December 2014

ADAPTIVE POWER MANAGEMENT FOR COMPUTERS AND MOBILE DEVICES

Hao Shen
Syracuse University

Abstract

Power consumption has become a major concern in the design of computing systems today. High power consumption increases cooling cost, degrades the system reliability and also reduces the battery life in portable devices. Modern computing/communication devices support multiple power modes which enable power and performance tradeoff. *Dynamic power management* (DPM), *dynamic voltage and frequency scaling* (DVFS), and dynamic task migration for workload consolidation are system level power reduction techniques widely used during runtime. In the first part of the dissertation, we concentrate on the dynamic power management of the personal computer and server platform where the DPM, DVFS and task migrations techniques are proved to be highly effective. A hierarchical energy management framework is assumed, where task migration is applied at the upper level to improve server utilization and energy efficiency, and DPM/DVFS is applied at the lower level to manage the power mode of individual processor. This work focuses on estimating the performance impact of workload consolidation and searching for optimal DPM/DVFS that adapts to the changing workload. Machine learning based modeling and reinforcement learning based policy optimization techniques are investigated.

Mobile computing has been weaved into everyday lives to a great extend in recent years. Compared to traditional personal computer and server environment, the mobile computing environment is obviously more context-rich and the usage of mobile computing device is clearly imprinted with user's personal signature. The ability to learn such signature enables immense potential in workload prediction and energy or battery life management. In the second part of the

dissertation, we present two mobile device power management techniques which take advantage of the context-rich characteristics of mobile platform and make adaptive energy management decisions based on different user behavior. We firstly investigate the user battery usage behavior modeling and apply the model directly for battery energy management. The first technique aims at maximizing the quality of service (QoS) while keeping the risk of battery depletion below a given threshold. The second technique is an user-aware streaming strategies for energy efficient smartphone video playback applications (e.g. YouTube) that minimizes the sleep and wake penalty of cellular module and at the same time avoid the energy waste from excessive downloading.

Runtime power and thermal management has attracted substantial interests in multi-core distributed embedded systems. Fast performance evaluation is an essential step in the research of distributed power and thermal management. In last part of the dissertation, we present an FPGA based emulator of multi-core distributed embedded system designed to support the research in runtime power/thermal management. Hardware and software supports are provided to carry out basic power/thermal management actions including inter-core or inter-FPGA communications, runtime temperature monitoring and dynamic frequency scaling.

ADAPTIVE POWER MANAGEMENT FOR COMPUTERS AND MOBILE DEVICES

by

Hao Shen

B.S., Southeast University, 2008

Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University

December 2014

Table of Contents

Chapter 1 Introduction 1

1.1 System level dynamic power management for general purpose computing 3

1.2 Battery aware power management for mobile computing 5

1.3 Dissertation Contributions 6

Chapter 2 Workload Consolidation and Adaptive Power Management for General Purpose Computing Systems 10

2.1 Chip Multiprocessor Performance Modeling for Contention Aware Task Migration and Frequency Scaling 10

2.1.1 Introduction 10

2.1.2 Motivational observations 15

2.1.3 Model construction 25

2.1.4 Model directed task mapping 31

2.1.5 Experimental Results 33

2.1.6 Conclusions 45

2.2 Learning Based DVFS and DPM for CPU and Peripheral devices 45

2.2.1 Introduction 45

2.2.2 Related Works 51

2.2.3 General Architecture of Q-learning based Power Management 54

2.2.4 Learning based Power Management for Peripheral Devices 59

2.2.5 Learning based CPU Power Management 73

2.2.6 Experimental results and analysis 78

2.2.7 Conclusions 99

2.3 Chapter Summary 99

Chapter 3 Adaptive Battery Management for Mobile Device 101

3.1 Battery Aware Stochastic QoS Boosting in Mobile Computing Device 101

3.1.1 Introduction 101

3.1.2 Battery Level Prediction Using Neural Networks 103

3.1.3 Stochastic Control for Smartphone Energy Management 106

3.1.4 Implementation and Evaluation 111

3.1.5 Conclusions 116

3.2 User-Aware Energy Efficient Streaming Strategy for Smartphone Based Video Playback Application 117
3.2.1 Introduction 117
3.2.2 Background 119
3.2.3 User Behavior Modeling and Downloading Strategy Optimization 121
3.2.4 Experimental Results 126
3.2.5 Conclusions 131
3.3 Chapter Summary 131
Chapter 4 An FPGA-based Distributed Computing System with Power and Thermal Management Capabilities 133
4.1 Introduction 133
4.2 Altera Nios II Embedded Evaluation Kit (NEEK) 136
4.3 System Architecture 137
4.3.1 Single FPGA multi-core system 137
4.3.2 Multi-FPGA distributed system 140
4.4 Frequency Scaling and Temperature Monitoring 140
4.4.1 Glitch free clock switching 141
4.4.2 Monitor the processor's temperature 144
4.5 Experiments 145
4.5.1 Characterization of communication latency 146
4.5.2 Parallel matrix multiplication 148
4.5.3 Evaluation of the temperature sensor and clock selection module 150
4.6 Conclusion 152
Chapter 5 Conclusions 153
Bibliography 155

List of Figures

Figure 1-1 Data center power consumption trend 1
Figure 1-2 Performance and battery capacity trend 2
Figure 1-3 Power Consumption of Opteron X4 processor 4
Figure 2-1 Performance sensitivity to resource contention and frequency scaling 17
Figure 2-2 Relation between target performance and LLC miss of its SMT neighbor 22
Figure 2-3 Percentage performance boost of target process when its SMT neighbor changes from lbm to gamess 24
Figure 2-4 Percentage performance boost of target process when its SMT neighbor changes from lbm to sleep task 24
Figure 2-5 Estimated performance is highly correlated to actual performance 29
Figure 2-6 Performance for all workloads 36
Figure 2-7 Energy and EDP of model_full and capping 39
Figure 2-8 Performance of model predictive task migration 41
Figure 2-9 Model directed hierarchical power management 44
Figure 2-10 Illustration of system under power management. 57
Figure 2-11 State transition diagram of SP, SR and SQ models. 60
Figure 2-12 Pseudo code for Q-learning power manager using the VSS technique. 66
Figure 2-13 Level 1 neural network 68
Figure 2-14 Relation between power and $lg\ \lambda$ for a given workload. 71
Figure 2-15 Block diagram of the power control flow of the Q-learning power manager. 73
Figure 2-16 Qualitative illustration of the relation between CPU temperature, performance, energy and clock frequency. 77
Figure 2-17 Response of Q-learning power manager to synthetic trace 1. 80
Figure 2-18 Response of Q-learning power manager to synthetic trace 2. 81
Figure 2-19 Three consecutive days' requests from HP hard disk traces. 83
Figure 2-20 Power/Latency tradeoff curves for workload. (a)HP-1; (b)HP-2; (c)HP-3; (d)Desktop-1; (e)Desktop-2 85
Figure 2-21 Q-value for observation-action pair(000,0) 86
Figure 2-22 Power/Latency tradeoff curves for (a) HP workloads (b) desktop workloads when Tbe= 8 seconds. 89
Figure 2-23 Relative average power deviation from user constraints. 90

Figure 2-24 Percentage MSE of instant power versus user constraints. 91
Figure 2-25 Relative average latency deviation for latency constrained power management. 92
Figure 2-26 Energy, temperature and performance results of Q-learning algorithm with constraints and expert-based algorithm without constrains: (a) energy versus performance; (b) temperature versus performance 98
Figure 3-1 Average prediction error. 105
Figure 3-2 Battery level at the beginning of 100 battery charges 105
Figure 3-3 Prediction error vs. time to next battery charge 106
Figure 3-4 Battery change histogram 108
Figure 3-5 Temporal correlation of phone usage. 109
Figure 3-6 MDP training process 109
Figure 3-7 actual depletion rate vs. depletion tolerance 114
Figure 3-8 QoS Boosts vs. (a) depletion tolerance (b) actual depletion rate 114
Figure 3-9 MDP violation percentage histogram 115
Figure 3-10 The radio resource state machine of 3G interface 120
Figure 3-11 Power trace and network activities during YouTube playback 120
Figure 3-12 Adjusting the PDF based on obtained usage information 126
Figure 3-13 Comparison of wasted seconds for different users and different buffering strategies 130
Figure 3-14 Buffering points of user 1 by the GMM approach 131
Figure 4-1 Hierarchical architecture 137
Figure 4-2 Processor configuration 138
Figure 4-3 Topology of on-chip inter-processor connections 139
Figure 4-4 Clock generation block 142
Figure 4-5 Simulation result of clock transition process 142
Figure 4-6 Temperature sensor and processor 145
Figure 4-7 Inter-FPGA communication latency 147
Figure 4-8 Inter-processor communication latency 147
Figure 4-9 Overall task execution time for processors with and without embedded multipliers 149
Figure 4-10 Computation and communication time for processors with embedded multipliers 149
Figure 4-11 Processor's temperature change under different working frequency 151

List of Tables

Table 2-1 Top 9 selected events sorted by its correlation to the performance for PEFS model... 27
Table 2-2 Accuracy of 3 different PEFS models 28
Table 2-3 Top 9 selected features sorted by its correlation to the performance for PPTM model 31
Table 2-4 Workloads used in the evaluation..... 37
Table 2-5 Average performance, power and violations..... 45
Table 2-6 Input selection vs. prediction error. 70
Table 2-7 Characteristics of Service Provider 78
Table 2-8 Characteristics of Different Reference Policies 80
Table 2-9 Characteristics of Workload Traces. 83
Table 2-10 Constraining performance and temperature. 94
Table 3-1 Comparison of user behavior models..... 127

Chapter 1 Introduction

Energy conservation and power management have become two of the most important challenges in today's computing systems, including general purpose computers and mobile devices.

It is estimated that datacenters consume about 2% of all US electricity, with an unsustainable annual growth of 15%. Datacenter power is projected to be over 8% of US power by 2020, and its carbon emission by that time will exceed those of the airlines by 2020. As Figure 1-1 shows, the amount of server shipments and internet traffics increase almost 4 times from 2006 to 2012, while the power consumption of data centers increases more than 8 times from 2000 to 2014. Improved operation techniques in data center are needed to make it more economically and ecologically sustainable and scalable. High power also means high temperature, which has many other adverse side effects such as decrease of system reliability, performance, increase of leakage power, cooling cost and etc..

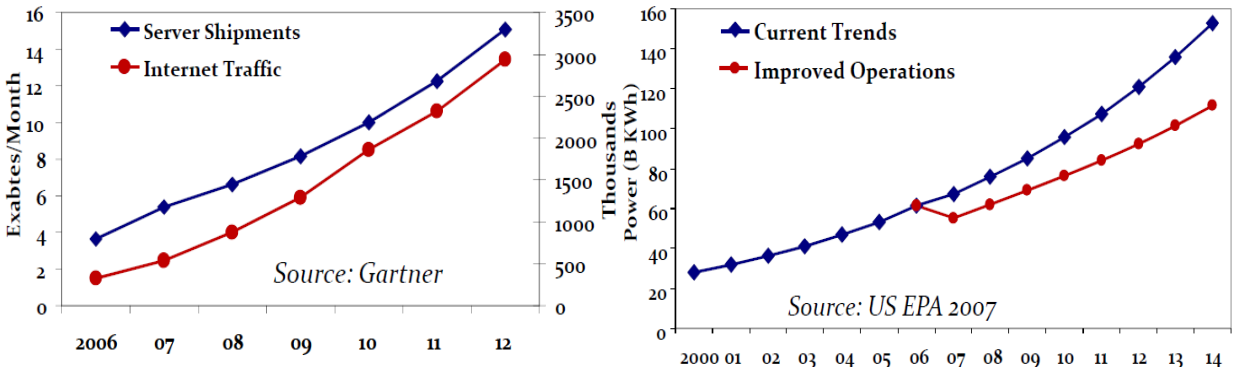


Figure 1-1. Data center power consumption trend[135]

Energy conservation is even more important to battery operated mobile computing devices (e.g., smart phone). Battery life has continuously been one of the top critical factors that affect user satisfaction. While mobile computing has become indispensable in everyday lives for communication, sensing, controlling and entertainment, the increasing complexity of hardware and applications in the mobile devices greatly outpaces the development of battery technology. Figure 1-2 shows that from Samsung Galaxy S to S3, the CPU performance increases 5.9 times, while the battery capacity only increases 1.4 times. To narrow such gap, there is an urgent demand to manage the energy usage and battery lifetime for mobile devices.

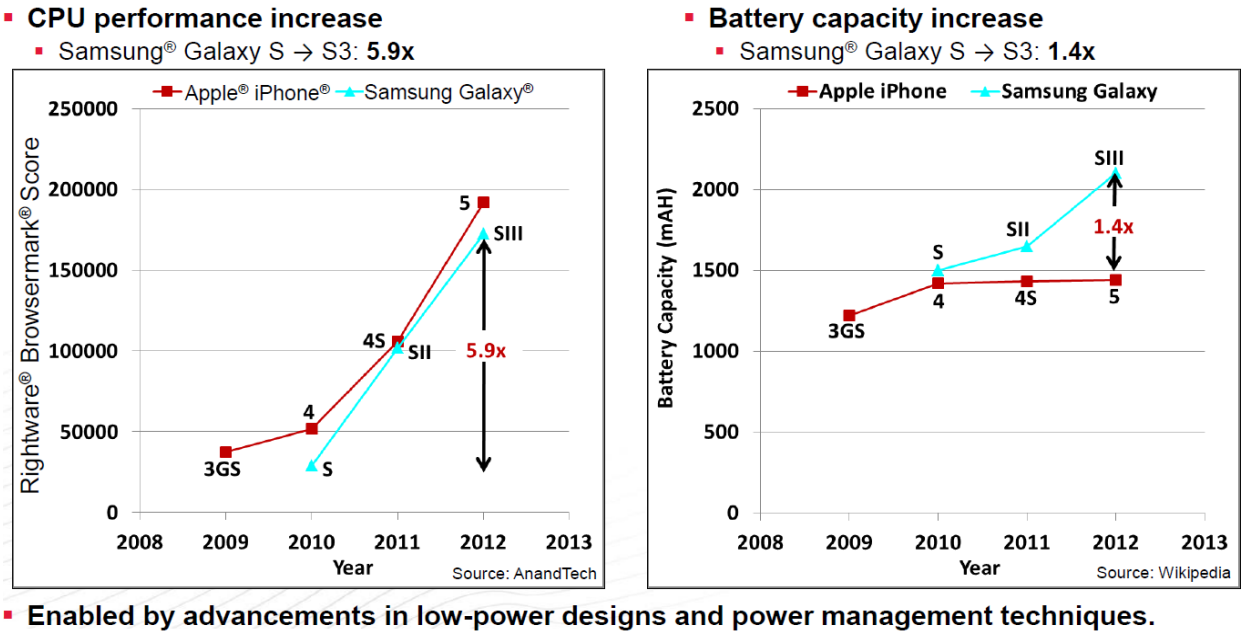


Figure 1-2. Performance and battery capacity trend[136]

In addition to increased energy dissipation, excessive power consumption has also become a major roadblock in the design of computing systems. High power consumption increases die temperature, which further reduces performance, accelerates the electromigration, and raises the leakage power.

Power consumption and performance of an optimized design are contradictory design metrics. Gaining one will lead to the sacrifice of the other. The ultimate goal of power management is to minimize power consumption while still maintaining certain performance levels required by different users on different platforms.

1.1 System level dynamic power management for general purpose computing

Modern computing/communication devices support multiple power modes, which enable power and performance tradeoff. *Dynamic power management* (DPM) has proven to be an effective technique for power reduction at the system level. It selectively shuts-off or slows-down system components that are idle or underutilized. The power manager needs to make wise decisions on when to put the devices into which power mode. *Dynamic voltage and frequency scaling* (DVFS) is another technique that has been widely used in modern processors for energy reduction or temperature control by dynamically changing the working frequency and voltage of the processor. Both DVFS and DPM provide a set of control knobs for runtime power management. From this perspective, they are fundamentally the same. While the DVFS is usually found as the power control knob for CMOS digital ICs, such as micro-controllers or microprocessors, during the active time; the DPM is usually for the peripheral devices, such as the hard disk drives and network interface, or for microprocessors running interactive applications accompanied with long idle intervals.

One of the major issues in today's general purpose computing devices is the lack of energy proportionality. Figure 1-3 shows the power consumption of Intel Opteron X4 processor. As we can see, even if the CPU utilization is approaching to 0%, the processor still consumes about 60%

of its peak power consumption. Since high utilization corresponds to high energy efficiency, workload consolidation is widely used in today's data centers to improve server utilization and energy efficiency. Through dynamic task migration, the workloads of a few servers are increased, which create more power management opportunities for the other servers.

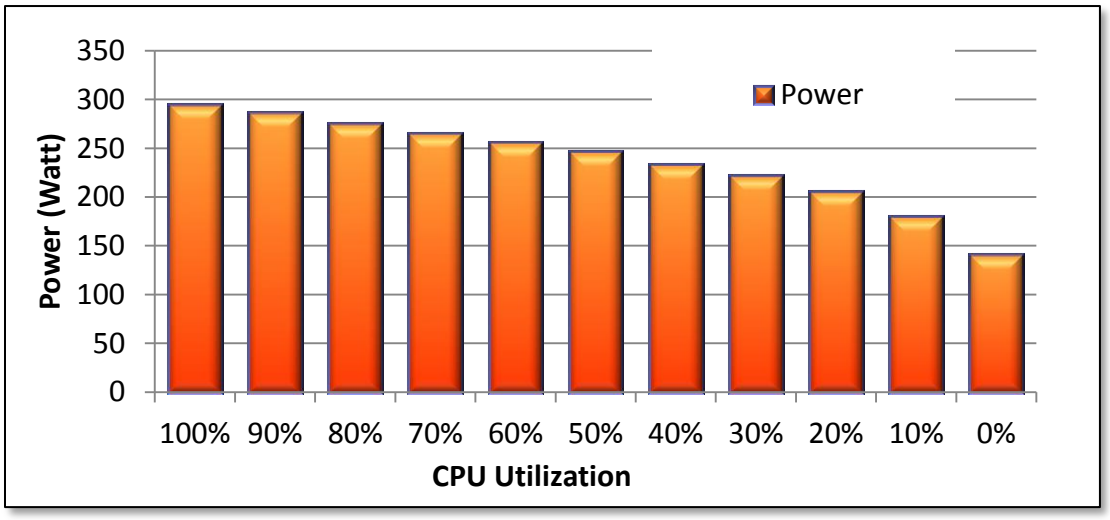


Figure 1-3. Power Consumption of Opteron X4 processor

In a hierarchical power management framework, the upper level is usually virtual machine management that performs workload consolidation, while the lower level is usually DPM or DVFS. What is the performance impact of those system level power management techniques, how to achieve the optimal tradeoff between power consumption and performance are two questions that need to be answered.

Robust power management must consider the uncertainty and variability that come from the environment, the application and the hardware. For example, the workload of a complex system is usually unpredictable as it strongly depends on the nature of the application, the input data and the user context. The workload variation changes the device usage pattern and has the

most significant impact on the system speed and power consumption. The contention of shared resources such as buses or I/Os in an MPSoC also increases the variability of hardware response time for communication and computation. Furthermore, the process, voltage, and temperature (PVT) variation results in a large fluctuation in hardware performance and power consumption. Therefore, statically optimized resource and power management policies are not likely to achieve the best performance when the input characteristics change. The ability to observe, learn and adapt to different hardware systems and different working environments is essential for a power management controller.

Tasks demand different resources at different levels. In a parallel computing system with dynamic workload activities, such demand varies from task to task and from time to time. The level of resource contention is determined by the selection of "co-runners" on the same core or the same processor. Task migration, which re-distributes tasks across multiple cores/processors during runtime, may effectively mitigate resource contention. Searching for the best task distribution is a non-trivial problem and if not handled properly, will lead to performance degradation instead of performance improvement.

1.2 Battery aware power management for mobile computing

The goal of power management for a battery powered mobile computing device is to minimize the chance of battery depletion while providing high quality of service. In addition to computing communication activities, battery charging and discharging activities as well as battery state of charge should also be included as part of the decision making framework.

The workload on a mobile computing device and its battery charging activities have clear imprint of user's personality. The power management controller has to adapt to the usage pattern of different users. Proper learning and decision making techniques must be developed.

Mobile device's context is the internal and external environment with which it runs. The context can be collected from sensors such as location, mobility, environmental light brightness and etc.. It also can be inferred from phone's interval working state such as the application launch history, call time, time of the day and etc. The context of a mobile device provides rich information that helps workload prediction and power management decision making. While traditional controller selects power management actions solely based on the status of the computing device, the power management of a mobile computing device may benefit from available context information collected from various sensors.

1.3 Dissertation Contributions

In the first part of the dissertation, we will concentrate on the dynamic power management of the personal computer and server platform where the hierarchical management frame work is adopted. The upper level performs virtual machine management for workload consolidation, and the lower level is performs adaptive DPM or DVFS.

Workload consolidation is usually performed in datacenters to improve server utilization for higher energy efficiency. One of the key issues related to workload consolidation is the contention for shared resources such as last level cache, main memory, memory controller, etc, which may degrade system performance. Furthermore, we have found that the degree of resource contention of a system affects its performance sensitivity to CPU frequency. There is a close coupling between the decision of workload consolidation at upper level and effectiveness of

DVFS in the lower level. Without detailed architecture level information, the complex relationship between contention, frequency and performance can only be retrieved through learning and modeling. In this thesis,, we apply machine learning techniques to construct a model for chip multiprocessor (CMP) Performance Estimation under Fixed workload Scheduling (PEFS). It quantifies performance degradation of target process caused by resource contention and frequency scaling for current CMP workload with the assumption of a fixed task mapping. The model is further generalized for performance prediction with task migration (PPTM), which predicts the performance degradation under new task mappings generated by potential intra-processor task migration. The inputs of both models are readings from Performance Monitoring Units (PMU) screened using standard feature selection technique. Both models are tested on an SMT-enabled chip multi-processor with 10~20% estimation error in average. Experimental results show that, guided by the performance model, better task migration and DVFS decisions can be made to explore tradeoffs between performance and energy dissipation.

We also present a novel on-line power management technique based on model-free constrained reinforcement learning (Q-learning). The proposed learning algorithm requires no prior information of the workload and it dynamically adapts to the environment to achieve autonomous power management. We focus on the power management of the peripheral device and the microprocessor, two of the basic components of a computer. Due to their different operating behaviors and performance considerations, these two types of devices require different designs of Q-learning agent. We will discuss system modeling and cost function construction for both types of Q-learning agent. Enhancement techniques are also proposed to speed up the convergence and better maintain the required performance (or power) constraint in a dynamic system with large variations. Compared with the existing machine learning based power

management techniques, the Q-learning based power management is more flexible in adapting to different workload and hardware and provides a wider range of power-performance tradeoff.

In the second part of the dissertation, we present our work on mobile device power management, which considers battery status and user behavior.

We first extend the learning based power management framework for battery energy management. The goal is to maximize the quality of service (QoS) provided by the mobile device (i.e., smartphone), while keep the risk of battery depletion below a given threshold. A Markov Decision Process (MDP) is learned from history user behavior. The optimal management policy is solved using linear programing. Simulations based on real user traces validate that, compared to existing battery energy management techniques, the stochastic control performs better in boosting the mobile devices' QoS without significantly increasing the chance of battery depletion.

A smartphone consists of different energy-consuming components from processor, LCD screen, to WiFi card, cellular interface and etc. Although the percentage of power consumption of different components varies in different applications and different system settings, the cellular interface consumes more than 50 percent of total power consumption of the smartphone [107], when it is used as the network interface. We further present a methodology to design user-aware streaming strategies for energy efficient smartphone video playback. The goal is to manage the streaming process to minimize the sleep and wake penalty of cellular module and at the same time avoid the energy waste from excessive downloading. The problem is modeled as a stochastic inventory system, where the real length of video playback requested by the smartphone user is considered as demand that follows a stochastic process. Through user

behavior analysis, a Gaussian Mixture Model (GMM) is constructed to predict the user demand in video playback, and then an energy efficient video downloading strategy will be determined progressively during the playback process. Experimental results show that compared to a static downloading strategy that is optimized by exhaustive trail, our method can reduce the wasted energy by 10 percent in average.

In the last part of the dissertation, we present an FPGA based emulator of distributed multi-core embedded system designed to support the research in runtime power/thermal management. The system consists of multiple FPGAs connecting through Ethernet with each FPGA configured as a multi-core system. Hardware and software supports are provided to carry out basic power/thermal management actions including inter-core or inter-FPGA communications, runtime temperature monitoring and dynamic frequency scaling.

Chapter 2 Workload Consolidation and Adaptive Power Management for General Purpose Computing Systems

2.1 Chip Multiprocessor Performance Modeling for Contention Aware Task Migration and Frequency Scaling

2.1.1 Introduction

It has been pointed out [56] that the server energy efficiency reduces super-linearly as its utilization goes down. Due to the severe lack of energy proportionality in today's computers, workload consolidation is usually performed in datacenters to improve server utilization for higher energy efficiency. When used together with power management on idle machines, this technique can lead to significant power savings [55].

Today's high-end servers have multiple processing units that consist of several symmetric multiprocessing (SMP) cores. Each physical core also comprises more than one logical cores enabled by the simultaneous multithreading (SMT) technique. One of the key issues related to workload consolidation is performance degradation due to the contentions for shared resources. At SMP level the shared resources include main memory, last level cache, memory

controller, etc. At SMT level, the shared resources also include execution modules such as instruction issue ports, ALU, branch target buffers, low level caches, etc. [59][60]. The degree of performance degradation is a function of the resource usage of all processes that are co-running and hence is hard to predict. Even if we can measure the execution time of an application accurately, there is no direct way to tell how much degradation that the process went through unless we have a reference copy of the same application running alone on an identical hardware machine.

Dynamic voltage and frequency scaling (DVFS) [66][67][68]is another effective low power technique that has widely been used. Compared to workload consolidation and runtime power management, DVFS provides finer adjustment in power-performance trade-offs with much less control overhead. In a hierarchical power management framework [55][62], the upper level is usually virtual machine management that performs workload consolidation, while the lower level is usually voltage and frequency scaling. Due to the gap between CPU and memory speed, the performance impact of DVFS is not linearly proportional to the scale of frequency reduction [66][67][68]. Different applications have different *sensitivity* to frequency scaling. A memory intensive application usually suffers less performance degradation from DVFS than a CPU intensive one, as the CPU speed is no longer the performance bottleneck. The same can be expected for many systems running multiple consolidated workloads. As their performance constrained by the contention for shared resources, power reduction can be achieved by applying DVFS without significant performance impact. However, similar to systems with resource contention, it is hard to directly tell an application's performance sensitivity to frequency scaling without having a reference copy to compare with during runtime.

Tasks demand different resources at different levels. In a parallel computing system with dynamic workload, such demand varies from task to task and from time to time. The level of resource contention is affected by the selection of "co-runners" on the same core or the same processor. Task migration, which re-distributes tasks across multiple cores/processors during runtime, may effectively mitigate resource contention. Searching for the best task distribution is a non-trivial problem and if not handled properly, will lead to performance degradation instead of performance improvement.

Performance degradation should be hidden from the customers, especially in a cloud environment, where the quality of service (QoS) is specified by the service level agreement (SLA) between service providers and customers and charges are determined based upon usage or reservation of cloud resources. How to guarantee the service level in a system that performs workload consolidation and DVFS for power control is an urgent research problem [71][73].

Previous works studied how to optimize process scheduling to mitigate the resource contention ([58], [61]~[65], [68]~[70]). Many of them aim at finding a metric that must be balanced across the running threads to minimize the resource contention. The metrics are normally related to the last level cache miss rate. These works make the best effort to mitigate the resource contention, however, they do not report the performance degradation during runtime. Hence, without a reference copy, it is almost not possible to tell at runtime if certain scheduling algorithm does improve the performance and how much it improves. After all, resource usage of a software program is dynamically changing. An increase in IPS (instruction per second) does not necessarily indicate the adoption of a more efficient scheduling algorithm. It may be simply because the program has entered a phase which requires less memory access. It would be beneficial if the service provider knows how much degradation the target process is undergoing

when it is co-scheduled with other processes competing for the shared resource and when the DVFS is applied. With such information, further adjustment in performance power tradeoff can be adopted. Another limitation of those previous works is the lack of ability to quantitatively predict the exact performance change caused by the change in task mapping. Therefore, they are not able to make fine-grained task migration decisions. Furthermore, their goal is to improve the average performance of all tasks. Given a mixed workload with both performance critical and noncritical tasks, this may lead to over-optimization for those noncritical tasks. To overcome the above limitations, a model that estimates the performance degradation of each individual target process under different task distributions will be extremely useful.

The problem is further complicated when CPU frequency scaling is performed in a system with resource contention, because its impact on different resources is not equal. Obtaining an analytical model to quantify performance degradation in a system with resource contention and frequency scaling is almost not possible. Machine learning techniques seem to be the only feasible solution [57].

Some previous works have been proposed to apply machine learning to model the performance change of tasks when their co-runners vary [71][57][61]. [71] trains a MIMO model online. Its inputs are different control actuators for different cores (e.g., CPU cycles scheduled to different cores and etc.). Its outputs are predicted QoS value. [57] uses the information from hardware performance counter to estimate the performance degradation of an SMP machine. [61] presents a model to predict the potential performance impact from different co-running neighbors to make better decision of task migration. However, none of these works consider the possibility that a system could also run at different voltage and frequency levels. All of these previous

works consider SMP machine where only single thread is running on each core, therefore, they ignore the contention for shared execution resources.

In this work, we apply machine learning techniques to develop a model for *Performance Estimation under Fixed task Scheduling* (*PEFS*). It estimates task performance degradation caused by existing resource contention and voltage/frequency scaling in current workload settings. We need to point out that, this model does not "predict" the performance of a given task schedule and frequency setting. Instead, it monitors the PMUs of current server, and estimates its performance degradation with the respect to an ideal system (i.e. the system without any resource contention and frequency scaling.) The information can be used as feedbacks to guide scheduling and DVFS. We further present a generalized model for *Performance Prediction under Task Migration* (*PPTM*). The second model "predicts" the performance degradation under new task mappings caused by potential intra-processor task migration. Based on the predicated results, the best task scheduling on the chip is found using either integer linear programming or graph analysis.

Compared to previous works (especially [57] and [61]), the contributions of this work are:

1. It studies the joint impact on performance from resource contention and frequency scaling. Our results demonstrate the necessity of considering them together at the same time for performance modeling.

2. It studies the effectiveness of traditional performance prediction using last level cache (LLC) miss rate. Our results show that the absolute value of LLC miss rate and performance in general do not have high correlation.

3. Performance estimation model (PEFS) and prediction model (PPTM) are presented to quantify performance degradation of a task under current (or potentially future) resource contention and frequency scaling in SMT-enabled chip multi-processor

4. The performance estimation information is used in a feedback control loop to guide voltage and frequency selection and the performance prediction results are used to guide task mapping/migration for reduced contention. The framework is flexible to handle variety of workload with mixed performance critical and non-critical tasks.

The rest of this section is organized as follows: sub-section 2.1.2 presents some observations that motivate the proposed performance model. Subsection 2.1.3 presents the model construction procedure. Subsection 2.1.4 talks about how to apply the model to find the best task mapping/migration. Experimental results are presented in 2.1.5, and Section 2.1.6 gives the conclusions.

2.1.2 Motivational observations

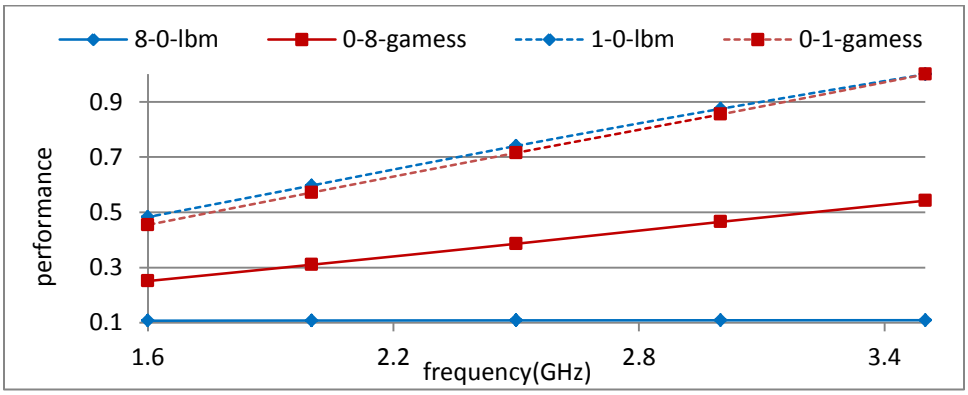
(1) Impact of co-running neighbors on DVFS sensitivity

In this section, we provide some experimental data that motivate the search for a model that captures the performance impact of both resource contention and frequency scaling. Our experimental system is an Intel Ivy Bridge i3770K CPU machine with 4 physical cores and 8 logical cores (SMT2). Each physical core has dedicated L1 and L2 cache (shared by two logical cores) while all cores share the same 8MB L3 cache. It supports frequency scaling from 3.5 GHz to 1.6 GHz with a step of 0.1 GHz. It is also equipped with 8GB two-channel 1600 MHz DDR3 memory. Ubuntu Linux is installed. The configuration of this experimental platform is representative among many commercial computers on the market nowadays.

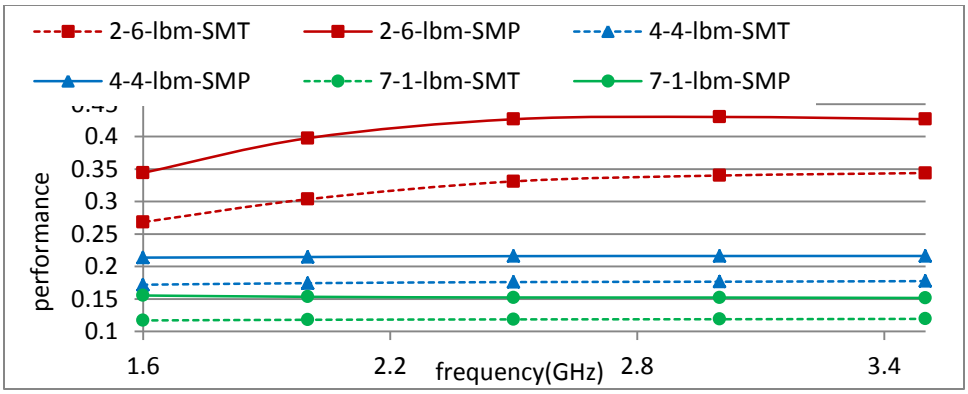
Though many research papers assume that frequency scaling can be applied at core level, Intel Ivy Bridge processors only have one voltage regulator. The per-core level frequency scaling is disabled by firmware and OS [77]. Each physical core can be put in deep sleep C state independently [77] when they become idle. This state has very low power consumption due to power and clock gating. The socket power of our experimental system is around 24W during idle state when deep sleep C state is enabled. When the deep C state is disabled, the idle power becomes 36W at lowest frequency and 63W at highest frequency.

Nowadays the memory subsystem becomes relatively fast. We observe that running a single memory intensive task will be far from saturating the memory subsystem of the server. The performance of the task scales almost linearly during frequency scaling as the CPU and cache speed are still the bottleneck even for memory intensive tasks. The linear relation stops only when multiple memory intensive tasks are actively running simultaneously.

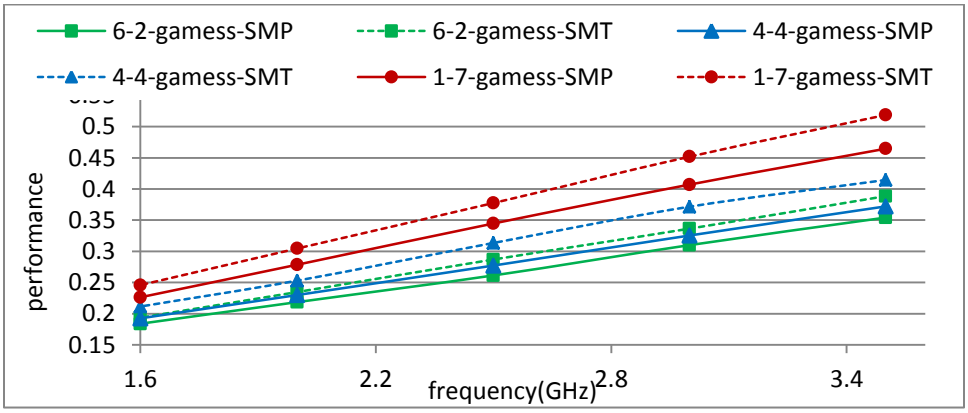
Our hypothesis is that different co-scheduled jobs not only affect the performance of an application by generating resource contentions, but also affect its sensitivity to frequency scaling. To demonstrate this, we create workload that has various levels of resource contention. Our workload consists of two benchmarks from SPEC CPU2006 [80]. One is lbm, which is memory



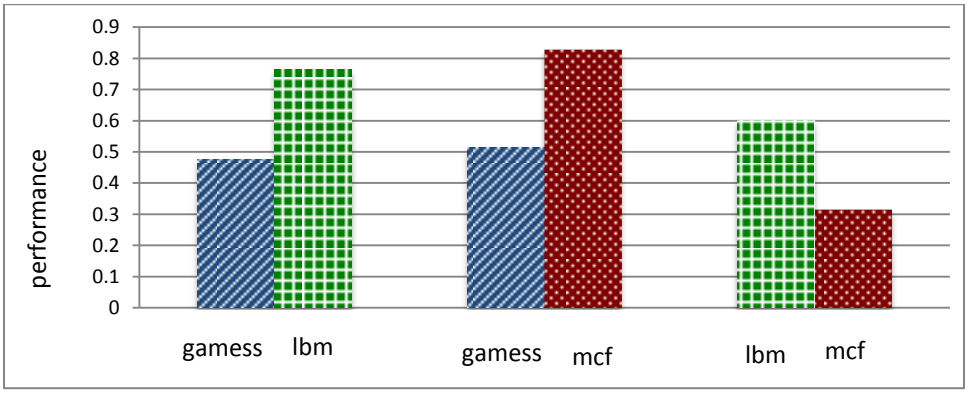
(a) Performance of uniform workload



(b) Memory intensive task in hybrid workload



(c) CPU intensive task in hybrid workload



(d) Impact from different SMT

Figure 2-1 Performance sensitivity to resource contention and frequency scaling

intensive; and the other is gamess, which is CPU intensive. Different workloads are generated using these two benchmarks. In these workloads, each logic core executes at most one benchmark program. We refer the two processes sharing the same physical core as *SMT neighbors* and the two processes running on different physical cores as *SMP neighbors*. The performance of these two benchmarks and their sensitivities to frequency scaling are tested in the context of different workload mappings. The test cases are labeled as *n*-*m*-*T*-SMT[SMP]. The parameters *n* and *m* specify that there are *n* lbm processes and *m* gamess processes running. The parameter "*T*" is the name of the target process whose performance we are interested in. The label "SMT" indicates that the SMT neighbor of our target process is the same benchmark program; otherwise the label "SMP" is attached to the workload.

Figure 2-1(a)~(c) show the performance degradations for each test case. The x-axis is CPU frequency and the y-axis is the normalized performance of the target benchmark program compared with the same target program running alone on a dedicated processor at the highest frequency (i.e. 3.5 GHz).

In Figure 2-1 (a) we can see that when only one task is running, regardless whether it is memory intensive or CPU intensive, the performance scales linearly with CPU frequency at the same rate. This is because of the high memory bandwidth of the modern server. When all 8 logic cores running the same task, the memory intensive task (lbm) suffers much more degradation than the CPU intensive task (gamess) due to the memory contention. However, it is also much less sensitive to frequency scaling than gamess, because the CPU is no longer the bottleneck of performance. Figures (b) and (c) show the performance of lbm and gamess separately when they are scheduled with different co-runners. Three major observations are made from the two figures.

First: Having lbm as the SMT neighbor causes more performance degradation than having gamess. For example, 2-6-lbm-SMT has less performance than 2-6-lbm-SMP and 6-2-gamess-SMT has better performance than 6-2-gamess-SMP. The similar trend can be observed for other test cases. This is mainly because a memory intensive SMT neighbor competes for the L1 and L2 cache.

