU speed control policy to complete a job within its deadline is to keep the CPU speed as close as possible to the MFS (by using the MFS consistently if it is supported, otherwise by using only the two immediately neighboring speeds).

Chapter 3 revealed that, however, achieving the optimal energy-performance trade-off for commodity-OS-based GP systems is impossible in the current DVS paradigm because the

CPU-occupancy-based task scheduling mechanism essentially limits the effectiveness of DVS in reducing CPU energy consumption and makes the prediction of the required future CPU speeds hard. The scheduling mechanism prioritizes tasks based only on their CPU occupancies regardless of their timeliness requirements and, therefore, often assigns a non-urgent task with a higher priority than an urgent task.

This property hinders the CPU speed from being kept close to the energy-optimal constant CPU speed making it demand an unnecessarily high CPU speed in order to complete both of the tasks (because the urgent task is blocked until the higher priority non-urgent task completes) within the urgent task's deadline (then CPU will idle until the deadline of the non-urgent task). This property fundamentally limits the achievable amount of energy savings even with a completely accurate *CMFS* (*Compound Minimum Feasible Speed*, i.e., the lower bound of the CPU speed required to complete a workload on time in the presence of other blocking workloads) prediction because CMFS is not kept close to the CPU energy-optimal constant CPU speed, violating the condition of *ICMFS* (*Ideal CMFS*, i.e., the total CPU utilization. which is the ideal lower bound of the CPU speed required to prevent the system from overload situation) that was proven to be the energy optimal constant CPU speed for a task set in Chapter 3 through an energy-optimal CPU speed control analysis. Moreover, this scheduling mechanism makes the accurate prediction of future CMFSs hard under DVS as it prioritizes tasks based on their CPU occupancies; The CPU speed adjustment affects tasks' CPU occupancies and, in turn, changes the relative priorities among tasks and, after all, keeps the CMFS of a task fluctuating as the blocking tasks vary.

Chapter 3 showed that these problems can be solved by prioritizing tasks according to their timeliness requirements. This DVS-friendly task scheduling, however, must be practical for actual commodity-OS-based GP systems unlike the conventional RT task scheduling and, therefore, is called *DVS-friendly GP task scheduling*. Then Chapter 3 suggested a new DVS paradigm for commodity-OS-based GP systems – the integrated approach of a DVS-friendly GP task scheduling and a DVS mechanism.

Then Chapter 4 described GPSched, a DVS-friendly GP BE task scheduler for Linux.

The GPSched BE task scheduler is an augmentation of the traditional CPU-occupancy-based task scheduler with the capability for a time constraint-based task scheduling. It prioritizes tasks in two steps. In the first step, the scheduler assigns a given task with one of four graded priority ranges based on the attribute of the service that the task is involved with and, therefore, is called 'GPSched' BE task scheduler. The four priority ranges in their order are *HRT (Hard Real Time)* priority range, *DBSRT (Deadline-Based Soft Real Time)* priority range, *RBSRT (Rate-Based Soft Real Time)* priority range, and *NRT (Non Real Time)* priority range mapped respectively to *non-UI (User Interface) hardware control services, UI services, system management services*, and the rest non-time constrained services, the four broad types of service that a commodity-OS-based GP system provides. As Chapter 4 showed, this approach allows the GPSched BE task scheduler to provide an accurate time constrained-based task scheduling (among the tasks having different priority ranges) for commodity-Linux-based GP systems in which the a priori knowledge of tasks' time constraints is not available and precedence constraints exist among tasks. In the second step, the scheduler determines the specific priority of the task within the priority range by applying the traditional CPU-occupancy-based task scheduling. In this way, the GPSched BE task scheduler inherits the advantages of the CPU-occupancy-based task scheduling in quickly responding to interactive applications and evenly sharing CPU among tasks.

