7-12-2012

Practical Dynamic Thermal Management on Intel Desktop Computer

Guanglei Liu
Florida International University, guanglei1025@gmail.com

DOI: 10.25148/etd.FI12080613

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

PRACTICAL DYNAMIC THERMAL MANAGEMENT ON INTEL DESKTOP COMPUTER

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Guanglei Liu

2012

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Guanglei Liu, and entitled Practical Dynamic Thermal Management on Intel Desktop Computer, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Kang K. Yen

Jean H. Andrian

Deng Pan

Nezih Pala

Gang Quan, Major Professor

Date of Defense: July 12, 2012

The dissertation of Guanglei Liu is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida Internatinal University, 2012

DEDICATION

I would like to dedicate this Doctoral dissertation to my parents and my beloved wife. Without their love, understanding, support, and encouragement, the completion of this endeavor would never have been possible.

ACKNOWLEDGMENTS

I wish to express my deepest appreciation to my major professor, Dr. Gang Quan, for his guidance, encouragement, support, and patience. His passion, dedication and sincere interests in scientific research have been a great inspiration to me.

I would also like to thank my committee members, Dr. Kang K Yen, Dr. Jean H. Andrian, Dr. Deng Pan and Dr. Nezih Pala, for their very helpful insights, comments and suggestions to improve the quality of this dissertation.

Finally, I would like to thank the National Science Foundation (NSF) for supporting the research described in this dissertation through grants CNS-0969013, CNS-0917021 and CNS-1018108.

ABSTRACT OF THE DISSERTATION

PRACTICAL DYNAMIC THERMAL MANAGEMENT ON INTEL DESKTOP COMPUTER

Guanglei Liu

Florida International University, 2012

Miami, Florida

Professor Gang Quan, Major Professor

Fueled by increasing human appetite for high computing performance, semiconductor technology has now marched into the deep sub-micron era. As transistor size keeps shrinking, more and more transistors are integrated into a single chip. This has increased tremendously the power consumption and heat generation of IC chips. The rapidly growing heat dissipation greatly increases the packaging/cooling costs, and adversely affects the performance and reliability of a computing system. In addition, it also reduces the processor's life span and may even crash the entire computing system. Therefore, dynamic thermal management (DTM) is becoming a critical problem in modern computer system design.

Extensive theoretical research has been conducted to study the DTM problem. However, most of them are based on theoretically idealized assumptions or simplified models. While these models and assumptions help to greatly simplify a complex problem and make it theoretically manageable, practical computer systems and applications must deal with many practical factors and details beyond these models or assumptions.

The goal of our research was to develop a test platform that can be used to validate theoretical results on DTM under well-controlled conditions, to identify the limitations of existing theoretical results, and also to develop new and practical DTM techniques. This dissertation details the background and our research efforts in this endeavor. Specifically, in our research, we first developed a customized test platform based on an Intel desktop. We then tested a number of related theoretical works and examined their limitations under the practical hardware environment. With these limitations in mind, we developed a new reactive thermal management algorithm for single-core computing systems to optimize the throughput under a peak temperature constraint. We further extended our research to a multicore platform and developed an effective proactive DTM technique for throughput maximization on multicore processor based on task migration and dynamic voltage frequency scaling technique. The significance of our research lies in the fact that our research complements the current extensive theoretical research in dealing with increasingly critical thermal problems and enabling the continuous evolution of high performance computing systems.

TABLE OF CONTENTS

CHAPTER PAGE

1 INTRODUCTION 1
1.1 The Thermal Design Challenges 1
1.2 Research Problem and Our Contributions 5

2 BACKGROUND AND RELATED WORK 10
2.1 Power Consumption and Power Management 10
2.1.1 Source of Power Consumption 10
2.1.2 Power Reduction Techniques 14
2.2 Power and Thermal Correlation 16
2.3 Thermal Management 18
2.3.1 Mechanical Cooling Technologies 19
2.3.2 Dynamic Thermal Management Techniques 22
2.3.3 Thermal-Aware Scheduling 24

3 THE INTEL BASED PRACTICAL HARDWARE PLATFORM 27
3.1 Related Work 27
3.2 The Intel i5 Based Hardware Platform 28
3.3 Dynamic Thermal Management Techniques 30
3.3.1 DVFS Technique 30
3.3.2 Task Allocation and Migration Technique 33
3.3.3 Computing Cooling System Configuration 34
3.4 Power and Temperature Measurement 35
3.4.1 Power Measurement 35
3.4.2 Temperature Capturing 37
3.4.3 Computing System Monitoring Tool 37

4 VALIDATING THERMAL MANAGEMENT PRINCIPLES ON PRACTICAL HARDWARE PLATFORM 39
4.1 Related Work 39
4.2 Preliminary 41
4.3 System Implementation 44
4.4 Experimental Results 46
4.4.1 Air-Cooling VS Thermal Management Technique 47
4.4.2 DPM VS DTM Technique 49
4.4.3 Scheduling Algorithm Evaluation 51
4.5 Summary 57

5 A PRACTICAL ON-LINE DYNAMIC THERMAL MANAGEMENT ALGORITHM ON SINGLE CORE MICROPROCESSOR 58
5.1 Related Work 58
5.2 Practical Hardware Limitations Analysis 59
5.3 An Enhanced Dynamic Thermal Management Algorithm 63
5.3.1 Non-Constant Sampling Interval 63
5.3.2 Offline Thermal Analysis for Safe Speed 66
5.4 Experimental Results 68
5.4.1 Experiment Setup 68
5.4.2 DTM Algorithm Performance Evaluation 69
5.4.3 DTM algorithm Throughput Analysis 71
5.5 Summary 74

6 NEIGHBOR-AWARE DYNAMIC THERMAL MANAGEMENT FOR MULTI-CORE PLATFORMS 75
6.1 Related Work 75
6.2 Preliminary 78
6.2.1 Motivation Example 78
6.2.2 Problem Description 80
6.3 Neighbor-Aware Temperature Prediction 81
6.3.1 Temperature Prediction Model 81
6.3.2 Neighbor-Different Prediction 85
6.3.3 Neighbor-Normalized Prediction 87
6.4 Proactive Algorithm 89
6.4.1 Candidate Processor for Migration 90
6.4.2 Thermal Management Algorithm 92
6.5 Experimental Results 93
6.5.1 Experiment Setup 93
6.5.2 Prediction Analysis 94
6.5.3 Throughput Analysis 95
6.6 Summary 98