Second: With more and more lbm processes running on the processor, the performance degradation of the target is exacerbated. For example, 2-6-lbm-SMT has better performance than 4-4-lbm-SMT and 6-2-gamess-SMT has less performance than 4-4-gamess-SMT. Such trend is more prominent for memory intensive target (i.e. lbm) than for CPU intensive target (i.e. gamess).

Third: The gamess is more sensitive to frequency scaling than the lbm. Figure 2-1 (c) shows that its performance decreases almost linearly with frequency scaling. However, as more lbm processes are added into the system, the decreasing ratio reduces, which indicates a reduced sensitivity to frequency scaling. For example, the performance of 6-2-gamess-SMT changes slower than 4-4-gamess-SMT with frequency scaling. On the other hand, lbm's performance is a nonlinear function of the CPU frequency. When the number of lbm processes increases, its performance is almost constant as shown in Figure 2-1 (b), indicating a low sensitivity to frequency scaling.

To sum up all the discussions above, the contention and DVFS both affect the workload's performance. To make things more interesting, a program's sensitivity to frequency scaling is not only determined by itself but also its SMT and SMP neighbors. And the performance does not always scales linearly. The performance model considering only one frequency will no

longer be accurate when DVFS is enabled. In order to provide accurate performance estimation to guide power management at different level, our performance model must provide accurate estimation across a wide range of CPU frequency.

Many previous works focus only at performance degradation due to SMP level contention; however, the SMT level contention has even greater performance impact. To further show this impact, we pick two processes from lbm, gamess and mcf (which is another memory intensive benchmark in SPEC CPU2006) and run them as SMT neighbors. Because only two processes are running, the SMP level contention is almost negligible.

Figure 2-1 (d) shows the normalized performance of each processes running with different neighbors. The two benchmarks running together are bundled. As we can see, gamess has large performance degradation when running with either lbm or mcf; while lbm has relatively less degradation in either cases, which indicates low sensitivity to SMT level contention. The performance of mcf exhibits the behavior of bimodal. It is has large degradation when running with lbm and marginal degradation when running with gamess. This suggests that, compared to other two, mcf is more sensitive to having a memory intensive SMT neighbor. In other words, its performance is a function of the characteristics of its SMT neighbor. These observations motivate the development of the PEFS model, which will be discussed in Section 2.1.3.

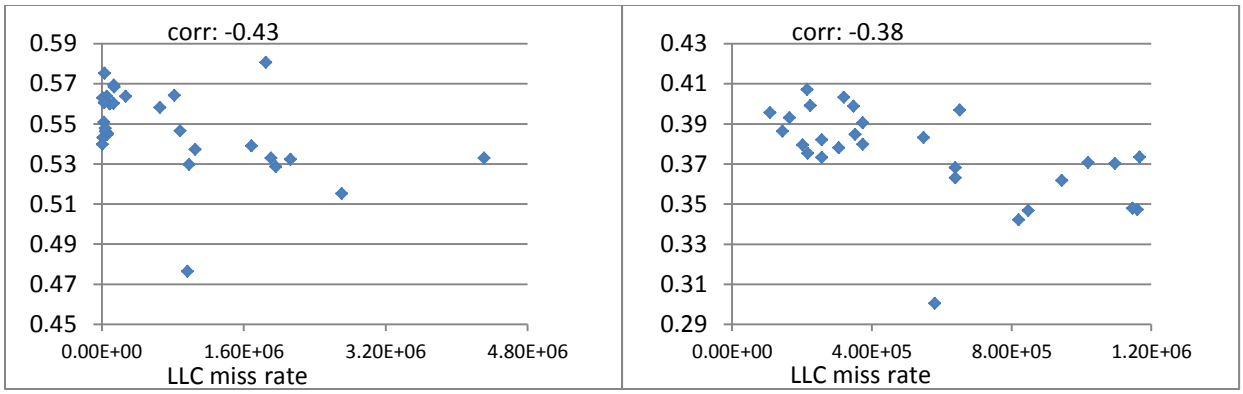
(2) Limitation of traditional performance model based on LLC miss

Many previous task mapping/migration algorithms try to minimize resource contention and performance degradation by balancing the last level cache miss across co-scheduled tasks. The rationale behind this is the assumption that performance degradation and LLC miss rate are

highly correlated. However, our experimental results show that such assumption is not always true. Although LLC miss rate in general is a good indicator of how severe tasks will compete for shared memory resources, it is not a comprehensive indicator of overall resource contention and performance degradation.

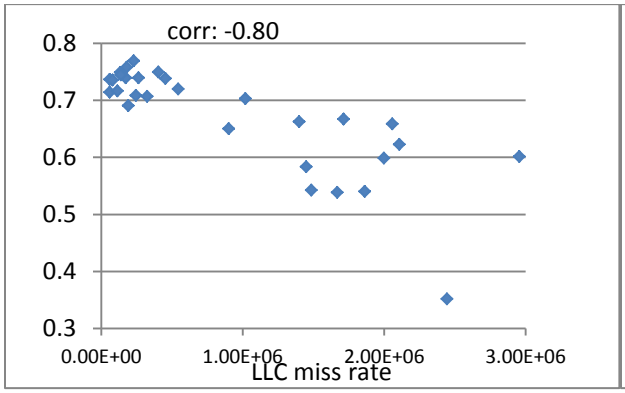
Two sets of experiments are conducted to evaluate the relationship between target task performance degradation and its LLC miss rate. In the first set of experiments, we use gamess as the target task and change its SMT neighbor from the remaining set of 29 benchmarks in SPEC CPU 2006. Its SMP neighbor is set to either lbm or gamess, which stereotypes memory intensive or CPU intensive environment. The relation between the target performance and the LLC miss rate of its SMT neighbor is plotted in Figure 2-2 (a) and (b). In the second set of experiments, the same procedure is repeated with the target process set to lbm and the results are given in Figure 2-2 (c) and (d).

In Figure 2-2 the Y-axis represents the normalized performance of the target task while the X-axis represents the LLC miss rate of its SMT neighbor. Correlations between X value and Y value of data points are listed on top of the figure. As we can see from the figures, the performance degradation of a CPU intensive target task (e.g. gamess) only has weak correlation with the LLC miss rate of its SMT neighbor. Even memory intensive target task (e.g. lbm) does not have high correlation between its performance and its SMT neighbor's LLC miss rate, if the rest of the tasks running are also memory intensive. That's because as the SMP neighbors become more memory intensive and the overall memory access gets heavier, the impact from the SMT neighbor will be less important. These observations motivate us to develop a new performance prediction model that considers information beyond the LLC miss rate.

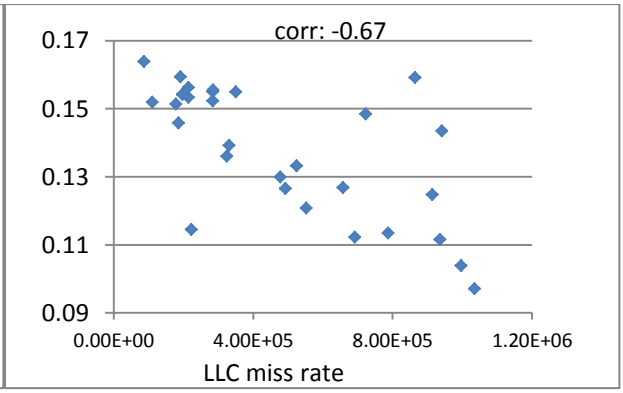


(a) gamess (CPU intensive SMP neighbor)

(b) gamess (memory intensive SMP neighbor)



(c) lbm (CPU intensive SMP neighbor)



(d) lbm (memory intensive SMP neighbor)

Figure 2-2. Relation between target performance and LLC miss of its SMT neighbor

(3) Potential performance improvement by task migration

The next set of experiments is performed to find out the potential of performance improvement by task migration. Figure 2-3 shows the performance boost of 29 benchmarks in SPEC CPU 2006 when its SMT neighbor changes from lbm to gamess. These benchmarks are indexed based on the ascending order of their LLC miss rate. The blue bars and red bars correspond to the experiments where the SMP neighbors of the target are CPU intensive and memory intensive respectively. In order to closely resemble the effect of task migration, the total workload running on the processor does not change before and after the switch of SMT neighbor.

The first observation from Figure 2-3 is that different benchmark benefit differently from replacing a memory intensive SMT neighbor to a CPU intensive one. The performance boosts varies from 5% to 35%. We also noticed that although switching the SMT neighbor from lbm to gamess always gives positive performance improvement, the magnitude varies for different SMP neighbors. For example, when running with CPU intensive SMP neighbors, task #12 ("gromacs") has 22% performance gain if its SMT neighbor changes from lbm to gamess. This number reduces to 7% if its SMP neighbors are memory intensive. Furthermore, the relative order of the performance gain among tasks changes in different SMP settings. For example, in a CPU intensive SMP setting, pairing gamess with task #12 ("gromacs") gives higher performance gain than pairing it with task #14 ("perlbench"). However, the reverse is observed if the SMP setting is memory intensive. A simple LLC miss rate model is not able to provide such detailed information on performance changes. First of all, some benchmarks with high LLC miss rate might have less demand on other resources, which allows certain co-runners run faster. Secondly, as mentioned in [58][72][75], LLC miss rate along cannot represent the cache contention characteristics as different programs have different access patterns of the cache. Different programs have different spatial preferences of the cache access, which cannot be captured by the LLC miss rate.

Although they show significant variations, the data in Figure 2-3 do not pose strong motivation for a more powerful and better task-mapping algorithm, as the average performance gain is only about 13%. This is because, running 8 SPEC benchmark tasks simultaneously, the processor already has very high utilization and there is not much room for performance improvement.

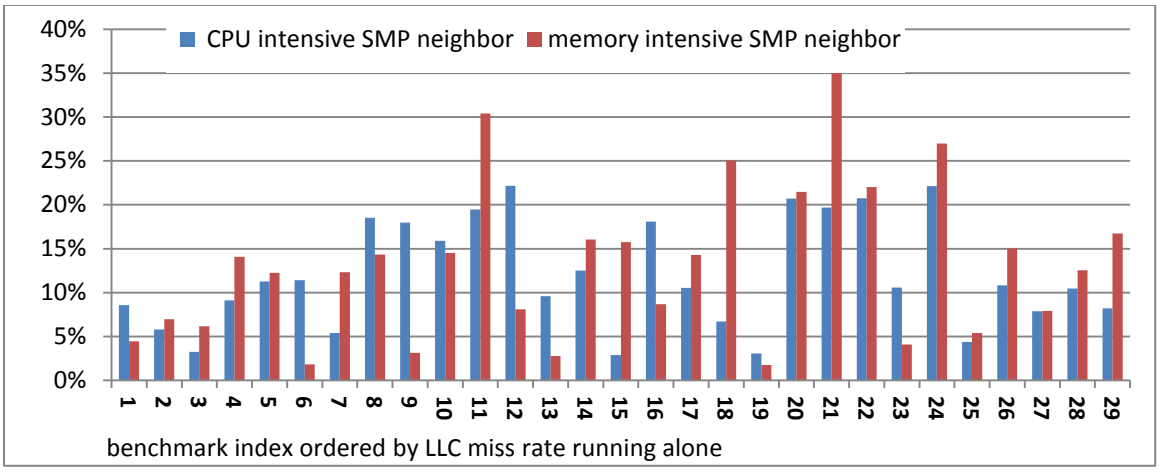


Figure 2-3. Percentage performance boost of target process when its SMT neighbor changes from lbm to gamess

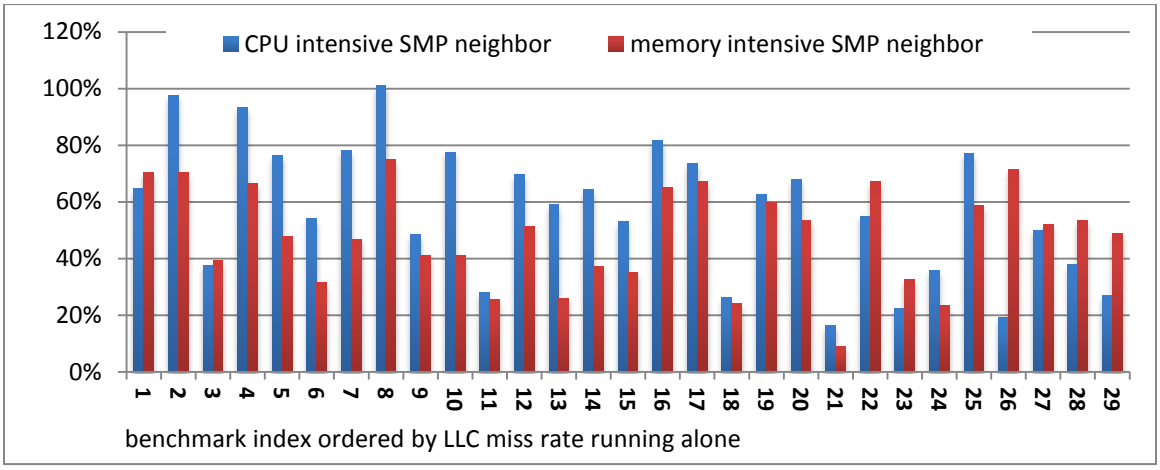


Figure 2-4. Percentage performance boost of target process when its SMT neighbor changes from lbm to sleep task

The high utilization in previous example is not very common in data center. More performance improvement can be achieved from appropriate task-mapping algorithm when the cores are not fully utilized. Figure 2-4 shows the performance boost of different benchmark when their SMT neighbor switches from lbm to a "sleep" task. The "sleep" task is introduced to represent an idle logic core or a task that has extremely low resource demand. As we can see, for

different target program, replacing its memory intensive SMT neighbor with a “sleep” (idle) task can provide 10% to 100% performance improvements. Furthermore, some benchmarks benefits more from such switch under CPU intensive SMP setting, while the others benefit more under memory intensive SMP setting. Obviously, good task mapping algorithm should pair the SMT co-runners so that the best performance gain can be achieved. The above analysis shows that LLC miss rate only provides a rough guideline for task mapping; searching for the best solution is something more subtle.

2.1.3 Model construction

(1) Performance estimation under fixed scheduling (PEFS)

In this section, we apply machine learning technique to construct a model that assesses the performance degradation of target application considering the impact from its current neighbors and the CPU frequency. The degradation is measured with the respect of a reference system which has no resource contention and frequency scaling. The discussion is carried out based on Intel Ivy Bridge i3770K CPU, which has 4 physical cores and 8 logical cores. However, the same method can be applied to other processors. Because we focus on CPU-bound workloads (i.e., SPEC CPU2006), in our model, we assume only one thread is running on each logical core [58].

We classify the processes running on the same processor into 3 categories: Target, SMT and SMP. Target is the process whose performance degradation needs to be characterized. The SMT process shares the same physical core with the Target, and the rest of the processes running on the same chip belong to the SMP category.

To train and test the model, we created 30 groups of workloads. Each workload consists of 4 benchmarks in SPEC CPU2006. One of them will be Target, and another one will be its SMT neighbor. The other two benchmarks will be duplicated to 6 processes and run on the rest of the 3 physical cores. During the selection, we try to involve as many benchmarks as possible while exploring different combinations of memory and CPU intensive benchmarks [74] to create variety. Each workload is run with 8 different frequencies swept from 1.6 GHz to 3.5 GHz.

(a) Feature selection

There are around 260 PMU candidates on each logical core. We use *perf* [78] to collect the PMU values. There are only 4 hardware performance counters for each logical core which means only 4 events can be monitored at the same time without loss of accuracy. If more events are to be recorded, the counters will be time-multiplexed. Even if we collect 8 events in each run, to collect around 260 PMU events requires running the same workload repeatedly for more than 30 times. As we can see, not only it is impossible to have all 260×8 events as inputs for model construction, to collect all of these events will also take a prohibitively long time. A feature selection step must be performed first to reduce the size of events to simplify modeling and data collection.

In this step, we run each workload for only 10 seconds and repeat this for about 40 times. Each time 6~8 PMU events are collected. The data forms the preliminary training set. First, we consolidate the PMU events of the 6 SMP processes by calculating their sum. To the target process, they are like background activities and it is not necessary to keep the individual information. After consolidation, we have 3 sets of PMU events from Target, SMT and SMP processes respectively plus the CPU frequency. Then we apply *Weka* [79] for feature selection.

The events are evaluated using *CFsSubsetEval* algorithm which evaluates the subset of events by considering the individual predictive ability of each feature along with the degree of redundancy between them. A set of 24 events is selected at the end. Table 2-1 shows the top 9 events that are selected and their correlation to the target performance. Interestingly, we found that the attribute *frequency* itself is not selected at last. However, the frequency information is reflected in the PMU readings.

(b) Model construction

After feature selection, a more comprehensive and accurate data collection is performed again. Each workload runs for 40 seconds to get more coverage and the 24 selected PMU events are recorded with 4 collected at each run. The model output is normalized performance (PF) with

Table 2-1 Top 9 selected events sorted by its correlation to the performance for PEFS model

PMU event name	Correlation
UOPS_DISPATCHED.PORT_3(Target)	0.83
CYCLE_ACTIVITY.CYCLES_NO_EXECUTE(SMP)	0.77
CYCLE_ACTIVITY.CYCLES_LDM_PENDING(Target)	0.67
IDQ.ALL_DSB_CYCLES_ANY_UOPS(SMP)	0.63
CYCLE_ACTIVITY.CYCLES_L1D_PENDING(SMP)	0.61
L2_LINES_OUT.PF_CLEAN(Target)	0.54
MEM_LOAD_UOPS_RETIRED.HIT_LFB(Target)	0.40
LOAD_HIT_PRE.HW_PF(Target)	0.36
MOVE_ELIMINATION.INT_ELIMINATED(SMT)	0.33

respect to the reference system. It is calculated as $PF = \frac{instruction_{test}}{instruction_{ref}}$, where $instruction_{test}$ is retired instruction of the target application running on the test system that has contention and frequency scaling, and $instruction_{ref}$ is the retired instruction of the target application running

on a reference system that has no contention and frequency scaling. Both are collected over the same amount of time. It is easy to see that performance degradation can be calculated as *1-PF*.

About 16 different modeling algorithms are evaluated for their relative absolute error through the 10 folds cross-validation process. The results show that MultilayerPerceptron (i.e., neural network) model yields the best accuracy. We refer this model as "model_full".

Two reference models are also constructed in the similar way. However, the first one ignores the impact of frequency scaling. Its training data is collected from systems performing no frequency scaling (i.e. running at 3.5GHz). The model is referred as "model_no_freq". The second one does not explicitly consider the impact of SMT neighbor. It's training set does not have PMU data for the SMT process. This model is referred as "model_no_SMT".

The accuracy of those three models and their correlation with the actual performance are given in Table 2-2 . As we can see, "model_full" gives the highest accuracy and correlation. The "model_no_SMT" also has a low error rate. This is because the impact from SMT process is partially reflected in the Target process's PMU change. Finally, Figure 2-5 shows the correlation between the PEFS estimated performance and the actual performance. As we can see they are highly correlated.

Table 2-2 Accuracy of 3 different PEFS models

	model_full	model_no_freq	model_no_SMT
Relative absolute error	11.2%	30.2%	13.5%
Correlation	0.994	0.940	0.985

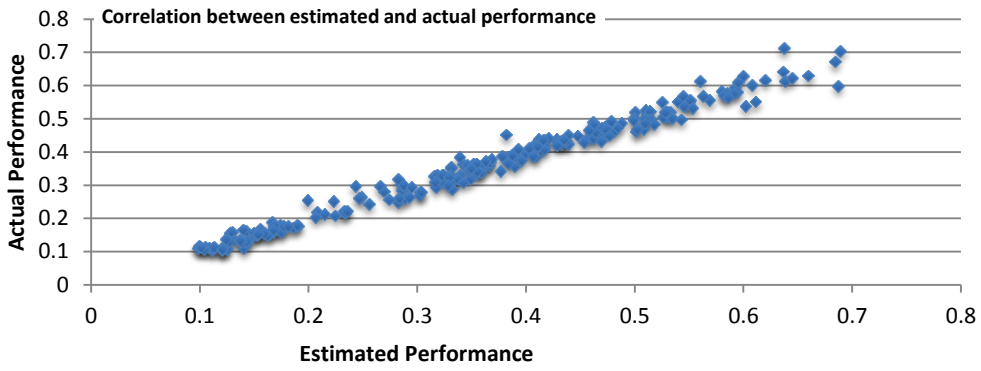


Figure 2-5 Estimated performance is highly correlated to actual performance

(2) Performance prediction under task migration (PPTM)

Our discussion in section 2.1.2 shows that co-runners have significant impact on performance of target process, and dynamic task migration, which remaps task during runtime when workload characteristic changes, is desirable. An effective task migration controller needs the ability to predict how different task mappings may affect the performance of the target process. In this section we generalize the PEFS model for Performance Prediction under Task Migration (PPTM). We limit our discussion to migrations within the single chip multi-core processor, which is referred as intra-processor task migration. We focus on intra-processor task migration because it has very low overhead but quite significant performance impact if performed correctly. The similar modeling technique can also be applied for performance prediction under inter-processor task migration.

Unlike PEFS, PPTM predicts how the performance of target process will change if a new task mapping is adopted. The prediction relies on the architectural activities observed under current task mapping and the knowledge obtained during the training process. For good prediction results, the new mapping should not be dramatically different from the current

mapping. In this work, the task migration is confined to switch the CPU affinity of only two processes running on different cores. One of them is the target process and the other is referred as the migration target (mTarget). Based on this definition, migrating the target process to an idle core can be done by switching it with an "idle process." Besides Target and mTarget, PPTM divide the rest of processes running on the processor into 3 categories based on current task mapping: SMT, mSMT, and SMP. SMT and mSMT share the same physical core with Target and mTarget respectively. However, this relation will be reversed after migration. In other words, after migration, SMT process will share the same physical core with mTarget and mSMT process will share the same physical core with Target. All processes running on other cores are SMP tasks.

To train and test the model, we created around 720 groups of workloads. Each workload consists of 6 benchmarks randomly selected from SPEC CPU2006. Four of them will be Target, SMT, mTarget and mSMT tasks. The remaining 2 benchmarks will be duplicated to 4 processes and run on the rest of the 2 physical cores. Except the Target, all other tasks can also be set as "idle task", which does nothing but sleep. Each workload is run twice. In the first round, PMU information is collected. In the second round, Target and mTarget processes will be switched and performance will be recorded. Each workload is run with 3 different frequencies swept from 1.6 GHz to 3.5 GHz. The rest of the feature selection and model construction steps are similar to that of the PEFS model introduced in Section 2.1.3.

Table 2-3 shows the top 9 selected features for the PPTM model and their correlation with the target performance. The average absolute error is 21.1% by 10 fold cross validation and the correlation between predicted performance and real performance is 0.967.

Table 2-3 Top 9 selected features sorted by its correlation to the performance for PPTM model

PMU event name	Correlation
UOPS_DISPATCHED_PORT.PORT_2(Target)	0.75
BR_MISP_EXEC.ALL_BRANCHES(SMP)	0.60
CYCLE_ACTIVITY.CYCLES_LDM_PENDING(Target)	0.52
RESOURCE_STALLS.SB(mSMT)	-0.36
LD_BLOCKS_PARTIAL.ADDRESS_ALIAS(mTarget)	0.34
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE (SMT)	0.32
LD_BLOCKS_PARTIAL.ADDRESS_ALIAS(mSMT)	-0.31
RESOURCES_STALLS.ROB(SMP)	0.27
MEM_LOAD_UOPS_RETIRED.LLC_HIT(Target)	0.25

2.1.4 Model directed task mapping

With the help of PPTM model, the task mapping can easily be formulated as an integer linear program. Similar problems are discussed in [76], however, with slightly different objectives.

We use N to denote the total number of logic cores in the processor. Similar to [76], we assume that each logic core runs only a single task including the idle task. Therefore, the total number of tasks running on the processor is also N. Let $\mathcal{T}$ denote the set of tasks, $|\mathcal{T}|=N$, and $\mathcal{T}_c$ denote the set of performance critical tasks. Our goal is to maximize the total performance of those critical tasks. We define the target variable x_{ij} to be 1 when task i and j are mapped to the same physical core, otherwise it is 0. We use p_{ij} to denote the performance of task i when it is co-scheduled with task j. The value of p_{ij} can be obtained using PTMM prediction if i and j are not scheduled together at current mapping, otherwise it can be obtained using PEFS estimation.

The following specifies the objective function and the constraints of the integer linear program for model directed task mapping:

$$\max \sum_{i \in Tc} \sum_{j \in T} x_{ij} p_{ij} \quad \text{s.t.}$$

$$x_{ii} = 0, \forall i \in T$$

$$x_{ij} = x_{ji}, \forall i \in T \; \forall j \in T$$

$$\sum_{j \in T} x_{ij} = 1 \, , \forall i \in T$$

The constraints ensure that each task is scheduled exactly once and it must be mapped with a different task other than itself. We used lp_solve [81] to solve this problem in our experiment.

The task mapping can be found by looking for a perfect match in a graph [76]. By considering each task as a vertex and setting the weight of edge between 2 vertices as the total performance of the two corresponding tasks when they are mapped together, the authors of [76] search for the performance optimal mapping by looking for a set of edges with maximum weight such that each vertex is connected to exactly one edge. This is a perfect matching problem and polynomial complexity algorithm exists for this problem.

The exact same graph model as [76] cannot be used in this work, because our goal is to only increase the performance of the set of critical tasks while their objective is to maximize the total performance of all tasks. The difference can be resolved with simple modification in the edge weight. By defining the weight of an edge e_{ij}, which connects task i and j, as the following:

$$e_{ij} = \begin{cases} p_{ij} + p_{ji}, & \text{if both } i \text{ and } j \text{ are critical tasks} \\ p_{ij} \;\; (or \; p_{ji}), & \text{if only } i \text{ (or } j \text{)is a critcal task} \\ 0, & \text{otherwise} \end{cases},$$

the same perfect matching algorithm can be used to find the mapping that maximizes the performance for the set of critical tasks.

2.1.5 Experimental Results

We apply the PEFS model and PPTM model to achieve runtime performance optimization and power management. Three sets of experiments are conducted that represent different application scenarios.

(1) Runtime power management without task migration

In the first set of experiments, we consider DVFS based power management on a multicore processor with fixed CPU affinity mapping. PEFS model is used to provide performance feedback to guide the DVFS controller. Four different workloads are generated and tested. Each workload contains 8 copies of SPEC CPU2006 benchmark. The first workload (WL1) has 2 memory intensive processes and 6 CPU intensive processes. The second workload (WL2) has 4 memory intensive processes and 4 CPU intensive processes. The third and fourth workloads consist of only memory intensive benchmarks and CPU intensive benchmarks respectively. Two different scheduling methods are applied to WL1 and WL2. The first one schedules a memory intensive process to be the SMT neighbor with a CPU intensive process. The second one schedules two memory intensive (or two CPU intensive) processes to be SMT neighbors to each other. The first method causes less resource contention [58] and hence will be denoted as “G”, which stands for “good” scheduling. The second method is denoted as “B” which stands for “bad” scheduling. The detailed information of workloads and their mappings is presented in Table **2-4**. Labels (M) and (C) indicate if the benchmark is memory or CPU intensive. In this work, we do not consider task migration. The performance feedback from the

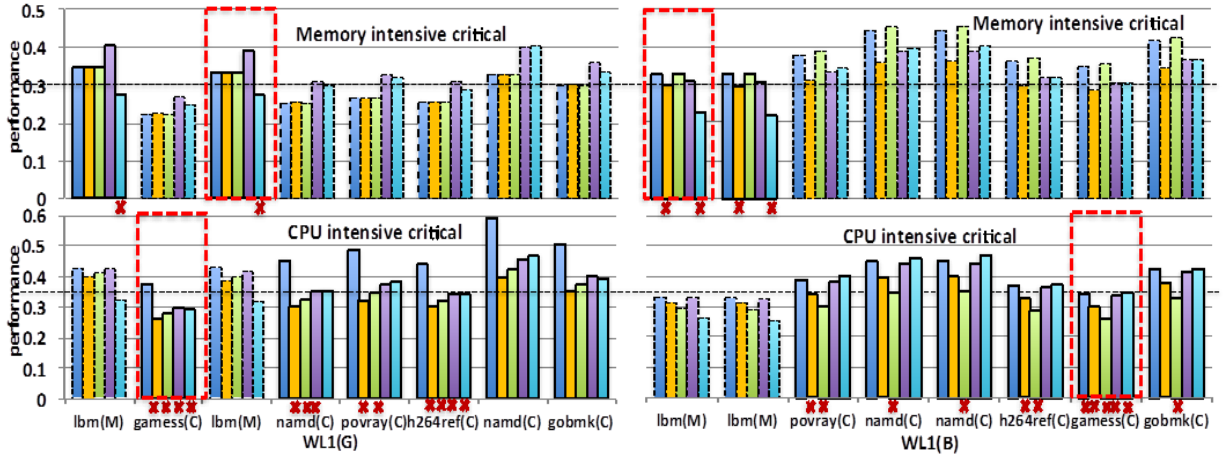
model is only used to guide DVFS settings. Please note that, the testing workloads are significantly different from the training set. None of the training workload has more than 50% similarity to a testing workload.

Each workload will run for 400 seconds (benchmarks will run iteratively if their actual length is less than 400 seconds). A user-level Shell script is developed for performance monitoring and DVFS control. It dynamically calls the Linux *perf* tool to collect the 24 PMU attributes from each logical core to form the inputs of the model. The interval of data collection is set to be 10 seconds, which is long enough to let each event be monitored for substantial period of time to get good sampling accuracy. We assume that a set of target processes are critical and have QoS constraints. The constraint is expressed as the normalized performance (PF) of the process with the respect to the reference system. If all critical tasks exceed performance threshold, the chip's frequency will be increased by 0.1GHz (chip voltage will be adjusted accordingly). Otherwise, the frequency will be decreased. Two sets of critical tasks are selected for WL1 and WL2. The first one consists of all memory intensive tasks, while the second one consists of all CPU intensive tasks. For WL3 and WL4, all tasks are critical.

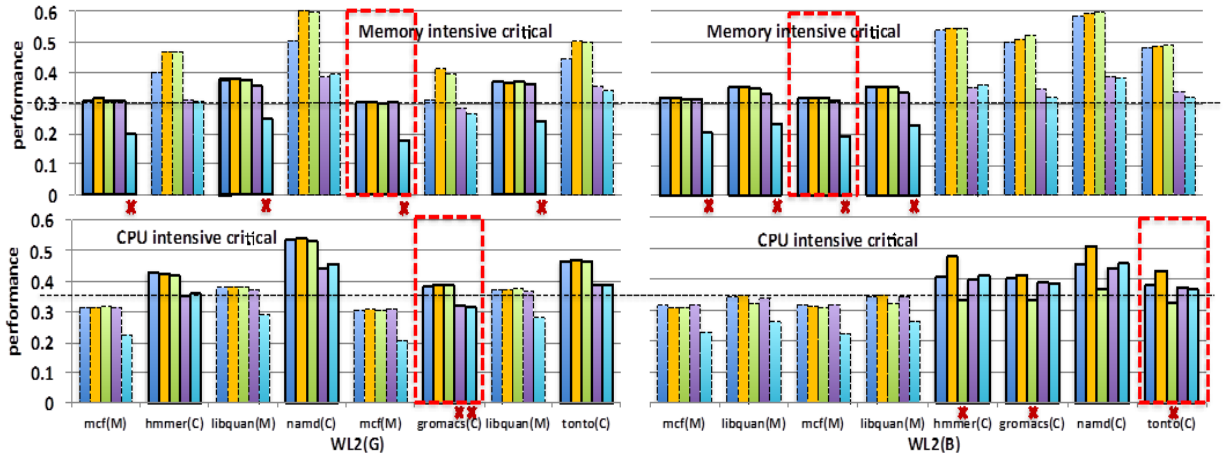
We refer to a system that uses our model as "model_full". It is compared with 4 reference systems: (1) model_no_smt: the system conducts performance assessment without considering SMT neighbor's impact explicitly, i.e. it uses model_no_smt specified in Table 2-2 for performance estimation; (2) model_no_freq : the system conducts performance assessment without considering the impact of frequency scaling, i.e. it uses model_no_freq specified in Table 2-2 for performance estimation; (3) direct_scaling: the system scales CPU frequency linearly according to the given performance threshold; (4) capping: instead of frequency scaling, the system set a cap on the CPU quotas that a task can take based on the given performance

threshold. The cap is set using Linux *cgroups*[82]. The same cap is given to all tasks on the chip. The processor will run at the highest speed and enter deep sleep mode when it is capped. Both "direct_scaling" and "capping" ignores SMP level resource contention. A constant 50% performance degradation is assumed for SMT level contention. Although not very accurate, this is the best approximation that we can have without dynamically tracking the performance, which is the purpose of using simple management approaches such as "direct_scaling" and "capping". The CPU frequency and cap are set accordingly. For example, if the performance threshold is 30% of reference system, then "direct_scaling" will set CPU frequency to $0.3/0.5 = 60\%$ of the maximum frequency, while "capping" will cap the CPU quota to 60%.

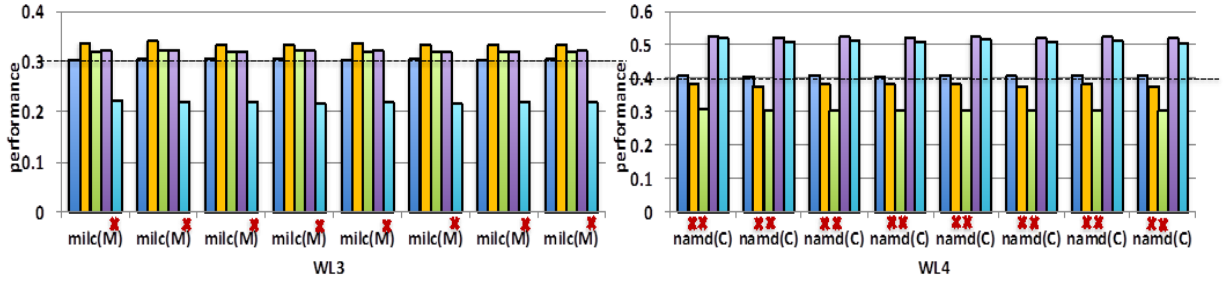
The performance for all 4 workloads running on 5 different systems is reported in Figure 2-6. Please note that WL1 and WL2 both have 2 different task mappings and for each mapping two sets of critical tasks are tested. Therefore, four plots are presented for each workload. The left two plots in Figure 2-6 (a) are for WL1(G) and the right two plots are for WL1(B). The top two plots in Figure 2-6 (a) are for systems where memory intensive tasks are critical, while bottom two plots are for systems where CPU intensive tasks are critical. Each bundle of bars corresponds to the performance of one task running at different systems. The bars with dark solid outlines are the critical tasks whose performance is important while the bars with dotted outlines are noncritical. The dotted horizontal lines indicate the performance thresholds. If a solid bar falls below this line then there is a performance violation. A task with performance violation is marked by a small red cross underneath the bar. Those critical tasks that have the lowest performance are referred as *bottleneck tasks*, as their performance is the bottleneck that determines the CPU frequency of the entire chip. They are marked with red boxes. In order to



(a) WL1



(b) WL2



(c) WL3 and WL4

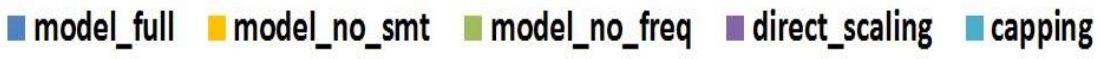


Figure 2-6. Performance for all workloads

Table 2-4. Workloads used in the evaluation

	WL1 (G)	WL1 (B)	WL2 (G)	WL2 (B)	WL3	WL4
0	lbm (M) gamess (C)	lbm (M) lbm (M)	mcf (M) hmmer (C)	mcf (M) libq (M)	milc (M) milc (M)	namd (C) namd (C)
1	lbm (M) namd (C)	povray (C) namd (C)	libq (M) namd (C)	mcf (M) libq (M)	milc (M) milc (M)	namd (C) namd (C)
2	povray (C) h264ref (C)	namd (C) h264ref (C)	mcf (M) gromacs (C)	hmmer (C) gromacs (C)	milc (M) milc (M)	namd (C) namd (C)
3	namd (C) gobmk(C)	gamess (C) gobmk(C)	libq (M) tonto(C)	namd (C) tonto(C)	milc (M) milc(M)	namd (C) namd(C)

From the figure we can see, systems using the PEFS model (i.e. model_full) have almost no performance violation except for WL1 (B). This only performance violation is because of the inefficient task mapping. All of the reference systems have performance violations for this test case. Furthermore, our model keeps the performance of those bottleneck tasks much closer to the performance threshold than all other techniques. This means that lower frequency level is used and hence more energy savings are achieved. Comparing systems with different mapping choices, our model can correctly identify the 'bottleneck' tasks and make frequency scaling decision accordingly. We also observed that "capping" gives large violation most of time when the critical tasks are memory intensive. This is because it runs the CPU at full speed, hence the memory becomes the performance bottleneck. Furthermore, when the CPU is throttled, the memory access is stopped too. The similar is not observed for DVFS based approaches, where both CPU and memory operate all the time.

The third thing we observed is that "model_no_smt", "model_no_freq" and "direct_scaling" lead to more violation when the critical tasks are CPU intensive. This is because frequency scaling based on inferior performance model or simple linear scaling obviously cannot accurately capture the performance degradation of CPU intensive tasks, which varies greatly during frequency scaling; while the performance of memory intensive tasks generally do not change that much. We also observed that there are fewer violations for WL2 than WL1 if the critical jobs are CPU intensive. It seems that the more memory intensive tasks are running, the easier for all the models to make the right decision since sensitivity to frequency scaling reduces.

Please note that all systems use the same task mapping. And all of the first four systems "model_full", "model_no_SMT", "model_no_frequency" and "direct_scaling" perform DVFS based power management. Since Intel Ivy Bridge processor only supports chip level frequency scaling, the system that has the minimum power consumption without performance violation is the bottleneck task, whose performance should exactly meet the threshold. Therefore, it is not necessary to compare the power consumption among "model_full", "model_no_SMT", "model_no_freq" and "direct_scaling". Because model_full brings the performance of the bottleneck task closest to the threshold, its power consumption must be lower than the other three reference models. However, the same comparison cannot be applied to "capping", because it performs power management using CPU capping instead of DVFS. Therefore, we still need to compare its power consumption with that of "model_full". The power consumption of the 10 test cases in Table 2-4 is measured using *Watts up?PRO* power meter.

Figure 2-7 shows the energy and energy delay product (EDP) of systems using "capping" and "model_full". Both systems execute the same amount of instruction. Here the whole system

idle power (around 24W) is removed from calculation. As we can see, in average “model_full” has 24% reduction in energy and 38% reduction in EDP compared to “capping”.

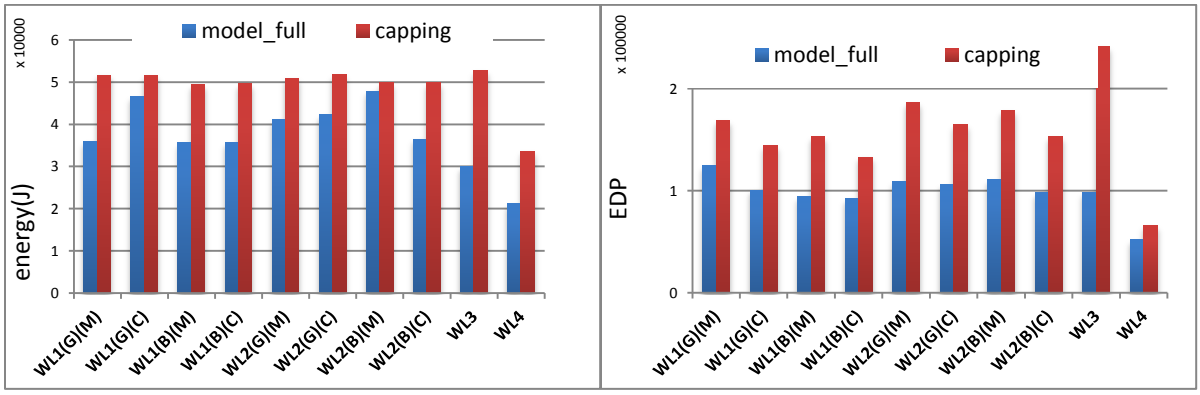


Figure 2-7. Energy and EDP of model_full and capping

(2) Task migration for optimal performance

In the second set of experiments, we apply the PPTM model to guide task migration. The input of the PPTM model

is the PMU information collected while the tasks are running under current mapping. The goal is to find a new mapping that maximizes the performance of a set of critical tasks. No DVFS power management is considered in this experiment. Since only the highest CPU frequency is used in this experiment, we train the PPTM model with only the data collected at the single clock frequency, and refer it as PPTM-SF.