Then Chapter 5 described *GPScheDVS*, a realization of the new DVS paradigm for commodity-Linux-based GP systems, which consists of a DVS-friendly GP task scheduler called *GPSched BE task scheduler*, a DVS mechanism called *GPSDVS,* and the *task type detector* that guides other two components by providing them with the information about the attribute of the service that each task is involved with. The task type detector is the key feature that makes GPScheDVS discernable from other possible realizations of the new DVS paradigm.

Chapter 6 presented the experimental results obtained under the real-life usage scenarios of a Linux-based laptop. The results showed a considerable improvement achieved by GPScheDVS over the existing autonomous DVS schemes confirming the effectiveness of the suggested DVS paradigm.

7.2 Summary of contributions

The research described in this thesis contributes to the areas of the autonomous CPU speed control for commodity-OS-based GP PCs in the following ways:

- The impact of task scheduling on a system-level CPU speed control was described.
- The concept of DVS-friendly task scheduling was described.
- GPSched, a DVS-friendly GP best-effort (BE) task scheduler for commodity-Linux-based GP PCs, was described.
- GPScheDVS, an integration of GPSched and a task-based DVS scheme customized to GPSched, was proposed.
- AIDVS, an improved interval-based algorithm, which can save larger amount of energy saving than existing interval-based algorithms, was described.
- DVS-suite, a Linux-based tool to perform the experiments on autonomous DVS schemes in commodity-Linux-based GP mobile PCs, was developed.
- Tweak-PM, a Linux-based tool to probe and set the features of an equipped Intel Pentium-M processor, was developed.

7.3 Future work

The future work to further improve the performance of GPScheDVS includes:

- Combining GPScheDVS-SE and GPScheDVS-CP: The current version of GPScheDVS has two alternative modes – system energy centric (SE) and CPU power centric (CP) modes. As Chapter 6 showed, however, the two modes sacrifice one metric to pursuing another metric. The goal of this future work is to find the optimal point that best trade-off the system energy consumption and the average CPU power consumption.

- Low overhead: A disadvantage of GPScheDVS is its relatively high overhead in time and system energy consumption, even though it improved the QoS of SRT applications and energy consumption over the existing DVS schemes with the overhead. An idea is to integrate GPScheDVS's data structure into the similar data structures that already exists in a commodity Linux kernel.

- Non-CPU components DPM: As Chapter 6 showed, the improvement by GPScheDVS in CPU energy savings is much higher than the improvement in the entire system energy consumption. This is natural because CPU is just a part of the entire system. Thus, if the non-CPU components' energy consumption can be reduced by a dynamic power management scheme which works in coordination with GPScheDVS, the benefit of GPScheDVS will be augmented.

- Within DBSRT priority group, isolation of workload: The final idea to extend GPScheDVS is to improve its time constraint-based task scheduling property by introducing a type of aperiodic server to provide the isolation among DBSRT tasks such as multimedia applications.

Bibliography

[1] IDC, "PC market outlook remains strong, while the shift to emerging regions accelerates, according to IDC," *idc.com*, September 2007. [Online]. Available: http://www.idc.com/getdoc.jsp?pid=23571113&containerId=prUS20870407. [Accessed April 2, 2008].

[2] R. N. Mayo and P. Ranganathan, "Energy Consumption in Mobile Devices: Why Future Systems Need Requirements-Aware Energy Scale-Down," *Internet Systems and Storage Laboratory, HP Laboratories Palo Alto*, August 2003. [Online] Available: http://www.hpl.hp.com/techreports/2003/HPL-2003-167.pdf. [Accessed April 7, 2008].

[3] CNET technical staffs, "CNET Reviews," *cnet.com*, [Online] Available: http://reviews.cnet.com/. [Accessed April 7, 2008].

[4] A. Mahesri and V. Vardhan, "Power consumption breakdown on a modern laptop," *Fourth International Workshop on Power-Aware Computer Systems (PACS)*, pp. 165-180, December 2004.

[5] C.B. Margi, K. Obraczka, and R. Manduchi, "Characterizing System Level Energy Consumption in Mobile Computing Platforms," *2005 International Conference on Wireless Networks, Communications and Mobile Computing*, Vol. 2, pp. 1142-1147, June 2005.