7 CONCLUSIONS 99
7.1 Concluding Remarks 99
7.2 Future Work 101

REFERENCES 102

VITA 114

LIST OF FIGURES

FIGURE PAGE

Figure 1.1 The trend of increasing number of cores in CMPs 2

Figure 1.2 System on chip (Soc) system power consumption trend . . . 3

Figure 2.1 CMOS switching mode for static power consumption 11

Figure 2.2 Three types of leakage current in CMOS 13

Figure 2.3 System-level thermal model for single-core processor 17

Figure 2.4 Architecture-level thermal model for single-core processor . . 19

Figure 2.5 3D-IC with micro-channel cooling system 21

Figure 3.1 Our test bed based on an Intel Desktop computer 30

Figure 3.2 High-level Cpufreq module infrastructure overview 32

Figure 3.3 Conventional air-cooling with heatsink 34

Figure 3.4 Overview of the system interconnection 36

Figure 3.5 Lm-sensor monitor tool overflow 38

Figure 4.1 Different thermal and power management algorithms 42

Figure 4.2 (a) Cool down processor temperature with traditional *Air-cooling* solution. (b) Cool down processor temperature by using the *DVFS* technique. 48

Figure 4.3 The DTM algorithm cooling efficiency comparison 50

Figure 4.4 The temperature profile of running RunStop schedule with different cooling conditions 52

Figure 4.5 The temperature profile with the TwoSpeed schedule under different cooling conditions 54

Figure 4.6 Thermal aware schedules comparison at different fans peed . 56

Figure 5.1 The conventional dynamic thermal management algorithm (CDTM). 61

Figure 5.2 An example of temperature trace. 63

Figure 5.3 The definition of *safe region* and *buffer zone* 64

Figure 5.4 The number of temperature violations after implementing different thermal management algorithms 70

Figure 5.5 Massive temperature violations occur with the CDTM approach 72

Figure 5.6 Temperature violations has been significantly reduced with the VS-DTM technique 72

Figure 5.7 ERDTM algorithm completely eliminates peak temperature violation 73

Figure 5.8 Throughput comparison with different thermal management algorithms 74

Figure 6.1 Temperature trace with Hot and Cool neighbor processors . . 79

Figure 6.2 Temperature history based prediction 82

Figure 6.3 Different processor location scenarios 84

Figure 6.4 NADTM compares with the conventional prediction method, which does not take the neighbor effect into consideration . . 93

Figure 6.5 Prediction accuracy comparison with different benchmarks . 95

Figure 6.6 Execution time comparison with four different approaches. *NP and CP represent the neighbor-aware and conventional prediction respectively. NM and CM represent neighbor-aware and conventional migration respectively* 97

LIST OF TABLES

TABLE PAGE

Table 2.1 Duality between thermal and electrical quantities 16

Table 3.1 Dell Precision T1500 Technical Specifications 29

Table 3.2 Intel Core i5-750 Processor Specifications 31

Table 3.3 Intel i5 available frequency levels (GHz) 33

Table 4.1 Execution times for different cooling methods 49

Table 4.2 Peak temperatures under different fan speeds 53

Table 4.3 Different speed group 55

Table 4.4 TwoSpeed schedule under different Cooling condition 55

Table 4.5 Temperature comparison under different fan speed 55

Table 5.1 Lookup table for SPEC 2000 benchmarks 69

CHAPTER 1
INTRODUCTION

As IC technology scaling continues, the total number of transistors integrated into an IC chip has doubled every two years to accommodate the increasing demand of higher computation capability. However, this increase in transistor density has made the power consumption increases exponentially, and also substantially elevated the heat generation. The rapidly growing heat dissipation has greatly increased system costs, adversely affected the performance and reliability and can even cause fatal hardware damage or crash the entire system, if the processor temperature is not properly managed. It is fair to say that this thermal issue has become one of the critical challenges in the design of computing system. In this chapter, we first introduce the motivation of our research problems, and then we summarize the major contributions of this dissertation.

1.1 The Thermal Design Challenges

As the technology scaling, hundreds of millions of transistors have been integrated into the processor chip. It is estimated that nearly 40 billion transistors are integrated into a single die today [81] and the number is growing very fast. It is expected that the number of transistors can reach 150 billion by 2015 [12]. With the dramatically increasing number of transistors, more complicated computer architectures, such as multi-core architectures, are designed to continuously deliver higher and higher peak computing performance. As shown in Figure 1.1[1], more and more computing cores are integrated into a single chip to deliver high performance. For example, the single-chip cloud computer (SCC) experimental processor with 48-core is developed by Intel lab as a platform for many-core software research [97]. And an 80-core prototype has

[1]Figure 1.1 from Intel Microprocessor Technology Lab, 2011

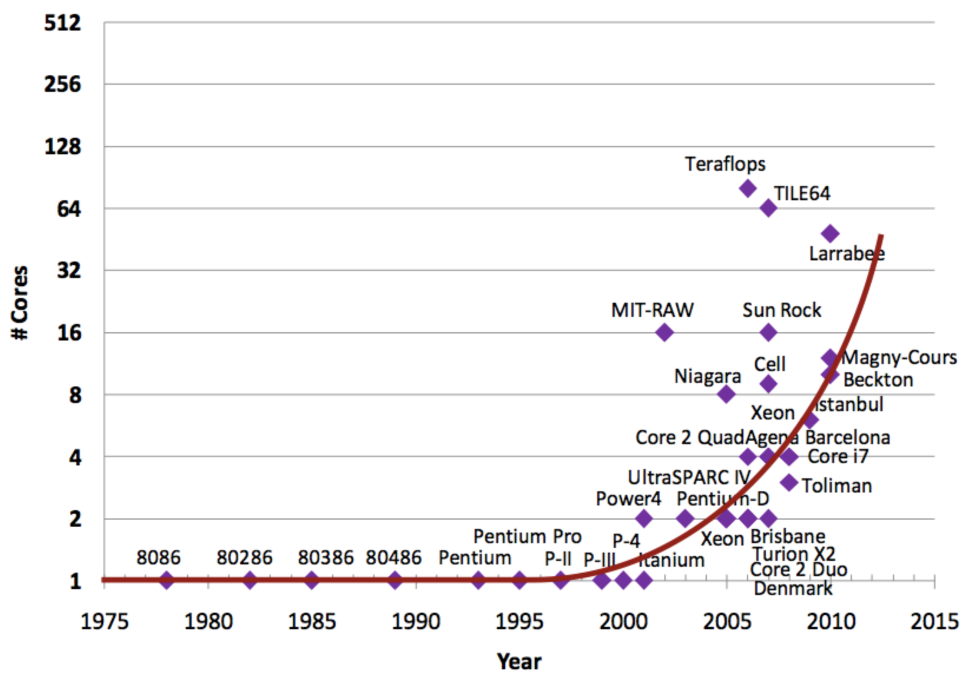


Figure 1.1: The trend of increasing number of cores in CMPs

recently been demonstrated by Intel [114]. In addition, some supercomputing centers have been developed [34], such as the IBMs BlueGene project, which integrates up to 65,536 computation nodes [86]. The multicore architecture can significantly improve the computing system performance. Valentini et al. [113] has shown that two smaller processor cores, instead of one large monolithic processor core, can provide 70-80% more performance.