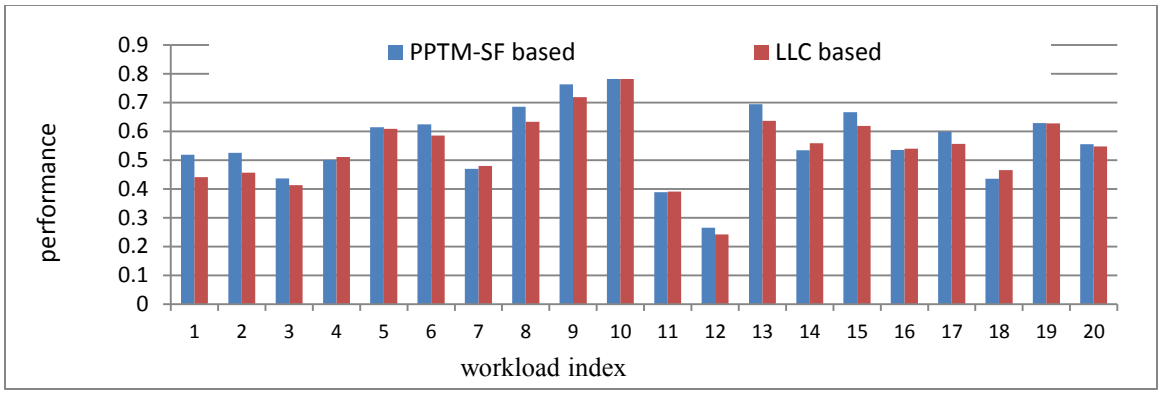
As we pointed out in section 2.1.2 that task migration will not provide much performance gain if the CPU utilization is very high. Here we assume that at least one of the cores is not fully utilized, i.e. there is at least one idle task in the workload. Two scenarios are evaluated. In the first scenario, the workload has two critical tasks and one idle task; in the second scenario, the workload has 4 critical tasks and 2 idle tasks.

Our reference algorithm is task migration based on LLC miss rate. It chooses the set of tasks with the minimum LLC miss rate during run time and pairs them with the critical tasks. For each of the two scenarios, 20 different workloads are created based on randomly selected SPEC benchmarks. We run each workload for 400 second. Every 40 seconds the PPTM-SF model will be called or the LLC information will be checked. Based on the prediction, a new task mapping is found. If it differs from the current one, tasks will migrate accordingly.

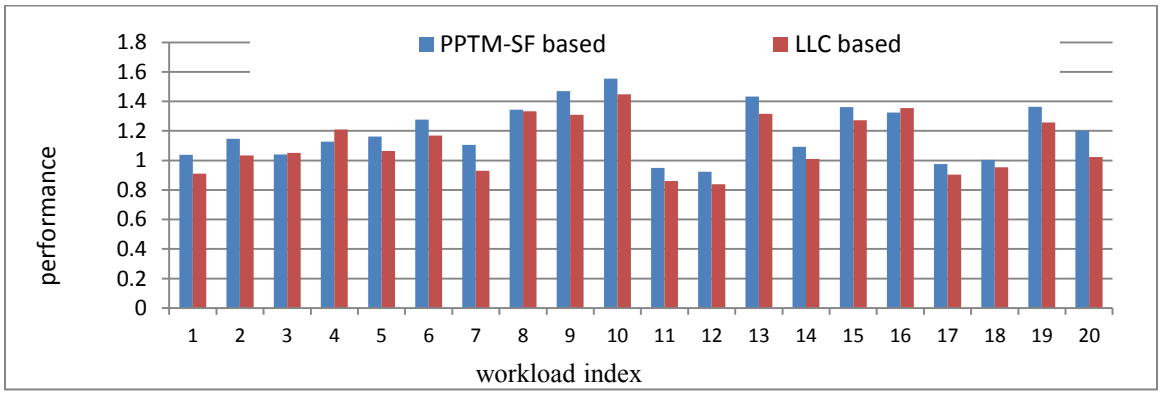
Figure 2-8 (a) and (b) give the performance comparison between PPTM based and LLC based system for both scenarios. The X-axis in both figures gives the index of workloads, and the Y-axis gives the summation of the performance of all critical tasks normalized with the respect of an ideal system. As we can see in the figure, the PPTM based task migration in average gives 4% better performance than LLC based migration for a system with 2 critical tasks and 1 idle task. It gives 9% better performance in average for a system with 4 critical tasks and 2 idle tasks. We can see that the more critical tasks we have, the better the PPTM-SF model performs than the LLC based migration. The results also show that, in general LLC miss rate can provide fairly good prediction of how the target performance will change after migration, if the rest of the workload remains that same. However, because the absolute value of the total LLC miss rate does not correlate to the absolute target performance very well, it cannot be used to provide accurate guidance for power and performance tradeoffs.

(3) Combining task migration with DVFS

In Section 2.1.5, we showed how much energy can be saved from model directed DVFS. In the next experiment, we demonstrate how task migration can create potential for further



(a) Overall performance of 2 critical tasks (scenario 1)



(b) Overall performance of 4 critical tasks (scenario 2)

Figure 2-8. Performance of model predictive task migration

energy savings and more opportunities for DVFS. Fourteen different workloads were created in this experiment consisting of randomly selected SPEC benchmarks. Each workload has 2 critical tasks and 1 idle task. Every 40 seconds new task mapping will be searched based on the performance predicated using the PPTM model. If the result is different from current mapping, task migration will be performed. Every 10 seconds the PPTM model will be used again to estimate the current performance of the critical tasks. This is done by setting both the Target and mTarget to be the same. Please note that, although PEFS model has higher accuracy in estimating performance under current task mapping, we choose to use PPTM model so that one

set of model needs to be trained and stored and the complexity of the approach could be reduced. We will increase (or decrease) one frequency step of the CPU if the performance is below (or above) the given threshold.

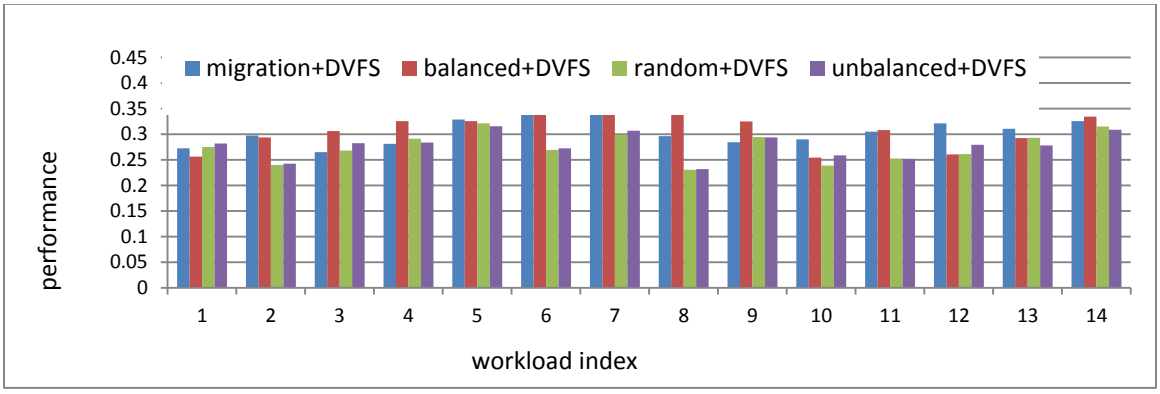
Our reference systems have static mapping, however they also performs DVFS power management based on the performance information estimated using the PPTM model. Three different static mappings are adopted. The first static mapping tries to balance the LLC miss rate and pairs the tasks with the lowest LLC miss rate with the critical tasks. We refer to this system as "balanced". The second static mapping works reversely. It pairs the tasks with the highest LLC miss rate with the critical tasks, and is referred as "unbalanced". The third static mapping randomly pairs uncritical and critical tasks, and is referred as "random". In the experiment, the average performance of 8 random mappings is reported. Please note that the LLC miss rate of an arbitrary process is unknown in a datacenter until the process has completed. Therefore, the "balanced" and "unbalanced" mappings are created simply for experimental purpose and the "random" mapping is the more realistic case.

Two different thresholds of normalized performance are used for the critical tasks. Figure 2-9 (a) and (c) report the performance of the worst critical task and the total system power consumption when the threshold is set to 0.25. Figure 2-9 (b) and (d) report the same information for the systems where the threshold is set to 0.35. The figure does not include the static power when the system is idle.

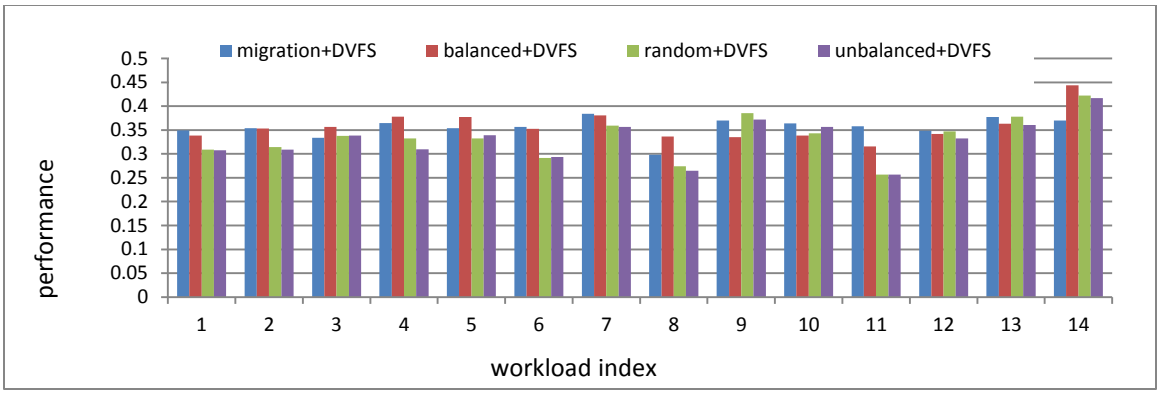
Table 2-5 gives the average of the minimum performance of the two critical tasks, the number of performance violations and the average power performance ratio of the four testing systems collected across the 14 workloads. Instead of energy, here we report the ratio between the system

power consumption and the average minimum performance of critical tasks. This is because due to different task mapping, it is difficult to make sure that all critical and noncritical tasks execute the same amount of instructions across different systems. Therefore, we use the ratio between power consumption and the average performance of the worst case critical tasks to represent power-performance tradeoffs. A smaller power performance ratio means higher energy efficiency.

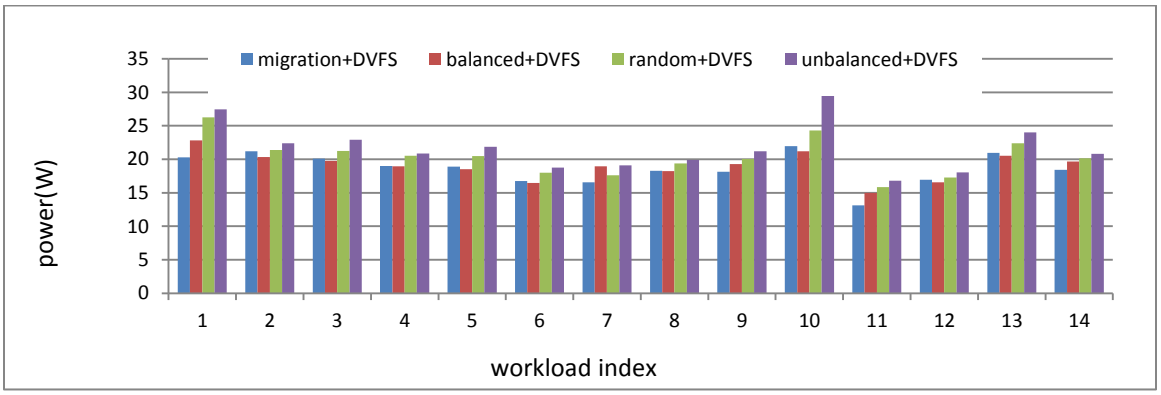
As we can see, in general all systems have more violations when performance threshold is tight (i.e. 35% of an ideal system), however our system has the least violation. This is because, although all systems perform the model based DVFS, our system is more flexible since it dynamically migrates tasks to better explore the opportunity of constraining the critical tasks' performance above threshold. At the same time, our system has the lowest power-performance ration. This is because the migration reduces part of the stress of meeting performance constraint, so our system does not simply rely on overclocking the CPU to improve performance. Therefore its power consumption is also lower than other systems.



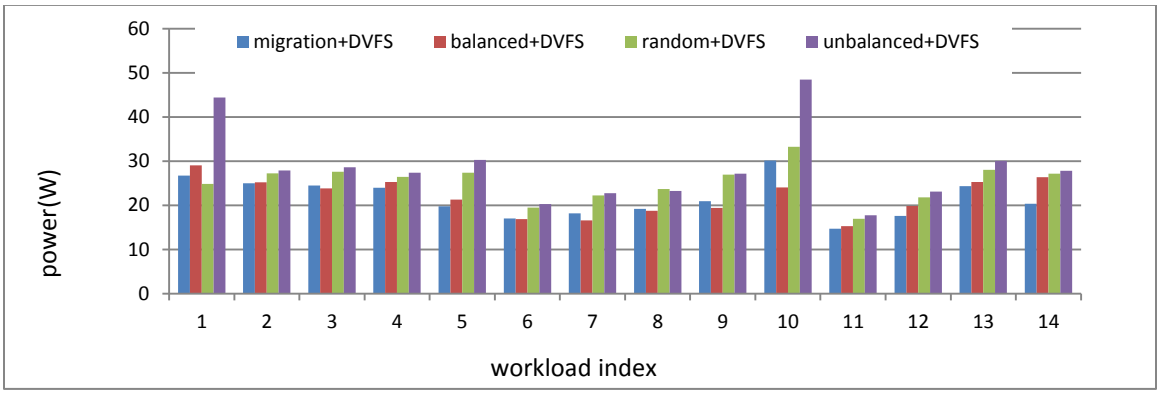
(a) Performance for threshold 0.25



(b) Performance for threshold 0.35



(c) Average power for threshold 0.25



(d) Average power for threshold 0.35

Figure 2-9 Model directed hierarchical power management

Table 2-5 Average performance, power and violations

	Performance		Number of Violations		Power- performance ratio	
	P_{th}=0.25	P_{th}=0.35	P_{th}=0.25	P_{th}=0.35	P_{th}=0.25	P_{th}=0.35
Our System	0.308	0.355	0	4	60	60
Balanced	0.312	0.358	0	6	60	61
Random	0.275	0.335	3	10	74	75
Unbalanced	0.278	0.330	2	9	78	87

2.1.6 Conclusions

In this work, we demonstrate the importance of considering both resource contention and frequency scaling in system performance modeling. A model is constructed to dynamic quantify task performance degradation with the respect to a reference system, where the target process is executed alone at the highest frequency. The propose model is used to provide performance feedback to guide DVFS control. The model is further extended to predict the performance of the target process under a new task mapping. The improved model is used to provide performance prediction to guide the task migration. Experimental results show that the proposed models effectively controls the system performance and keeps it close to the given constraint, hence leads to lower power consumption with minimum performance violation.

2.2 Learning Based DVFS and DPM for CPU and Peripheral devices

2.2.1 Introduction

Robust power management must consider the uncertainty and variability that come from the environment, the application and the hardware. For example, the workload of a complex system is usually unpredictable as it strongly depends on the nature of the application, the input data and the user context. The workload variation changes the device usage pattern and has the most significant impact on the system speed and power consumption. The contention of shared resources such as buses or I/Os in an MPSoC also increases the variability of hardware response time for communication and computation. Furthermore, the process, voltage, and temperature (PVT) variation results in a large fluctuation in hardware performance and power consumption. Therefore, statically optimized resource and power management policies are not likely to achieve the best performance when the input characteristics change. The ability to observe, learn and adapt to different hardware systems and different working environments is essential for a power management controller.

In this work, we present a novel approach for system level power management based on online *reinforcement learning* (*RL*). The proposed power manager learns a new power control policy dynamically at runtime from the information it receives. This is achieved by trying an action in a certain system state, and adjusting the action when this state is re-visited next time, based on the reward/cost received. This is a model-free approach as the power manager learns the policy directly. The technique does not require any prior information of the system or workload. However, if such knowledge is available, it can help to speed up the convergence of the learning algorithm and better track the performance (or power consumption) constraints.

A reinforcement learning model consists of three basic elements: a state space that describes the environment status, an action space that defines the available control knobs and a cost function that evaluates the reward/cost of different actions in different states. How these

three elements should be defined is determined by the available environment information, the nature of the system under control, as well as the user objectives and constraints. Therefore, it varies from problem to problem.

In this work, we investigate RL model construction for the power management of two most common types of devices in a computer, the peripheral device and the microprocessor. The peripheral device is an interactive system that processes the I/O requests generated by software applications. Its performance is measured by its response time (which is proportional to the average length of request waiting queue.) It assumes that each I/O request is an atomic task. For such peripheral device, its workload is captured by the distribution of idle intervals and the request generation rate. For the microprocessor, we focus on its power management during the time of batch processing. The performance is measured by the execution time, and the workload characteristic is captured by its CPU intensiveness. From the power management algorithm design perspective, a microprocessor in batch mode and a peripheral device differ in many ways.

First of all, they have different power management control knobs. The power consumption of a peripheral device can be controlled by configuring the device into discrete power modes. For example, a wireless adaptor usually has 4 power modes: receiving, transmitting, idle and standby modes. However, the power consumption of a microprocessor performing batch processing is usually controlled by dynamic voltage and frequency scaling. For example, the AMD Opteron processor is implemented with 5 levels of voltage and frequencies. This leads to different action spaces in the Q-learning algorithm design.

Secondly, their performances are measured by different criteria and affected by different factors. While the performance of a peripheral device is measured by its response time, which is

determined by the request incoming rate and processing speed; the performance of a microprocessor working on batch processing is measured by its execution time, which is affected by the CPU clock frequency and architectural events such as pipeline stalls. Therefore, the two devices should be modeled by different sets of parameters. This leads to different state classification methods in Q-learning model construction.

Furthermore, their optimization objectives are different. The ultimate goal of power management is to minimize the energy dissipation under the given performance constraint. For peripheral devices, an infinite horizon is usually assumed. Therefore, minimizing energy dissipation is the same as minimizing the average power consumption over a long time. However, for a microprocessor working on batch processing, we usually focus on minimizing the energy dissipation over the execution time. While the average power consumption of a peripheral device is a monotonic decreasing function of its response time; such monotonic relation does not always exist between the total energy dissipation and execution time of a modern microprocessors. On one hand, although lowering the voltage and frequency of a CPU effectively reduces the dynamic energy, it also increases the leakage energy because the system has to be kept active for a longer time [25]; therefore, the total energy may not necessarily decrease. On the other hand, as the limited memory bandwidth has already become the performance bottleneck for some high performance computers, lowering CPU voltage and frequency to a certain extent may not have significant impact on performance, since the memory subsystem still works under a constant frequency [9][25][27]. As we can see, in addition to different objective functions, the relations between objectives and constraints are also different for the two power management problems. Hence, different cost function must be constructed.

Finally, other constraints sometimes are usually considered in microprocessor power management. For example, temperature has significant impact on the chip performance and reliability, and hence should be considered as another constraint. This requires a cost function design that is flexible enough to incorporate multiple constraints.

Above discussions show that different Q-learning models (i.e. different environment state classification methods and different cost function formulations) are needed for peripheral device power management and microprocessor power management. These two power management problems are both interesting and are complementary to each other. It is important to discuss their Q-learning models separately. In this work, we focus on how these Q-learning models are constructed and how effective they will be. In addition to model construction, techniques are developed to enhance the performance of the RL based controller in a power managed system. Novel techniques are proposed that speed up the convergence of learning and better maintain the required performance (or power) constraint in a dynamic system. The following summarizes the main contribution of this work:

1. We present a power manager that does not require any prior knowledge of the workload. It learns the policy online with real-time information and adjusts the policy accordingly. After a certain set-up time, the optimal policy can positively be found.

2. We propose a set of enhancement techniques that utilize the partial knowledge of the system to accelerate the convergence speed and enable the runtime tracking of the performance (power consumption) constraint.

3. We apply the RL based controller to perform power management of peripheral devices and microprocessors. Different model construction approaches are discussed for these two types

of devices. The performance of the proposed power management controller is evaluated by either simulation or real measurement.

Compared to the previous works in [29][43], this work has the following major contributions.

1. In addition to the peripheral devices, we also apply the RL based power management to microprocessors. Model construction and cost function formulation are discussed.

2. The RL model construction for peripheral devices is improved to handle real application scenarios with more diversified workload and practical constraints. For example, we improved the state partition techniques to cover workloads with large variations.

3. While the traditional stochastic power management is able to satisfy the given constraints on long term average performance (or power consumption), they usually have large performance (or power consumption) variations during short period of time. In this work, a two level controller is proposed to find the weight factor that balances the power-performance tradeoff of the learning based power management policy so that it operates at a relatively constant performance (or power consumption) that is close to the given constraint.

4. More experimental data are provided. In addition to traces collected from personal PCs, the proposed power management technique is evaluated using the HP hard disk traces that resemble the workload of large data centers. It is also implemented on a Dell Precision T3400 workstation to control the runtime voltage and frequency scaling for simultaneous energy, performance and temperature management.

The rest of this section is organized as follows: Subsection 2.2.2 talks about the related work including the expert-based DPM algorithm, which will be used as a comparison with our algorithm. Subsection 2.2.3 introduces the general RL model for power management. Subsections 2.2.4 and 2.2.5 discuss model construction and enhancement techniques for the power management of peripheral devices and microprocessors, respectively. Subsection 2.2.6 presents the experimental results. Finally 2.2.7 gives the conclusions.

2.2.2 Related Works

Based on when it is applied, system level low power techniques can be categorized into design time approaches and run time approaches. The former modifies and optimizes the system and component architecture during design time for a lower power consumption or to facilitate runtime power reduction[10][18][2][42]; while the latter performs online to dynamically control the power with the respect of performance constraints. Dynamic power management (DPM) and dynamic voltage frequency scaling (DVFS) belong to the second category.

The simplest and most widely used DPM algorithm is the timeout policy which puts the device into low power mode after it has been idle for certain amount of time. Though it is easy to implement and relatively effective in many computer systems, the timeout policy is far from optimized because it wastes energy during the timeout period. Furthermore, the traditional timeout policy uses a fixed timeout value which cannot adapt to the change of workload or user context.

In order to best adjust itself to the dynamic system, many DPM works on a system model that is learned from the history information. For example, the predictive DPM[21] predicts the next idle time based on previous idle time and makes power mode switching decision based on

the predicted value. The previous works in stochastic power management [38][39][44] model the system as a Markov decision process. The model construction requires offline learning. [48] proposed a user-based adaptive power management technique that considered user annoyance as a performance constraint. [3] converts the scheduling task on multiprocessor into a cooperative game theory problem to minimize the energy consumption and the makespan simultaneously, while maintaining deadline constraints.

Many research works have been proposed to find the optimal DVFS scheduling for energy reduction. [9] uses runtime information on the statistics of the external memory access to perform CPU voltage and frequency scaling. Its goal is to minimize the energy consumption while translucently controlling the performance penalty. [45] first presents a workload prediction model for MPEG decoder and the predicted workload is further used to guide the voltage and frequency scaling. [11] considers processors as producers and consumers and tunes their frequencies in order to minimize the stalls of the request queue while reducing the processors' energy.

Multidimensional constraints sometimes are considered in power management. Performance and temperature are two typical constraints in designing a power management policy for microprocessors. [13][28] propose to use mathematical programming to solve voltage and frequency scheduling problem for energy optimization with temperature constraints. Both works assume that the workload is known in advance. [51][54] apply model predictive control (MPC) to find the best sequence of voltage and frequency settings for minimum energy under given temperature constraint over a finite horizon. A temperature model is required for the MPC controller to work effectively. [6] controls active memory modules and schedules workload

between CPU sockets to achieve balanced thermal distribution for energy management with temperature and performance constraints.

All of the above works either assume given workload model or require offline model construction and policy optimization, therefore they cannot adapt to the workload changes in real-time. Online learning algorithms are natural choices for real-time adaptive power management. [7] presents a method that periodically adjusts the size of physical memory and the timeout value to turn off the hard disk to reduce the average energy consumption. The joint power management predicts the next hardware accesses frequency and idle interval based on previous information. [20] uses program counters to learn the access patterns of applications and predicts when an I/O device can be shut down to save energy. [52] uses a skewed striping pattern to adaptively change the number of powered disks according to the system load. They also enhanced the reliability of the storage system by limiting disk power cycles and using different RAID encoding schemes. [30] and [23] propose a machine learning approach for multicore resource management using on-chip hardware agents that are capable of learning, planning, and continuously adapting to changing demands. Those works also use the machine learning technique to perform DRAM bandwidth scheduling for a maximum throughput. In [16], the authors propose a learning algorithm that dynamically selects different experts to make power management decisions at runtime, where each expert is a predesigned power management policy. This approach leverages the fact that different experts outperform each other under different workloads and hardware characteristics. The similar approach is applied in [14] to perform power management with performance and temperature constraints. There is also a large body of works that learn workload/temperature model online for thermal management [4][15][53], or

uses random policy with temperature aware adaptation [12]. However, these works do not consider energy (or power) minimization.

Reinforcement learning has been applied to resource allocation and further extended to microprocessor power management in [46] and [47], respectively. Both works focus on the web application servers. The environment state is describe by the rate of the incoming request and the policy is optimize offline using the Sarsa(0) algorithm. In order for the offline trained policy to work effectively, the implied assumption is that the workload is highly repeatable. Our approach differs from these two works in that, we adopt different state classification; focus on general purpose computing applications and our policy is optimized online.

2.2.3 General Architecture of Q-learning based Power Management

In this chapter, we will first introduce the principle of Q-learning and then we will discuss how to extend the traditional Q-learning algorithm to solve the dynamic power management problem.

(1) Q-learning algorithm

Reinforcement learning is a machine intelligence approach that has been applied in many different areas. It mimics one of the most common learning styles in natural life. The machine learns to achieve a goal by trial-and-error interaction within a dynamic environment.

The general learning model consists of

- An agent
- A finite state space *S*
- A set of available actions A for the agent

- A penalty function $P: S \times A \rightarrow P$

The goal of the agent is to minimize its average long-term penalty. It is achieved by learning a policy π, i.e. a mapping between the states and the actions.

Q-learning is one of the most popular algorithms in reinforcement learning. At each step of interaction with the environment, the agent observes the environment and issues an action based on the system state. By performing the action, the system moves from one state to another. The new state gives the agent a penalty which indicates the value of the state transition. The agent keeps a value function $Q^{\pi}(s, a)$ for each state-action pair, which represents the expected long-term penalty if the system starts from state *s*, taking action *a*, and thereafter following policy π. Based on this value function, the agent decides which action should be taken in current state to achieve the minimum long-term penalties.

The core of the Q-learning algorithm is a value iteration update of the value function. The Q-value for each state-action pair is initially chosen by the designer and then it will be updated each time an action is issued and a penalty is received based on the following expression.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\varepsilon_t(s_t, a_t)}_{learning\ rate} \times \left[\overbrace{\underbrace{p_{t+1}}_{Penalty} + \underbrace{\gamma}_{Discount\ Factor} \underbrace{\min_a Q(s_{t+1}, a)}_{Min\ Future\ Value}}^{Expected\ Discounted\ Penalty} - \overbrace{Q(s_t, a_t)}^{Old\ Value} \right] \tag{2.1}$$

In the above expression, s_t , a_t and p_t are the state, action and penalty at time *t* respectively, and $\varepsilon_t(s, a) \in (0,1)$ is the learning rate. The discount factor γ is a value between 0 and 1 which gives more weight to the penalties in the near future than the far future. The next time when state *s* is visited again, the action with the minimum Q-value will be chosen, i.e.

$\pi(s) = min_{a \in A} Q(s, a)$. The value of $Q(s_t, a_t)$ is updated at the beginning of cycle *t+1*, i.e., the *Q*-value for the state-action pair of the previous cycle is updated at the beginning of current cycle.

As a model-free learning algorithm, it is not necessary for the Q-learning agent to have any prior information about the system, such as the transition probability from one state to another. Thus, it is highly adaptive and flexible.

(2) Q-learning Model for Power Management

Figure 2-10 shows the general architecture of a Q-learning based power management system. It consists of two parts, the environment and the controller. The environment can further be divided into hardware and software environments. The hardware environment could be any peripherals device such as hard disk and network card or the microprocessor. The software environment includes OS, application software, user inputs, etc. The controller continuously observes the environment and manages the control knobs (also denoted as the actuators in the figure). The environment information can be obtained through different channels. Some of the I/O requests and software activities can be observed through the operating system, the architecture event can be observed by reading the performance counters, and some of the device physical information (such as temperature) can be obtained by reading the embedded sensors. Based on the environment information, the current system state will be classified and the penalty of current state action pair will be calculated. This penalty information will be used to update the Q-values. The best action (i.e. a setting of the control knobs) that has the lowest Q-value will be selected to control the states of the actuators. A discrete-time slotted model is used throughout this work, which means all the decision making and system state updating occur on a cycle basis.

A time slot *n* is defined as the time interval [*nT*, *(n+1)T*], and the power manager makes decision for this time slot at the beginning of this interval at time *nT*.

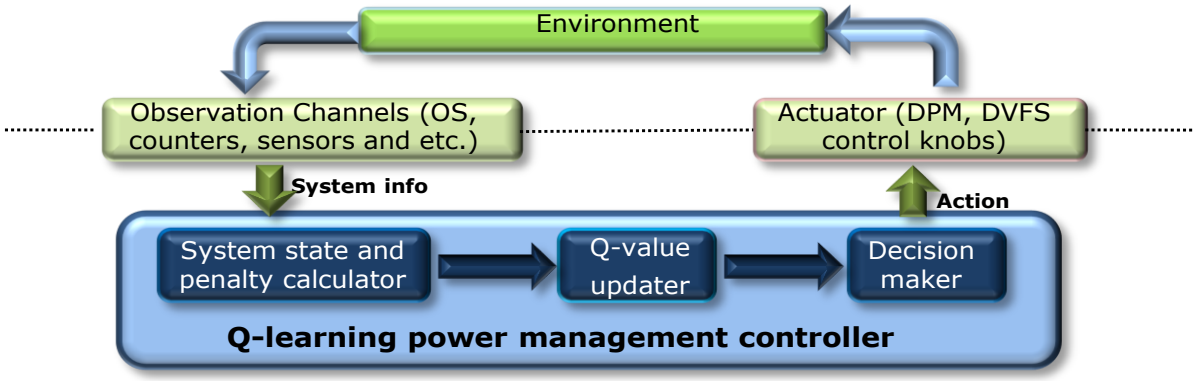


Figure 2-10. Illustration of system under power management.

To construct the Q-learning model for a given control problem, three questions need to be answered: 1. How to classify the environment status into different state space of the Q-learning model? 2. How to formulate cost function from the observed information? 3. Given the set of actuators, what controls are available? In the rest of the work, we will answer these questions to design a Q-learning model for the power management of peripheral devices and microprocessors. Although the objectives of the two problems are similar, because the performance and operation status of peripheral devices and microprocessor are usually characterized by different parameters and requires different control knobs, different Q-learning models must be constructed for these two power management problems. Detailed discussion in model construction will be provided in later sections.

(3) Enhanced Q-learning

Q-learning is originally designed to find the policy for a Markov Decision Process (MDP). It is proved that the Q-learning is able to find the optimal policy when the learning rate α is reduced to 0 at an appropriate rate, given the condition that the environment is MDP. However,

it is important to point out that a computing system for power management is typically non-Markovian. First of all, the workload of most computing system exhibits long range similarity [50] and hence the request pattern generated by the environment in our power management system is most likely to be non-Markovian. Furthermore, even if the underlying system is Markovian, what the power manager observes may not be Markovian due to the noise and disturbance, such as state aggregation, during the observation. As we mentioned earlier, the Q-learning may not be able to find the optimal policy in a non-Markovian environment. Nevertheless we still choose Q-learning to solve this problem because of its simplicity and also because of its robustness to endure noise.

Reinforcement learning in a non-Markovian environment is an open problem. Many research works have investigated the feasibility of applying the traditional RL algorithms to solve the decision problem in a non-Markovian environment or a partially observable Markovian environment [35][41]. The author of [35] applies five RL algorithms in a noisy and non-Markovian environment and compared their performance and convergence speed. Their results show that the Q-learning exhibits the highest robustness at low noise level and medium robustness at high noise level. However, the convergence speed of Q-learning reduces the most drastically when the noise level increases. In [41] the similar results are reported. Q-learning is capable to achieve the same performance as the other two reference learning algorithms at the cost of slower convergence speed. Based on the study we conclude that the major limitation of Q-learning, when being applied in a non-Markovian environment, is its convergence speed.

Traditional Q-learning assumes no prior information of the environment. However, in a power management system, the model of system can be pre-characterized. We know exactly how many power modes the system has and how it switches its power mode given a power

management command. In other words, we have partial information of the power management system. Based on this information, we are able to design an improved Q-learning algorithm with faster convergence speed. More details are provided in Section 2.2.4.

2.2.4 Learning based Power Management for Peripheral Devices

In this section, we will introduce the details of designing a Q-learning based power management algorithm to achieve the performance and power tradeoff for a peripheral device. The peripheral devices, also known as input/output devices, can be considered as a *service provider* (*SP*). The request to the device is buffered in a *service request queue* (*SQ*) maintained by the OS. The software application that accesses the device is considered as *service requestor* (*SR*).

A peripheral device usually has several different working modes and low power modes. The state of SP can be partitioned based on its power modes. The time to transit from one state to another is hardware specific and is assumed to be known. The state of SR can be classified by its request generation rate, which is time varying. The transition rate from one SR state to another is usually unknown. Furthermore, such transitions are usually non-Markovian. SQ is a queuing model and its states are classified based on the number of waiting requests. Obviously, the state transition rate of the SQ is determined by the request generation rate and request processing rate, which can be derived from the status of SP and SR. Finally, the environment of the power management controller is modeled as the composition of the SP, SQ and *SR*.

Figure 2-11 gives an example of SP, SR and SQ models. The SP has two power modes, active mode and sleep mode. They can switch to each other based on the power management command. The SR has three request generation mode, high speed, low speed and idle mode.

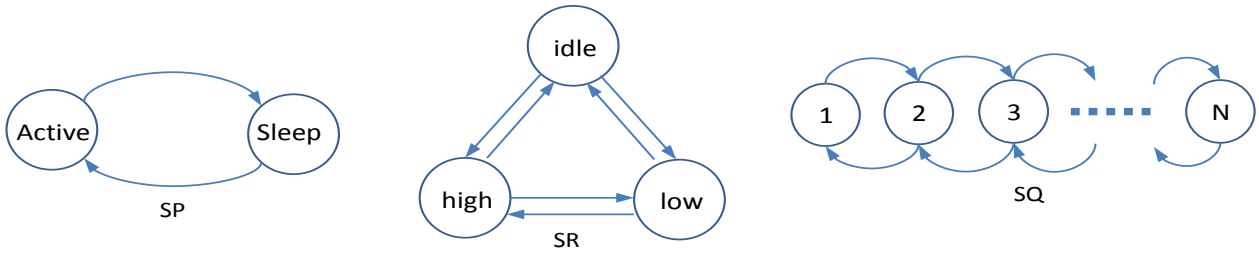


Figure 2-11. State transition diagram of SP, SR and SQ models.

They can also transit to each other. The SQ can hold up to *N* requests and each time the number of requests can only increment or decrement by one.

(1) State Partition and Penalty Calculation

The observed power mode of SP can naturally be used to represent its state. SP has two types of states, stable state and transient state. During the stable state (e.g., active states and sleep states), the SP stays at a specific power mode. It processes the request at a certain speed (which could be as low as zero in sleep state). The learning agent observes the environment and issues power management command periodically. During the transient state, the SP switches from one power mode to another. It does not process any request. The learning agent halts during the transient state because the SP does not respond to any power management command.

The state of SR is classified based on the rate of the incoming request. Due to the high variation of the workload, this value is a random variable distributed over a wide range and it can almost be considered as continuous. In order to reduce the state space, it is necessary to discretize the values into fewer states. In order to adapt to different workload intensities, we propose to partition the rate of incoming request based on its exponential moving average (EMA) [Hwang and Wu 2000]. The EMA of current cycle i is calculated as $EMA_i = \alpha \cdot EMA_{i-1} + (1-\alpha) \cdot sr_i$, where EMA_{i-1} is the exponential moving average request rate calculated in previous

cycle and sr_i is the observed incoming request rate of current cycle. Let N denote the total number of states of SR. The SR is in state 0 and N-1 when the incoming request rate is in the range $\left[0, 2^{-\lceil N/2 \rceil + 1} EMA\right]$ and $\left[2^{\lceil N/2 \rceil - 1} EMA, \infty\right]$ respectively. The SR is in state i, $0 < i < N - 1$ when the incoming request rate is in the range $\left[2^{-\lceil N/2 \rceil + i} EMA, 2^{-\lceil N/2 \rceil + i + 1} EMA\right]$. The state of SQ is classified based on the length of the queue. State aggregation is also adopted to reduce state space.

In order to find the best tradeoff between power and performance, we define a Lagrangian cost for each state and action pair (s, a) that combines the costs of power consumption ($power(s, a)$) and performance penalty ($q(s,a)$) :

$$C(s, a; \lambda) = Power(s, a) + \lambda q(s, a) \tag{2.2}$$

When SP is in a stable state, $Power(s,a)$ and $q(s,a)$ represent the system power consumption and the number of waiting request of current state. When SP is in a transient state, because the Q-learning agent will be suspended as mentioned before, we are not able to update the cost until the end of the transient state. Therefore, the accumulated cost during the entire switching period should be calculated. Furthermore, many systems have non-symmetric penalty for switching into and switching out from a low power mode. Sometime turning off the device may be effortless, but we still need to anticipate the difficulty to turn it back on in the future. Based on these motivations, for a transient state s where SP switches from power mode A to power mode B, the power cost is calculated as the average of the energy dissipation to switch from A to B and from B to A, i.e. $Power(s, a) = (P_{A2B} * T_{A2B} + P_{B2A} * T_{B2A})/2$, where P_{A2B}, P_{B2A} are power consumptions during A to B and B to A switch respectively, and T_{A2B}, T_{B2A} are delays of those switches. The performance cost is calculated as the average accumulated request

delays during the time the SP switches from A to B and from B to A, i.e. $q(s,a) = (q_{A2B} * T_{A2B} + q_{B2A} * T_{B2A})/2$, where q_{A2B}, q_{B2A} is the average request incoming rate during the power mode switching along the history. To give an example, consider a hard disk drive. To transit this hard disk from sleep state to active state usually associates with long latency and high power consumption because we have to spin up the disk mechanically. During the transition, all the new incoming requests will be accumulated in SQ. This transition will not be necessary if the disk didn't go to sleep state at all in previous decision. With this knowledge, we distribute the penalty evenly between the sleep to active and active to sleep transitions so that SP will not transit to sleep state (normally taking little effort) aggressively. Our experiment shows that such cost function calculation for the transient state leads to better result.

Given the next state *s'* and its Q values, the learning agent updates the Q-values of the state action pair (*s*, *a*) periodically using the following equation.

$$Q(s,a;\lambda) = \left(1 - \varepsilon_{(s,a)}\right)Q(s,a;\lambda) + \varepsilon_{(s,a)}(C(s,a;\lambda) + min_{a'}\, Q(s',a';\lambda) \qquad (2.3)$$

The Q-value of state action pair (*s*, *a*) reflects the expected average power and request delay caused by the action a taken in state *s*. The new action a' with minimum Q-value $min_{a'}\, Q(s',a';\lambda)$ will be issued at state s'.

(2) Accelerating the Speed of Convergence of Q-learning

The convergence of the Q-learning relies on the recurrent visits of all possible state-action pairs. Based on equation (2.3) we can see, each time a state *s* is visited and an action *a* is taken, a corresponding *Q* value *Q*(*s*, *a*) is updated. It is calculated as the weighted sum of itself and the best Q value of the next state *s*', i.e.