[6] A. Krishnamoorthy, "Minimize IC power without sacrificing performance," *eetimes.com*, July 2004. [Online]. Available: http://www.eetimes.com/showArticle.jhtml?articleID=23901143. [Accessed March 20, 2008].

[7] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," *Seventh International Symposium on High Performance Computer Architecture (HPCA-7)*, pp. 171-182, January 2001.

[8] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM*

Transactions on Architecture and Code Optimization (TACO), Vol. 1, pp. 94-125, March 2004.

[9] M. Bidewell, "Software-Based Dynamic Thermal Management for Linux Systems," *43rd annual southeast regional conference*, Vol. 2, pp. 229-234, March 2005.

[10] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, "Thermal Response to DVFS: Analysis with an Intel Pentium M," *2007 international symposium on Low power electronics and design*, pp. 219-224, August 2007.

[11] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Second ed. Reading, MA: Addison-Wesley Publishing Company, 1993.

[12] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall, "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, Vol. 5, No. Q1, pp. 1-9, February 2001.

[13] P. P. Gelsinger, "Microprocessor for the New Millenium: Challenges, Opportunities, and New Frontiers," *IEEE International Solid-state Circuits Conference*, pp.22-23, February 2001.

[14] R. F. Sechler and G. F. Grohoski, "Design at the system level with VLSI CMOS," *IBM Journal of Research and Development*, Vol. 39, No.112, pp. 5-22, January/March 1995.

[15] S. Sun and P. Tsui, "Limitation of CMOS Supply-Voltage Scaling by MOSFET Threshold-Voltage Variation," *IEEE Journal of Solid-state Circuits*, Vol. 30, No. 8, pp. 947-949, August 1995.

[16] S. Borkar, "Design Challenges of Technology Scaling," *IEEE Micro*, Vol. 19, Issue 4, pp. 23-29, July 1999.

[17] Y. Taur, "CMOS design near the limit of scaling," *IBM Journal of Research and Development*, Vol. 46, No. 2/3, pp. 213-222, March/May 2002.

[18] H. Iwai, “CMOS down scaling and process induced damages,” *8th International Symposium on Plasma- and Process-Induced Damage*, pp. 1-11, April 2003.

[19] M. Bohr, “High Performance Logic Technology and Reliability Challenges,” *irps.org*, 2003. [Online]. Available: http://www.irps.org/03-41st/Intel_IRPS03_Bohr_Keynote.pdf. [Accessed March 20, 2008].

[20] S. Rusu, “Trends and Challenges in High High-Performance Performance Microprocessor Design,” *eda.org*, 2004. [Online]. Available: http://www.eda.org/edps/edp04/submissions/presentationRusu.pdf. [Accessed March 20, 2008].

[21] L. Benini, A. Bogliolo, and G. De Micheli, “A Survey of Design Techniques for System-Level Dynamic Power Management,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 8, No. 3, pp. 299-316, June 2000.

[22] K. Lahiri, S. Dey, D. Panigrahi, and A. Raghunathan, “Battery-Driven System Design: A New Frontier in Low Power Design,” *15th International Conference on VLSI Design and the Seventh Asia and South Pacific Design Automation Conference (ASPDAC)*, pp.261-267, 2002.

[23] O. S. Unsal and I. Koren, “System-Level Power-Aware Design Techniques in Real-Time Systems,” *Proceedings of the IEEE*, Vol. 91, No. 7, pp. 1055-1069, July 2003.

[24] V. Venkatachalam and M. Franz, “Power Reduction Techniques for Microprocessor Systems,” *ACM Computing Surveys*, Vol. 37, No.3, pp. 195-237, September 2005.

[25] C. S. Ellis, *Controlling Energy Demand in Mobile Computing System (Synthesis Lectures on Mobile and Pervasive Computing)*, First ed. San Rafael, CA: Morgan & Claypool Publishers, 2007.