At the same time, however, the increasing number of transistors has caused the power consumption of a chip to increase exponentially. As an example, Figure 1.2[2] illustrates the power consumption trend of the system on chip (Soc) computing system between the year of 2005 to 2020. Even with the projected technology advances in material, circuit design, etc, the power consumption is expected to increase by nearly 10 times from 2005 to 2020. The high power density causes thermal hotspots in the system, and substantially increases the packaging and cooling costs. It is

[2]Figure 1.2 from ITRS, 2005.

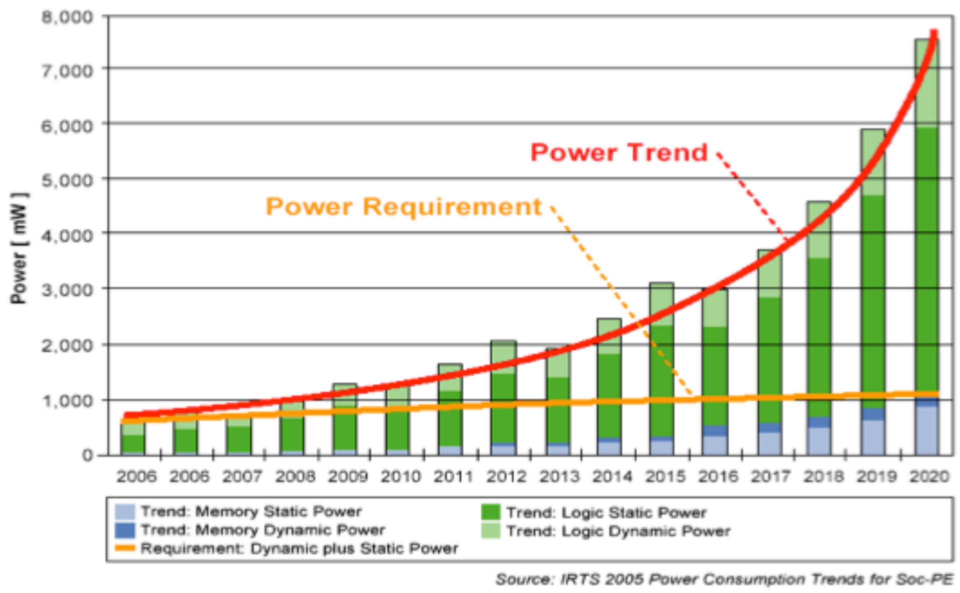


Figure 1.2: System on chip (Soc) system power consumption trend

estimated that the thermal packaging increases the total packaging cost at 1-3 dollar per watt [103, 51]. On the other hand, for the high-performance computing system, the cooling cost of a 30,000 square feet data center with 1000 standard computing racks is around 4-8 million dollar each year [5, 10, 13, 103]. This cooling cost has already became an economic problem, which significantly affected the development of computing systems.

Moreover, when the computing system temperature increases, it also negatively impacts the reliability, and degrades system performance. It is reported more than 50% of all integrated circuit failures are related to the thermal issues [88]. Even a small temperature change of 5^oC can cause a $2\times$ difference in the mean time to failure of the devices [116]. According to Yeh et al [130], every 10^oC increase in operating temperature can cut a devices life span by half. From Santarini et al. [94], every 15^oC increase in temperature can lead to as much as 10-15% in circuit delay.

In addition, high temperature also increases leakage power consumption. The leakage power refers to the power consumption caused by the leakage current flowing through the transistor channel when the transistor is turned off. It increases rapidly as transistor size reduces, and is becoming comparable if not exceeds the dynamic power consumption in the IC circuit. As shown in Figure 1.2, the leakage power consumption is estimated to increase around 5 times from the year of 2005 to 2020. It is reported that for the 65nm technology, the leakage power consumption can be 2-3 times higher than the dynamic power consumption [65]. To make things worse, leakage power increases with the temperature. Based on the research proposed by Liao et al. [66], increasing temperature from 65^oC to 110^oC can increase the leakage power by 38% for IC circuits. The increased power consumption generates more heat and elevates the temperature. At the same time, high temperature will increase the leakage power consumption and thus the overall power consumption. Evidently, high temperature is becoming more and more critical in the design of computing systems.

Traditionally, designers deal with the thermal issues by using the air-cooling method, which consists of a heat sink and cooling fan. However, as we discussed before, the cooling cost can be extremely high for the modern high computation computing system. In addition, the air-cooling method cools down the entire surface of IC chip. Thus, it cannot effectively eliminate the thermal hotspots. Because of the limited thermal conductance of air, it becomes impossible to satisfy the cooling requirement for cutting-edge computing systems such as the 3D-ICs. Therefore, developing an effective cooling solution becomes an interesting research topic from both industry and academy. For example, new cooling techniques such as the micro-channel with liquid cooling method have been developed recently. In addition, the dynamic thermal management (DTM) techniques, which control the processor temperature by dynamically adjusting the computing performance, have also been proposed. Based on the DTM techniques, numerous researches have been done to address the thermal issues. Many of them are focusing on the thermal-aware throughput maximization problem. Since the modern computing systems are integrated with a self-protection mechanism, the computing system can automatically shut down itself to avoid the permanent physical damages cased by the thermal violation [93]. In this case, to ensure the system operates can deliver the maximal performance; the processor temperature must be carefully maintained under the thermal limitation.