$Q(s,a) = (1-\varepsilon)Q(s,a) + \varepsilon(C(s,a) + \min_{a'} Q(s',a'))$. The frequency that state *s'* occurs after state-action pair (s, a) reveals the information of the system transition probability. In traditional Q-learning, only the Q value corresponding to the actual visited state-action pair will be updated. This is because the controller has no information of the system dynamics, and it totally relies on the actual execution trace to find out the next state information for a given state-action pair.

In contrast to conventional Q-learning systems, we do have some partial information of our target power management system. The state of a power management system is a composition of the states of SP, SR and SQ. Among these three, only SR has unknown behavior. The state space SP is the set of available power modes and its power consumption, processing speed and power mode transition overhead are known. We also know that SP and SR are independent to each other, and when SP and SR are given, the behavior of SQ is determined.

Based on the available information on SP and SQ, we propose to update more than one Q values each cycle to speed up convergence. For each visited state-action pair ((*sp*, *sr*, *sq*), *a*) we will update the *Q* values for a set of state-action pairs $\{((sp', sr, sq'), a') | \forall sp' \in SP, \forall sq' \in SQ, \forall a' \in A\}$. These state actions pairs are referred as virtual states and actions, because we assume that the system has (virtually) visited these state action pairs and will update their Q values. Note that all virtual state has the same SR state which is the actual SR state that the system has recently visited. In order to update the *Q* values of a virtual state-action pair ((*sp'*, *sr*, *sq'*), *a'*), we need to know, starting from this state action pair, what the next system state will be even though it is not currently being visited. More specifically, given the information that the system was in state (sp_t, sr_t, sq_t) and it switched to (sp_{t+1}, sr_{t+1}, sq_{t+1}) after action *a* is taken, we would like to guess what the next state (sp_{t+1}', sr_{t+1}', sq_{t+1}') will be if the system is currently in a different state (sp_t', sr_t, sq_t') and another action *a'* is taken.

Given the current state sp_t' and action a', it is not difficult for us to find the next state sp_{t+1}' as the SP model is pre-characterized. We also know that the SR works independently to the SP. Regardless of the state of SP, the requests are generated in the same way. Therefore, sr_{t+1}' is the same as sr_{t+1}. The value of sq_{t+1}' (i.e. the number of waiting requests) depends on both the number of incoming requests and the number of requests that have been processed. The former is determined by the state of SR and can be measured from the actual system, while the later is determined by the processing speed of SP at current power mode sp_t'. Because SP has been pre-characterized, this information can also be estimated fairly accurately. After the next state is determined, the Q values of the state-action pair $((sp_t', sr_t, sq_t'), a')$ that has been virtually visited can easily be calculated. In the rest of the work, we refer to this technique as *Virtual State Switching* (*VSS*).

Using VSS, the number of Q-values that would be updated in each cycle is $|SP| \times |SQ| \times |A|$, where $|SP|$, $|SQ|$ and $|A|$ are the cardinality of the *SP*, *SQ* and *A*. The complexity of the constrained Q-learning is $O(|SP| \times |SQ| \times |A|)$. The size of SP state space and action space is fixed for a given hardware. With a carefully controlled *SQ* state partition, this computation complexity is affordable.

We further improve the convergence speed of the proposed Q-learning algorithm by adopting a variable learning rate. Compared to the traditional Q-learning, the learning rate $\varepsilon_{(s,a)}$ is not fixed in our algorithm. Instead, it is dependent on the frequency of the visit to the state-action pair (s, a) and is calculated as:

$$\varepsilon_{(s,a)} = \frac{\mu}{Visit(s,a)} \tag{2.4}$$

where $Visit(s,a)$ is the number of times that the state-action pair (s,a) has been visited, and μ is a given constant.

Figure 2-12 gives the pseudo code for the power management controller using enhanced Q-learning algorithm with VSS. The algorithm is executed at the beginning of each time slot. Its input is current environment state s_t, the previous environment state s_{t-1}, the action a_{t-1} in last cycle, and the weight coefficient λ. Each time, it updates the Q values of the real state action pair (S_{t-1}, a_{t-1}) as well as all the virtual state action pairs (S'_{t-1}, a'_{t-1}).

It is important to point out that the more information we know about the system, the more accurate projection we can make about the virtual state switching. If we do not have enough information or cannot find solid reasoning to project the virtual state switching, we may apply VSS only to a small set of state-action pairs.

(3) Power (Performance) Tracking using 2-level Controller

For general peripheral devices, power and performance are two metrics inversely proportional to each other in many computing systems. In general, a performance constrained system achieves the lowest power dissipation when delivering just enough performance as required (or vice versa for a power constrained system). The Lagrange cost function defined in equation (2.2) enables us to find tradeoff between power and performance by varying the parameter λ. However, what is the right value of λ that exactly meets the power (or performance) constraint is difficult to find out.

It is known that when the value of λ increases, the Q-learning algorithm will favor policies that have better performance and vice versa. By comparing the actual power

Q_Learning_Power_Manager(S_t, S_{t-1}, a_{t-1}, λ)

Input: Current state $S_t = (sp_t, sr_t, sq_t)$, last state $S_{t-1} = (sp_{t-1}, sr_{t-1}, sq_{t-1})$, action a_{t-1}, and weight coefficient λ;
Calculate the cost $C(S_{t-1}, a_{t-1}; \lambda)$ using Equation (2.2);
Calculate the Q-value $Q(S_{t-1}, a_{t-1}; \lambda)$ using Equation (2.3);
For each ${sp_{t-1}}' \in SP$ {
  For each ${sq_{t-1}}' \in SQ$ {
    For each action ${a_{t-1}}' \in A$ {
      If (${sp_{t-1}}' \neq sp_{t-1} \parallel {sq_{t-1}}' \neq sq_{t-1} \parallel a'_{t-1} \neq a_{t-1}$) {
      / *Do not update the Q-value of the real state action pair twice*/
        Given the virtual state action pair $(S'_{t-1}, a'_{t-1}) = ((sp'_{t-1}, sr_{t-1}, {sq_{t-1}}'), a'_{t-1})$, find the projected next state ${S_t}' = ({sp_t}', {sr_t}', {sq_t}')$;
       Calculate the cost $C({S_{t-1}}', {a_{t-1}}'; \lambda)$ using Equation (2.2);
       Calculate the Q-value $Q({S_{t-1}}', {a_{t-1}}'; \lambda)$ using Equation (2.3);
        } /* end if*/
      }
    }
} /* end for */

Choose action a_t with $min_{a_t} Q(S_t, a_t; \lambda)$;

Figure 2-12. Pseudo code for Q-learning power manager using the VSS technique.

consumption (or performance) to the power (or performance) constraint, we can adjust the value of λ using a feedback control. However, without knowing the exact relation among power, performance and λ, the feedback control method will easily generate large overshoot or undershoot in measured output (i.e. power consumption or performance) and hence lead to an unstable system [1]. To limit the overshoot and undershoot, we propose to further confine the value of λ in a predefined range. The upper bound and the lower bound of the range are estimated from the long term average workload characteristics and the given power (performance) constraints using a neural network.

The above analysis leads to a 2-level control unit that tunes the value of λ to keep the system aligning to the given constraint. The proposed 2-level constraint tracking unit has a neural network based coarse grained controller in the first level to set the upper and lower bound

of λ based on the long term average workload. It also has a feedback controller in the second level to fine tune the value of λ based on the instantaneous workload variations.

Here we consider the problem of maximizing performance for a given power constraint as an example to illustrate the 2-level controller. Its dual problem, i.e. minimizing power consumption for a given performance constraint can be solved in a similar way.

The concept of two level controllers has been successfully applied in many power/thermal management works. For example, [51] uses online model estimator to predict the workload and generate the desirable trajectory of reference power consumption at lower level and feedback control to keep the actual power consumption close to the trajectory. [5] performs core level proactive thermal management at lower level and socket level task scheduling at upper level. Here we need to point out that the controller in this section does not directly manage the control knobs. Instead, it controls the value of the Lagrange multiplier (λ) which is used to realize power-performance tradeoff in the Q-learning algorithm.

(a) Boundary Prediction using Neural Network Models (Level 1 Controller)

The goal of the first level controller is to estimate the value of λ that exactly meets the performance/power constraint considering only the long term average workload. The estimated value is denoted as $\hat{\lambda}$. Using $\hat{\lambda}$ as a reference, we set the upper and lower bound of the second level controller which fine tunes the value of λ based on the instantaneous workload variation.

We found from the experiments that it is difficult to construct a model that estimates $\hat{\lambda}$ directly from power (performance) constraint. This is probably because our Q-learning algorithm has discrete behavior and it is very likely that slight change in λ does not make difference in

control policy. In other words, the relation from the average power consumption (or performance) to λ is a one to many mapping instead of a properly defined function, and hence it is difficult to obtain. Fortunately, power (or performance) of a peripheral device is a monotonic increasing (or decreasing) function of λ. This means that we can use binary search to find the appropriate value of $\hat{\lambda}$, if there is a model that predicts the average achieved power (performance) based on the given λ. A neural network is used for such modeling purpose.

The neural network model adopted in this work has an input layer, an output layer and a hidden layer as shown in Figure 2-13. The hidden layer consists of 5 neurons. For a given service provider, the neural network model predicts the average power consumption based on the selected tradeoff factor λ and workload information.

In our experiments we observed that, when controlled by the learning based power management unit, the average power consumption of the device has a log-linear relation with the tradeoff factor λ. Figure 2-14 gives the relation of the simulated power consumption and the

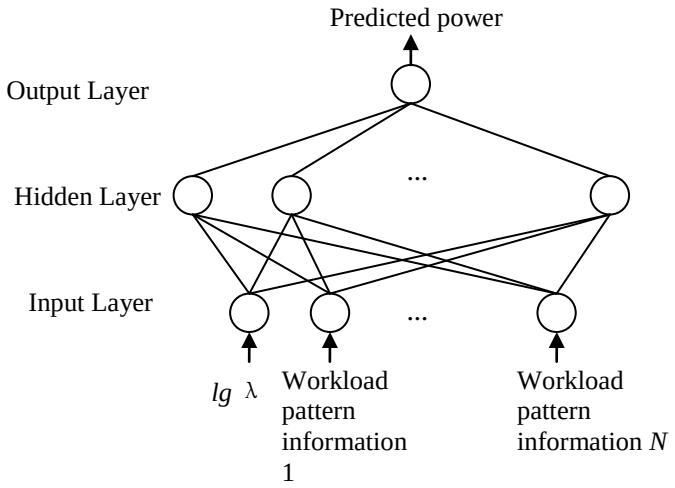


Figure 2-13. Level 1 neural network.

value of $lg\lambda$ of a hard drive whose read/write activities follows the HP Cello99 trace [40][34]. As we can see that their relation is approximately linear. To reduce the nonlinearity of the neural network model, we choose $lg\lambda$ instead of λ as one of its inputs.

What input variables should be selected for the neural network to represent the average workload characteristics is a nontrivial problem. For those peripheral devices where service speed is much faster than the request incoming speed, (for example, in general a hard disk drive can process all accumulated read/write request in very short time after the disk has been spun up), the input variables could be the probability distribution of the request inter-arrival time which reflects current workload pattern.

The probability distribution of the request inter-arrival time is represented by a set of variables. The *i*th variable gives the probability that the inter-arrival time t_{int} is greater than or equal to iT, where T is a user defined time period. Similar to many other estimation models, an accurate power estimator needs to have both good specificity and high sensitivity. Selecting too few input variables may lead to low sensitivity of the model as it misses much useful information. However, including too many input variables may cause low specificity because useful features will be covered by noises. We propose to use the greedy feature selection method [8] to select only those variables that give the most information to the prediction of average power consumption.

In our experiment, a neural network is constructed to predict the power consumption of a hard disk drive under learning based power management. We select T as $1/4\,T_{be}$, where T_{be} is the break-even time which is the minimum amount of time that a device must stay in low power mode for the energy saving to equal the overhead of power mode switching.

Table **2-6** gives the prediction error for different input selections for the HP Cello99 workload. For more details of the hard disk drive model and the Cello99 workload, please refer to Section 2.2.5. As we can see, including too many features does not help to increase the accuracy of the model because this introduces more noise in the input and will actually decrease the specificity of the model. On the other hand, a model based on extremely few inputs is not accurate either because it does not have enough sensitivity to detect a workload change.

Table 2-6. Input selection vs. prediction error.

Input selection	$i = 1$	$i = 6$	$i = 12$	i=1,2,3…15	i = 1, 6, 12
Prediction error	27%	17%	30%	14%	8%

Considering the fact that T_{be} is $4T$ for our experiment device, the selection of probabilities for idle intervals longer than T, $6T$ and $12T$ is reasonable as they represent the short idle, medium idle, and long idle intervals, thus form relatively complete spectrum of idle interval information of the workload.

The training of the neural network relies on recorded operation information of the system. For better accuracy, different models may be constructed for different types of workload if they can be classified.

With the neural network, we predict the tradeoff factor that exactly satisfies the given power (performance) constraint and denote the value as $\hat{\lambda}$. We confine the range of the tradeoff factor to be$(\hat{\lambda}/C\,, C\hat{\lambda})$, where C is a constant that is greater than 1. Consequently, the value of $lg\lambda$ is confined to the range $(lg\hat{\lambda} - C, lg\hat{\lambda} + C)$.

(b) Fine Adjustment using Feedback Control (Level 2 Controller)

In order to fine tune the value of λ, we use a linear model to approximate the relation between $lg\lambda$ and the power consumption P for a given workload, i.e. $P = A * \lg(\lambda) + B$, where A and B are unknown coefficients. Such linear relationship has been observed in our experiment as shown in Figure 2-14. The values of A and B are assumed to be constant when workload does not change abruptly and λ is confined to a limited range. Let P_{curr} and λ_{curr} be the current power consumption and current value of λ, also let P_{goal} and λ_{goal} be the power constraint and the

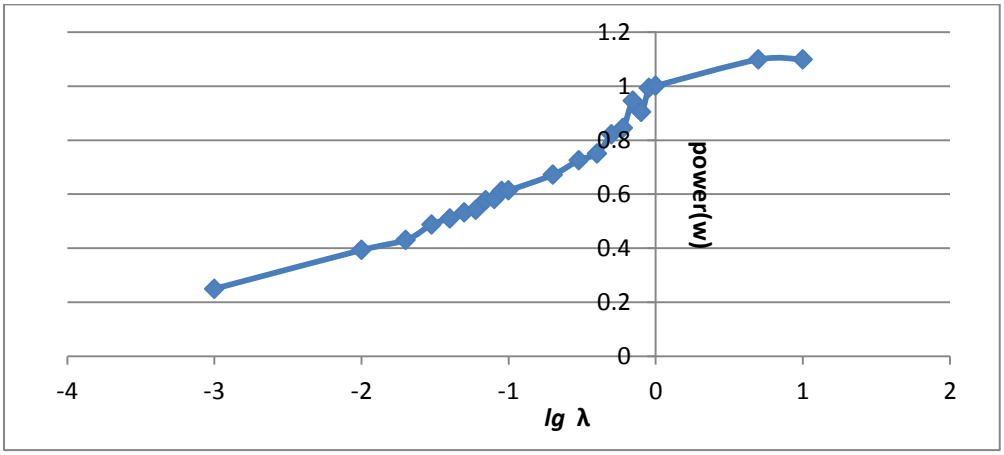


Figure 2-14. Relation between power and *lg λ* for a given workload.

corresponding value of λ that exactly achieves this power constraint. If λ_{curr} and λ_{goal} are not too far from each other, we will have equation (2.5) and (2.6):

$$P_{curr} = A * lg\,\lambda_{curr} + B \quad (2.5)$$

$$P_{goal} = A * lg\,\lambda_{goal} + B \quad (2.6)$$

Combining Equation (2.5) and (2.6), the goal value of λ can be calculated as the following:

$$\lambda_{\text{goal}} = \lambda_{curr} * 10^{\frac{P_{goal} - P_{curr}}{A}} \quad (2.7)$$

The value of A can be obtained by observing the average power consumption of the system under different λ's. Let P_1 and P_2 be the average power consumption of the system using λ_1 and λ_2, A can be calculated using Equation (2.8).

$$A = (P_1 - P_2)/(lg\lambda_1 - lg\lambda_2)\ . \quad (2.8)$$

(c) Overall Flow

Figure 2-15 gives the block diagram of the overall flow of the Q-learning power manager with constraint tracking. The function *Q-learning_power_manager*() is the basic Q-learning function shown in Figure 2-12. Both level 1 and level 2 controllers are triggered periodically. The level 2 controller is triggered at a higher frequency than the level 1 controller. In our experiment, the periods are set to 1000 and 200 cycles for level 1 and level 2 controllers respectively. When level 2 controller is triggered, A and λ_{goal} will be calculated using Equation (2.8) and (2.7). If $lg\ \lambda_{goal}$ is out of the range $(lg\hat{\lambda} - C, lg\hat{\lambda} + C)$, it would be rounded to $lg\hat{\lambda} - C$ or $lg\hat{\lambda} + C$. When level 1 controller is invoked, a new $\hat{\lambda}$ will be predicted and the allowed range of λ will be adjusted accordingly. The learning rate factor (i.e. *Visit*(*s*,*a*) in Equation (2.4) will be reset every time when level 1 or level 2 controller is triggered because a new tradeoff factor is found.

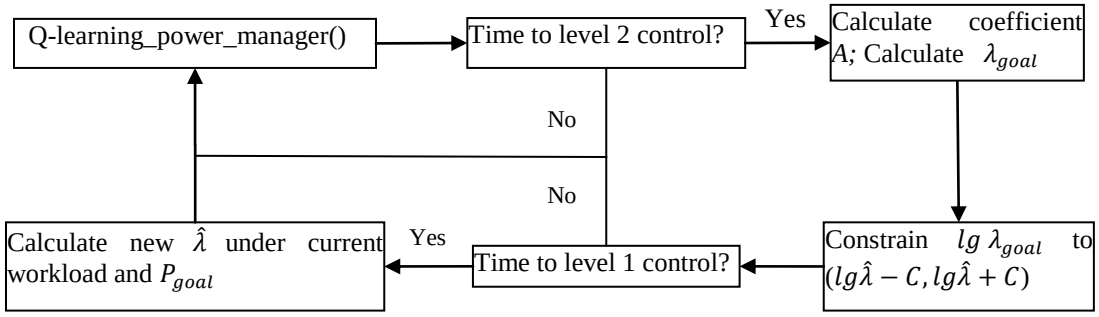


Figure 2-15. Block diagram of the power control flow of the Q-learning power manager.

2.2.5 Learning based CPU Power Management

As explained in Section 2.2.1, a microprocessor working in batch mode has very different characteristic compared to the peripheral devices and hence requires different Q-learning model. In this section, we discuss how to apply the Q-learning algorithm to the power management of such microprocessor. Our goal is to minimize the CPU energy dissipation for the batch processing under the given execution time constraint. In order to demonstrate the Q-learning's capability of handling multidimensional constraints, we add the average die temperature as the second constraint. Please note that user may select any other measurement as the second constraint, because the construction of the Q-learning algorithm does not rely on any temperature mode or thermal analysis. The relationship of the CPU frequency and its energy, performance (i.e. inverse of the execution time), and temperature is qualitatively shown in Figure 2-16. The energy first decrease as the frequency reduces. If we further reduce the frequency, the energy will increase as the leakage power becomes dominant and power reduction is slower than the

runtime increase. The CPU frequency that gives the minimum energy dissipation is denoted as f_E^*. The performance and temperature increases as the CPU frequency rises. However, since the clock speed for the memory subsystem does not change, the performance gain due to fast CPU will gradually slow down. Therefore, the performance is a concave function of CPU frequency.

We use *f* and *v* to denote the scaling ratios of the CPU voltage and frequency. They are calculated as $f = F/F_{max}$ and $v = V/V_{max}$, where V_{max} (F_{max}) and V (F) represent the maximum voltage (frequency) and the scaled voltage (frequency) of the processor respectively. We assume that *f* and *v* have one to one correspondence, i.e. for each CPU frequency there is a matching supply voltage level. We use μ to represent the percentage of time the application is processed on CPU and cache. It is referred as *CPU intensiveness*. It is calculated as the following [16]:

$$\mu = 1 - \frac{cycles\_l1i\_stalled + cycles\_l1d\_stalled}{total\ number\ of\ cycles} \tag{2.9}$$

where $cycles\_l1i\_stalled$ and $cycles\_l1d\_stalled$ are the number of cycles during which the CPU is stalled for instruction and data fetches. They can be recorded periodically in many commercial processors. Though there are other architectural events related to μ, such as the cycle of stalls due to TLB miss, branch prediction miss and etc., they are less dominant than the cache miss event and usually cannot be monitored at the same time with the cache miss events. Hence, they will be ignored in this formula.

The CPU intensiveness varies from application to applications or even inside the same application. Its value affects how the energy and execution time change with voltage and frequency scaling. When the value of μ reduces, the CPU spends more time waiting for the memory reads/writes. Less performance gain can be achieved by increasing the CPU frequency.

On the other hand, because most of the time is spent on memory subsystem, reducing the clock frequency will cause less increase of the execution time. Therefore, the energy optimal frequency $f_E^*(\mu)$ is lower. In this work, we assume that the CPU has been characterized so that for specific μ, the minimum energy frequency $f_E^*(\mu)$ is known.

At the end of each time slot, three cost variables are updated, which include energy cost (C_E), performance cost (C_P) and temperature cost (C_T). While the die temperature can be read from on-chip temperature, the other two cannot be obtained directly. This is because both of them depend on the execution time, which is unknown during the runtime as we assume no prior knowledge of the workload. To overcome this limitation, in this work we define the energy cost at cycle *t* as the normalized deviation from the energy minimum frequency of current workload (i.e. $f_E{}^*(\mu_t)$) to the energy cost, i.e. $C_{E,t} = |f_t - f_E{}^*(\mu_t)|/(f_{\max} - f_{\min})$, where f_t and μ_t are frequency and CPU intensiveness during cycle *t*, f_{max} and f_{min} are the maximum and minimum frequency of the CPU. The similar energy cost definition is also used in [16]. We also define the performance as the normalized deviation from the maximum clock frequency, i.e. $C_{P,t} = (f_{max} - f_t)/(f_{max} - f_{min}) * \mu_t$. Finally, the temperature cost of cycle *t* is defined as the temperature increase from cycle *t*-1, i.e. $C_T = (T_t - T_{t-1})/T\_range_{intervals}$ where $T\_range_{intervals}$ is the maximum temperature change in two adjacent time intervals. It is about 2℃ in our experiment system.

We partition the environment state so that the cost functions remain relatively constant during the same state. Based on this criterion, the state is classified based on four parameters: (*f*, *T*, *IPS*, μ). They represent the clock frequency, the temperature, the instructions per second (*IPS*) and the workload CPU intensiveness respectively. Let *N* be the total number of clock frequencies

supported by the processor, we use f_i to denote the ith clock frequency, with f_0 representing the minimum frequency. We discretize the possible range of temperature into M levels, with T_0 representing the ambient temperature and $T_{M\text{-}1}$ representing the maximum temperature threshold.

In section 2.2.4, we solve the performance constrained power optimization problem by dynamically adjusting the weight coefficient of the Lagrange cost function to find minimum power policy that exactly meets the performance constraint. The rationale of this approach is that power is a decreasing function of response time for the peripheral device. Such relation no longer exists between energy and performance for a batch mode CPU as shown in Figure 2-16. Multi-dimensional constraints make things even more complicated. For example, as shown in Figure 2-16, assume the minimum frequency that satisfies the performance constraint is f_P and the maximum frequency that satisfies the temperature constraint is f_T. Our goal is to constrain the performance and temperature, while at the same time minimizing the energy. As we can see, it is not possible to find a frequency that satisfies both performance and temperature constraints exactly. Hence we have to modify the cost function of the Q-learning algorithm to decouple these two constraints.

We denote the performance and temperature constraints as con_P and con_T. We also use Δ_P, and Δ_T to represent the difference between the constraint and the actual average penalty during a history window for performance and temperature respectively. The value of Δ will be positive if the system outperforms the user constraint during the history window, otherwise it will be negative. Because we are interested in constraining only the average performance and average temperature, we consider the system to be performance and temperature *bounded* when $C_P \leq con_P + \Delta_P$ and $C_T \leq con_T + \Delta_T$, otherwise, the system is *unbounded*. In this way, if the system

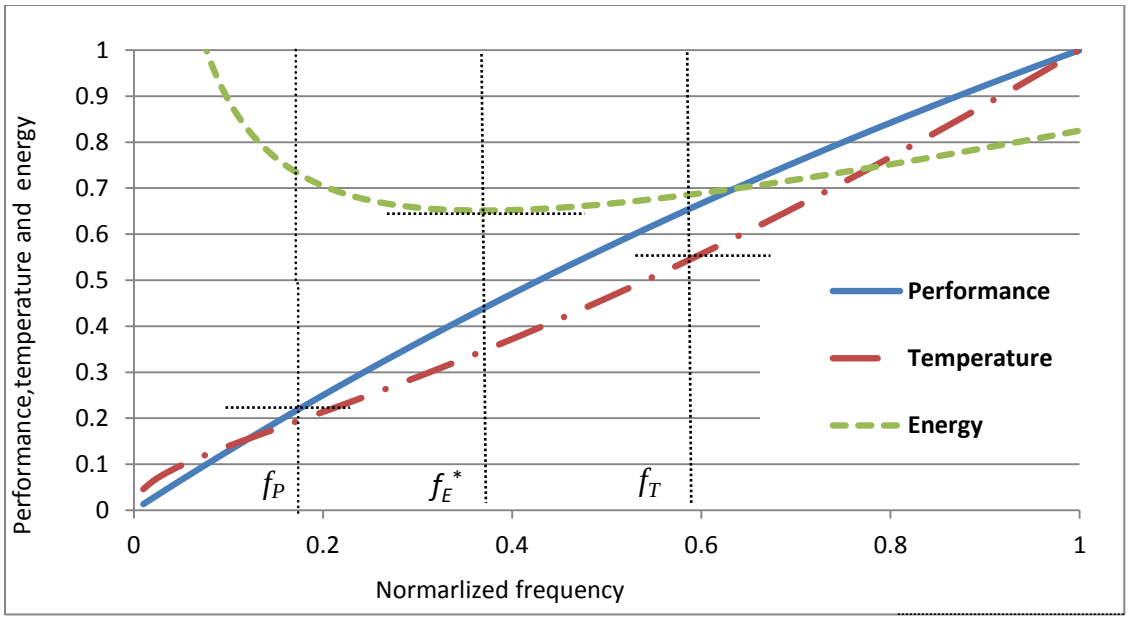


Figure 2-16. Qualitative illustration of the relation between CPU temperature, performance, energy and clock frequency.

has been outperforming the user constraint during the past, it will be considered performance (or temperature) bounded even if the cost of the current cycle is a little higher than the constraint.

The modified cost function considers 3 scenarios:

$$C = \begin{cases} C_E & if\ C_P \leq con_P + \Delta_T\ \&\ C_T \leq con_T + \Delta_T \\ C_E + \alpha \cdot C_P & if\ C_P > con_P + \Delta_P\ and\ C_T \leq con_T + \Delta_T \\ C_E + \alpha \cdot C_T & if\ C_P \leq con_P + \Delta_P\ and\ C_T > con_T + \Delta_T \end{cases}$$

In the above equation, α is a large positive number. Based on the modified cost function, when the system is bounded in both performance and temperature, the Q-learning algorithm will search for policies that minimize the energy cost. As soon as the system becomes unbounded in either performance or temperature, the cost function will be modified and the Q-learning algorithm will put more emphasis on improving the performance or temperature that has violated

the constraint. It can be proved that as long as the performance and temperature constraints are feasible, they will not be violated at the same time.

We need to point out that the proposed approach manages the voltage and frequency of a single CPU. For a multi-core system, this approach is viable if cores work independently to each other. It can be extended to manage multiple cores with interactions simultaneously by augmenting the state space of the Q-learning model to consider the joint state of different cores.

2.2.6 Experimental results and analysis

(1) Experimental Results for Peripheral Device Power Management

(a) Experimental Setup

In this section, we will present the simulation results of learning based power management for peripheral devices. The target SP in the experiment is a hard disk drive (HDD). Table 2-7 summaries the power and performance of the hard disk drive. These parameters are obtained from real hard disk datasheet [49]. The T_{be} value is round up to the nearest integer for the simplicity of calculation. In the table, the P_{tran} and T_{tran} are power and performance overhead of sleep to active transition. The active to sleep transition is not mentioned in the datasheet and hence will be ignored in the model.

Table 2-7. Characteristics of Service Provider

$P_{active}(W)$	$P_{sleep}(W)$	$P_{tran}(W)$	$T_{tran}(s)$	$T_{be}(s)$	Speed (MB/s)
1.1	0.1	1.42	3	4	16.6

In order to evaluate the performance of our learning based power management policy, we developed a fast performance evaluation framework of the HDD using OMNeT++ [OMNeT+]. OMNeT++ is a discrete event simulation environment written in C++.

The performance of the Q-learning based power management is compared with the expert based learning algorithm proposed in [16]. Table 2-8 lists five fixed timeout policies, an adaptive timeout policy, and an exponential predictive policy. These 7 heuristic policies form the set of experts for the expert-based learning algorithm. Hence, the expert-based learning algorithm overcomes the limitation of any of these single heuristics by dynamically selecting one of them to adapt with the changing workload. A control knob factor α is provided for power performance tradeoff [16].

(b) Results for Synthetic Workload

In this experiment, we use two synthetic workload to intuitively illustrate how the Q-learning based power manager is able to adapt to the workload change.

In Figure 2-17, the blue dots represent the state of SR. It is categorized into 2 states, with 0 represents zero incoming rate and 1 represents non-zero incoming rate. We assume that when there are incoming request, they come in at a constant rate. The red solid line represents the state of SP, with 0 representing sleep mode and 1 representing active mode. The SP is controlled by a Q-learning based power manager. The synthetic workload trace we created shows a changing pattern during the time. At the beginning of the experiment, the SR's idle time is always 2 seconds, which is smaller than the system T_{be}, hence the system should not go to sleep during the idle interval. While later in the experiment, the SR's idle time is increased to 8 seconds which is longer than T_{be} .From the behavior of the SP we can see that the power manager

Table 2-8. Characteristics of Different Reference Policies

Policy	Characteristics
Fixed Timeout(1~5)	Timeout = 1* T_{be} ~ 5* T_{be}
Adaptive Timeout	Initial timeout = 3 * T_{be} Adjustment = +/-1 * T_{be}
Exponential Predictive	$I_{n+1} = \beta i_n + (1-\beta) I_n \, , \beta = 0.5$
Expert-based Learning	Uses the above seven policies as experts.

undergoes 4 phases:

Phase 1: The power manager learns the pattern of the workload.

Phase 2: The pattern has been learnt and the power manager decides to keep the SP active during the short idle period.

Phase 3: After the workload changed, the power manager start learning again..

Phase 4: The new pattern has been learnt and SP will go to sleep during long idle period.

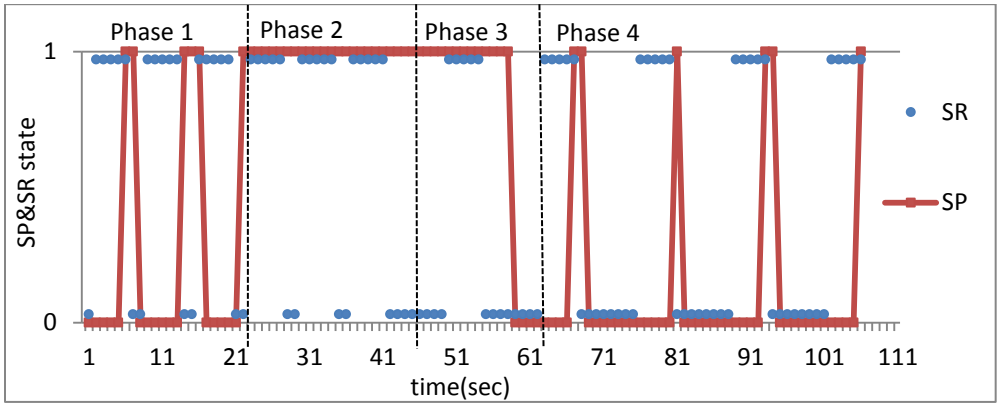


Figure 2-17. Response of Q-learning power manager to synthetic trace 1.

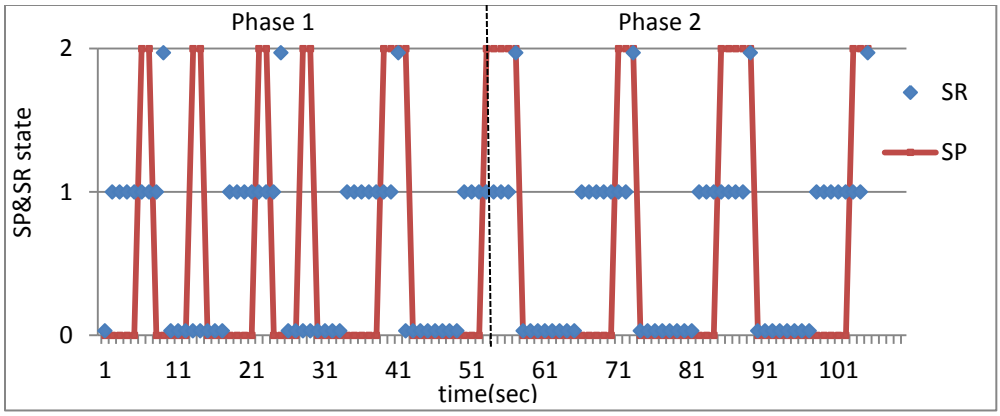


Figure 2-18. Response of Q-learning power manager to synthetic trace 2.

Note in our system, the SP service rate is always much higher than the request incoming rate. The SP only takes a short time to process the accumulated requests after activated.

In the second example shown in Figure 2-18, the SR has 2 different incoming rates, and hence overall 3 states. States 0, 1 and 2 represent idle, low incoming rate and high incoming rate respectively. The workload has a clear pattern which always starts with a long idle period followed by a long period of low incoming rate and then a short period of high incoming rate. After that the pattern repeats itself. As we can see in the Figure 2-18, during the learning phase (i.e. phase 1) the power manager tried different control policies by turning the SP on and off at different time. Eventually, it found that the best policy for this workload is to turn on the device in the middle of the low rate incoming period and turn it off immediately after the high incoming rate period is over. Note that none of the seven heuristic policies in Table 2-8 classifies SR into different states; hence they are not able to detect the workload pattern in this example.

(c) Q-learning Power Management for Real Workload

In this experiment, we evaluate the performance of the Q-learning based power manager using two different types of workloads:

1) Workloads extracted from HP cello99 traces [40][34]. Cello99 trace records file system read write activities of HP data center. All the requests with the same PID within one microsecond are merged into one large request. One interesting observation we have found is that hourly request incoming rate has strong correlation to the time of a day. Figure 2-19 shows the hourly request incoming rate for 3 days. As we can see, the peak and bottom occurs at approximately the same time. This observation agrees with reference [40] and it indicates that similar applications are running at the same period of time on different days. Such property can be used to gather training data to construct the neural network based power (performance) prediction model presented in subsection 2.2.4. We extracted 3 workloads (i.e. HP-1,HP-2 and HP-3) at different time of the day.

2) Workloads collected from the desktop computer [43]. Using Windows Performance Monitor, we collected hard disk read/write request sequences from two different desktop workstations whose hard disk usage level differs significantly. We stopped collection when the generated file size reaches 5MB, which is equivalent to 70,000 read/write requests in the sequence. The first trace was collected in the afternoon when a set of applications were running simultaneously with high disk I/O activities, resulting in a short collection time (i.e., about 1000 seconds). The other trace was collected at night when only two applications were running and it takes more than 20000 seconds to complete the collection.

Table 2-9 summaries the characteristics of the HP and desktop workload traces that we use.

Both Q-learning algorithm and expert-based algorithm can achieve different power-performance tradeoff by controlling the tradeoff factors λ and α respectively. By varying these

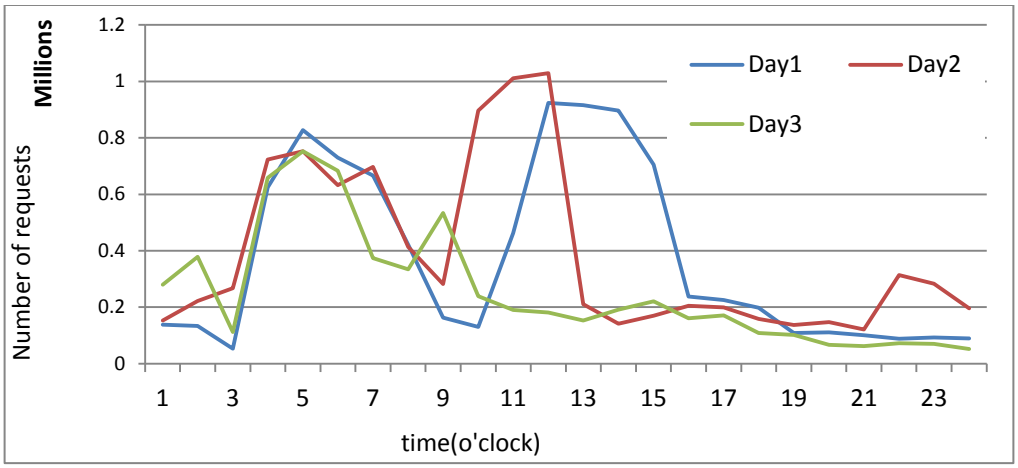


Figure 2-19. Three consecutive days' requests from HP hard disk traces.

Table 2-9. Characteristics of Workload Traces.

Trace name	Duration(sec)	No. of total requests after merging	No. of idle time≥ T_{be}(4 sec)
HP-1	14322	14994	1127
HP-2	14375	44468	332
HP-3	14387	151404	742
Desktop-1	21634	18036	1166
Desktop-2	1026	27782	43

tradeoff factors, we generate multiple power management policies with different power/latency tradeoffs. Figure 2-20 shows these power latency tradeoff points for these two learning based power management algorithms tested using 5 real workload traces. The results for power management using the traditional Q-learning algorithm without VSS enhancement are also shown in those figures. To better show the trend of power/latency tradeoff, we use solid line to sequence the power/latency points following the decreasing order of the value of corresponding

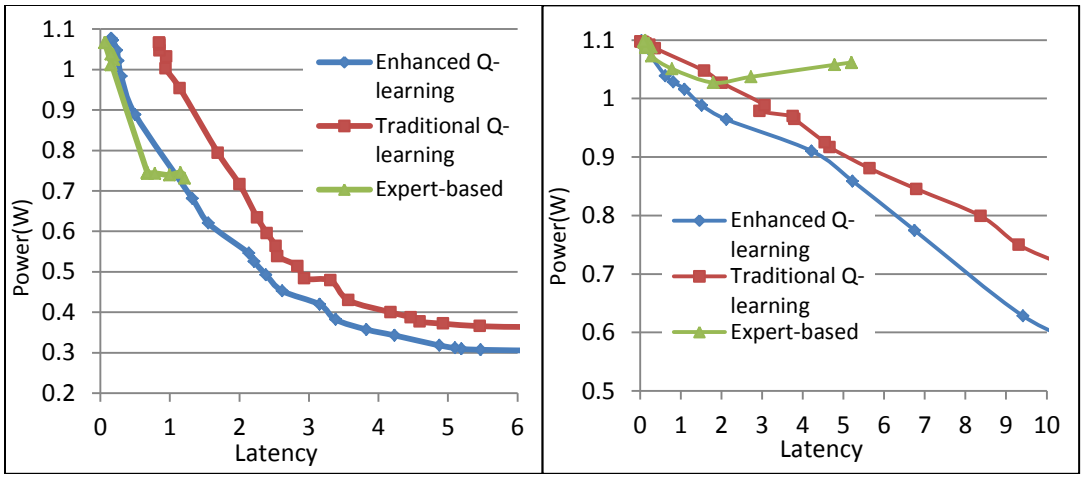
tradeoff factors. Note in this set of experiment, learning rate $\varepsilon_{(o,a)}$ in Equation (2.4) is reset to 1 periodically to adapt to the change of the workload patterns.