[26] W. Liu, “Techniques for Leakage Power Reduction in Nanoscale Circuits: A Survey,” *orbit.dtu.dk*, 2007. [Online]. Available: http://orbit.dtu.dk/getResource?recordId=200963&objectId=1&versionId=1. [Accessed March 20, 2008].

[27] AMD Inc. technical staff, *AMD-K6-2 Processor Data Sheet*, AMD Inc., February 2000.

[28] Intel cooperation technical staff, "Intel 80200 Processor," *intel.com*. [Online]. Available: http://www.intel.com/design/iio/80200.htm. [Accessed March 20, 2008].

[29] Intel cooperation technical staff, *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*, White Paper, Intel cooperation, March 2004.

[30] Intel cooperation technical staff, *Intel Pentium M Processor on 90 nm Process with 2-MB L2 Cache*, Intel cooperation, January 2006.

[31] Intel cooperation technical staff, *IA-32 Intel Architecture Software Developer's Manual: Documentation Changes*, Intel cooperation, March 2006.

[32] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "The limit of dynamic voltage scaling and insomniac dynamic voltage scaling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 13, Issue 11, pp. 1239-1252, November 2005.

[33] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," *41st Design Automation Conference*, pp. 868-873, June 2004.

[34] R. Jejurikar, C. Pereira, R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," *41st Design Automation Conference*, pp. 275-280, June 2004.

[35] K. Nose, M. Hirabayashi, H. Kawaguchi, S. Lee, and T. Sakurai, "V_{TH}-Hopping Scheme for 82% Power Saving in Low-Voltage Processors," *IEEE Conference on Custom Integrated Circuits*, pp. 93-96, May 2001.

[36] C. Kim and K. Roy, "Dynamic Vth scaling scheme for active leakage power reduction," *The conference on Design, automation, and test in Europe (DATE'02)*, pp. 163–167, April 2002.

[37] J. T. Kao, M. Miyazaki, and A. P. Chandrakasan, “A 175-mV Multiply-Accumulate Unit Using an Adaptive Supply Voltage and Body Bias Architecture,” *IEEE Journal of Solid-State Circuits*, Vol. 37, No. 11, pp.1545-1554, November 2002.

[38] D. Duarte, N. Vijaykrishnan, M. Irwin, and Y-F. Tsai, “Impact of technology scaling and packaging on dynamic voltage scaling techniques,” *15th Annual IEEE International ASIC/SOC Conference*, pp. 244-248, September 2002.

[39] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, “Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads,” *2002 International Conference on Computer-Aided Design (ICCAD)*, pp. 721-725, November 2002.

[40] L. Yan, J. Luo, and N. K. Jha, “Joint Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-Time Embedded Systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 7, pp. 1030-1041, July 2005.

[41] F. Yao, A. Demers, and S. Shenker, “A scheduling model for reduced CPU energy,” *36th Annual Symposium on Foundations of Computer Science*, pp. 374-382, October 1995.

[42] T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors,” *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 197-202, 1998.

[43] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, “Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads,” *2002 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pp. 721–725, 2002.

[44] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar, “Critical power slope: Understanding the runtime effects of frequency scaling,” *International*

Conference on Supercomputing (ICS), pp. 35–44, June 2002.

[45] S. Irani, S. Shukla, and R. Gupta, "Online strategies for dynamic power management in systems with multiple power-saving states," *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 2, No. 3, pp. 325–346, August 2003.

[46] J. R. Lorch and A. J. Smith, "PACE: A new approach to dynamic voltage scaling," *IEEE Transactions on Computers*, Vol. 53, No. 7, pp. 856–869, July 2004.

[47] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," *Design Automation Conference (DAC)*, pp. 275–280, 2004.

[48] R. Rao and S. Vrudhula, "Energy Optimal Speed Control of Devices with Discrete Speed Sets," *42nd annual conference on Design automation (DAC)*, pp.901-904, June 2005.

[49] G. M. Candea and M. B. Jones, "Vassal: Loadable Scheduler Support for Multi-Policy Scheduling," *Second USENIX Windows NT Symposium*, Vol. 2, pp.157-166, August 1998.