1.2 Research Problem and Our Contributions

Extensive theoretical researches have been conducted on dynamic thermal management techniques in recent years. For example, significant research efforts have been made on thermal-aware performance maximization problem [18, 134, 136], peak temperature minimization [71, 72, 112], real-time guarantee under peak temperature constraints [118, 47, 20] and overall energy reduction under peak temperature constraints [9, 44, 8]. These researches differ by their system models, architecture types, and design constraints and objectives. Most of these works are based on some simplified mathematical models and idealized assumptions. By simplifying a complex theoretical problem, these theoretical work can be extremely useful in uncovering the fundamental principles in DTM. However, real computing systems and applications must deal with practical factors and conditions beyond the theoretical models. For example, many researches employ the lumped first order RC thermal model to model temperature dynamics [56, 92, 104]. However, to get accurate values for a practical processor platform can be challenging [70]. Also, some approaches assume that the exact knowledge of the actual temperature is always available, which may not always be possible [128, 95]. Other assumptions such as ambient temperature [20, 44, 19] remaining constant indefinitely, which is not always true as evidenced in [108]. Furthermore, it has been a common practice to evaluate the theoretical results based on software simulation, where the theoretical results can be evaluated in a software environment closer to the real world. Though some properties of a computing system or application can be derived analytically and parameterized in the software environment, it is highly desirable that these results be verified using practical hardware platforms and applications, since the analytically derived specifications of the systems can be very different from those in the real application scenarios [68].

The goal of our research is to study the dynamic thermal management problem from the perspective of a practical hardware platform. Specifically, we intend to develop a practical and customized hardware test platform that enables us to investigate theoretical DTM techniques under well controlled conditions, to study the limitations of existing theoretical algorithms and to develop new and practical DTM techniques that can accommodate factors and conditions existing in practical systems and applications. Compared with the related work, we made the following contributions in this dissertation:

1. We developed a customized, flexible, and practical hardware platform based on an Intel i5-750 quad-core processor, running the Ubuntu 10.04.1 Linux operating system with kernel version 2.6.32. It has the flexibility to adopt most of the advance thermal management techniques, such as dynamic voltage frequency scaling (DVFS), clock gating, idle period injection and task migration. Thus, many theoretical works can be implemented and validated on our platform. Besides, we are able to directly read the temperature of each individual core from the on-chip thermal sensors. The temperature of entire chip can be obtained from thermal circuits as well. In addition, our platform adopted the power measurement technique proposed by a related research work [48]. Thus, the power consumption of the microprocessor can be measured at runtime. In order to get accurate experimental results, we were able to adjust the cooling system to guarantee that all the experiments were carried out with the same

cooling constant. Moreover, the well-known benchmark, SPEC CPU2000, has been used to provide reliable and comparable experimental results. Lastly, the hardware monitor tool, *Lm-sensor*, which can collect both thermal and power information, has been used to provide a user interface. Thus, the thermal and power management designers are able to analyze their research based on the practical system information.

2. With our hardware test platform, we first studied how effective DTM techniques can be in a practical hardware environment. We investigated the cooling potential of the DTM technique and its computing performance tradeoffs on our hardware platform, compared it with the traditional air-cooling method (e.g. the cooling fan and heat sink). We found that the DTM technique can outperform the traditional air-cooling method, because it can control the temperature in a wider range. Next, after we verified several theoretical works based on our platform, we proved the two widely used thermal management principles still validate in reality. Furthermore, by comparing the thermal and power management algorithms on our platform, we found that even though the power management techniques can effectively reduce the power consumption and further bring down the overall temperature, the thermal management techniques, which consider special thermal characteristics, must be developed to achieve better temperature management performance. We also analyzed how the cooling solution can affect the performance of the thermal management algorithm by comparing those approaches under different cooling conditions.

3. We identified several limitations in the assumptions of existing theoretical researches on our hardware platform. Based on our findings, we proposed an *Enhanced reactive dynamic thermal management* (ERDTM) algorithm for single-core processor to maximize the program throughput under a given temperature

constraint. Comparing with the conventional reactive approach, our ERDTM schedule takes the practical thermal sensor hardware characteristic into consideration, and implement a non-constant sampling technique to improve the temperature detection accuracy. In addition, by doing offline thermal analysis, we are able to built up a thermal profiling look-up table, which can guide the ERDTM algorithm selecting the optimal working frequency to further improve the throughput. Our experimental results show that the ERDTM algorithm can significantly reduce the number of temperature violation by 88%. Also, the overall system throughput can be improved by 8.1% compared with the conventional reactive approach.

4. We extended our research work from single-core to multicore platform and proposed a proactive *Neighbor-aware dynamic thermal management* (NADTM) algorithm with temperature prediction technique. Compared to the reactive DTM approaches, the proactive DTM algorithm can detect the thermal emergency in advance, and leave enough time for the DTM algorithm to react to temperature changes. Other than simply using the temperature history to predict temperature, our NADTM algorithm takes the heat transfer from neighboring processor into consideration. It can significantly improve the prediction accuracy by 38% compared to the temperature history based prediction method. And the prediction error is as small as 1^oC. In addition, we showed that simply migrating a task from the hottest core to the coolest core is not always the optimal solution. Instead, our NADTM algorithm incorporates the neighboring temperature information for task migration and thus can achieve better performance. Our experimental results show that our NADTM can effectively maintain the temperature under the pre-defined temperature limit. The overall system throughput can be improved by as much as 5.8%.

The rest of the dissertation is organized as follows. In Chapter 2, we first introduce the background of this work, and then discuss the related works in thermal and power management. In Chapter 3, we discuss the practical hardware platform. With this platform, we validated several existing theoretical works in Chapter 4. After we analyzed the limitations of the theoretical work on the practical platform, we developed a reactive EADTM algorithm for the single-core platform described in Chapter 5. Then, a proactive thermal-aware throughput maximization algorithm NADTM for the multicore platform is proposed in Chapter 6. Finally, we conclude this dissertation and discuss future works in Chapter 7.

CHAPTER 2
BACKGROUND AND RELATED WORK

The research background is presented in this chapter. We first introduce the sources of the power consumption of IC chips and some common power reduction techniques. We then discuss the power and thermal relationship and thermal management techniques with an emphasis on the thermal-aware scheduling techniques.