From Figure 2-20, four observations can be obtained:

1) Expert-based algorithm generally outperforms Q-learning algorithm for low-latency high performance scenario. This is because all the experts used in the expert-based algorithm are designed for high performance and they will turn on the device as soon as a request comes in. In contrast to the expert based algorithm, the Q-learning algorithm allows the device to buffer the requests.

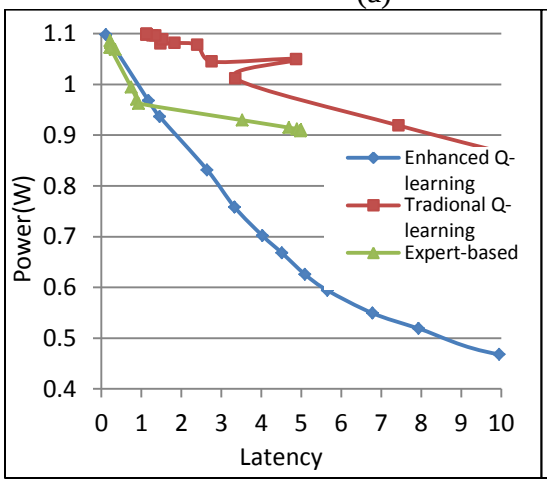
2) The Q-learning outperforms the expert based policy when the performance is relatively less important than the power consumption and it provides wider range of power-performance tradeoff. The tradeoff curve for Q-learning based power management is also much smoother than the curve for expert based power management. For Q-learning based management, power is a decreasing function of performance in all cases except the last one (i.e. desktop workload 2). While for expert-based power management, such monotonic relation is not obvious for several test cases (i.e. HP-1, HP-2, Desktop-1 and Desktop-2).

3) For workload Desktop-2, the red curve moves forward and backward. This means the latency (and the power) of the device does not have a monotonic relation with the tradeoff factor. This is probably because the workload is very intensive and changes so rapidly, the traditional Q-learning algorithm does not have enough time to find the best policy before the workload changes. Our enhanced Q-learning algorithm exhibit better monotonic relation between power/latency and the tradeoff factor, due to its fast convergence speed.

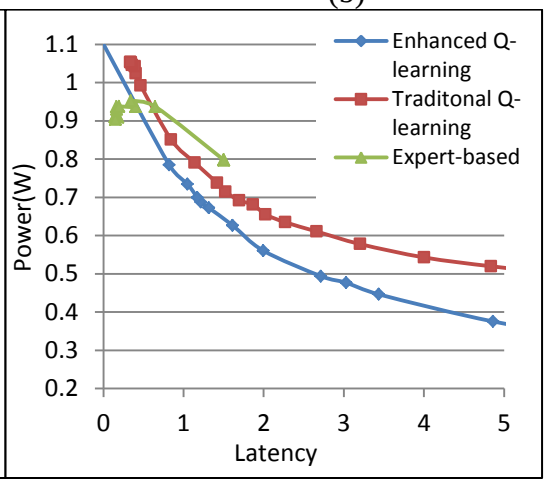


(a)

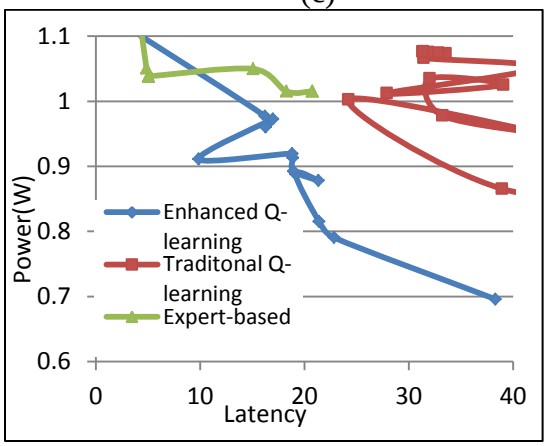
(b)



(c)



(d)



(e)

Figure 2-20. Power/Latency tradeoff curves for workload. (a)HP-1; (b)HP-2; (c)HP-3; (d)Desktop-1; (e)Desktop-2

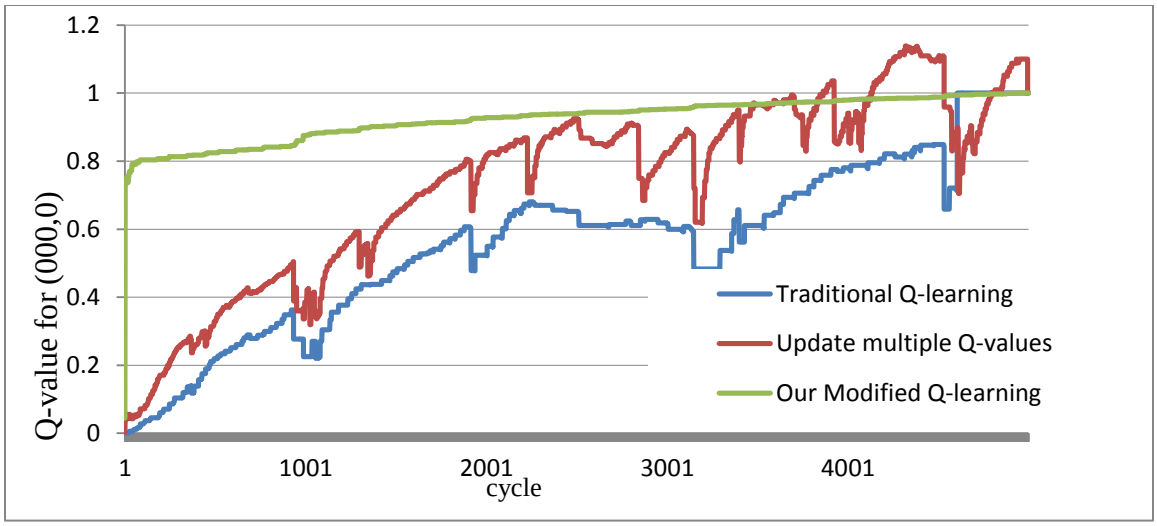


Figure 2-21. Q-value for observation-action pair(000,0).

4) Our proposed VSS technique in Section 2.2.4 significantly improves the power latency tradeoff curves due to the faster speed of Q-learning convergence. Figure 2-21 compares their convergence speed.

As we mentioned earlier, two enhancement techniques are used to speed up the convergence. First, the learning rate ε is modified as an adaptive factor associated with the observation-action pair. Second, we update multiple Q-values instead of only one Q-value in each learning step using the VSS technique. Figure 2-21 shows the change of the Q-value of state-action pair (000, 0) for 3 different Q-learning algorithms: the traditional Q-learning (without variable learning rate and multiple Q-value update), the Q-learning algorithm with multiple Q-value update (but no variable learning rate), and our enhanced Q-learning algorithm. The state action pair (000, 0) represents the scenario when there are no incoming requests, no waiting requests in queue, and HDD is in *sleep* mode, and the power management command is to continue sleeping. As we can see, comparing to the other two learning algorithms, the changes of

Q-value for the proposed modified Q-learning is smoother. Moreover, it converges much faster to the stable state. The similar trend can be found with all other state action pairs.

In terms of complexity, as we mentioned before, the enhanced Q-learning is O($|SP| \times |SQ| \times |A|$), the expert-based algorithm is O(n) where n is the number of simple experts used, and the traditional Q-learning is O(1).

(d) Adaptivity of the Learning based Power Management to Different Hardware

In the third experiment, we consider power management of systems with special hardware that has a large penalty to go to sleep mode. The purpose of this experiment is to test the robustness of the Q-learning algorithm in working with different types of service provider. Different devices will have different power and transition characteristics. For example, the server's hard disk or the CD-ROM will always have longer T_{be} than personal computer's hard disk.

In this experiment, we increase the T_{be} of the HDD from 4 seconds to 8 seconds by increasing the P_{tran} and run the simulation again. Figure 2-22 shows the results for 3 HP workload traces and 2 desktop traces respectively. As we can see, the policies found by the expert-based algorithm do not give proper tradeoff between power and performance. When the latency increases, the power consumption of the system increases too. The policies found by the Q-learning based power management are still capable of trading performance for power reduction. This is because the expert-based algorithm is restricted by the selected experts, which are a set of time-out policies whose time out values are multiples of T_{be}. When the value of T_{be} gets larger, the flexibility of these time-out policies reduces. Compared to the idle intervals in the request pattern, these timeout values are either too small or too large. When the performance

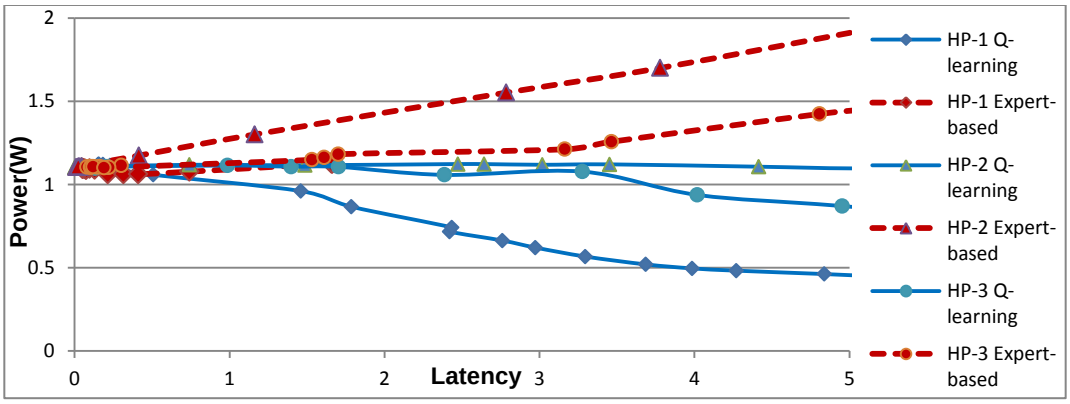
requirement reduces, the power manager will put the device into sleep mode more frequently. However, without proper timeout threshold, there will be lots of mistakes and frequent on-off switches. Hence, not only latency, the power will also increase. This problem can be solved if more timeout polices with finer resolution of timeout threshold are added as experts. However, this means higher complexity. This experiment also shows that with different workload patterns and different hardware devices, the performance of expert-based algorithm depends highly on the right selection of different experts.

In contrast to the expert based policy, the Q-learning power management algorithm not only learns and adapts to different workloads, but also adapt to different hardware, both of which are the requirements of a good power management algorithm [37].

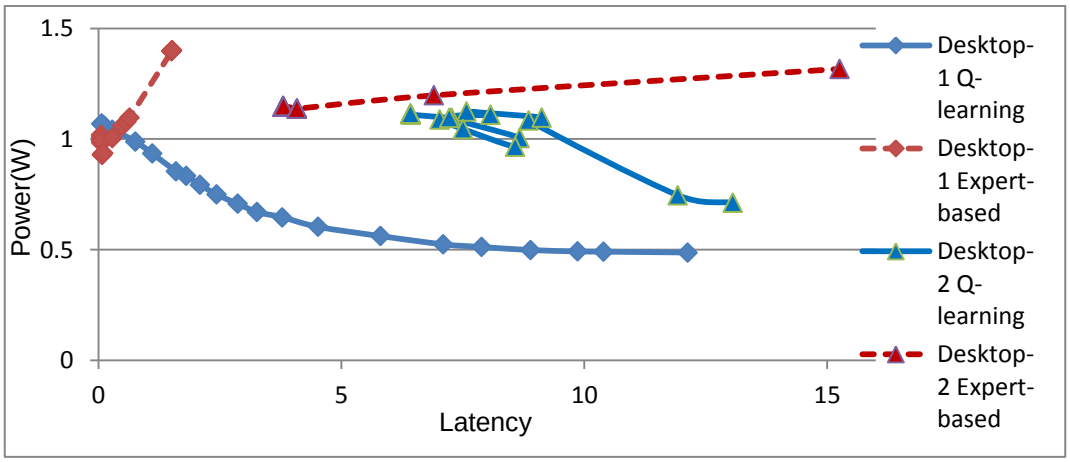
(f) Tracking the Power (Latency) Constraint

In this section, we will demonstrate the effectiveness of our proposed power (latency) constraint tracking algorithm. It is measured by the difference between the actual average power consumption (or latency) and the user specified constraint. The closer the actual value and the constraint are, the more effective the constraint tracking algorithm is.

In the first set of experiments, we consider the problem of performance optimization with power constraint and show the effectiveness of level 1 and level 2 controllers. First we compare our Q-learning algorithm with the same algorithm that has level 2 constraint tracking controller disabled (i.e. the value of λ is set exactly equal to $\hat{\lambda}$ predicted by the neural network). Please note that we divide each workload trace into 2 segments, a training sequence and a testing sequence. The neural network is trained using the training sequence, and then applied to the testing sequence to collect the comparison results.



(a)



(b)

Figure 2-22. Power/Latency tradeoff curves for (a) HP workloads (b) desktop workloads when T_{be}= 8 seconds.

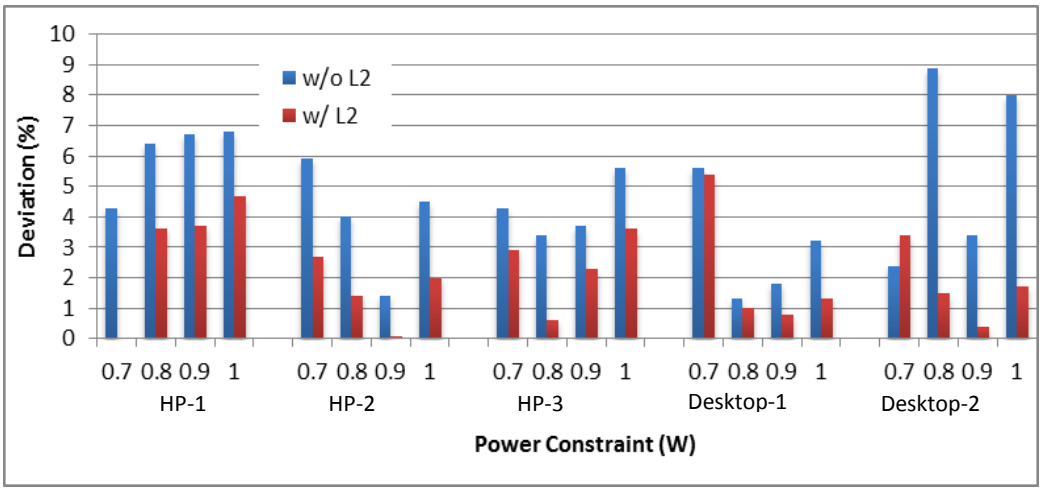


Figure 2-23. Relative average power deviation from user constraints.

As we mentioned in Section 2.2.4, the function of level 2 controller is to keep the average power consumption close to the power constraint using feedback control. In this experiment, we focus on how much the average power consumption deviates from the power constraint. We vary the power constraint from 0.7 to 1. The average power over the entire simulation time is measured and the relative deviation of the average power is calculated which is the relative difference between actual average power and the power constraint. The comparison results are shown in

Figure 2-23. As we can see, adding level-2 controllers can reduce the constraint tracking error from 4.58% to 2.15%, which stands for approximately 50% improvement. The capability of being able to stay close to the power constraint is very useful for systems powered by battery [19] or energy harvesting units [26], where the budget of available power is predefined.

While using level-2 controller helps to keep the average power consumption close to the constraint, using level-1 controller helps to reduce the variation of the instantaneous power consumption. In next experiment, we compare out Q-learning algorithm with the same algorithm

that has level 1 controller disabled (i.e. the value of λ is controlled only by the feedback controller.) The percentage mean square errors (MSE) between the instantaneous power consumption and the power constraint is calculated. Here we use the average power consumption over 200 cycles to represent the instantaneous power. The comparison results are given in Figure 2-24. As we can see, including level-1controller reduces the variation of the power by 15.6% in average.

The previous experiment shows that including a level 2 controller could reduce the average power deviation from 4.58% to 2.15%. Although this represents 50% relative improvement, the absolute improvement is only 2%, which is quite small. This is because our level 1 predictor is already accurate in predicting the average power for the given λ. Therefore, using level 1 control we can find the tradeoff factor close to the right value. However, level 2 controller is especially important when we couldn't construct a good model to predict λ in level 1. In the next set of experiments, we consider the problem of power minimization with performance constraint.

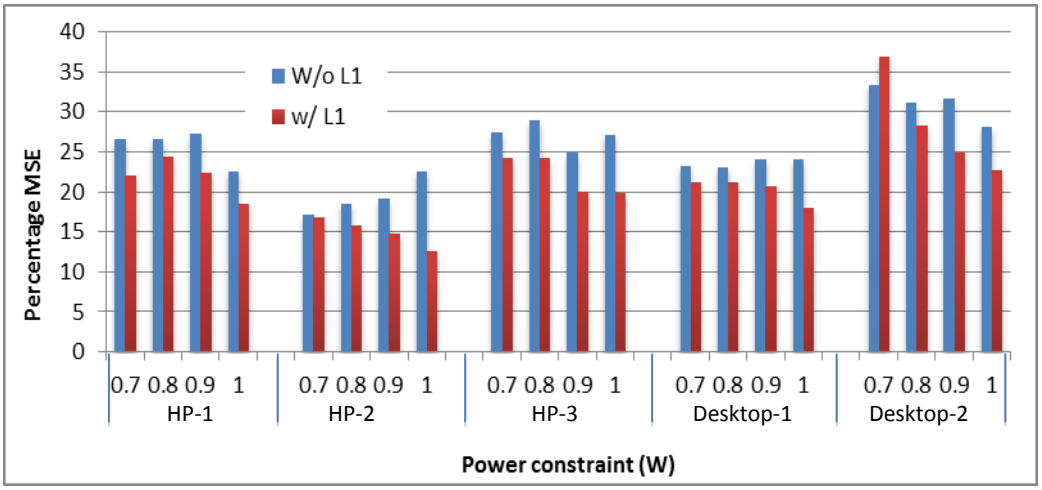


Figure 2-24. Percentage MSE of instant power versus user constraints.

Because the rate of incoming request to a hard disk drive has very large variation, it is difficult to train a neural network model that could accurately predict the average latency. Therefore, the level 1 control can only provide a very loose bound and the search for the appropriate tradeoff factor largely depends on the level 2 controller. Instead of confining λ around $\hat{\lambda}$ which is predicted by the neural network, we constrain it within the range $(C\ \lambda_{curr},\ \lambda_{curr}/C)$ where λ_{curr} is value of λ that have recently been used. By doing this, we

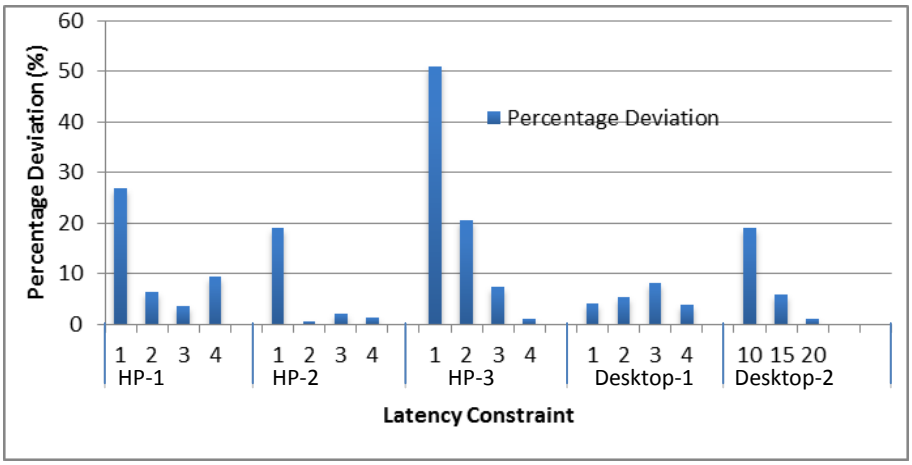


Figure 2-25. Relative average latency deviation for latency constrained power management.

prevent λ from changing too abruptly and stabilize the latency change through the time.

Figure 2-25 shows the percentage deviation of the average latency compared to the latency constraint. We vary the latency constraint from 1 to 4 for the three HP traces as well as Desktop-1. Note that a different set of latency constraints is used for trace Desktop-2. This is because it is extremely intensive and no power management policy except the "always on" policy can meet the same latency constraints as we use for the other 4 traces. The experimental results show that in average the Q-learning based power manager can maintain the system performance within about 10% of the given constraint. Furthermore, it is much easier to track a

loose performance constraint than a tight performance constraint. Note that the data shown in Figure 2-25 are relative deviation. When the latency constraint is tight, although the relative deviation is large, its absolute value is still small.

(2) Q-learning Based Microprocessor Power Management

In the second set of experiments, we evaluated our Q-learning algorithm for microprocessor power management. We implemented the Q-learning based DVFS controller on a Dell Precision T3400 workstation with Intel Core 2 Duo E8400 Processor [22]. The processor supports 4 frequency levels: 2GHz, 2.33GHZ, 2.67GHz, 3GHz. The Linux kernel we use is version 2.6.29.

We used *coretemp* driver in the Linux kernel to read the temperature sensor of the processors. The default driver updates temperature readings once every second and we modified it to be every 10ms to achieve our required granularity. We used *cpufreq* driver in Linux based on Enhanced SpeedStep technology [17] of Intel Core 2 processor to adjust the processor's frequency. We used Perform2 tool [36] to monitor performance events of the processors. We ran the experiments on one core and fixed the frequency of the other core to be the minimum. The Q-learning controller was triggered every 20ms. Empirically, this interval will not exert too much overhead to the processor while still capable of tracking the change of workload. The overhead of frequency change is only about 20us. We use the option "–print-interval=20" provided by pfmon to control its sampling period to also be 20ms.

We use benchmarks from MiBench [32] and MediaBench [31] to form the workload of the evaluation system. Our goal is to generate workloads with changing CPU intensiveness. The benchmarks we selected are: bitcount_small, basicmath_small, qsort_large, tiff2rgba, mpeg4dec,

and jpeg200dec together with a simple custom application with only CPU busy loops. Their CPU intensiveness varies from 11% to almost 100% with an average of 82% according to our measurement. Each benchmark running a little more than 0.2s under minimum frequency is a running unit. We serialized 100 running units of different benchmarks in 4 different random orders to construct 4 different "workloads". Every experiment result reported here is the average of the 4 "workloads". We need to point out that MiBench is a benchmark mainly designed for embedded systems, while our experiment is done on a Core 2 Duo workstation. However, it is our goal is to test how well the power management controller adapts to different workload. Our objective in selecting the benchmark is to create a variety of workload with different CPU intensiveness. We found that the two programs from MediaBench (i.e. mpeg4dec and jpeg200dec) has relatively lower CPU intensiveness (in the range of 70% to 80%) while the program in MiBench has much higher CPU intensiveness which is above 95%. So the combination of MiBench and MediaBench gives us such variety.

Since we have 4 frequency levels (i.e. f_0~f_3) on our platform, we partition the workload CPU intensiveness μ into 4 states, so that f_i is corresponding to the ideal frequency f_E^* when $\mu = \mu_i$, $0 \le i \le 3$. Such partition enables us to measure the energy penalty using the deviations from the ideal frequency. The temperature and *IPS* are also empirically partitioned into 4 states.

Table 2-10. Constraining performance and temperature.

Performance / Temperature	0.34 (constraint)	0.67 (constraint)	1 (constraint)
0.34(constraint)	0.33 / 0.46	0.58 / 0.34	0.78 / 0.30
0.67(constraint)	0.24 / 0.58	0.38 / 0.51	0.62 / 0.44
1 (constraint)	0.23 / 0.61	0.37 / 0.55	0.60 / 0.43

As discussed in Section 2.2.5, the performance is measured by the deviation from the maximum frequency; the energy is measured by the deviation from the energy optimal frequency. The temperature is the average normalized temperature of the CPU observed by the temperature sensor.

We run the Q-learning based power management for minimum energy under different performance and temperature constraints. The results are shown in Table 2-10. Each column in the table represents a performance constraint and each row represents a temperature constraint. Because our platform only supports 4 frequency levels and the frequency increases linearly at an equal step from level 0 to level 3, the corresponding normalized temperature and performance for those frequency levels should also change roughly at an equal step. To better show our results, we set the constraints to be 0.34, 0.67 and 1 as shown in the tables. Each entry gives the actual performance and temperature of the system under the power management. For example, the cell in row 1 and column 1 of Table 2-10 shows that the actual normalized temperature and performance of the system is 0.46 and 0.33 respectively when the performance and temperature constraint are both set to 0.34. The entries are shaded differently according to the energy dissipation of the system. The lighter the cell is, the lower energy dissipation we achieve. As we can see, the upper left cell has the darkest color because it corresponds to the most stringent user constraints and hence leaves almost no room for the optimization of the 3rd metrics. On the contrary, the bottom right cell has the lightest color because it corresponds to the most relaxed constraints.

We can see that sometimes the Q-learning controller cannot find a policy that satisfies both user constraints. For example, the entry (0.34, 0.34) in Table 2-10 has constraint violation. Sometime, the controller finds policies that exactly satisfies one of the constraints and

outperforms the other (e.g. entry (0.34, 0.67) in Table 2-10). For the rest of times, the controller finds policies that outperform both user constraints. This clearly shows that the relation among T, P and E are not monotonic. We cannot optimize one metric by setting the other (one or two) metrics exactly to the given user constraints. For example, consider cell (0.67, 0.67) in Table 2-10. The user set a loose performance and temperature constraint (con_P=con_T=0.67) in order to optimize the energy. However the result shows that the policy that minimizes the energy actually does not have to work so slowly and will not generate so much heat. Clearly in this test case, we have $f_P \leq f_E^* \leq f_T$ for the average μ of the workloads, where f_E^* is the energy optimal frequency, f_P and f_T are the frequencies that exactly satisfy the performance and temperature constraints respectively. However, we need to point out that the data reported here is the average of 4 different workloads over 80 seconds simulation. Although in average the CPU intensiveness satisfies the condition $f_P \leq f_E^* \leq f_T$, the instantaneous value of μ for each individual workload may not always satisfy this condition. That is why the entry (0.67, 0.67) has a darker shade than the cell (1.0, 1.0), which indicates a higher energy. The later, due to the extremely loose performance and temperature constraints, can always reach the energy optimal point f_E^*.

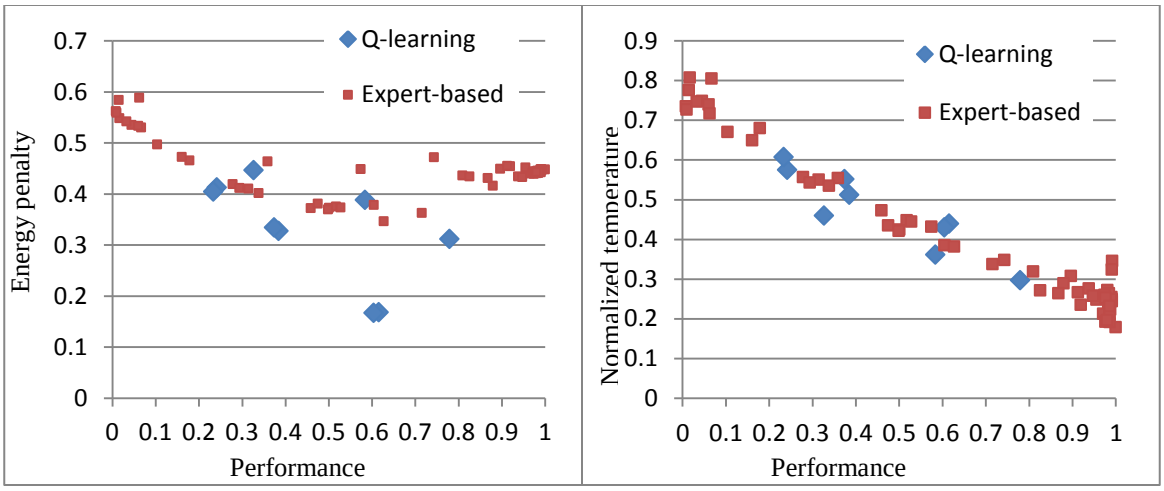
The experimental results also show that, generally without the prior knowledge of hardware and software, our Q-learning based controller can correctly learn the tradeoff space and give effective control policies. The only information we need to know related to the hardware is the mapping of different workload CPU intensiveness to the ideal working frequency f_E^* for the energy optimization purpose. This requirement can be removed if the processor's power consumption can be measured during the runtime.

In order to compare the performance of the proposed learning algorithm with the state-of-art approach, we modified the expert-based algorithm in [16] for energy management with the

consideration of performance and temperature. In our modified expert based approach, we choose different voltage and frequency configurations as experts. The cost function is weighted sum of energy cost, performance cost and temperature cost defined in 2.2.5, i.e. $C = \alpha C_E + \beta C_P + \gamma C_T$. The modified expert based approach is actually a reduced version of the expert based controller proposed in [14]. While their work considers energy and thermal management for multi-core system running interactive applications, our problem is a little different. If we remove some control knobs that are specific to multi-core system (e.g. task migration) and interactive applications (e.g. adaptive random and DPM) from their work, and also remove thermal gradient and thermal cycle from their cost function, then it will be reduced to the our modified expert based controller.

As we mentioned previously, because the energy and performance no longer have monotonic relation, and also because there are two constraints (i.e. performance and temperature), it is very difficult to find a set of weight factors that minimizes the energy while satisfying the given constraints. Therefore, we sweep the weight factors $\alpha, \beta,$ and γ to generate a set of power management policies that gives different energy/performance/temperature tradeoffs. Their corresponding energy, performance and temperature values are plotted in Figure 2-26, represented by the red dots. These policies do not try to meet any performance/temperature constraints. They provide the best effort to minimize the weighted sum of energy, performance and temperature costs.

Our modified expert based control is a reduced version of the controller proposed in [14], which is a comprehensive work on multi-core system energy and thermal management using expert based framework. The scope of their problem is a little different from ours. They



(a) (b)

Figure 2-26. Energy, temperature and performance results of Q-learning algorithm with constraints and expert-based algorithm without constrains: (a) energy versus performance; (b) temperature versus performance

consider multi-core system running interactive applications (i.e. web server), while we consider single core management for CPU running batch processing. In addition to energy, performance and temperature, they also consider thermal gradient and thermal cycles in the optimization. If we remove those control knobs that are specific to multi-core system (e.g. task migration) and interactive applications (e.g. adaptive random and DPM) from their work, and also remove thermal gradient and thermal cycle from their cost function, then it will be reduced to the same modified expert based controller implemented in our experiment.

From Table 2-10 we have already shown that our approach can meet the performance and temperature constraints. Figure 2-26 also shows that, for the same performance level, our approach results in the same or even less energy and similar temperature, compared to the expert

based approach, which does not guarantee the performance and temperature constraints. Therefore, the Q-learning based approach performs better for this problem.

2.2.7 Conclusions

In this work, we propose a general model solving the dynamic power management problem using Q-learning. The Q-learning power manager does not require any prior knowledge of the workload or the system model while it can learn the policy online with real-time incoming tasks and adjusts the policy accordingly. Convergence speed acceleration techniques are proposed that make the Q-learning algorithm more efficient in non-Markovian environment. A 2-level power or performance control model is proposed to accurately keep the system at the given power (or performance) constraint, to achieve maximum performance (or minimum power consumption). Simulation results prove that our Q-learning power management algorithm is able to achieve better power performance tradeoff than the existing expert-based power management algorithm.

The Q-learning algorithm is also extended for the CPU power management by controlling its DVFS settings. The control algorithm is capable to achieve minimum energy while meeting the user constraints in performance and temperature.

2.3 Chapter Summary

In this chapter, we presented two works targeting system level power management in PC and server environment.

In the first work, we create a model that is used to dynamic quantify task performance degradation with the respect to a reference system, where the target process is executed stand

alone at the highest frequency. The propose model is used to provide performance feedback to guide DVFS control. The model is further improved to predict the performance of the target process under a new task mapping. The improved model is used to provide performance prediction to guide the task migration. Experimental results show that the proposed models effectively controls the system performance and keeps it close to the given constraint, hence leads to lower power consumption with minimum performance violation.

In the second work, we propose a general model solving the dynamic power management problem using Q-learning. The Q-learning power manager does not require any prior knowledge of the workload or the system model while it can learn the policy online with real-time incoming tasks and adjusts the policy accordingly. The Q-learning algorithm is firstly applied to the HDD power management by putting HDD into sleep and waking it up. Simulation results show that can achieve good power performance tradeoff. Then it is also extended for the CPU power management by controlling its DVFS settings. The control algorithm is capable to achieve minimum energy while meeting the user constraints in performance and temperature.

Chapter 3 Adaptive Battery Management for Mobile Device

3.1 Battery Aware Stochastic QoS Boosting in Mobile Computing Device

3.1.1 Introduction

Mobile computing has been weaved into everyday lives for communication, sensing, controlling and entertainment to a great extend. Many of the applications running on mobile devices can be configured into different levels of quality of service (QoS). For example, by increasing the synchronization frequency between the mobile device and the email server, an email application can receive incoming mails more promptly; by boosting the duty cycle of built-in sensors such as GPS, more accurate environment information can be gathered. The increase of QoS of an application running on mobile computing device always associates with extra energy dissipation [120]. While the progress of battery technology still cannot keep up with the increasing energy demand of the computing devices, traditional energy management of mobile device aims at minimizing energy dissipation. The common practice is to adopt a conservative QoS configuration to trade for longer battery life.

Compared to PC and server environment, the workload of mobile devices is more user dependent as different users use their mobile device in differently ways. For example, some users may have strong preference of certain applications and also, different users are likely to have

different interaction behaviors of the devices (e.g., some users would access the devices more frequently than others). On the one hand, the workload of a mobile device is context dependent. Such context includes time, location, battery status, quality of WiFi or cellular connection, and etc.

The energy in a mobile computing device is not simply expenditure but rather a dynamic flow that has generation and consumption. For example, most users recharge their smartphones every night. Recent study shows that about 72% of smartphone users will recharge their phone before the battery is low. The advances in energy harvesting techniques also make it possible for future generation smartphones to scavenge ambient RF or solar energy from environment when they are available. Given that the energy is replenishable, it is not necessary to overemphasize on energy saving. A more challenging research topic is how to exploit the potential of future battery recharge to deliver higher QoS. User behavior and preference plays an important role in determining the availability of external energy resources [126]. It is clear that the amount of incoming energy is a stochastic process that is strongly influenced by user behavior.

Smartphone usage and energy management have been considered in many previous researches. Authors of [122] conducted a thorough study on the diversity in smartphone usage and discovered immense diversity among users. These discoveries laid the basis of the user centric mobile device management. Authors of [119], [121], [123], [124] and [125] focus on context-aware mobile device power and performance management. Among these works, reference [119] is the most similar to ours as it assigns excessive energy to boost the performance. It considers the remaining battery energy at the time of battery charge as a random variable and predicts the lower bound of this value with required confidence. The predicted

remaining energy is considered as an extra and will then be redistributed to applications to increase their QoS proportionally.

Both [124] and [121] showed that predicting the battery level of mobile device is difficult. The former achieves only 40% average accuracy in battery level prediction during one-day period, while the later predicts the battery charging opportunity at merely 37% accuracy in average and 84% accuracy at best when the model parameters are optimized empirically. Clustering users according to their charging preference can reduce the average prediction error from 60% to about 25% [124]. However, the peak error is still as high as 38% and it always coincides with the initiation of battery charge, which means poor prediction when the battery is low. The low accuracy can be explained by discoveries in [119] and [126], which shows that in terms of battery use and recharge behavior, significant variation exists not only across different users, but also within individual user across his/her own pattern.

In this work, we investigate more sophisticated models based on neural network for battery prediction. Our results confirmed that deterministic prediction of battery level usually has low accuracy and hence not suitable to guide energy management. We then apply stochastic control to solve the energy management problem. Markov Decision Process (MDP) based and Q-learning based management policies are evaluated and compared.

The research in this work is enabled by the smartphone usage traces collected by the Livelab [118] project. The traces record various information including battery change, charging time, application usage, IO statistics, Cell Tower ID, Wifi availability and etc. for 34 users over 6~12 months. All of our analyses are carried out on this set of traces.

3.1.2 Battery Level Prediction Using Neural Networks

In this section, we study the potential of predictive energy management. The basic idea is to periodically predict the remaining energy level at the time of battery charge and distribute this extra energy to boost the QoS of applications. The key of predictive energy management is the accuracy of battery energy prediction.

We refer to remaining energy level at the beginning of battery charge as our *target variable*, because it is what we need to predict. Three neural networks with different input vectors are trained and tested to predict the next target variable. The first model makes the prediction simply based on current time and battery level and is referred as 2-input model. Compared to the first model, the second model has 24 more input variables that represent the battery level changes of past 24 hours. It is referred as 26-input model. Similarly, the last model, has 5 more input variables than the first model. These 5 variables give the battery level at the beginning of 5 recent battery charges, in other words it uses the 5 recent values of the target variable to predict the sixth one. The model is referred as 7-input model. All three models are trained using the first half of the collected data of different users and tested using the second half of the data.

Figure 3-1 gives the prediction error for 34 users. The prediction is made every hour and the reported error is the average error of the testing set. As we can see, the 2-input and 26-input model have higher accuracy than the 7-input model. The prediction error of the former ranges from 15% to 22%, which is a little better than the results reported in [121] and [124]. However, there is no significant improvement. The 7-input model has the worst accuracy. This indicates the lack of strong temporal correlation in the target variable.

Figure 3-2 plots the battery level at the beginning of 100 consecutive battery charges. The sequence has large temporal variations and is hard to be predicted based on its previous values. We also found that the prediction error reduces as the prediction time gets closer to the next battery charge. Figure 3-3 gives the relation between prediction error of the 2-input model and the time to next battery charge. As we can see, when predicted 2 hours ahead of next battery charge, the average absolute error is 15% of overall battery capacity and the relative error is 50%.

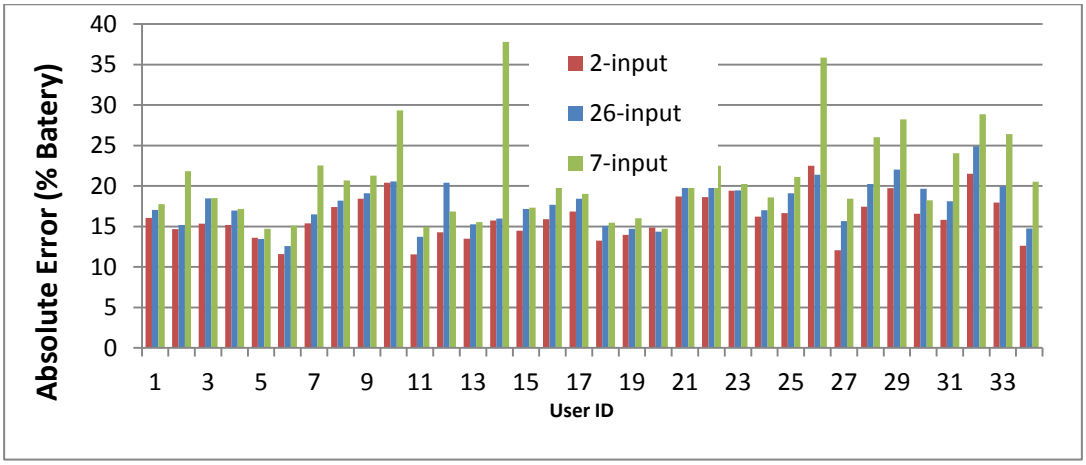


Figure 3-1 Average prediction error.