[50] A. Silberschatz, G. Gagne, and P. B. Galvin, *Operating System Concepts*, Sixth ed. New York: John Wiley & Sons, Inc., 2003.

[51] N. Audsley and A. Burns, "Real-Time System Scheduling," University of York - Department of Computer Science, Report YCS 134, 1990.

[52] K. Ramamritham and J. A. Stankovic, "Scheduling algorithms and operating systems support for real-time systems," *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 55-67, January 1994.

[53] J. A. Stankovic, "Real-Time and Embedded Systems," *ACM Computing Surveys*, Vol. 28, No. 1, pp. 205-208, March 1996.

[54] K. Gopalan, "Real-time support in general purpose operating systems," Stony Brook

University - Department of Computer Science, ECSL Technical Report TR92, 2001.

[55] L. Sha, T. Abdelzaher, K. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. Mok, "Real Time Scheduling Theory: A Historical Perspective," *Real-Time Systems*, Volume 28, Numbers 2-3, pp. 101-155, November 2004.

[56] G. Buttazzo, "Rate Monotonic vs. EDF: Judgment Day," *Springer Real-Time Systems*, Vol. 29, Issue 1, pp. 5-26, January 2005.

[57] G. Buttazzo, "Research Trends in Real-Time Computing for Embedded Systems," *ACM SIGBED Review*, Vol. 3, Issue 3, pp. 1-10, July 2006.

[58] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of ACM*, Vol. 20, Issue 1, pp. 46-61, January 1973.

[59] M. Spuri and J. A. Stankovic, "How to integrate precedence constraints and shared resources in real-time scheduling," *IEEE Transactions on Computers*, Vol. 43, No. 12, pp. 1407-1412, December 1994.

[60] L. Abeni and G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," 19th IEEE Real-Time Systems Symposium, pp. 4-13, December 1998.

[61] L. Abeni, G. Lipari, and G. Buttazzo, "Constant bandwidth vs. proportional share resource allocation," IEEE International Conference on Multimedia Computing and Systems, Vol. 2, pp. 107-111, July 1999.

[62] J. Regehr, M. B. Jones, and J. A. Stankovic, "Operating System Support for Multimedia: The Programming Model Matters," Technical Report MSR-TR-2000-89, Microsoft Research, December 2000.

[63] S. Banachowski and S. A. Brandt, "Toward a taxonomy of time-constrained applications," 24th IEEE Real-Time Systems Symposium, pp. 3-6, December 2003.

[64] Red Hat technical staffs, *Red Hat Linux 9 Shrike OS*. [CD-ROM]. Raleigh, NC: Red Hat Cooperation, 2003.

[65] A. Azevedo, I. Issenin, and R. Cornea, "Profile-based Dynamic Voltage Scheduling using Program Checkpoints," *2002 Conference on Design, Automation and Test in Europe*, pp. 168-175, March 2002.

[66] J. Pouwelse, K. Langendoen, and H. J. Sips, "Application-Directed Voltage Scaling," IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 5, pp. 812-826, Octorber 2003.

[67] X. Liu, P. Shenoy, and M. Corner, "Chameleon: Application Level Power Management with Performance Isolation," *13th annual ACM international conference on Multimedia*, pp. 839-848, November 2005.

[68] C. Hsu and U. Kremer, "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction," *ACM SIGPLAN 2003 conference on Programming language design and implementation*, pp. 38-48, June 2003.

[69] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *First Symposium of Operating Systems Design and Implementation*, November 1994.

[70] V. Pallipadi and A. Y. Starikovskiy, "The Ondemand Governor: Past, Present, and Future," *2006 Linux Symposium*, pp. 223-238, Vol. 2, July 2006.

[71] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," *First International Conference on Mobile Computing and Networking*, pp. 13-25, November 1995.

[72] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *International Symposium on Low Power Electronics and Design*, August 1998.