2.1 Power Consumption and Power Management

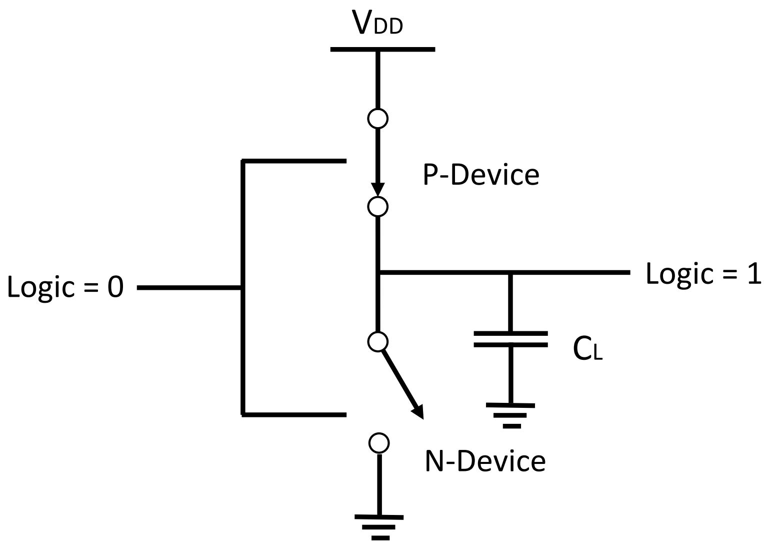
Thermal phenomena is closely related to the power consumption of IC chips. In this section, we briefly discuss the sources of power consumption of the IC circuits and techniques for power managements.

2.1.1 Source of Power Consumption

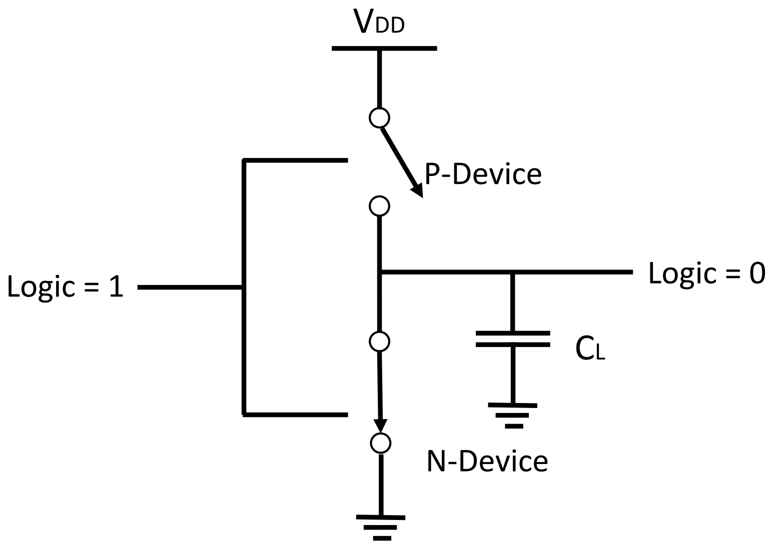
The power consumption of an IC chip consists of two types of power consumptions, i.e. the *dynamic power consumption* and the *static power consumption* [43]. The *dynamic power consumption* is caused by charging and discharging of load capacitance; and the *static power consumption* is caused by leakage current. Traditionally, the dynamic power has been the major sources of the power consumption in a CMOS circuit. However, as the semiconductor technology enters the deep submicron domain, leakage power consumption increases rapidly and is comparable or even surpass the dynamic power consumption in an IC circuit. In what follows, we discuss each of them respectively.

Dynamic Power Consumption

The dynamic power consumption, P_{dyna}, also known as the capacitive-load power consumption, is consumed by charging and discharging external load capacitance [2,



(a) Charging mode



(b) Discharing mode

Figure 2.1: CMOS switching mode for static power consumption

91, 89]. Figure 2.1 shows the different modes of transistors when charging and discharging a CMOS circuit. The capacitive-load power consumption can be formulated as

$$P_{dyna} = C_L \times V_{dd}^2 \times f \times N_{sw}, \tag{2.1}$$

where f is the clock frequency, C_L is the external load capacitance, N_{sw} is the number of bits switching and V_{dd} is the supply voltage. Since the working frequency is the proportional to the supply voltage level, we can further formulate the dynamic power consumption as

$$P_{dyna} \propto C_L v_{dd}^3 \tag{2.2}$$

from equation (2.1), we can see the dynamic power consumption is highly related with the supply voltage, working frequency, the complexity of the logic gate and the switching activity.

Static Power Consumption

In contrast, the static power is caused by the leakage current. Figure 2.2 shows how a transistor controls the flow of the current between *source* and *drain*. In the ideal case, when the transistor is at the "*OFF*" state, there is no current flowing through the drain and the source node. This is because there is an insulating material called *channel* placed in between. When the voltage at the gate node increases high enough to reach the threshold voltage, $V_{threshold}$, the conductivity of the channel increases as well. It turns the transistors to the "*ON*" state, and allows the current to flow between the drain and the gate. However, in reality, although the voltage at the gate does not reach the threshold $V_{threshold}$, there is still leakage current flowing through the transistor.

As shown in Figure 2.2, the leakage current can be classified into three categories

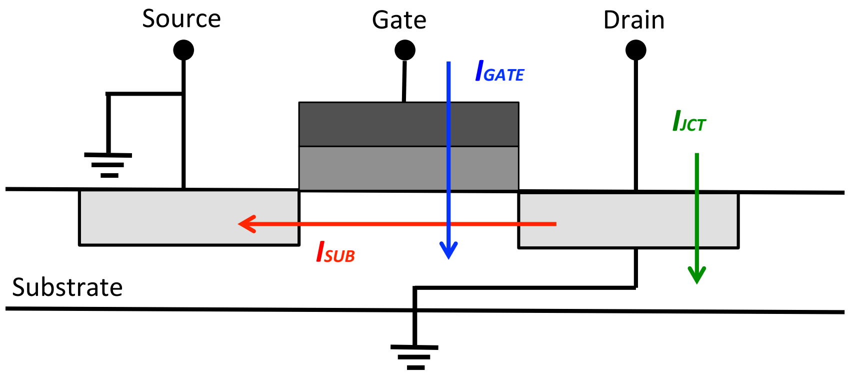


Figure 2.2: Three types of leakage current in CMOS

based on the path of current flow [89]. The first one is the *junction leakage current* (i.e. I_{JCT}), which is the current of the source or the drain that goes through the reverse-biased diode to the substrate, when the transistor is in off state. The junction leakage current heavily depends on the physical characteristics of the transistor. Next, the *gate direct tunneling leakage current* (i.e. I_{GATE}) flows from the gate terminal through the dielectric material to the substrate. It can be affected by the thickness of the gate oxide material. The last one is the *sub-threshold leakage current* (i.e. I_{SUB}), which is caused by the diffusion current of the minority carriers in the channel of a transistor. It flows directly from the drain to the source even when a transistor is in off state. It depends on the temperature of the chip, threshold voltage, supply voltage, as well as some other process dependent technology constants. Among those three leakage current powers, the sub-threshold leakage is more critical than the others [51]. Thus, the leakage current can be formulated by a non-linear exponential equation as [65]