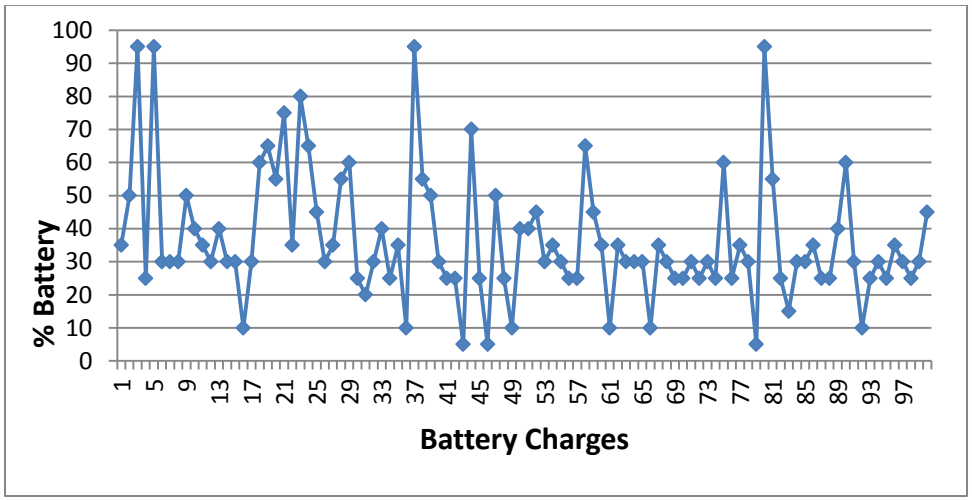


Figure 3-2 Battery level at the beginning of 100 battery charges

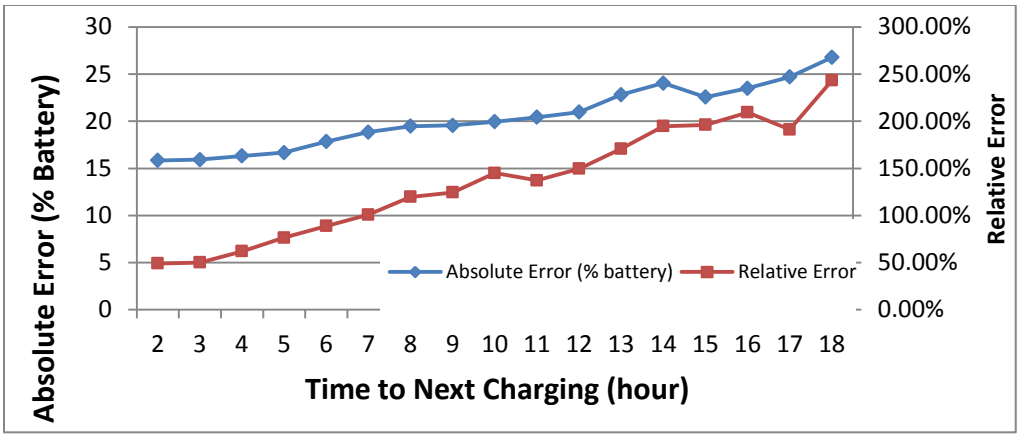


Figure 3-3 Prediction error vs. time to next battery charge

These numbers increase to 25% and 250% if the prediction is made 17 hours ago. Unfortunately, prediction at earlier time is more important as it provides higher reward.

3.1.3 Stochastic Control for Smartphone Energy Management

(1) State space

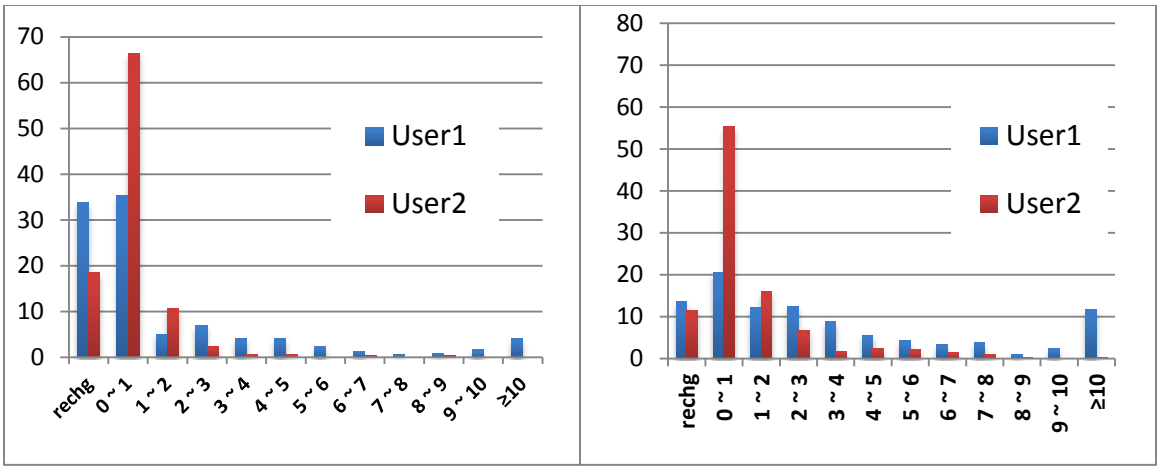
Using real user usage statistics, [122] confirmed that the usage behavior of different users could be described as the same mathematical model, though probably with different parameters. From [126], we can also see that though very different from user to user, the distribution of recharging level for each user has certain fixed patterns. These findings motivate us to explore stochastic control for the smartphone energy and QoS management. We consider smartphone battery change as a Markov Decision Process (MDP) [116][128] and consider QoS settings of the phone as control actions. The objective is to maximize the average QoS level while keeping the possibility of battery depletion under given threshold. Different users have different recharging behaviors. Some users (Type-A in [126]) charge the phone regularly regardless of the

battery level. Some other users (Type-B in [126]) charge the phone only when the battery is low. Such psychological effect is hard to model. In our work, we target at those users whose activities are relatively independent to the battery level.

The first step of model construction is to identify the state space of the MDP. The MDP tracks the change of battery; therefore the first feature that we included is the current battery level itself. Secondly, we found that time is highly correlated to the phone usage and user charging behavior. For the same user, the battery charging usually happens around the same time and the phone usage during the day is also quite stable. For example, Figure 3-4 gives the histogram of the battery change for two different users during 2~3am (Figure 3-4(a)) and 2-3pm (Figure 3-4(b)). The left most set of data in both figures gives the percentage of time that the smartphone is recharging. It is labeled as "rechg". The next 11 sets of data give the percentage of time that the smartphone consumes 0~1%, 1~2%,…., 9~10%, and 10~100% of battery energy during the recorded time period. As we can see, both users have higher possibility to charge their phone during 2~3am than 2~3pm. The chance to have nonzero battery energy dissipation is higher during 2~3pm than 2~3am. We also see that User1 has more intensive smartphone usage than User2 and also charges more often as a consequence.

Figure 3-4 further confirms the rationale of using stochastic model for smartphone energy management.

In addition to time and battery level, we are also interested to find out if other features, such as battery change rate, current and previous location, phone sleep time, should be included in the state space. We have found that they are all highly correlated to the time and battery level, thus do not provide much new information.



(a) Usage during 2~3am (b) Usage during

Figure 3-4 Battery change histogram

Correlations between future phone usage and previous phone usage are also calculated. We use hourly battery change rate $\Delta B(t)$ and smartphone sleep time $T_{sleep}(t)$ to represent phone usage during time slot *t*. The duration of each time slot is set to 1 hour. Figure 3-5 gives the correlations between $\Delta B(t)$ and $\Delta B(t-i)$, as well as the correlations between $\Delta B(t)$ and $T_{sleep}(t-i)$, $0 \leq i \leq 3$. As we can see, the battery change rate and phone sleep time in even one hour ago has low correlation with current battery change rate. And the correlation keeps on reducing when the distance in time increases. This indicates that the previous phone usage does not provide much help in predicting the future battery change either. Including it in the state space will not improve the model accuracy but add model complexity.

We denote the state vector of the MDP as (*t*, *B*(*t*)), where *t* is the time at the beginning of current time slot and *B*(*t*) is the battery level at time *t*. A state *i* is a *depletion* state if *B*(*t*) = 0, which indicates the depletion of the battery. Associated to each state *i*, there is a set of *K* actions $a_i^k \in A(i) = \{a_i^0, a_i^1, a_i^2 \ldots, a_i^{K-1}\}$. Each action corresponds to a QoS level of the phone and we

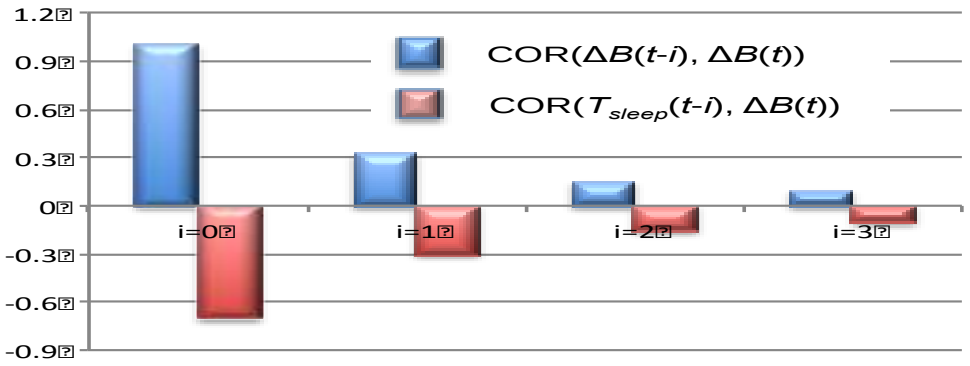


Figure 3-5 Temporal correlation of phone usage.

assume that the average power consumption of the phone at the *k*th QoS level is known. The reward of state *i*, denoted as $r(i, a_i^k)$, is set proportional to the chosen QoS level a_i^k.

(2) MDP training and solving

By observing the history of smartphone activities, we train the MDP model for each user. The training process is to determine the transition probability ($p_{i,j}(a_i^k)$) from state *i*, to state *j* under action a_i^k and the reward $r(i, a_i^k)$.

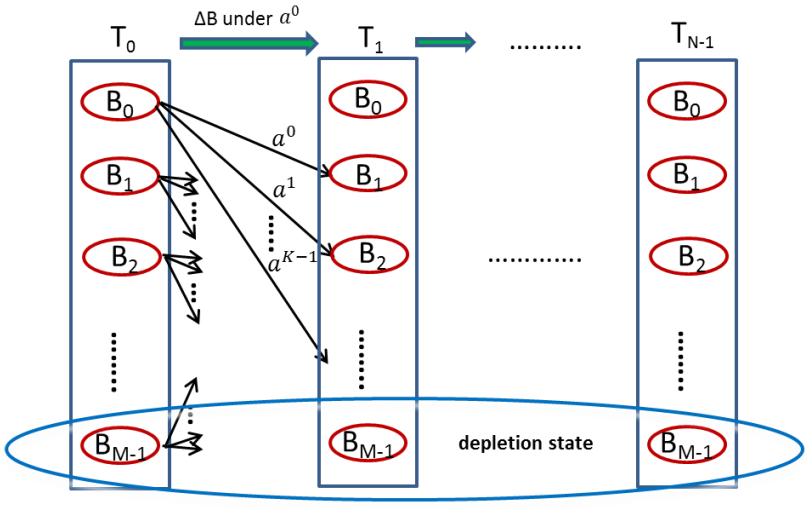


Figure 3-6 MDP training process

The detail of our MDP model training is shown in Figure 3-6. Each red circle in the figure represents a distinct state. The entire state space is divided into *N* groups. Each group corresponds to a specific time (i.e. *t*) in the state vector. With each group there are *M* states corresponding to *M* battery levels (i.e. *B* in the state vector). B_{M-1} is the battery depletion state which should be avoided. From a state in group T_n, the system will go to a state in group T_{n+1}. If $n=N-1$, then the system will go to a state in group T_0. The training sequence is collected from system without any QoS boost, in other words, only the action a^0 is taken in the original training trace and the battery change ΔB under a^0 is recorded. We assume 70% hourly battery increase rate during battery charging. Because we target only at users whose activities are independent to battery level, we assume the same usage activity happens regardless of the current status of battery. The recorded information ΔB reflects the workload activities, from which we can estimate the battery change for other QoS settings $a^k, k \neq 0$. The amount of battery change decides which state the system will transit into. In this way, for every remaining battery state *B* belonging to the same group and every action *a* of this state, we can calculate the next battery state *B'* and reward *r*. Then after training, the transition probability ($p_{i,j}(a_i^k)$) can easily be calculated and the reward $r(i, a_i^k)$ is calculated as the average of the rewards received under this state action pair.

The optimal policy is found by solving the mathematical program as following [117]:

$$\max \sum_{i \in S} \pi_i \sum_{a \in A(i)} f(i,a) r(i,a) \tag{3.1}$$

subject to

$$\pi_j = \sum_{i \in S} \pi_i \sum_{a \in A(i)} p_{ij}(a), \ j \in S \tag{3.2}$$

$$\sum_{a \in A(i)} f(i,a) = 1, \tag{3.3}$$

$$f(i,a) \geq 0, \quad (3.4) \qquad \sum_{i \in S_{Depletion}} \pi_i < L, \quad (3.5)$$

where π_i is the stationary distribution for state i. $f(i,a)$ is the probability that an action a is taken if the state of the system is i and it is the set of variables that we need to optimize. $r(i,a)$ is the reward received if action a is taken in state i. $p_{ij}(a)$ is the transition probability from state i to state j, given that action a is taken at state i. S is the state space of the system. Equation (3.2) constraints the balance of the state probability and transition probability. Equation (3.3) specifies that probabilities of all actions taken in a state should add up to 1. The constraint (3.5) specifies that the overall probability of those depletion states should be less than L. L is a small number possibly given by the user based on their specific tolerance of battery depletion. Higher tolerance usually will lead to more performance boost opportunities.

By introducing a set of new decision variables x_{ia}, $x_{ia} = \pi_i f(i,a), i \in S, a \in A(i)$, the above non-linear problem can be transformed into a linear one and solved.

3.1.4 Implementation and Evaluation

(1) Implementation

We use the real user traces from the Livelab project [118] as we mentioned before. We use four-way cross-validation [129] in the experiments. Traces of every user are partitioned into 4 equal sized subsets, 3 of which form the training set and 1 of which will be the testing set. The final result is the average of all tests. A simulator is implemented using C++ to evaluate the policy. If the user tolerance of battery depletion (i.e. L in constraint (5)) is too low, no feasible solution can be found. A best effort solution will be used instead.

It is assumed that a phone has three QoS levels corresponding to 1x, 1.5x and 2x of the default setting. The energy dissipation of the phone is assumed to be proportional to its QoS level and the original energy dissipation when no boost was performed. This assumption is used only to simplify the experiment setup. How to adjust the QoS of different applications and what is the relationship between the QoS and power consumption are nontrivial problems outside the scope of this work [120].

In addition to the MDP based approach, three reference approaches are also simulated. The first one is Q-learning based approach[127] which generally shares the same underlying model with MDP but is an online learning method. We refer to this method as '*ML*' in our simulation. The second one is based on [119], which profiles the histogram distribution of the remaining energy at the time of battery recharge. The profiled information will be used to predict the lower bound of the remaining energy and the QoS of the phone will be raised proportionally based on the estimation. A confidence level is determined based on the profiled distribution that specifies the probability that the prediction is correct. We refer to this method as '*HIST*' in our simulation. The profiling is performed on the training set and the policy is tested on the testing set. The third reference policy is prediction-based approach, which predicts the remaining energy using the 2-input neural network as described in Section 3.1.2. Similar to HIST, the QoS will be raised based on the prediction. We refer to this method as '*Nnet*'.

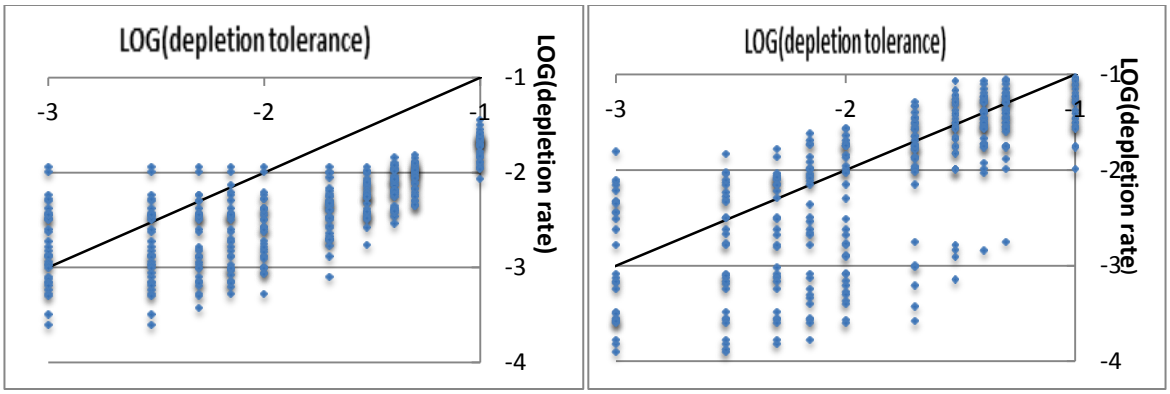
(2) Evaluation Result

Different level of user tolerance of battery depletion leads to different potential of QoS boost. For the MDP method, varying L in Equation (3.5) gives the indication of different tolerance of the battery depletion. For example, L=0.01 means that the user can tolerate 1%

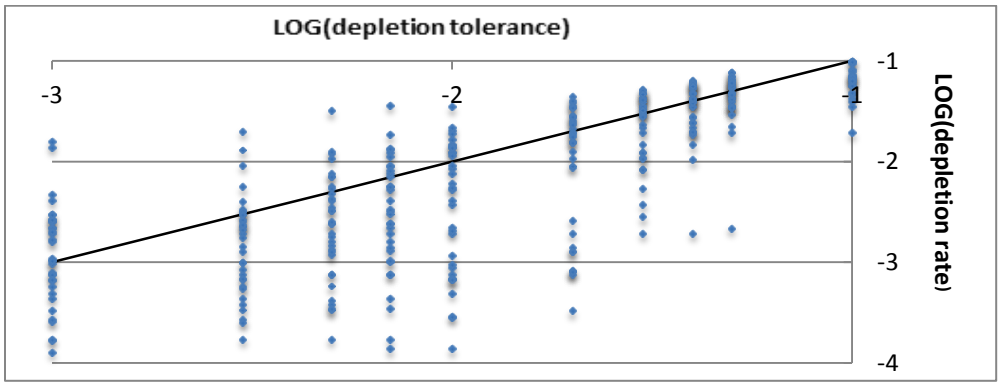
chance of battery depletion during the entire smartphone usage in exchange for performance boost. For HIST, the confidence level is set to be $1 - tolerance$. For ML method, the battery depletion tolerance is tracked using a feedback control method by dynamically changing the penalty of the battery depletion state. All the results following are based on four-way cross validation.

For all the 34 users, we vary the depletion tolerance from 10% to 0.1% and recorded the amount of QoS boosts and actual battery depletion rate. The results are shown in Figure 3-7.

Figure 3-7 (a)~(c) compares the actual battery depletion rate (Y-axis) with the depletion tolerance (X-axis) for all 34 users under different management algorithms. The performance of Nnet is not shown here, because there is no way to integrate the user depletion tolerance with the prediction based management. The black line in the figure represents the ideal cases where the actual depletion exactly meets the constraint. The points above the line correspond to systems that are under-constrained and have depletion violation while the ones below the line are systems over-constrained. Note that both X and Y axes are logarithmic, therefore the difference between the actual and the constraint is magnified when depletion tolerance is low. As shown in the figure, in most cases all 3 algorithms tend to over-constrain than under-constrain points. When depletion tolerance is high (i.e. loose constraint), HIST is more conservative than ML and MDP and all cases using HIST are over-constrained. The correlation between actual depletion and depletion tolerance is calculated and given in the figure. The higher correlation means more précised management. As we can see, the system using MDP management achieves the highest correlation (i.e, 0.78).

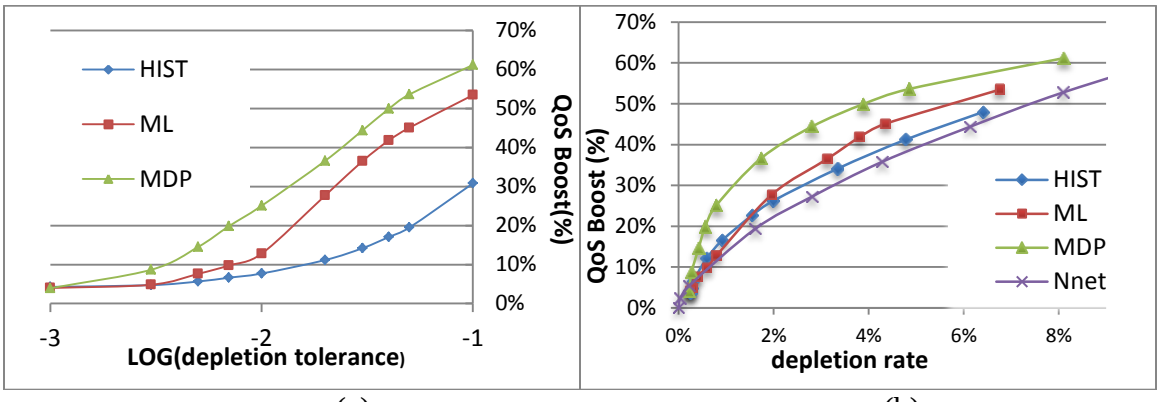


(a) HIST (correlation: 0.7) (b) ML (correlation: 0.71)



(c) MDP (correlation: 0.78)

Figure 3-7 actual depletion rate vs. depletion tolerance



(a) (b)

Figure 3-8 QoS Boosts vs. (a) depletion tolerance (b) actual depletion rate

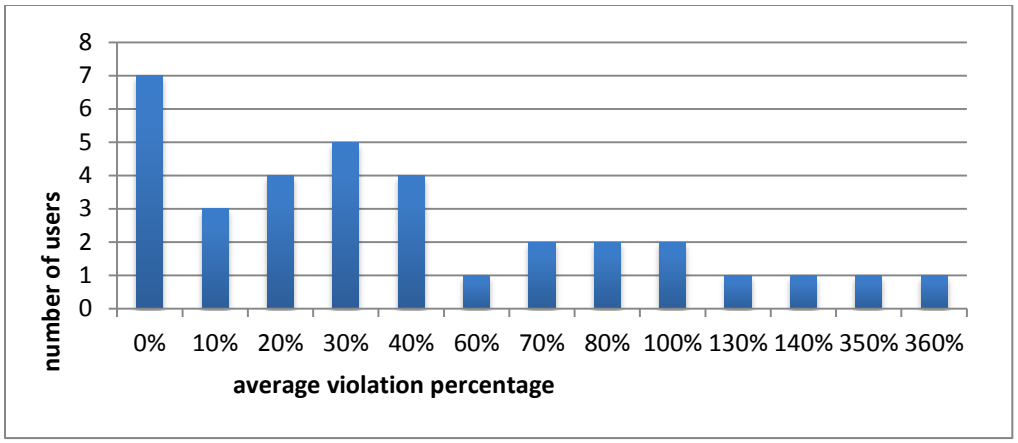


Figure 3-9 . MDP violation percentage histogram

Figure 3-8 (a) shows the percentage QoS boost under different depletion tolerance compared to nominal case without any QoS boost. The results reported here is the average of all 34 users. As we can see, MDP gives the most QoS increase and the HIST gives the least. The low violation and low performance boost shows that the HIST is much more conservative than MDP and ML especially when the depletion tolerance is loose. Figure 3-8 (b) shows the relation between the QoS boosts and the actual battery depletion rate collected from simulations. In the figure, the curves at the upper left are better than the curves in the lower right. Again, the results reported here is the average of all 34 users. Note that this figure shows the tradeoff between QoS boost and the actual battery depletion rate. Those data points with similar X values might not correspond to the same depletion tolerance constraint. For Nnet, the tradeoff curve is obtained by varying the amount of energy that is distributed for QoS boosting as different portions of the predicted target variables. As we can see, the MDP method gives the highest QoS improvement than others with the same battery depletion, while the Nnet gives the lowest QoS improvement. It shows that the stochastic control method outperforms others as it achieves better QoS and energy reliability tradeoffs by better tracking different user's battery usage and recharge patterns.

One of the concerns for the stochastic approaches is that they still have a few violations (under-constraints) when depletion tolerance is not very tight while the HIST has not violation at all as shown in Figure 3-7. The foremost reason is that setting confidence level of HIST to $(1 - tolerance)$ is very conservative and will prevent QoS boost. On the other hand, the phone usage and battery charging pattern for some users are not always consistent during all the time and it is hard to capture their behavior using simple MDP models as we did. Figure 3-9 shows the histogram distribution of average degree of violation for those 34 users under the MDP based energy management. The average degree of violation is calculated as $\max[0, \frac{depletion_{actual} - depletion_{tolerance}}{depletion_{tolerance}}]$. Please note that the degree of violation is defined as the relative increase of battery depletion compared to the tolerance. For example, if the depletion tolerance is 1% and the actual depletion is 2%, then the degree of violation is 100%. The X-axis in Figure 3-9 gives the range of average degree of violation, 0% stands for the range [0%, 10%]. The Y-axis is the number of users whose average depletion rate falls in the corresponding range. For instance, among 34 users, there are 7 users whose average degree of violation is between 0%~10% and 23 users whose degree of violation is less than 50%. As we can see, the majority of users have reasonable degree of violations, only some "inconsistent" users deteriorate the results greatly. For those users, more sophisticated models may need to be developed.

3.1.5 Conclusions

In this work, we aim at increasing the QoS of mobile devices considering the fact that many users recharge the battery before depletion. Neural network model that predicts the remaining energy at next battery charge is first investigated. The results show that accurate prediction is difficult. Then we present a stochastic framework for QoS boosting under the user

specified battery depletion tolerance. The model is trained using real user traces and the framework is simulated and compared with existing approaches.

3.2 User-Aware Energy Efficient Streaming Strategy for Smartphone Based Video Playback Application

3.2.1 Introduction

Battery life has continuously been one of the critical factors for smartphone user satisfaction. The increasing complexity of hardware and applications in the smartphones outpaces today's battery technology [106]. Although the percentage of power consumption of different components (e.g., processor, LCD, WiFi card, cellular interface) varies in different applications and different system settings, the cellular interface will soon become the most dominant energy consumer, which consumes more than 50 percent of total power consumption of the smartphone, when it is used as the network interface [107].

Video downloading and playback is one of the most common activities on the smartphone that has high energy consumption. When a user starts to watch a video, how long he (or she) is going to continue is unknown. Many traditional video players (application programs) try to download as many video data to the memory as possible during the time the user is watching in order to reduce glitches. Such buffering strategy may download more data than the user is going to watch if the user quits the video at an early time. It is shown that in Youtube, 60% videos are only being watched for less than 20% of their duration due to various reasons [109]. So a lot of downloaded data is useless as a result. On the other hand, to avoid excessive downloading, some video players buffer small amount of data periodically as a burst. Whether

the next chunk of data will be downloaded is based on the user's watching progress. However, such downloading strategy will either keep the network interface active all the time or keep on switching it back to active from low power mode. In both cases, there will be energy wastes due to idle power consumption and the switching overhead of the network interface. Although such overhead is negligible for today's WiFi interface with the efficient implementation of power saving mode [108], it is still quite substantial for the cellular interface.

Downloading just enough video that the user is going to watch in a burst is the most energy efficient strategy, however, it is not possible to accurately predict the real playback length as it is a random variable that has strong dependency on the user behavior and the video contents. The problem is very similar to a stochastic inventory system [114] in operation research where user demands follows stochastic distributions and the supply over-stocking and shortage are associated with costs.

We analyze video watching activities of different smartphone users and propose a stochastic model to capture the distribution of the real playback length. Based on the model the amount of data to be buffered is determined so that the expected energy waste on the cellular interface is minimized. As the video progresses, the estimated distribution of the remaining demands is dynamically updated and consequently affects the amount of data to be downloaded in the next burst. To the best of the authors' knowledge, this is the first work aiming at finding the best buffering strategy during video playback for energy saving in the network interface. The main technical contributions of this work are summarized as the following: (1) A Gaussian Mixture Model (GMM) is proposed to capture the distribution of the video playback length for different users. (2) Based on the GMM generated, we apply stochastic inventory theory to find the best buffering strategy (i.e. how much video data should be buffered in each burst) for

minimum energy wastes on the cellular network interface. (3) An Android application is developed to collect real video playback activities. The proposed model and buffering strategy are evaluated using the real data.

3.2.2 Background

(1) Power consumption of cellular interface

The cellular interface is in general far less energy efficient than the WiFi interface. Its transmission energy per bit is usually much higher and varies with the signal strength [111]. It also has ramp energy and long tail energy state when no useful data is transferred [107][110], therefore putting cellular interface into lower power idle state and bringing it back when needed always associate with a non-negligible energy waste. Figure 3-10 shows the state machine of the 3G cellular interface [107][110]. As soon as the 3G interface receives a transmission request, it leaves the low power mode and enters the CELL_FACH (Forwarded Access Channel) state, which has very low throughput (less than 15kbps). If the amount of data to be sent or received is greater than the threshold, it will soon enter the CELL_DCH (Dedicated Channel) state, which has high throughput and energy consumption. When downloading is completed, the 3G interface will enter CELL_FACH state again after Inactivity Timer 2 has expired, and later switches to the low power IDLE state after the Inactivity Timer 1 has expired. Our measurement found that the Inactivity Timer 1 is about 4 seconds while the Inactivity Timer 2 is about 7 seconds. Please note that this information is vendor specific and it changes for different phones using different cellular networks ([110]).

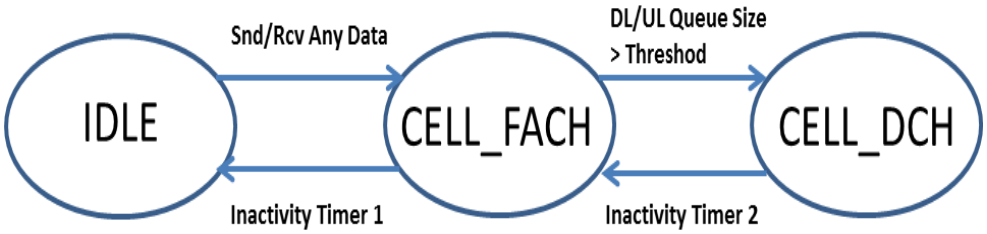


Figure 3-10. The radio resource state machine of 3G interface

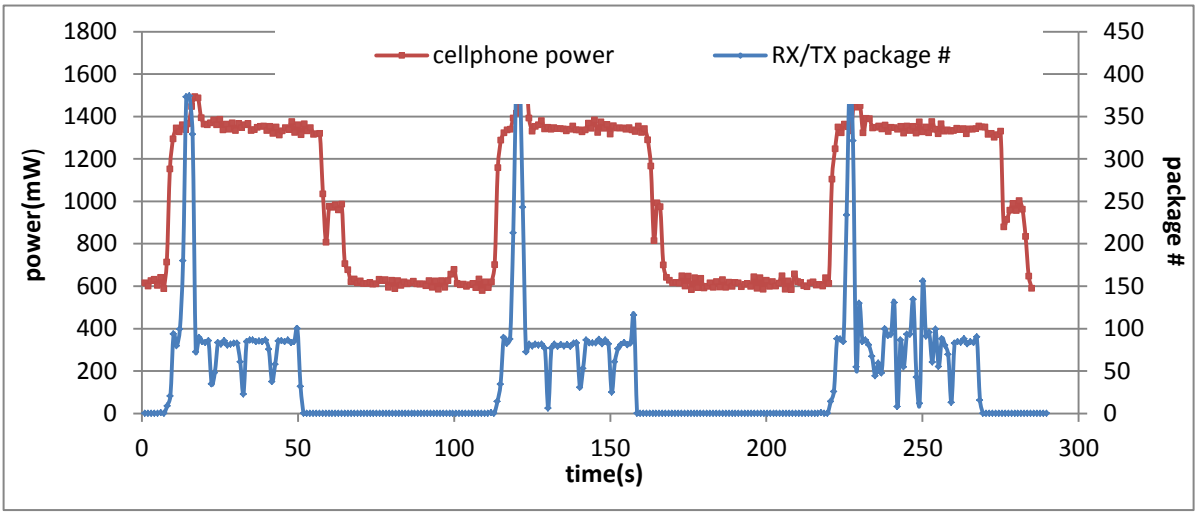


Figure 3-11. Power trace and network activities during YouTube playback

(2) Existing buffering strategy in YouTube

Very little information about the downloading strategy used by YouTube can be found in published works. We tried our best to test it and found that it's quite different from phone to phone and potentially place to place. Typically the YouTube software always pre-buffers certain amount of video data ahead of user's current watching progress. If the signal strength is high and data can be transferred quickly, then the cellular interface will go to low power idle state when 'enough' data has been downloaded. Otherwise the cellular interface will keep buffering video data to keep up with the user watching progress. Our study shows that the amount of pre-

buffering is a fixed value for a particular phone, although it varies over different phones. Based on the data provided by PCWorld, the average 3G data download speed is around 2 Mbps (Megabits per second) [115]. This is much faster than the data rate of a regular quality YouTube video, which is measured to be around 300 kbps. As we can see, if each pre-buffer process provides enough data to support the playback for a time that is longer than the count down period of Timer 1 and 2, it creates an opportunity for the cellular interface to go to low power state.

Figure 3-11 shows the measured power trace of a Nexus S smartphone with AT&T 3G network during the time YouTube is playing. The amount of packages received and transmitted is also shown in the figure. As we can see, the power consumption of the phone is periodically reduced to about 50% of its peak value when there is no transmission and receiving activities. This is because the 3G interface enters low power mode. We can also see that there is a visible delay from the time that transmission and receiving activity stops to the time that the power reduces. This timeout period is the overhead of turning off the 3G interface as discussed in Section 3.2.2.

The goal of this work is to model different users' watching behavior (i.e., length of video playback) and then try to find the best strategy of streaming (i.e. the optimal size of pre-buffering) to minimize the energy wastes on cellular interface.

3.2.3 User Behavior Modeling and Downloading Strategy Optimization

(1) GMM based playback duration modeling

The actual length of video playback is a random variable that is affected by user habits and the content of video clips. Although it cannot be accurately predicted in advance, its distribution can be learned through user behavior analysis of the past. Such model must be user

specific, because different users watch videos clips with largely different average length [109]. Even for a single user, the length of video playback will have large variations. A user may have different video watch behavior (i.e. the watch time) toward different genres of clips. For example he or she may watch a music show for a very short time while spend longer time on a movie. There are different classes of scenarios of video watch (e.g., music show, movie and etc.); some of these scenarios probably even cannot be clearly identified. For each scenario, the video playback length is a random variable that has relatively small variance, while overall video playback length is a mixture of these random variables.

Based on the above discussion, we propose to model the distribution of playback length using a *Gaussian Mixture Model* (*GMM*) [113]. A GMM is a weighted sum of *M* component Gaussian densities given by the following equation: $p(x) = \sum_{i=1}^{M} w_i g(x|\mu_i, \Sigma_i)$, where x is a D-dimensional feature vector, w_i, i=1, …M are weights, and $g(x|\mu_i, \Sigma_i)$, i = 1, …M, are the component Gaussian densities with mean vector μ_i and variance vector Σ_i. In our problem, the feature vector has one element, that is, the length of playback. The values of M, μ_i and Σ_i are learned from training set, which is collected from past user behavior.

(2) Energy efficient streaming strategy

(a) Deciding pre-buffering size

In the next, we will present our energy efficient streaming strategy, which determines the amount of data to be pre-buffered at minimum energy waste. We model the streaming process as a stochastic inventory system [114]. Deciding how much data to be pre-buffered for a video player is analogous to finding how much inventory to replenish for a company. An inventory system has two cost components [114]: *holding cost* which is the cost associated with the storage

of the inventory until it is consumed and *shortage cost* which is the cost when the amount of the commodity demanded exceeds the available stock. In our video buffering system, the *holding cost* is the energy wasted for downloading those excessive video data that are not used, while the *shortage cost* is the energy wasted to wake up a cellular interface in order to download more data when the user tries to watch beyond current buffering point. Please note that both holding cost and shortage cost are wasted energy. We do not consider the energy required to download the video data as cost, as long as these data are actually used.

We use D to denote the amount of video data that is actually watched by user (i.e. demand). D is a random variable. We use y to denote the decision on the size of pre-buffering (i.e. inventory). Both D and y are measured by the length of video playback time in seconds. The holding cost $h(D, y)$ is defined using Equation (3.6):

$$h(D,y) = \begin{cases} 0 & when\ y \le D \\ (y - D)e_h & when\ y > D \end{cases} \tag{3.6}$$

,where constant parameter e_h gives the energy needed to download one second of video data. Based on the definition, if the amount of data pre-buffered is less than the demand, then there is no holding cost. Otherwise, holding cost will be the amount of energy needed to download the extra data.

If there is a shortage, the cellular interface will be turned on in the future to download more data. The more shortage we have, the more frequent the cellular interface will be turned on and hence more switching overhead. Therefore the shortage cost is defined as (3.7):

$$s(D,y) = \begin{cases} (D - y) * e_{sw}/\alpha & when\ y < D \\ 0 & when\ y \ge D \end{cases} \tag{3.7}$$

, where constant parameter e_{sw} is the energy overhead to switch off and wake up the cellular interface. The parameter α represents the expected downloading size in the future. Therefore, $(D-y)/a$ gives the expected times that the cellular interface will be turned on. Obviously, the real value of α is unknown at the time when the shortage cost is calculated. In our experiment, we learn this parameter from the analysis of the training set. We denote the ratio of e_{sw} and α as e_s, $e_s = e_{sw}/\alpha$. Based on the definition, if the pre-buffered data is more than the demand, then there is no shortage cost. Otherwise, the more shortage we have, the more frequently the cellular interface will be turned on to fill in the data, hence the shortage cost is proportional to the size of shortage.

The overall *cost* is the sum of holding cost and shortage cost given by Equation (3.8):

$$C(D,y) = h(D,y) + s(D,y) \tag{3.8}$$

If we use $\varphi_D(\varepsilon)$ to represent the probability density function (PDF) of *D*, then the following equation gives the expected cost *C(y):*

$$C(y) = E[C(D,y)] = \int_0^y (y-D)e_h\varphi_D(\varepsilon)d\varepsilon + \int_y^\infty (D-y)e_s\varphi_D(\varepsilon)d\varepsilon \tag{3.9}$$

The maximum and minimum value of *C(y)* can be found by solving the equation:

$$C'(y) = 0 \tag{3.10}$$

We can also prove that:

$$C''(y) = (e_h + e_s)\varphi_D(y) \geq 0 \tag{3.11}$$

Therefore, the minimum value exists.

Theorem 1 ([114]). The optimal size (y^0) of video data to be pre-buffered with minimum energy waste satisfies the following equation:

$$\Phi(y^0) = e_s/(e_h + e_s) \tag{3.12}$$

where Φ is the cumulative density function (CDF) of the playback length *D*. Because all CDF are monotonically increasing functions, the value of y^0 can be found easily using binary search.

(b) Adaptive streaming strategy

As mentioned in Section 3.2.3, we use GMM model to capture the distribution of playback length. The model is learned from the usage history of the specific user, and its CDF is denoted as $G(D)$. *D* is an estimation of playback time without any prior knowledge. As the playback process goes on, we gather more information about *D*. For example, if the video has played for *A* seconds and it is still going on, we know that $D > A$. Therefore, the distribution model of *D* should be updated to reflect this information. As shown in Figure 3-12, let *g*(*D*) be the original PDF of the *D*. After knowing that *D* is greater than *A*, we can set the PDF to be 0 for any *D* less than *A*. In order for the remainder of the PDF to have an integration of 1, the area of the blue shaded part will be redistributed to the rest of the *g*(*D*), and we get an updated PDF $g'(D|D > A)$ as the following:

$$g'(D|D > A) = \begin{cases} 0 & if\ D < A \\ g(D)/(1 - G(A)) & if\ D \geq A \end{cases} \tag{3.13}$$

where *G*(*A*) is the CDF of *g*(*D*) at point A, in other words, the area of the blue shaded part in Figure 3-12. Note that *G*(*A*) is known after the model is given. The updated CDF $G'(D|D > A)$ can be calculated as:

$$G'(D|D > A) = \int_A^D g'(t|t > A)dt = [G(D) - G(A)]/[1 - G(A)] \quad (3.14)$$

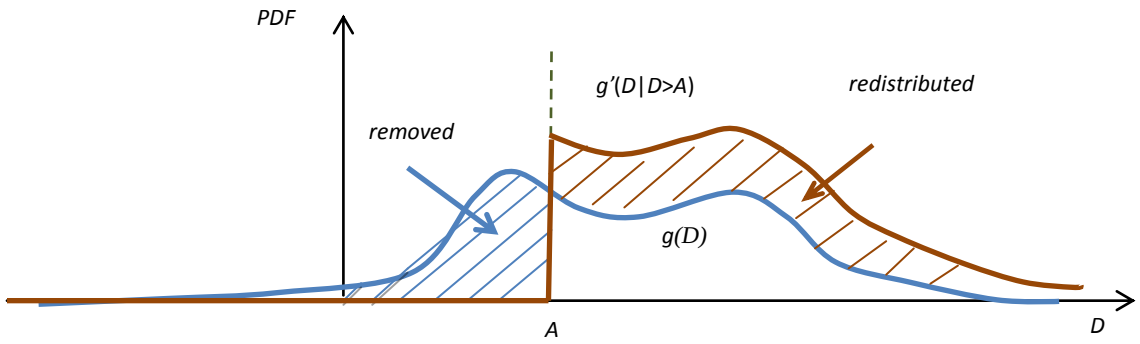


Figure 3-12 Adjusting the PDF based on obtained usage information

After that, the optimal size of data to be pre-buffered next is determined based on the updated CDF *G*'(*D*|*D*>A). This procedure will continue as the video watching activity goes on.