[73] D. Grunwald, P. Levis, K. I. Faras, C. B. Morrey III, and M. Neufeld, “Policies for dynamic clock scheduling, “ *Fourth Symposium on Operating Systems Design and Implementation*, October 2000.

[74] K. Flautner, S. Reinhardt, and T. Mudge, “Automatic performance setting for dynamic voltage scaling,” *Seventh annual international conference on mobile computing and networking*, pp. 260-271, May 2001.

[75] K. Flautner and T. Mudge, “Vertigo: Automatic performance-setting for Linux,” *Fifth Operating Systems Design and Implementation (OSDI)*, pp. 105-116, December 2002.

[76] J. R. Lorch and A. J. Smith, “Using user interface event information in dynamic voltage scaling algorithms,” *11th international symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2003)*, pp. 46-55, October 2003.

[77] J. R. Lorch and A. J. Smith, “Operating system modifications for task-based speed and voltage scheduling,” *First International Conference on Mobile Systems, Applications, and Services (MobiSys)*, May 2003.

[78] S. Gurun and C. Krintz, “Autodvs: an automatic, general-purpose, dynamic clock scheduling system for hand-held devices,” *Fifth ACM International Conference on Embedded Software (EMSOFT’05)*, pp. 218-226, September 2005.

[79] F. Xie, M. Martonosi, and S. Malik, “Efficient Behavior driven Runtime Dynamic Voltage Scaling Policies,” *Third IEEE ACM IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 105-110, September 2005.

[80] P. Pillai and K. G. Shin, “Real-time dynamic voltage scheduling for low-power embedded operating systems,” *18th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 89-102, October 2001.

[81] F. Gruian, “Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and

DVS Processors," *2001 international symposium on Low power electronics and design*, pp. 46-51, September 2001.

[82] H. Aydin, D. Mosse, and P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks," *IEEE Transactions on Computers*, Vol. 53, No. 5, pp. 584-600, May 2004.

[83] W. Yuan and K. Nahrstedt, "Integration of dynamic voltage scaling and soft real-time scheduling for open mobile systems," *12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 2002.

[84] G. Buttazzo, "Achieving scalability in real-time systems," *IEEE Computer*, Vol. 39, Issue 5, pp. 54-59, May 2006.

[85] B. Shneiderman, "Designing the User Interface: Strategies for Effective Human-Computer Interaction," 2nd ed. Reading, MA: Addison-Wesley Publishing Company, 1992.

[86] R. VanRullen and C. Koch, "Is perception discrete or continuous?," *TREND in Cognitive Sciences*, Vol. 7, No. 5, pp. 207-213 May 2003.

[87] Microsoft Cooperation, "PC Video Synchronization and Playback," *microsoft.com*, December 2001. [Online]. Available: http://www.microsoft.com/whdc/archive/VidSynch.mspx. [Accessed February 2, 2008].

[88] D. H. Nowlin, "Video Frame Display Synchronization," *intel.com*, September 2005. [Online]. Available: http://softwarecommunity.intel.com/articles/eng/2644.htm. [Accessed February 18, 2008].

[89] Hewlett Packard Cooperation, "HP 54600B, HP54601B, HP 54602B, and HP 54603B Oscilloscopes User and Service Guide," Publication number 54600-97021, November 1997.

[90] LEM cooperation technical staff, *Current Probe Model PR30*, LEM cooperation.

[91] D. Nash, “An Intrusion Detection System for Battery Exhaustion Attacks on Mobile Computers,” Master’s thesis, Virginia Polytechnic Institute and State University, 2005.

[92] Agilent cooperation technical staff, *Agilent 3458 multimeter data sheet*, Agilent cooperation.

[93] S. Kim and T. Martin, “DIP: A Double-interval-based Dynamic Voltage Scaling Scheme for Dynamic Priority-based Task Scheduling Systems”, *Proceedings of GLSVLSI’05*, April 2005.

[94] S. Kim and T. Martin, “GPSDVS: An Improved Task-Based Dynamic Voltage Scaling Scheme for General-Purpose Systems”, *Proceedings of SOCC 2005*, September 2005.