$$I_{leak} = I_s \cdot (\mathcal{A} \cdot T^2 \cdot e^{((\alpha \cdot V_k + \beta)/T)} + \mathcal{B} \cdot e^{(\gamma \cdot V_k + \delta)}), \tag{2.3}$$

where I_s is the leakage current at certain reference temperature and supply voltage, T is the operating temperature, $\mathcal{A}, \mathcal{B}, \alpha, \beta, \gamma, \delta$ are empirically determined technology constants.

In addition, the leakage power can be formulated as $P_{leak} = N_{gate} \cdot I_{leak} \cdot v_k$, where N_{gate} represents the number of gate and v_k is the kth supply voltage level. Furthermore, as reported in [74], the leakage current changes super linearly with temperature. The leakage and temperature dependency can be estimated by using the linear approximation method. Thus, the leakage power for the processor running in mode k can be effectively estimated as [19]

$$P_{leak}(k) = C_0(k)v_k + C_1(k)Tv_k, \tag{2.4}$$

where $C_0(k)$ and $C_1(k)$ are constants. Equation (2.4) illustrated the close correlation between temperature and leakage power consumption. We will give a detailed discussion about the leakage temperature dependency in later sections.

2.1.2 Power Reduction Techniques

There have been extensive researches conducted on power/energy consumption reduction at different design abstraction levels, from logical level, circuit level, architecture level, and all the way to the system levels [57, 82, 91, 115, 117, 80, 76].

As we discussed before, the dynamic power consumption depends on the transistor size, wire lengths and transistor switching activity. Therefore, the fundamental principles to reduce the dynamic energy consumption are to shrink the transistor size and also to reduce the switching activity when the transistor is idle. For example, at the logic level, the *logic gate restructuring* technique is developed to arrange logic gates and their input signals to reduce switching activities; at the architecture-level, the *clock gating* technique is a straightforward approach to reduce the circuits switching activity. It simply disables the clock signals of the processor components, which are not currently active; at the circuit-level, the designer implements the *Transistor sizing* technique that tries to adjust the size of the transistor so that the power consumption can be minimized without significant performance lost. *Transistor sizing* is another effective technique, which rearranges the order of transistors to minimize their switching activities. This approach places the most frequently switched transistor closer to a circuit output, thus the unnecessary switching activities of other transistors can be reduced; at the system-level, the *dynamic voltage scaling* (DVS) technique is one of the widely used methods to minimize the dynamic power. Because of the convex correlation between power and supply voltage, the power consumption can be reduced to one fourth of the original value, if the supply voltage is reduced to half.

As transistors become smaller and smaller and leakage becomes more and more prominent, many techniques have also been developed to reduce the leakage power consumption during the different states. For example, *body bias control* is a technique to reduce the leakage during the idle state. This approach can bias the source terminal of an "off" transistor, so that the leakage current can be reduced. Specifically, it applies a positive bias voltage during the standby state to the source terminal, so the threshold voltage is raised and the transistor is turned off more strongly. When the logic circuit is in the standby state, the *sleep transistor* technique is developed to reduce the leakage power. With this method, all functional units are implemented with low V_t transistors, which are fast but leakage prone. On the other hand, a high V_t sleep transistor is inserted between functional units and the ground. Thus, in the active state, the sleep transistor is turned on to allow normal operation. When functional units are in the standby state, the sleep transistor is turned off to cut off the leakage current path to the ground.

The power reduction techniques described above can effectively reduce the overall system power consumption. In the next section, we will explore the power and thermal correlation.

2.2 Power and Thermal Correlation

Power consumption and heat dissipation have a very close correlation because the high power consumption can bring up the on-chip temperature and generates a great amount of heat. In return, high temperature can increase the leakage power that increases the total power consumption as well. In this section, we discuss the relationship between power and thermal.

Table 2.1: Duality between thermal and electrical quantities

Thermal Quantity	Electrical Quantity
Power consumption: P (W)	Current flow: I (A)
Temperature: T (^oC)	Voltage: V (V)
Thermal resistance: R $(^oC/W)$	Electrical resistance: R (Ω)
Thermal capacitance: C $(J/^oC)$	Electrical capacitance: C (F)

Based on the duality between the heat transfer and the electrical current flow as shown in Table 2.1, this critical characteristic is captured by using the lumped RC model. Therefore, for the system-level thermal analysis, many researches formulate the correlation between the power and temperature by using a mathematical equation (i.e. equation 2.5) to analyze how they interact with each other [46, 44, 19, 125, 17, 23, 85, 16, 23, 75, 90]. Specifically, as shown in Figure 2.3, $T(t)$ and T_{amb} are the chip temperature and ambient temperature respectively. $P(t)$ denotes the power consumption (in $Watt$) at time t, and R, C are the thermal resistance (in $^oC/Watt$) and thermal capacitance (in $J/^oC$) respectively. Then we have

$$RC\frac{dT(t)}{dt} = RP(t) + (T(t) - T_{amb}). \tag{2.5}$$

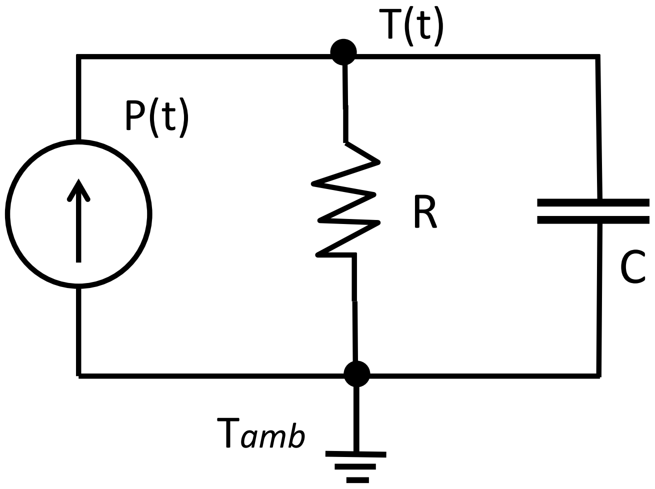


Figure 2.3: System-level thermal model for single-core processor

We can scale T such that T_{amb} is zero and then we have

$$\frac{dT(t)}{dt} = aP(t) - bT(t), \tag{2.6}$$

where $a = 1/C$ and $b = 1/RC$.