3.2.4 Experimental Results

(1) User behavior model evaluation

We developed a lightweight Android application that monitors and collects the users' YouTube video playback activities. We take advantage of the Android 'logcat' tool, which shows the debug messages ejected from both operating system and applications. Our study shows that each time a new video in YouTube starts, a "WatchActivity" intent of YouTube will be written in the log buffer. Whenever the video stops playing because the user quits or pauses the video, an audio hardware sleep message is ejected. Similarly, whenever the video starts playing because the user starts a new video or resumes an old one, an audio hardware wakeup message is ejected. Combining the audio hardware message and "WatchActivity" information, we are able to detect those time when the video is paused and remove it from the playback length.

Our application doesn't require root privilege and we installed it to five Android phones belonging to five different users. The application is running as a background service. The users are encouraged to watch YouTube videos as usual. The application records the history of their video playback length in a log file. After one and a half months of experiment, the log file is collected. In order to collect enough data, we recorded YouTube playback activities no matter the user uses WiFi or cellular access. We assume that the user behavior does not change significantly in both cases.

We use the first half of the collected traces as training samples and apply Matlab function "*gmdistribution.fit*" to learn the GMMs for each user. The function performs maximum likelihood estimation. We kept on increasing the number of Gaussian components until there is no significant improvement of the log-likelihood of the training sequence. For the purpose of comparison, we also model the training samples using normal distribution and exponential distribution. After constructing the model, we use the second half of the collected traces to test our model and calculate RLH (root likelihood) of each model.

Table 3-1. Comparison of user behavior models

Users	1	2	3	4	5
Means of Gaussian Components	{107,724, 2004}	{59,294, 2193}	{61, 1808}	{45, 361}	{43, 327}
RLH(GMM)	0.0009	0.0020	0.0039	0.0022	0.0061
RLH(Exp)	0.0009	0.0010	0.0016	0.0019	0.0049
RLH(Norm)	0.0003	0.0004	0.0004	0.0013	0.0022

Table 3-1 shows the comparison results. Each column represents different user. The second row gives the means of the Gaussian components in the GMM model for each user. As we can see, there are noticeable differences from user to user in terms of the distribution of their YouTube watching time. For users 1 and 2, their GMM models contain 3 Gaussian components; while for the other users there are only 2 Gaussian components. The mean of these Gaussian components also varies significantly. We believe that such difference is mainly due to the different watching interests of various users. The third, fourth and fifth rows in Table 3-1 give the RLH (root likelihood) of the testing sequence for GMM, exponential distribution and normal distribution models respectively. As we can see, the RLH of GMM model is 1.6 times and 4.4 times of the RLH for exponential and normal distribution models respectively. This indicates that the GMM is more suitable to describe the distribution of playback time.

(2) Comparison of energy savings

In the second set of experiment, we compare inventory theory based streaming strategy with heuristic strategy, which buffers fixed amount of data each time. For the inventory theory based approach, different user behavior models, including GMM, exponential, and normal distribution models, are tested to show the impact of good user behavior modeling. In order to demonstrate the effectiveness of the inventory theory based streaming strategy without worrying about the accuracy of user behavioral model, we created a synthetic testing trace that follows ideal GMM distribution. It has three Gaussian components whose means are (50, 500, 2000).

As mentioned in Section 3.2.3, in this work, we focus on reducing the energy wastes due to either excessive downloading or frequent on-off switches. The power consumption on cellular interface varies from phone to phone and from time to time on the same phone because of

different signal strength. To have a simple comparison, we use the amount of wasted video data to represent energy waste and it is measured by the length of playback time in seconds. Based on the definition of stocking cost, we know that e_h is equivalent to 1 second (of wasted video data). Using Monsoon power monitor[112] we measured the energy dissipation of a Nexus S smartphone with AT&T 3G network. Our data show that, with moderate wireless signal strength, the energy overhead to switch the cellular interface off (i.e. the energy dissipation in the timeout period) is approximately equal to the energy that is needed to download 16 seconds video data for YouTube. Therefore, e_{sw} in Equation (3.7) is set to 16.

Two static streaming strategies that have constant pre-buffering size are implemented. The first one buffers 100 seconds video data each time and is referred as Static(100s). We then sweep the pre-buffering size and test it using the training set of each user. The one that gives the minimum energy waste is selected and be applied to the testing sequence. We refer this approach as Static(best).

Figure 3-13 compares the wasted seconds (as the measurement of wasted energy) for different users and different buffering strategies. For the synthetic trace, the inventory theory based strategy reduces about 38% energy waste compared to the static strategies. We can also see that, the downloading strategy based on GMM model has 52% and 36% lower wasted energy that the strategies based on normal and exponential distribution models respectively. This indicates the importance of having accurate model.

However, for the testing traces from real users, the inventory theorem based strategy using GMM model has only 10% and 20% reduction of wasted energy in average compared to the Static(best) and Static(100s). And compared to the strategies based on normal and

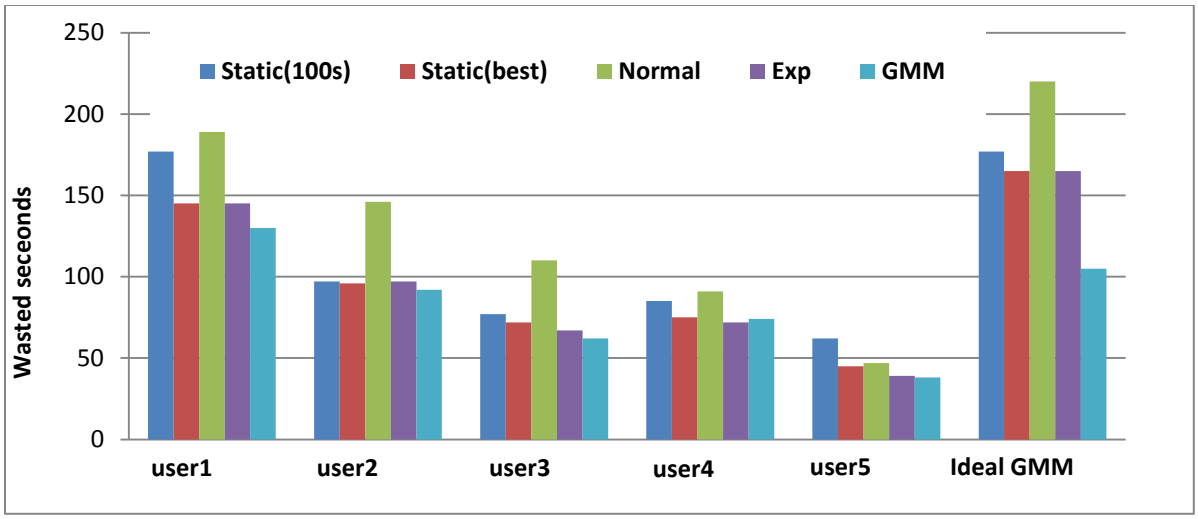


Figure 3-13 Comparison of wasted seconds for different users and different buffering strategies

exponential distribution models, our model has 31% and 5% less energy waste in average. The relative efficiency of our strategy reduces for real trace partly because the GMM based user behavior model is not perfectly trained probably due to insufficient training data. On the other hand, exponential distribution is not a bad model to characterize the statistics of playback time, because there is much less probability that a user watches a long video than he/she watches a short video. Therefore, streaming strategy using exponential model also gives good energy reduction compared to others.

In order to show how the proposed streaming strategy works, we performed a detailed analysis for trace collected from user1. Figure 3-14 shows the buffering points determined based on the proposed streaming strategy. We can observe that the buffering points are generally around the means of the Gaussian components (107, 724, 2004). Furthermore, the points are sparse before 2000 and then become very dense after that. The streaming strategy algorithm tries to determine when the user will most likely to stop the video and it will set the buffer point to

around that time. For example, after watching the video for 200 seconds, the user usually will not stop for another 300 seconds. So the next buffer point will be set to a little over 500 second and more than 300 seconds of video data will be downloaded to reduce the switching overhead. After the user watches for 2000 seconds, the video could end at anytime. Therefore, very few data will be downloaded each time to avoid waste.

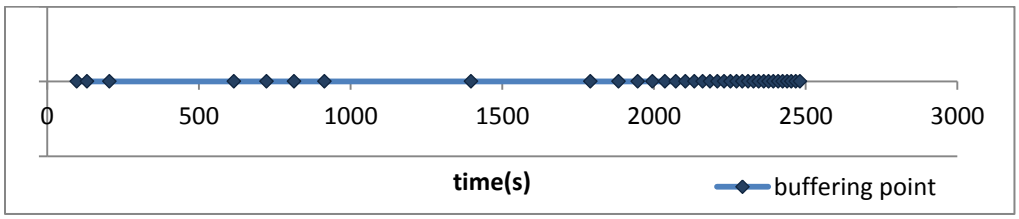


Figure 3-14 Buffering points of user 1 by the GMM approach

3.2.5 Conclusions

In this work, we use GMM to model different user's YouTube playback time and then apply Inventory Theory to find out the optimal buffering points during the video playback that minimizes energy waste on the cellular interface. Our evaluation based on real field study and simulation demonstrates that our approach can save about 10% more energy than the best static buffering method.

3.3 Chapter Summary

In this chapter, we presented two works targeting system level power management in the context rich mobile environment.

In the first work, we aim at increasing the QoS of mobile devices considering the fact that many users recharge the battery before depletion. We present a stochastic framework for QoS

boosting under the user specified battery depletion tolerance. The model is trained using real user traces and the framework is simulated and compared with existing approaches.

In the second work, we use GMM to model different user's YouTube playback time and then apply Inventory Theory to find out the optimal buffering points during the video playback that minimizes energy waste on the cellular interface. Our evaluation based on real field study and simulation demonstrates that our approach can save about 10% more energy than the best static buffering method.

Chapter 4 An FPGA-based Distributed Computing System with Power and Thermal Management Capabilities

4.1 Introduction

High power consumption and high working temperature have become a major issue in the design of today's embedded systems. They increase cooling cost, degrade the system reliability and also reduce the battery cycle time in portable devices. Recently, research in Dynamic Power Management (DPM) and Dynamic Thermal Management (DTM) has attracted substantial interests. The DPM controller dynamically slows down or turns off the computing device when it is idle or under-utilized to reduce the system power consumption [103]. The similar actions are taken by the DTM controller when the die is (or is predicted to be) over heated [104]. The effectiveness of DTM and DPM relies heavily on the workload pattern of the computing device, which is determined by task scheduling and ordering policies, the nature of applications running in the system as well as user input activities. Furthermore, the impact from the OS, such as the overhead of context switch and power mode switch, also affects the efficiency of DTM and DPM.

In this work, we present an FPGA-based distributed computing platform designed to support the research in distributed embedded computing with power/thermal management capabilities. The system consists of multiple FPGAs connecting through Ethernet links with each

FPGA configured as a multi-core system. Hardware and software supports are provided to carry out basic power/thermal management actions including inter-core and inter-FPGA communications, runtime temperature monitoring and dynamic frequency scaling. In the experiments, we evaluated the inter-FPGA and inter-core communication delay, tested the function of the distributed computing system using a case study of parallel matrix multiplication, and evaluated the function of dynamic frequency scaling and temperature sensing by measuring the temperature change of the CPU when it is running at different frequencies.

Compared to software simulation, an FPGA-based evaluation platform provides fast emulation speed which enables us to test the performance of power/thermal management techniques with real-life applications and OS. Compared to computer clusters, an FPGA-based platform has the flexibility to be configured to any hardware architecture and network topology. Hardware based sniffers for performance monitoring and traffic analysis can also easily be added.

Many works have been performed to emulate or develop multiprocessor distributed system using FPGA. Reference [1] gives a good overview of FPGA-based multiprocessor systems. References [85][86] aim at creating an FPGA-based multiprocessor or distributed platforms for teaching purpose. References [87][88] focus on the memory and synchronization problems in an FPGA-based multi-processor system. The authors of [89][90][91] propose different architectures for FPGA-based multiprocessor or distributed system for video applications.

There are also works focusing on FPGA-based multi-processor or distributed system emulator for power and thermal optimization. For example, the authors of [92] present a hardware and software co-systhesis framework for FPGA-based distributed embedded system in

order to achieve low power design. Reference [93] uses FPGA to emulate the power consumption of a multiprocessor system in order to guide task migration. Reference [84] presents the HW/SW of an FPGA-based emulation framework that enables the rapid extraction of a large range of statistics, including thermal modeling, at different architectural levels of MPSoC designs. Both [93] and [84] demonstrate that the FPGA-based HW/SW emulation achieves much faster speed than the software simulation.

Our work differs from all the previous works as we created a generic distributed platform using FPGA where power and thermal managements are possible. The main contributions of this work are:

1. We created a multi-core distributed computing platform using *Nios II Embedded Evaluation Kit* (*NEEK*). The platform consists of multiple FPGAs with each FPGA configured as a multi-core system. The processors on the same FPGA communicate with each other through shared memory and different FPGAs communicate with each other through Ethernet links using the TCP/IP stack.
2. Each core is on a separate clock domain and has the dynamic frequency scaling capability.
3. Each core has its own temperature sensor that mapped to its local address space.
4. Software supports for inter-core and inter-FPGA communication, dynamic frequency scaling and temperature monitoring are provided. Their latency and overhead are analyzed.
5. A case study is provided that shows how to do distributed matrix multiplication using the proposed platform.

The rest of the work is organized as follows: Section 4.2 briefly introduces the software and hardware resources on Altera Nios II Embedded Evaluation Kit, which is the base device of our distributed computing platform. Section 4.3 presents the details of the system architecture and Section 4.4 talks about the design of frequency scaling and temperature monitoring modules. Section 4.5 gives the experimental results and analysis. Section 4.6 is the conclusion.

4.2 Altera Nios II Embedded Evaluation Kit (NEEK)

Although the proposed distributed computing platform can be implemented on any major FPGA, its architecture selection is more or less affected by the hardware and software resources on the FPGA that is chosen for this project. In this section, we give a brief introduction of the resources on the Altera Nios II Embedded Evaluation Kit (NEEK). We will only focus on those resources that will be used in our work.

The Altera NEEK Cyclone III edition has one Cyclone III EP3C25F324 FPGA with 25,000 Logic Elements and 594 Kbits embedded memory, 32 MB DDR SDRAM, 1MB SRAM, 16 MB Intel P30/P33 Flash, 800 X 480 touch-screen LCD, Ethernet 10/100 Mbps, and PS2 and RS-232 connector,. The Nios II is a soft IP core of the embedded processor designed specifically for Altera FPGA. In this work, we choose the MicroC/OS-II RTOS [94] whose full ANSI C source code is included in the Nios II Embedded Design Suite. The MicroC/OS-II is a multitasking operating system that supports maximally 64 tasks with distinct task priorities (ranging from 1 to 64). It supports preemptive scheduling, and always runs the highest priority task that is ready.

The full ANSI C source code of NicheStack TCP/IP Network Stack (Nios II Edition) is also distributed by Altera as part of the design suite. The network stack can be used with the

10/100 Mbps Ethernet Controller (PHY) provided by Altera to establish socket communication via Ethernet links.

4.3 System Architecture

The proposed distributed computing platform has hie-rarchical architecture, which is shown in Figure 1. The system consists of multiple FPGAs connecting via Ethernet links. Each FPGA is further configured as a multi-core system. The cores on the same FPGA communicate via shared memory, while different FPGAs communicate via Ethernet Links.

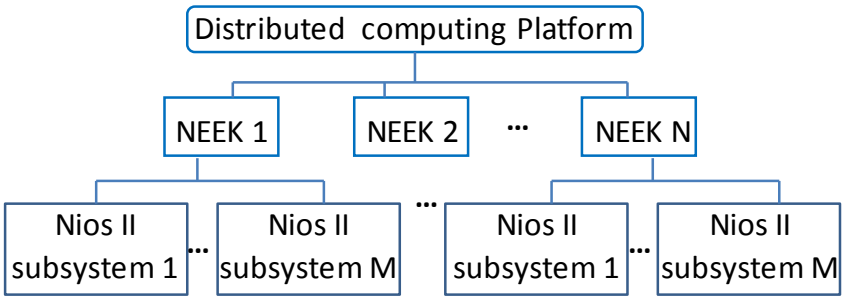


Figure 4-1. Hierarchical architecture

4.3.1 Single FPGA multi-core system

There are multiple Nios II subsystems on a single FPGA. Figure 4-2 shows the block diagram of one of the Nios II subsystem including its memory and some basic peripherals. The JTAG UART provides the debugging port interface. A High Resolution Timer is included to measure the program execution time. The *parallel I/O* (*PIO*) provides control and monitoring to clock generation block and the temperature sensor, which will be discussed in Section 4.4.

One or multiple shared memories are connected to each Nios II subsystem. These shared memories are configured as hardware mailbox for communications among processors on the

same FPGA. Among all the Nios II subsystems on the same FPGA, there is one that has Ethernet interface module. The Ethernet interface module consists of a DMA Controller, a Descriptor Memory and the Ethernet MAC. The Nios II subsystem that has the Ethernet interface acts as a gateway for inter-FPGA communications in the distributed computing system. All peripherals are connected to the Nios II processor via Avalon memory mapped interface [105]. The Avalon streaming interface [105] is used to connect the DMA to the Ethernet controller.

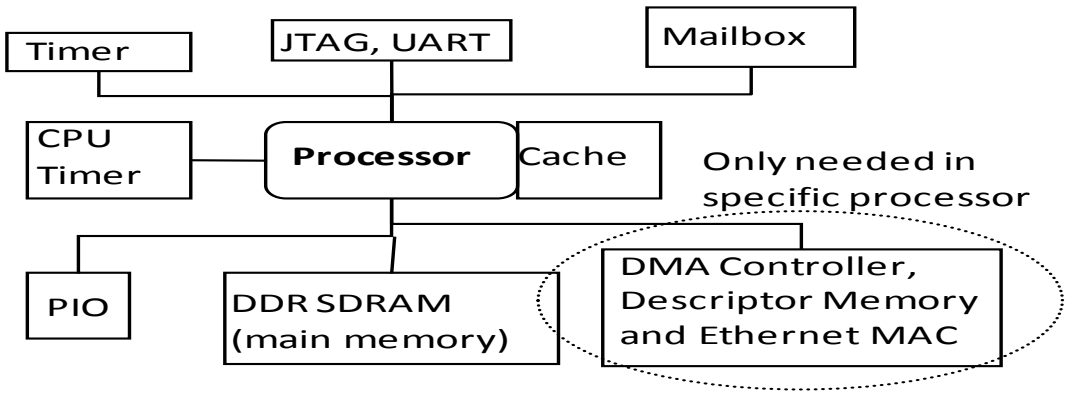


Figure 4-2. Processor configuration

The on-chip inter-processor communication is achieved using the mailbox [95]. Figure 4-3 shows an example of the topology of connections among 4 on-chip processors. Between any two processors that communicate to each other, a shared memory is inserted. The shared memory is configured into 2 mailboxes. One of them stores outgoing messages and the other stores incoming messages. The mailbox core contains hardware mutexes to ensure the mutually exclusiveness during communication through the shared memory where the actual messages are stored. In Figure 4-3, MB_{12} is used to store the messages sent from P_1 to P_2 while MB_{21} is for those sent from P_2 to P_1. Both MB_{12} and MB_{21} are attached to the same on-chip memory block that connects to both processors. Note that Figure 4-3 shows only a simplified system where the communication can only happen between P_1 and other 3 processors. The system can be

configured to implement different topologies of inter-processor communication network, such as a ring or a mesh.

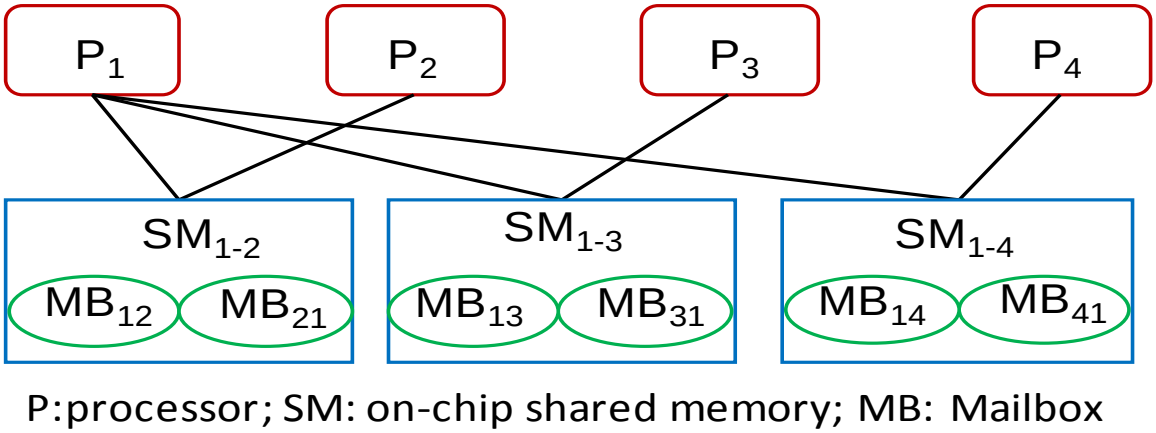


Figure 4-3. Topology of on-chip inter-processor connections

To post a message to an outgoing mailbox, the Mailbox API function altera_avalon_mailbox_post() must be used. The Mailbox API function altera_avalon_mailbox_pend() can be used to read a message from an incoming mailbox. The altera_avalon_mailbox_pend() function provides blocking wait for a message in the specified mailbox. In a multi-tasking OS such as MicroC/OS-II, if the mailbox is empty when the function is called, the process will be blocked and switched to be inactive. The process will be switched to the ready list when the mailbox is no longer empty.

It is necessary to point out that although the mailbox core and its supporting API provide a convenient way to achieve inter-processor communication, its speed is relatively slow for large data communication as we will show in the experimental results. This is because the mailbox API allows the processor to read or write only one 32 bit message each time. For high speed inter-processor communication of large amount of data, hardware mutex core [96] with shared memory should be used.

4.3.2 Multi-FPGA distributed system

The overall distributed computing platform consists of multiple FPGAs connecting via Ethernet links. We use the notation K_iP_j to represents the jth processor on the ith NEEK. All NEEKs are connected through their Ethernet port, which is connected to the first processor of the kit. Therefore, the K_iP_1, $1 \leq i \leq 4$ acts as the communication gateway. The gateway processor establishes the socket TCP/IP connections with all other gateway processor in the distributed system. For each TCP/IP socket, a receiving process is created that performs blocking wait for the incoming data. To send data from K_iP_x to K_jP_y, where $i \neq j$ and $x, y \neq 1$, the processor K_iP_x first sends the data to the gateway processor K_iP_1 via the mailbox, the data is then transmitted from K_iP_1 to K_jP_1 through TCP/IP socket, and finally the data is sent from K_jP_1 to K_jP_y via the mailbox.

As we can see from the above analysis that, in addition to the user applications, each processor must maintain a set of receiving processes waiting for the incoming data from either the mailboxes or the socket interfaces. We use semaphores to coordinate the execution of different processes. For example, in the matrix multiplication example that will be discussed in Section 4.5, the calculation process will wait on a semaphore which will be released by the receiving process when all data for the calculation are ready.

4.4 Frequency Scaling and Temperature Monitoring

Each Nios II subsystem has a hardware clock generation and selection block and a temperature sensor that provides the processor with dynamic frequency scaling and temperature monitoring capabilities.

4.4.1 Glitch free clock switching

One way to achieve dynamic clock frequency scaling is to reconfigure the *Phase-Locked Loop* (*PLL*) during the runtime. The Altera Phase-Locked Loop Reconfiguration Megafunction [98] can be used for this purpose. However, to reconfigure the Altera PLL [99] during runtime takes several microseconds, which is quite large performance overhead. Furthermore, the Reconfiguration Megafunction consumes lots of reconfigurable logic resources as well as on-chip memories, and it requires complicated control.

The second way to achieve dynamic frequency scaling is to switch among a set of clocks running at discrete frequency levels. Each Altera PLL generates up to 5 different output clocks, which is the same as the number of frequency levels supported by an XScale processor. The biggest challenge in designing a clock selection module is how to avoid the clock glitch during the transition period.

Figure 4-4 shows the glitch free clock generation and selection module implemented in our system. Based on the 4 bit control signal (i.e. *control*[3:0]) , it performs 4-to-1 selection from the output clocks generated by the PLL. In this design, the ALTPLL block is the Altera PLL Megafunction; the ALTCLKCTRL block is the Altera Clock Control Megafunction which performs clock enable/disable and clock selection [100]; and the LPM_MUX is a normal multiplexer. The DFF0~DFF3 are D-flip flops. They introduce delays during certain clock transitions to prevent glitch. More details about their function will be discussed later.

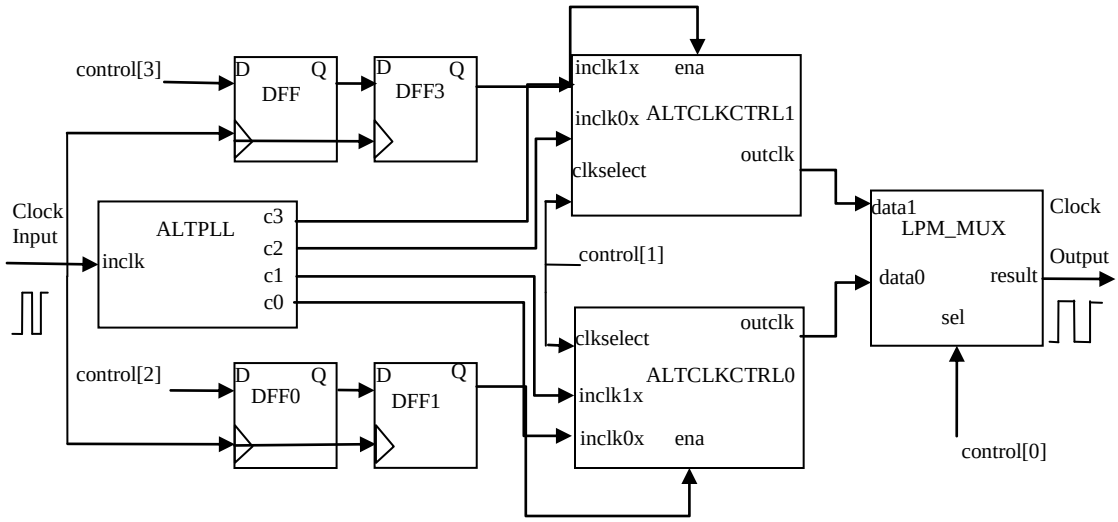


Figure 4-4.Clock generation block

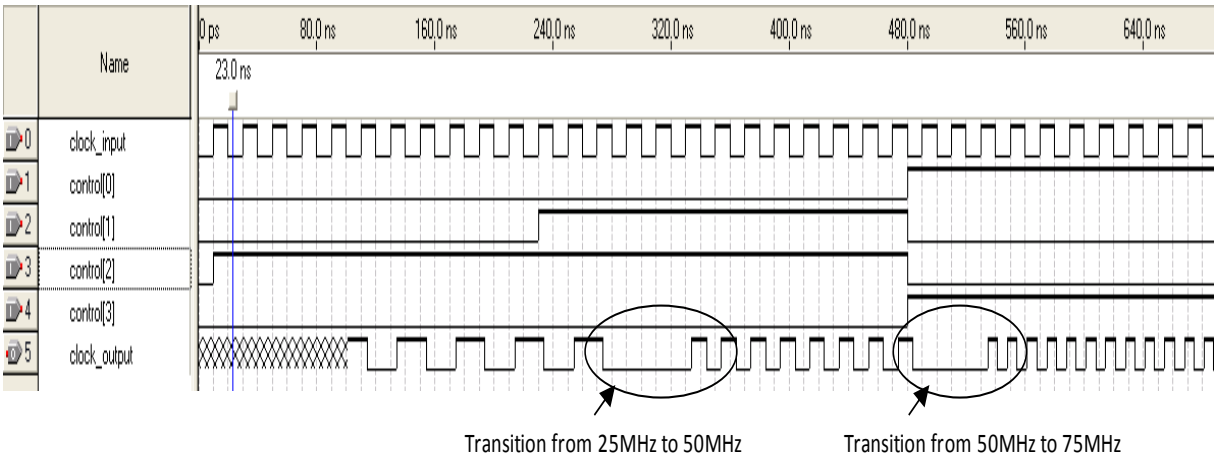


Figure 4-5. Simulation result of clock transition process

Note that the ALTCLKCTRL can only performs 2-to-1 clock selection, if the clock signals are generated by the PLL. Therefore, we have to used 2 ALTCLKCTRL blocks to multiplex 4 clock signals generated by PLL. To guarantee glitch free clock transition, the “ensure

glitch-free switchover implementation" option of the ALTCLKCTRL must be enabled. At anytime, at most one ALTCLKCTRL is active.

The Quartus II software does not support concatenation of clock controllers. That is why a normal multiplexer is used to select the outputs of the 2 ALTCLKCTRL blocks. In Quartus II Analysis & Synthesis settings, we enable the "Clock Mux Protection" option. This option ensures that the multiplexers in the clock network to be decomposed to 2-to-1 multiplexers, each of which will be synthesized to one LUT and hence be glitch free [97].

A clock generation/selection module is associated to each Nios II subsystem. Its control signals (*control*[3:0])are connected to the parallel I/O of the Nios II subsystem. The processor changes its frequency by writing to the corresponding parallel I/O ports. Because at most one ATLCLKCTRL is enabled at anytime, the value of control[3] and control[2] cannot be 1 at the same time.

Figure 4-5 illustrates the simulated waveform of an example of clock switching. Assume that the PLL clocks *c*0, *c*1, *c*2 and *c*3 are running at frequency 25MHz, 50MHz , 75 MHz , and 100MHz. The input clock of PLL is 50MHz. Also assume that the control[3:0] is initially "0100". The ALTCLKCTR1 controller is disabled and gives constantly low output while the ALTCLKCTR0 controller is enabled.

The output clock is connected to PLL output *c*0 through LPM_MUX and ALTCLKCTR0. After the processor write "0110" to control[3:0], the ALTCLKCTRL switches from *c*0 to *c*1. There is no glitch in the output clock because the switching occurs in the Altera glitch free clock control block. In the next, the processor writes "1001" to control[3:0]. This command will eventually turn off the ALTCLKCTR0 and switch the LPM_MUX from ALTCLKCTR0 to

ALTCLKCTR1. Although the LPM_MUX made the switch immediately after the control signal changed its value, the ALTCLKCTR1 is still disabled at this time because its enable signal is delayed by 2 clock cycles. The output clock will stop for a very short period of time during transition and resume afterwards. In this way, glitch on the clock is prevented.

The waveform shows that our clock selection unit takes only about 50 nanoseconds for clock transition. Compare to the PLL Reconfiguration Megafunction which has several microseconds delay, the clock selection unit has much lower performance overhead and needs much less hardware resource.

Since the processor's frequency will be changed at runtime and other components (e.g., memory and other peripherals) work under a fixed clock frequency, we need to bridge the gaps between processor's clock domain and other components clock domain. Avalon Memory Mapped Clock Crossing Bridge [101] is inserted between the processor and memory while Avalon Memory Mapped Pipeline Bridge [101] is inserted between the processor and peripherals.

4.4.2 Monitor the processor's temperature

In order to provide our system with the thermal awareness, an on-chip temperature sensor built up-on the programmable logic resources are implemented and attached to each Nios II subsystem.

We adopted the architecture of the temperature sensor described in [102]. The temperature sensor uses a delay line whose delay increases with the increasing of the temperature around it so that the delay time can be a good measurement of the temperature. In order to monitor the temperature change of a processor, we implemented this sensor and put it inside the corresponding processor using the chip planar in Quartus II. The resource used for the sensor is

very little. Only a little more than 200 logic elements are needed to implement the temperature sensor in our system.

The simplified schematic of the sensor is shown in Figure 4-6**.** After writing to the 'START' signal at the very beginning, the processor can continuously read the temperature value from the sensor via the PIO interface.

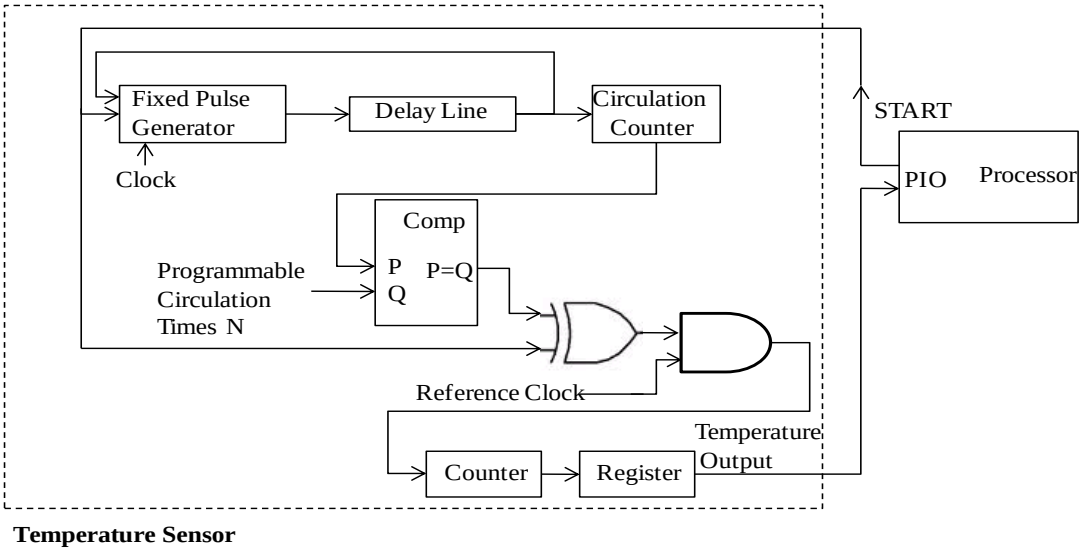


Figure 4-6. Temperature sensor and processor

4.5 Experiments

We used Quartus II and SOPC builder to create the hardware prototype configured in the FPGA and then used the Nios II Software Build Tools (SBT) to develop the user applications and to port them with the hardware abstraction layer (HAL) and operating system to the FPGA board. Our distributed computing platform consists of 4 NEEKs and each NEEK is configured into 4 Nios II subsystems. The 4 Nios II processors have point-to-point communication between

each other and each mailbox between 2 processors is 256 byte large. By default, the CPUs are running at 66.5 MHz. Standard Nios II cores are used with 1 KB instruction cache.

Three experiments have been carried out. The first experiment characterizes the latencies of inter-FPGA communication and inter-processor communications. The second experiment verifies the function of the distributed computing platform using a parallel matrix multiplication program. Finally, the third experiment evaluates the function of the clock selection module and the temperature sensor, by measuring core temperature while changing the core clock frequency.

4.5.1 Characterization of communication latency

Communication latency is an important characteristic of a distributed computing system. This information is usually required by task scheduling and mapping algorithms for power and performance optimization. In the first experiment, we measured the latency of inter-FPGA and inter-processor communications.

The communication latency between K_iP_x and K_jP_y ($1 \le i, j \le 4$ and $1 \le x, y \le 4$) is defined as the duration starting from the time when K_iP_x begins to send the data till the time when K_jP_y receives all the data. K_iP_x and K_jP_y are asynchronous to each other. We cannot find a common time reference to measure the time between events on these two systems. Therefore, once K_jP_y receives the packet, it will immediately send back the same packet to K_iP_x. We measure the round trip latency of the packet and divide it by 2 to calculate the one-way latency. This measurement is applicable because our system is homogenous and symmetrical.

Figure 4-7 shows the communication latency between K_iP_1 and K_jP_1, where $1 \le i, j \le 4$ and $i \ne j$. As both processors are gateway on different NEEKs, this communication involves only the

Ethernet Links. As we can see that the latency is almost a linear function to the transmission size. It takes about 8 milliseconds to send 5 KB data.

Figure 4-8 shows the communication latency between K_iP_x and K_iP_y, where $1 \le i \le 4$, $1 \le x, y \le 4$ and $x \ne y$. Both of the processors are on the same FPGA and their communication involves only the mailbox links. Again, the communication latency is a linear function of the transmission size. As we mentioned earlier, the mailbox based communication is relatively slow for large size

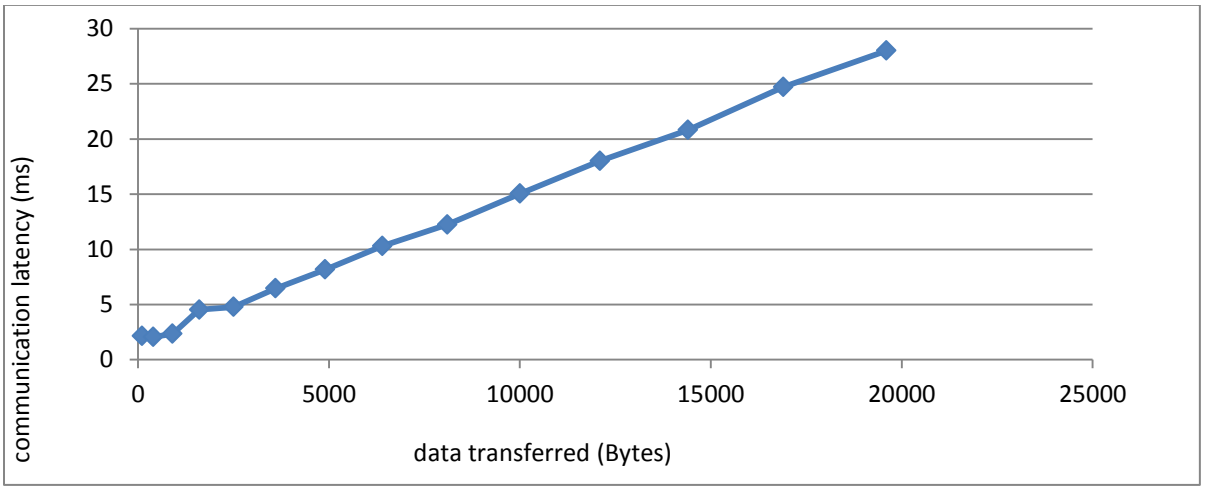


Figure 4-7. Inter-FPGA communication latency

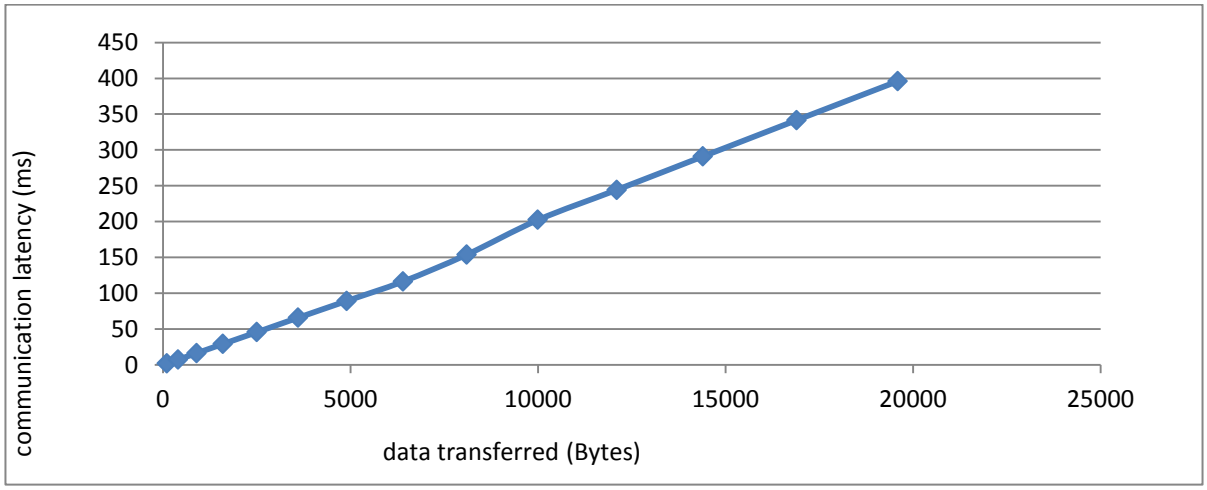


Figure 4-8. Inter-processor communication latency

of data because each time only one 32 bit message can be sent or received.