The overall power consumption of the processor is composed of two parts: the dynamic power P_{dyn} and leakage power P_{leak}. The dynamic power consumption is independent of the temperature and can be formulated as $P_{dyn} = C_2 v_k^3$ in equation (2.2), and the leakage power consumption can be formulated as $P_{leak}(k) = C_0(k)v_k + C_1(k)Tv_k$ in equation (2.4). Thus, the total power consumption of processor is formulated as

$$P(k) = C_0(k)v_k + C_1(k) \cdot Tv_k + C_2 v_k^3. \tag{2.7}$$

Accordingly, the temperature dynamics at the speed level k can be formulated as

$$\frac{dT(t)}{dt} = A(k) - B(k)T(t), \tag{2.8}$$

where $A(k) = a(C_0(k)v_k + C_2 v_k^3)$ and $B(k) = b - aC_1(k)v_k$. Hence, for a given time interval $[t_0, t_e]$, if the initial temperature is T_0, by solving equation (2.8), the ending temperature can be formulated as below:

$$\begin{aligned} T_e &= \frac{A(k)}{B(k)} + (T_0 - \frac{A(k)}{B(k)})e^{-B(k)(t_e - t_0)} \\ &= G(k) + (T_0 - G(k))e^{-B(k)(t_e - t_0)}. \end{aligned} \tag{2.9}$$

Many thermal and power simulation tools have been developed by exploiting the duality of thermal characteristics and electrical circuit. *HotSpot* is one of the most widely use simulation tool to conduct the architectural-level thermal analysis [104]. The HotSpot simulator works as follow. Once the floorplan of the functional units, as well as the physical properties of the processor are given (i.e. chip dimension and material), the HotSpot can automatically generate a three-dimensional RC network to model the heat transfer within the RC network. From the RC network as shown in Figure 2.4[1], a differential equation can be implemented to do thermal analysis. In addition, R and C represent the thermal resistance and the thermal conductance respectively. Current sources represent the power consumption of a given functional unit. The node voltages are the expected temperature value of the corresponding components. When processor is running tasks, the current of the RC network will change accordingly. Thus, by solving the differential equations, the corresponding temperature variation can be calculated. However, the architecture-level thermal model must be simple enough to allow architects to simulate thermal effects.

2.3 Thermal Management

Traditionally, thermal issues in computer systems are dealt with using the mechanical cooling methods such as heat sink, cooling fan, etc. As the thermal problem became

[1]Figure 2.4 from research paper [104].

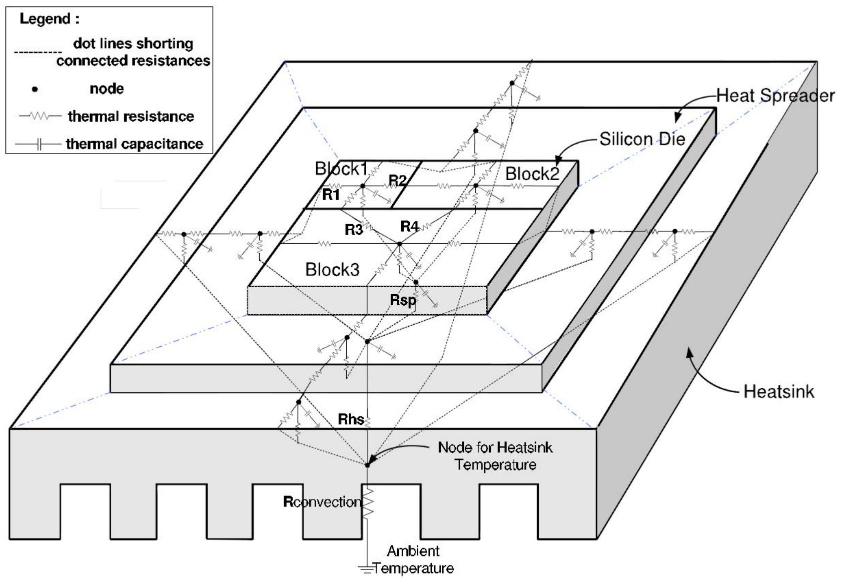


Figure 2.4: Architecture-level thermal model for single-core processor

more and more critical, new cooling methods such as the liquid cooling were developed. At the same time, DTM has drawn more and more research attentions. In this section, we introduce the traditional mechanical thermal management methods and dynamic thermal management techniques.

2.3.1 Mechanical Cooling Technologies

Since the development of the first electronic computers, the cooling system has played a key role in accommodating power increments and maintaining temperature at a satisfactory level to satisfy performance and reliability objectives. For the past 30 years, a number of mechanical cooling technologies have been developed to provide temperature control in computing systems. These techniques intend to effectively dissipate the heat generated by the system components to the ambient environment. In this section, we introduce some of the primary mechanical cooling technologies.

Heat Sink and Cooling Fan

The heat sinks and cooling fans are commonly used technology for cooling IC chips in computer systems. They are designed based on the concept of controlling the airflow rate between the heat sink and the ambient, so that the computing system can operate under the desired temperature. The heat sinks are made of metal with high thermal conductivity, and are directly connected with the small processor chip. They allow heat to transfer to its large surface area by conduction. Then, the computer fans remove the heat from the heat sink to the ambient environment. The thermal performance of the heat sink depends on many parameters, such as the thickness, surface area, and thermal conductivity. It is reported that the thermal conductivity of aluminum ranges from 150-200 W/mK, and the thermal conductivity of copper ranges from 350-390 W/mK [26]. Once the heat sink is manufactured, its cooling efficiency is fixed. Thus, the air-cooling performance is adjusted by changing the fan speed.