The communication latency between non-gateway processors on different FPGAs consists of the time spent on the 2 mailbox links and one Ethernet links. For example, if we want to transfer 4KB data from K_2P_2 to K_1P_2, from Figure 4-7 and Figure 4-8 , the overall transfer time will be 2*(transferring 4KB data time through mailbox)+1*(transferring 4KB data time through Ethernet) ≈(2*70.5+6.9) = 147.9 ms. The measured communication latency is roughly 148.3 ms which is very close to our estimation.

4.5.2 Parallel matrix multiplication

In this experiment we test the function of the distributed computing platform using a parallel matrix multiplication program.

Given three *N*x*N* matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$. Let $\boldsymbol{C} = \boldsymbol{AB}$. If we equally divided each matrix into *M* rows and *M* columns, then we create *M*x*M* sub-matrices from the original matrix. Denote those sub-matrix using their row and column index, we will have matrices, $\boldsymbol{A}_{i,j}$, $\boldsymbol{B}_{i,j}$ and $\boldsymbol{C}_{i,j}$, $1 \le i, j \le M$. It can be proved that $\boldsymbol{C}_{x,y} = \sum_{i=1}^{M} \boldsymbol{A}_{x,i}\boldsymbol{B}_{i,y}$. The matrix multiplication can be rewritten as the following:

$$\boldsymbol{C} = \boldsymbol{AB} = \begin{bmatrix} \boldsymbol{A_{1,1}} & \cdots & \boldsymbol{A_{1,M}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A_{M,1}} & \cdots & \boldsymbol{A_{M,M}} \end{bmatrix} \begin{bmatrix} \boldsymbol{B_{1,1}} & \cdots & \boldsymbol{B_{1,M}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{B_{M,1}} & \cdots & \boldsymbol{B_{M,M}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C_{1,1}} & \cdots & \boldsymbol{C_{1,M}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{C_{M,1}} & \cdots & \boldsymbol{C_{M,M}} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{M} \boldsymbol{A}_{1,i}\boldsymbol{B}_{i,1} & \cdots & \sum_{i=1}^{M} \boldsymbol{A}_{1,i}\boldsymbol{B}_{i,M} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{M} \boldsymbol{A}_{M,i}\boldsymbol{B}_{i,1} & \cdots & \sum_{i=1}^{M} \boldsymbol{A}_{M,i}\boldsymbol{B}_{i,M} \end{bmatrix}$$

Based on such decomposition method, we designed the parallel matrix multiplication

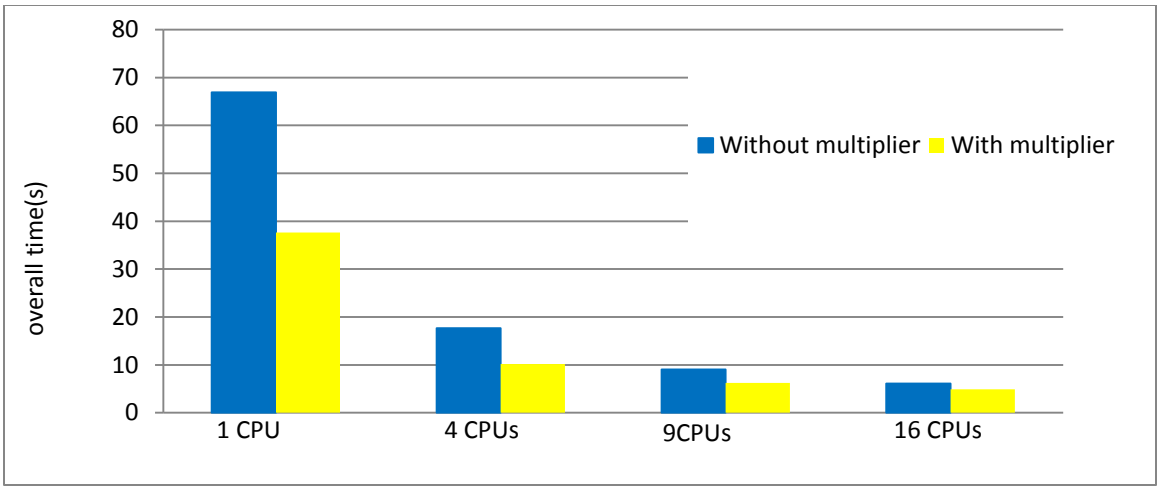


Figure 4-9. Overall task execution time for processors with and without embedded multipliers

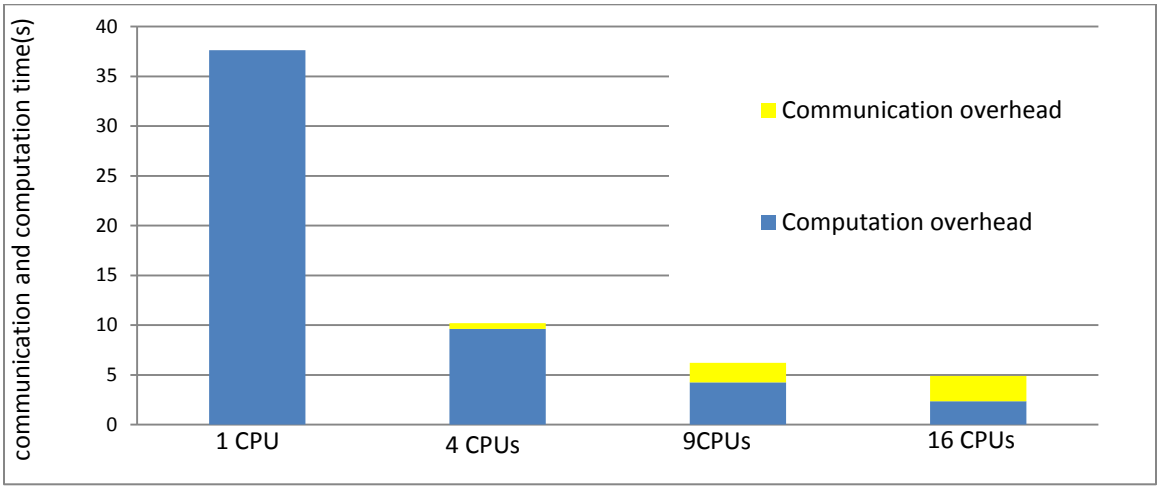


Figure 4-10. Computation and communication time for processors with embedded multipliers

program. The matrix size N is 100 in our experiment and M is set to 1, 2, 3, and 4. The matrix is initially stored in K_1P_1. After establishing the TCP/IP socket, K_1P_1 sends the sub- matrices $A_{x,1\sim M}$ and $B_{1\sim M,y}$, $1 \le x,y \le M$, to processors either on the same FPGA or on a different FPGA. Upon receiving the data, each processor then performs the calculation $\boldsymbol{C}_{x,y} = \sum_{i=1}^{M} \boldsymbol{A}_{x,i}\boldsymbol{B}_{i,y}$ and sends

the result matrix $\boldsymbol{C}_{x,y}$ back to K_1P_1. Based on the algorithm, the computation is distributed onto M^2 processors.

Figure 4-9 shows the overall task execution time of those 4 scenarios where M varies from 1 to 4 and the number of CPUs involved in the computing varies from 1 to 16. The execution time for processors with or without hardware multipliers are both presented. Figure 4-10 shows the communication and computation time break down for processors with embedded multipliers. As we can see, with the processor number increasing, the overall task execution time decreases almost linearly. The embedded multipliers can significantly reduce the execution time because the computation accounts for the major part of the execution time. And when we increase the number of processors involved in the computation, the communication time increases and becomes comparable to the computation time since more data need to be transferred while every processor performs less computation.

4.5.3 Evaluation of the temperature sensor and clock selection module

In the third experiment, we dynamically switches the processor's working frequency from 1MHz to 10MHz, 50 MHz , and 100MHz and evaluate the thermal impact of the frequency scaling. During the entire experiment, the processor is executing the floating point matrix multiplication program discussed in previous subsection. A separate process is created which reads the temperature sensor every 2 minutes. The processor stays at each frequency level for 10 minutes. Therefore, 5 temperature data are collected for each clock frequency levels.

When all frequency levels have been tested, the processor stays idle at 100MHz for 10 minutes and go back to execute the matrix multiplication program at 1MHz. And the previous procedure repeats.

Figure 4-11 shows the trace of temperature changes in the above mentioned procedure. As we can see, the reading of the temperature sensor increases when the processor is running at a higher frequency. And at the same time, when processor is idle, thought the working frequency doesn't change, the temperature goes down as we expected. However, because the Nios II subsystem is not clock gated, even though no instruction is executed in the processor, the clock is still toggling and there is still switching activities. Therefore, the temperature of the idling 100 MHz processor is still higher than the temperature of an active 10 MHz processor.

In average, the temperature sensor reading is 8670 when the CPU is running at 1MHz and 8730 when the CPU is running at 100 MHz. As in Figure 4-6, we use 150 logic elements to build the delay line and the circulation times is 4096. Based on this and according to [102], 1 unit difference in sensor reading corresponds to about 0.03°C temperature difference. Therefore, running the CPU at 100 MHz and 1MHz generates about 1.8°C temperature difference. The temperature change is much smaller than that can be generated in a general purpose CPU because the Nios II processor is a soft core processor and hence has larger area and lower power density.

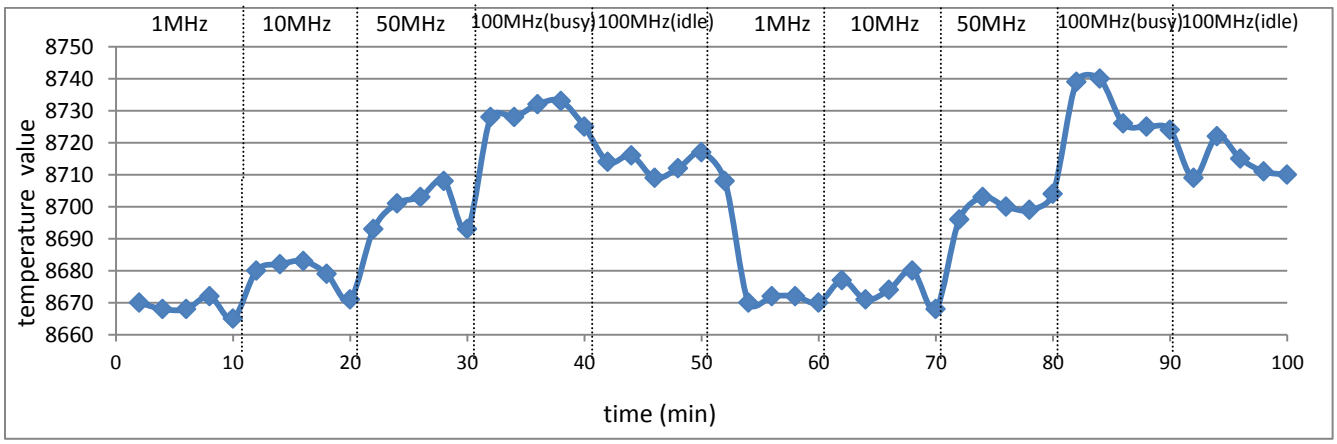


Figure 4-11 .Processor's temperature change under different working frequency

4.6 Conclusion

In this chapter, we proposed an FPGA-based distributed computing platform using Altera Nios II soft-core processors. The system consists of multiple FPGAs with each FPGA configured as a multi-core system. Each core has its own temperature sensor and has the dynamic frequency scaling capability. The platform is designed to support research in dynamic power/thermal management. Our experiment verifies the correct functionality of the distributed computing platform and evaluated the communication latency between cores and between FPGAs. It shows almost linear performance improvements as the number of cores increases.

Chapter 5 Conclusions

In this dissertation, we have studied several learning based dynamic power management techniques where the learning agent monitors the system dynamic and taking appropriate power management actions intelligently. We targeted on different platforms, i.e., (1)personal computer and server; (2)mobile devices and (3)FPGA.

We firstly presented two works targeting system level power management in PC and server environment. Our first work created a model that is used to dynamic quantify task performance degradation with the respect to a reference system, where the target process is executed stand alone at the highest frequency. The proposed model is used to provide performance feedback to guide DVFS control. The model is further improved to predict the performance of the target process under a new task mapping. The improved model is used to provide performance prediction to guide the task migration. Experimental results show that the proposed models effectively controls the system performance and keeps it close to the given constraint, hence leads to lower power consumption with minimum performance violation. The second work proposed a general model solving the dynamic power management problem using Q-learning. The Q-learning power manager does not require any prior knowledge of the workload or the system model while it can learn the policy online with real-time incoming tasks and adjusts the policy accordingly. The Q-learning algorithm is firstly applied to the HDD power management by putting HDD into sleep and waking it up. Simulation results show that can achieve good power performance tradeoff. Then it is also extended for the CPU power

management by controlling its DVFS settings. The control algorithm is capable to achieve minimum energy while meeting the user constraints in performance and temperature.

In the second part, for mobile devices environment, we also presented two works targeting system level power management in the context rich mobile environment. The first work aimed at increasing the QoS of mobile devices considering the fact that many users recharge the battery before depletion. We present a stochastic framework for QoS boosting under the user specified battery depletion tolerance. The model is trained using real user traces and the framework is simulated and compared with existing approaches. The second work used GMM to model different user's YouTube playback time and then applied Inventory Theory to find out the optimal buffering points during the video playback that minimizes energy waste on the cellular interface. Our evaluation based on real field study and simulation demonstrates that our approach can save about 10% more energy than the best static buffering method.

In last part of the dissertation, we proposed an FPGA-based distributed computing platform using Altera Nios II soft-core processors. The system consists of multiple FPGAs with each FPGA configured as a multi-core system. Each core has its own temperature sensor and has the dynamic frequency scaling capability. The platform is designed to support research in dynamic power/thermal management. Our experiment verifies the correct functionality of the distributed computing platform and evaluated the communication latency between cores and between FPGAs. It shows almost linear performance improvements as the number of cores increases.

Bibliography

References

[1] T. Abdelzaher, Y. Diao, J.L. Hellerstein, C. Lu, And X. Zhu, "Introduction to Control Theory and Its Application to Computing Systems," *Sigmetrics Tutorial*, Annapolis, Md, 2008.

[2] Y. Agarwal, S. Savage, and R. Gupta, "Sleepserver: A Software-Only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments," In *Proceedings Of The USENIX 2010 Annual Technical Conference* (*USENIX ATC'10*), pp. 22-22, 2010.

[3] I. Ahmad, S. Ranka, and S.U. Kham, " Using Game Theory for Scheduling Tasks On Multi-Core Processor for Simultaneous Optimization Of Performance and Energy," In *Proceedings Of IEEE International Symposium On Parallel And Distributed Processing*, 2008, pp.1-6, 2008.

[4] R. Ayoub and T. Rosing, "Predict and Act: Dynamic Thermal Management for Multi-Core Processors," In *Proceedings Of The 14th ACM/IEEE International Symposium On Low Power Electronics And Design* (*ISLPED*'09), pp. 99-104, 2009.

[5] R. Ayoub, K. Indukuri, and T.S. Rosing, "Temperature Aware Dynamic Workload Scheduling in Multisocket CPU Servers," In *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems* (*TCAD'11*), vol. 30, issue 9, pp. 1359-1372, 2011.

[6] R. Ayoub, R. Nath, and T. Rosing, "Jetc: Joint Energy Thermal And Cooling Management For Memory And CPU Subsystems In Servers," In *Proceedings Of IEEE 18th International Symposium On High Performance Computer Architecture* (*HPCA'12*) ,pp. 1-12, 2012.

[7] L. Cai, N. Pettis, And Y. Lu, "Joint Power Management Of Memory And Disk Under Performance Constraints," *IEEE Trans. On Computer-Aided Design Of Integrated Circuits And Systems,* vol. 25, pp. 2697-2711, 2006.

[8] R. Caruana and D. Freitag, "Greedy Attribute Selection," In *Proceedings Of The Eleventh International Conference On Machine Learning*, 1994.

[9] K. Choi, R. Soma and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off based on the ration of Off-Chip Access to On-Chip Computation Times," *TCAD'2004,* vol.24,pp.18-28, 2004.

[10] C.L. Chou and R. Marculescu, "Designing Heterogeneous Embedded Network-On-Chip Platforms With Users in Mind," *IEEE Trans. On Computer-Aided Design Of Integrated Circuits And Systems*, vol.29, pp.1301-1314, 2001.

[11] P. Choudhary and D. Marculescu, "Power Management Of Voltage/Frequency Island-Based System Using Hardware-Based Methods," *TVLSI*, vol.17, issue3, pp.427-438, 2009.

[12] A.K. Coskun, T.S. Rosing and K. Whisnant, "Temperature Aware Task Scheduling In MPSOCs," In *Proceedings Of Design Automation And Test In Europe* (*DATE'07*) , pp. 1659-1664, 2007.

[13] A.K. Coskun, T.S. Rosing, K. Whisnant, and K. Gross, "Temperature-Aware MPSOC Scheduling for Reducing Hot Spots and Gradients," In *Proceedings Of The 2008 Asia And South Pacific Design Automation Conference* (*ASPDAC*'08) , pp. 49-54, 2008

[14] A. Coskun, T.S. Rosing and K.G. Gross, "Temperature Management In Multiprocessor SOCs Using Online Learning," In *Proceedings Of The 45th Annual Design Automation Conference* (*DAC'08*), pp. 890-893, 2008

[15] A. Coskun, T.S. Rosing and K.C. Gross, "Utilizing Predictors For Efficient Thermal Management In Multiprocessor SOCs," In *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems* (*TCAD'09*), vol. 28, issue 10, pp. 1503-1516, 2009.

[16] G. Dhiman and T.S. Rosing, "System-Level Power Management Using Online Learning," *IEEE Trans. On Computer-Aided Design Of Integrated Circuits And Systems*, vol.28, pp.676-689, 2009.

[17] Enhanced Intel Speedstep® Technology - How To Document. Accessed Dec 06, 2014. http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm

[18] Y. Fei, S. Ravi, A. Raghunnathan and NK.Jha, "Energy-Optimizing Source Code Transformation For Operating System-Driven Embedded Software," *ACM TECS*, vol.7, 2009.

[19] Y. Fei, L. Zhong and NK. Jha, "An Energy-Aware Framework For Dynamic Software Management In Mobile Computing Systems," *ACM TECS*, vol.7, 2008.

[20] C. Gniady, A.R. Butt. Y. Charlie amd Y. Lu, "Program Counter-Based Prediction Techniques for Dynamic Power Management," *IEEE Transactions On Computers*, vol. 55, pp. 641-658, 2006.

[21] C-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *ACM TODAES*, vol.5, pp.226-241, 2000.

[22] Intel® Core™2 Duo Processor E8000 And E7000 Series. Accessed Dec 06, 2014. http://download.intel.com/design/processor/datashts/318732.pdf

[23] E. Ipek, O. Mutlu, J.F. Martinez and R. Caruana, "Self-Optimizing Memory Controllers : A Reinforcement Learning Approach*,"* In *Proceedings Of The 35th Annual International Symposium On Computer Architecture*,(*ISCA'08*), pp. 39-50, 2008.

[24] ITRS. Accessed Dec 06, 2014. http://www.itrs.net/

[25] R. Jejurikar, C. Pereira and R. Gupta, "Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems," In *Proceedings Of The 41st Annual Design Automation Conference* (*DAC'04*), pp. 275-280, 2004.

[26] A. Kansal, J. Hsu, S. Zahedi and M.B. Srivastava, "Power Management In Energy Harvesting Sensor Networks," *ACM TECS*, vol.6, iss.4, 2007.

[27] P. Langen and B. Juurlink, "Leakage-Aware Multiprocessor Scheduling For Low Power. In *Proceeding Of International Parallel And Distributed Processing Symposium* (*IPDPS'06*,) pp. 60, 2006.

[28] Y. Liu, H. Yang, R.P. Dick, H. Wang and L. Shang, "Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems. In *Proceedings Of 8th International*

Symposium On Quality Electronic Design (*ISQED*'07), pp. 204-209, 2007

[29] W. Liu. Y. Tan and Q. Qiu, "Enhanced Q-Learning Algorithm For Dynamic Power Management With Performance Constraint," In *Proceedings Of The Conference On Design, Automation And Test In Europe* (*DATE'10*), pp. 602-605, 2010

[30] J.F Martinez and E. Ipek, "Dynamic Multicore Resource Management :A Machine Learning Approach," *IEEE Micro* Vol.29, pp8-17, 2009

[31] Mediabench. Accessed Dec 06, 2014. http://euler.slu.edu/~fritts/mediabench/

[32] Mibench. Accessed Dec 06, 2014. http://www.eecs.umich.edu/mibench/

[33] Omnet++. Accessed Dec 06, 2014. http://www.omnetpp.org/ .

[34] Open Source Software at tesla.hpl.hp.com. Accessed Dec 06, 2014. http://tesla.hpl.hp.com/opensource/ .

[35] M. Pendrith, "On Reinforcement Learning of Control Actions in Noisy and Nonmarkovian Domains. Technical Report Nsw-Cse-Tr-9410," School Of Computer Science And Engineering, The University Of New South Wales, Sydney, Australia, 1994.

[36] Perfmon2. Accessed Dec 06, 2014. http://perfmon2.sourceforge.net/

[37] N. Pettis and Y. Lu, "A Homogeneous Architecture for Power Policy Integration in Operating Systems," *IEEE Trans. On Computers,* vol.58, pp.945-955, 2009

[38] Q. Qiu, Y. Tan and Q. Wu, "Stochastic Modeling And Optimization For Robust Power Management In A Partially Observable System," In *Proceedings Of The Conference On Design, Automation And Test In Europe* (*DATE'07*), pp. 779-784, 2007

[39] T.S Rosing, L. Benini, P. Glynn and G. De Micheli, "Event-Driven Power Management," *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, vol. 20, pp. 840-857, 2001.

[40] C. Ruemmler and J. Wilkes, "Unix Disk Access Patterns," *USENIX Winter* , pp.405-420, Jan25-29, 1993.

[41] K. Sikorski and T. Balch, "Model-Based And Model-Free Learning in Markovian and Non-Markovian Environments, " *Proc. Of Agents-2001 Workshop On Learning Agents*, 2001

[42] C.W. Smullen, J. Coffman and S. Gurumurthi, "Accelerating Enterprise Solid-State Disks With Non-Volatile Merge Caching," In *Proceedings Of International Green Computing Conference* (*IGCC*'10), pp. 203-214, 2010

[43] Y. Tan, W. Liu and Q. Qiu, "Adaptive Power Management Using Reinforcement Learning," In *Proceedings Of The 2009 International Conference On Computer-Aided Design* (*ICCAD'09*), pp. 461-467, 2009

[44] Y. Tan and Q. Qiu, "A Framework Of Stochastic Power Management Using Hidden Markov Model," In *Proceedings Of The Conference On Design, Automation And Test In Europe* (*DATE'08*), pp. 92-97, 2008

[45] Y. Tan, P. Malani, Q. Qiu and Q. Wu, "Workload Prediction and Dynamic Voltage Scaling for MPEG Decoding," In *Proceedings Of The 2006 Asia And South Pacific Design Automation* (*ASP-DAC '06*), pp. 911-916, 2006

[46] G. Tesauro, R. Das, N. Jong and M. Bennani, "A Hybrid Reinforcement Learning Approach To Autonomic Resource Allocation," In *Proceedings Of 3rd IEEE International Conference On Autonomic Computing* (*ICAC*'06), pp. 65-73, 2006

[47] G. Tesauro, R. Das, H. Chan, J. Kephart, D. Levine, F. Rawson and C. Lefurgy, " Managing Power Consumption And Performance Of Computing Systems Using Reinforcement Learning," In *Proceedings Of The 21st Annual Conference On Neural Information Processing Systems* (*NIPS*'07), 2007

[48] G. Theocharous, S. Mannor, N. Shah, P. Gandhi, B. Kveton, B. Siddiqi and C-H. Yu, "Machine Learning For Adaptive Power Management," *Intel Technology Journal*, vol.10, pp.299-312, 2006.

[49] TOSHIBA Hard Disk Drive Specification 1.8 inch Hard Disk DriveMK6006GAH/MK4006GAH/MK3006GAL.

[50] G.V. Varatkar and R. Marculescu, "On-Chip Traffic Modeling and Synthesis For MPEG-2 Video Applications," *IEEE Trans. On Very Large Scale Integration Systems*, vol.12, no.1, 2004.

[51] Y. Wang, K. Ma and X. Wang, "Temperature-Constrained Power Control For Chip Multiprocessors With Online Model Estimation," *In Proceedings Of The 36th Annual International Symposium On Computer Architecture, (ISCA'09)*, pp. 314-324, 2009.

[52] C. Weddle, M. Oldham, J. Qian and A.A. Wang, "PARAID: A Gear-Shifting Power-Aware RAID," *ACM TOS*, vol.3, no.13, 2007.

[53] I. Yeo, C. Liu and E. Kim, "Predictive Dynamic Thermal Management For Multicore Systems," *In Proceedings Of The 45th Annual Design Automation Conference (DAC'08),* pp. 734-739, 2008

[54] F. Zanini, D. Atienza, L. Benini and G. De Micheli, "Multicore Thermal Management With Model Predictive Control," In *Proceedings Of The 19th European Conference On Circuit Theory And Design* (*ECCTD*'09) , pp. 90-95, 2009

[55] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No 'Power' Struggles: Coordinated Multi-level Power Management for the Data Center," *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, March, 2008.

[56] L. Barroso and U. Holzle, "The case for Energy-proportional Computing," *Computer,* vol. 40, Issue 12, pp. 33-37, 2007.

[57] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud and J. Pei, "A Practical Method for Estimating Performance Degradation on Multicore processors, and its Application to HPC Workloads," *SC'12*, Article No.83, 2012

[58] S. Zhuravlev, S. Blagodurov and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," *ASPLOS XV*, pp.129-142, 2010

[59] J. R. Funston, K. E. Maghraoui, J. Jann, P. Pattnaik and A. Fedorova, "An SMT-Selection Metric to Improve Multithreaded Applications' Performance," *IPDPS'12*, pp.1388-1399, 2012

[60] M. E. Thomadakis, "The architecture of the Nehalem processor and Nehalem-EP SMP platforms," Texas A&M University, Tech.Rep.,2011

[61] K. K. Pusukuri, D. Vengerov, A. Fedorova and V. Kalogeraki, "FACT: a Framework for Adaptive Contention-aware Thread Migrations," *CF'11*, Article No.35, 2011

[62] G. Dhiman, G. Marchetti and T. Rosing, "vGreen: A System for Energy-Efficient Management of Virtual Machines," *ACM TODAES*, vol.16, iss.1, Nov.2010

[63] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a Simultaneous Multithreading Processor," *ASPLOS IX*, pp.234-244, 2000

[64] K. Deng, K. Ren and J. Song, "Symbiotic Scheduling for Virtual Machines on SMT Processors," *CGC'12*, pp.145-152, 2012

[65] R. Knauerhase, P. Brett, B. Hohlt, T. Li and S. Hahn, "Using OS Observations to Improve Performance in Multicore Systems," *IEEE Micro*, vol.28, iss.3, May.2008

[66] H.Shen, Y.Tan, J.Lu ,Q.Wu and Q.Qiu, "Achieving autonomous power management using reinforcement learning," *TODAES 2013*, vol.18, iss.2, no.24, Mar.2013

[67] H.Shen, J.Lu and Q.Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," *ISQED 2012*, pp.19-21, Mar.2012

[68] A. Merkel, J. Stoess and F. Bellosa, "Resource-conscious Scheduling for Energy Efficiency on Multicore Processors," *EuroSys'10*, pp.153-166, 2010

[69] D.Gaurav, V.Kontorinis, D.Tullsen, T.Rosing, E.Saxe and J.Chew, " Dynamic workload characterization for power efficient scheduling on CMP systems," *ISLPED'10*, pp.437-442, Aug.2010

[70] C.Bae, L.Xia, P.Dinda and J.Lange, "Dynamic adaptive virtual core mapping to improve power, energy, and performance in Multi-socket multicores," *HDPC'12*, pp.247-258, 2012

[71] R.Nathuji, A.Kansal and A.Ghaffarkhah, "Q-Clouds: Managing performance interference effects for QoS-Aware clouds," *EuroSys'10*, pp.237-250, 2010

[72] J.L.Kihm and D.A.Connors, "Implementation of fine-grained cache monitoring for improved SMT scheduling," *Computer Design: VLSI in Computers and Processors, 2004*, pp.326-331, Oct.2004

[73] S.Blagodurov, D.Gmach, M.Arlitt, Y.Chen, C.Hyser and A.Fedorova, "Maximizing server utilization while meeting critical SLAs via weight-based collocation management," *IM 2013*, pp.277-285, May.2013

[74] A. Phansalkar, A. Joshi and L. K. John, "Subsetting the SPEC CPU2006 Benchmark Suite," *ACM SIGARCH*, vol.35, iss.1, Mar.2007

[75] A.Settle, J.Kihm and A.Janiszewski, "Architectural Support for Enhanced SMT Job Scheduling," *PACT 2004*, pp.63-73, Sep.2004

[76] Y.Jiang, X.Shen, J.Chen and R.Tripathi, "Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors," *PACT'08*, pp.220-229, 2008

[77] J. D. Gelas, "Dynamic Power Management: A Quantitative Approach," *AnandTech*, Jan.2010. Accessed Dec 06, 2014. http://www.anandtech.com/show/2919

[78] Perf tool . Accessed Dec 06, 2014. *https://perf.wiki.kernel.org/index.php/Main_Page*

[79] Weka 3: Data Mining Software in Java. Accessed Dec 06, 2014. http://www.cs.waikato.ac.nz/ml/weka/index.html

[80] SPEC CPU2006. Accessed Dec 06, 2014. *http://www.spec.org/cpu2006/*

[81] lp_solve. Accessed Dec 06, 2014. *http://lpsolve.sourceforge.net*

[82] Linux cgroups. Accessed Dec 06, 2014. https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[83] T.Dorta, J.Jimenez, J.L.Martin, U.Bidarte, and A.Astarloa, "Overview of FPGA-Based Multiprocessor Systems", in *Reconfig'09*, pp.273-278,Dec.2009

[84] D.Atienza, P.G.D Valle, G.Paci, F.Poletti, L.Benini, G.De Micheli, and J.M.Mendias, "A Fast HW/SW FPGA-Based Thermal Emulation Framework for Multi-Processor System-on-Chip,", *DAC'06,* pp.618-623, Jul.2006.

[85] N.Fujii and N.Koike, "Work in Progress-Developmemt of a New Multi-CPU Parallel and Distributed Processing Experiment Platform for Remote hardware laboratory," in *Frontiers in Education Conference, 2009. FIE '09. 39th IEEE* ,pp.1-2, Oct.2009

[86] M.Manzke and R.Brennan, "Extending FPGA based Teaching Boards into the area of Distributed Memory Multiprocessors," in *Proc. WCAE '04*, Article No.5, 2004

[87] L.Yan and etc., "Performance Evaluation of the Memory Hierarchy Design on CMP Prototype Using FPGA,"in *ASICON'09*, pp.813-816, Oct.2009

[88] A.Tumeo,C.Pilato,G.Palermo,F.Ferrandi and D.Sciuto, "HW/SW Methodologies for

Synchronization in FPGA Multiprocessors," in *Proc.FPGA'09* , pp.265-268, 2009

[89] J.J.Martinez-Alvarez and etc., "A Multi-FPGA Distributed Embedded System for the Emulation of Multi-Layer CNNs in Read Time Video Applications," in *CNAA'2010*, pp.1-5, Feb.2010

[90] A.Kulmala, O.Lehtoranta, T.D.Hamalainen and M.Hannikainen, "Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC," in *RASIP Journal on Embedded Systems*, vol.2006, issue 1 , Jan.2006

[91] A.Tumeo and etc., "Prototyping Pipelined Applications on Heterogeneous FPGA Multiprocessor Virtual Platform," in *ASP-DAC'2009,*pp.317-322, Jan.2009

[92] L.Shang, R.P.Dick and N.K.Jha, "SLOPES: Hardware-Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems with Dynamically Reconfigurable FPGAs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol.26,pp.508-526, Mar.2007

[93] A.Bhattacharjee, G.Contreras and M.Martonosi, "Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation," in *Proc.ISLPED'08,* pp.335-340,2008

[94] Micrium, "uC/OS-II Kernel". Accessed Dec 06, 2014. http://www.micrium.com/page/products/rtos/os-ii .

[95] Altera Corporation , "Mailbox Core," Quartus II Handbook Version 9.1 Volume 5, Nov.2009

[96] Altera Corporation , "Mutex Core," Quartus II Handbook Version 9.1 Volume 5, Nov.2009

[97] Altera Corporation , "Clock Multiplexing," Quartus II Handbook Version 10.1 Volume 1, Dec.2010

[98] Altera Corporation, Phase-Locked Loop Reconfiguration (ALTPLL_RECONFIG) Megafunction User Guide, Aug.2010

[99] Altera Corporation, Phase-Locked Loop(ALTPLL) Megafunction User Guide, Nov.2009

[100] Altera Corporation, Clock Control Block (ALTCLKCTRL) Megafunction User Guide, Sep.2010

[101] Altera Corporation , **"**Avalon Memory-Mapped Bridges," Quartus II 10.0 Handbook, Volume 4 **,** Dec.2010

[102] P.Chen, M.C.Shie, Z.Y.Zheng, Z.F.Zheng, C.Y.Chu, "A Fully Digital Time-Domain Smart Temperature Sensor Realized With 140 FPGA Logic Elements," in *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol.54,pp.2661-2668, Dec.2007

[103] L.Benini, A.Bogliolo and G.De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans on VLSI*, vol.8, issue3,pp.299-316,2000

[104] Y.Ge, P.Malani and Qinru Qiu, "Distributed task migration for thermal management in many-core systems," *DAC2010*, pp.579-584, Jun.2010

[105] Altera Corporation , SOPC Builder User Guide , Dec.2010

[106] F.Ding, F.Xia, W.Zhang, X.Zhao and C.Ma, "Monitoring Energy Consumption of Smartphones," *iThings/CPSCom'11*, pp.610-613, Oct.2011.

[107] L.Zhang, B.Tiwana, Z.Qian, Z.Wang, R.P.Dick, Z.M.Mao and L.Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," *CODES/ISSS '10*, pp.105-114, 2010.

[108] N.Ding, A.Pathak, D.Koutsonikolas, C.Shepard, Y.C.Hu and L.Zhong, "Realizing the Full Potential of PSM using Proxying," *INFOCOM 2012*, pp.2821-2825, Mar.2012.

[109] A.Finamore, M.Mellia, M.M.Munafo, R.Torres and S.G.Rao, "Youtube Everywhere: Impact of Device and Infrastructure Synergies on User Experience," *IMC'11,* pp.345-360, 2011.

[110] F.Qian, Z.Wang, A.Gerber, Z.M.Mao, S.Sen and O.Spatscheck, "Characterizing Radio Resource Allocation for 3G Networks," *IMC'10*, pp.137-150, 2010.

[111] A.Schulman, V.Navda, R.Ramjee, N.Spring, P.Deshpande, C.Grunewald, K.Jain and V.N.Padmanabhan, "Bartendr: A Practical Approach to Energy-Aware Cellular Date Scheduling," *MobiCom'10*, pp.85-96, 2010

[112] Monsoon Solutions Inc. Accessed Dec 06, 2014. http://www.msoon.com/LabEquipment/PowerMonitor/ .

[113] Mixture model. Accessed Dec 06, 2014. http://en.wikipedia.org/wiki/Mixture_model

[114] E. Porteus, "Foundations of stochastic inventory theory," *Stanford Business Books,* Aug,2002.

[115] M. Sullivan, "3G/4G Performance Map: Data Speeds for AT&T, Sprint, T-Mobile, and Verizon," *PCWorld*, 2012. Accessed Dec 06, 2014.

http://www.pcworld.com/article/254888/3g4g_performance_map_data_speeds_for_atandt_sprint_tmobile_and_verizon.html

[116] Y. Tan and Q. Qiu, "A Framework of Stochastic Power Management Using Hidden Markov Model," *DATE'08*, pp.92-97, 2008

[117] D. Bello and G. Riano, "Linear Programming solvers for Markov Decision Processes," *SIEDS'2006*, pp.90-95, Apr 2006

[118] C. Shepard, A. Rahmati, C. Tossell, L. Zhong and P. Kortum, "Livelab: Measuring Wireless Networks and Smartphone User in the Field," *ACM SIGMETRICS Perfrom. Eval. Rev.*, vol.38, no.3. Dec 2010

[119] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins and L. Zhong, "Users and Batteries: Interactions and Adaptive Energy Management in Mobile Systems," *UbiComp'07*, pp.217-234, 2007

[120] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," *SOSP'99*, pp.48-63, 1999

[121] N. Ravi, J. Scott, L. Han and L. Iftode, "Context-aware Battery Management for Mobile Phones," *PerCom 2008*, pp.224-233, Mar 2008

[122] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan and D. Estrin, "Diversity in Smartphone Usage," *MobiSys'10*, pp.179-194, 2010

[123] Y. S. Lee and S. B. Cho, "An Efficient Energy Management System for Android Phone Using Bayesian Networks," *ICDCSW'2012*, pp.102-107, Jun2012

[124] E. A. Oliver and S. Keshav, "An Empirical Approach to Smartphone Energy Level Prediction," *UbiComp'11*, pp.345-354, 2011

[125] T. Yan, D. Chu, D. Ganesan, A. Kansal and J. Liu, "Fast App Launching for Mobile Devices Using Predictive User Context," *MobiSys'12*, pp.113-126, 2012

[126] A. Rahmati, A. Qian and L. Zhong, "Understanding Human-Battery Interaction on Mobile Phones," *MobileHCI'07*, pp.265-272, 2007

[127] H. Shen, Y. Tan, J. Lu, Q. Wu and Q. Qiu, "Achieving Autonomous Power Management Using Reinforcement Learning," *TODAES*, vol.18, issue.2, 2013

[128] Wiki "Markov Decision Process ". Accessed Dec 06, 2014. http://en.wikipedia.org/wiki/Markov_decision_process

[129] Jiawei Han and Micheline Kamber, "Data Mining Concepts and Techniques," Second Edition, Morgan Kaufmann Publishers, March 2006

[130] H.Shen and Q.Qiu, "User-Aware Energy Efficient Streaming Strategy for Smartphone Based Video Playback Applications", *DATE'*13, pp.258-261, 2013

[131] A.Rahmati and L.Zhong, "Context-Based Network Estimation for Energy-Efficient Ubiquitous Wireless Connectivity*", IEEE Transactions on Mobile Computing*, vol.10, iss.1, pp.54-66, Jan 2011

[132] Apple Working On Location-Aware Battery Management For iPhone. Accessed Dec 06, 2014. http://techcrunch.com/2013/07/25/apple-working-on-location-aware-battery-management-for-iphone/

[133] Taker. Accessed Dec 06, 2014. http://tasker.dinglisch.net/index.html

[134] N.Eagle and A.Pentland, "Reality mining: sensing complex social systems," *Journal of Personal and Ubiquitous Computing*, vol.10, iss.4, pp.255-268, May 2006

[135] M.Pedram, "Energy Efficient Enterprise Computing and Green Datacenters," *Workshop on IT and Future Society*, South Korea, Oct.2010.

[136] K. Sekar, "Power and Thermal Challenges in Mobile Devices" Broadcom Corp, Oct, 2013. Accessed Dec 06, 2014. http://www.sigmobile.org/mobicom/2013/MobiCom2013_IndustrialTalks_KrishnaSekar.pdf

Vitae

Hao Shen (S’14) receives a B.S. degree in Electrical Engineering from Southeast University, Nanjing, China, in 2008 and M.S. degree in Electrical and Computer Engineering from Binghamton University, NY, USA in 2011. His primary research interest is system level power management. He joins Marvell Semiconductor after graduation as firmware engineer.