Usually in today's desktop computers, the fan speed is controlled by a closed loop controller, which regularly collects the temperature information from CPU thermal sensors, and adjusts fan speed accordingly to accommodate the corresponding temperature change. As we discussed before, the power cost for cooling down computing systems is high. It is reported that the air-cooling system in server station can consume as much as 80W in 1U rank servers and 240W or more in 2U rack servers [87]. The cooling power consumption can reach up to 51% of the overall server power budget [64]. Moreover, the increasing fan speed can introduce system noise, which not only leads to an uncomfortable working environment, but also causes vibrations and impacts the reliability. It is estimated that the acoustic noise levels increase by 10dB as the fan speed increases by 50% [78].

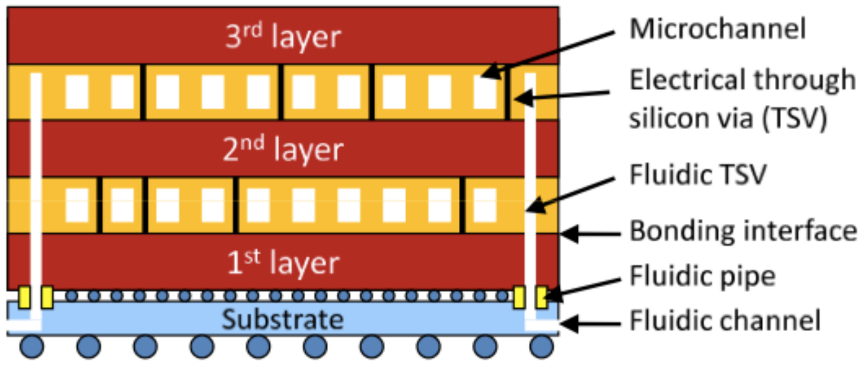


Figure 2.5: 3D-IC with micro-channel cooling system

Liquid-cooling

The *Liquid-cooling* technique is developed for the situation when air-cooling is not able to satisfy the cooling requirements. A good example of this cooling solution is the IBM 3081 processor that uses liquid to remove the heat generated from his processing units. Compared to the air-cooling method, the liquid-cooling solution has a higher thermal conductivity. In addition, because of the higher density of liquid, it can absorb more heat, which is about 3500 times more than air [26]. The cooling efficiency of the liquid-cooling depends on a lot of variables, such as liquid chemistry, flow rate, temperature, and pressure. It has been reported that the peak temperature can be reduced from 85^oC to 57^oC on the liquid-cooled 2D processor, while the maximum temperature variation is also reduced from 25^oC to 6^oC [26].

Micro-channel cooling system

The *micro-channel cooling system* is an extension of the liquid-cooling to further improve the cooling capability. The concept was originally developed by Tuckerman and Pease [111]. They etched $50\mu m$ wide and $300\mu m$ deep micro-channel into a $1cm \times 1cm$ processor chip. By directing water through those micro-channels, they can remove 790W with a temperature difference of 71^oC. Moreover, this technique has been extended to three-dimensional Integrated circuit (3D-ICs) designs.

One of the practical works from IBM demonstrated the feasibility of integrated micro-channel 3D ICs [96]. As shown in Figure 2.5[2], the micro-channels are distributed at each layer. Then, the cooling fluid is pushed to each micro-channel from the fluidic channel [81]. Since the 3D-ICs techniques already became the next generation computing system design objective, extensive research has been done on the 3D-ICs with micro-channel cooling. Hitoshi et al. [81] proposed a fast and accurate thermal-wake aware thermal model for integrated micro-channel 3D ICs. Sridhar et al. [105] presented a transient 3D thermal model with multiple inter-tier micro-channel to analyze the 3D-IC cooling problem. Wang et al. [120] developed a transient thermal simulator for temperature estimation in a 3D environment. Xu et al. [124] proposed a methodology to obtain accurate thermal profile from a closed-form thermal model with complex interconnect structures.

2.3.2 Dynamic Thermal Management Techniques

Since the mechanical cooling techniques consume tremendous energy, and sometimes the traditional cooling techniques such as the air-cooling method are not able to satisfy the cooling demand, the *dynamic thermal management* DTM techniques have emerged to address the thermal issue. Compared to the mechanical cooling techniques, the DTM techniques are more straightforward. They can directly reduce the temperature of the overheated area, either by reducing the frequency or migrating the task to a cooler processor. Therefore, numerous researches have been done based on the DTM techniques [14, 67, 39, 14, 104]. In this section, we introduce some of the most widely used DTM techniques.

[2]Figure 2.5 from research paper [81].

Dynamic Voltage and Frequency Scaling

The *dynamic voltage and frequency scaling* DVFS technique can dynamically adjust the working frequency and the supply voltage based on certain requirements (e.g. DVFS can be adjusted based on the workload or temperature threshold). It can be used either to conserve power consumption or to reduce the heat generated by the processor. With the DVFS technique, the demand for conventional cooling system can be eased, and the cooling cost can also be reduced. However, the DVFS technique reduces the temperature by sacrificing the computing system performance. Based on the DVFS technique, a lot of researches have been published [31, 58, 60]. They either tried to analyze the trade off between the performance and power and thermal dissipation, or focused on optimizing the thermal issue by reducing processor peak temperature. It has already become one of the most widely used thermal management techniques, which has been integrated on the modern computing system (e.g. Intel CPU throttling technology, SpeedStep and AMD's Cool'n'Quiet technology).

Clock Gating

The *clock gating* technique is an effective thermal management technique. It can reduce the temperature by cutting off the clock signal of the overheated processor component [6]. The clock gating technique is implemented based on the thermal management demands, such as cutting the clock signal when temperature will exceed the predefined thermal limit [133].

Fetch Toggling

The *fetch toggling* technique is another effective thermal management technique that is used on most computing systems [32]. Other than cutting off the clock signals, the fetch toggling technique throttles the instruction cache and make the task execution idle when the temperature is going to exceed the threshold. But, this approach has to sacrifice the performance to bring down the temperature.

Task Migration

The *task migration* technique has the flexibility to migrate a task among cores on the multicore platform. This technique is majorly used for the purpose of workload balancing and temperature balancing (i.e. it can cool down the processor by migrating the task from a hot core to cooler core). Compared to the other thermal management techniques discussed before, the task migration technique has very small effect on the system performance. However, the migration algorithm should be carefully designed to avoid the Ping-Pong effect (i.e. the particular case when the temperature difference between the cores is very small, and a task is constantly migrated from the hot core to the cool core), that only increases a huge amount of context switching overhead without reducing the temperature. A lot of researches have been done based on the task migration technique [36, 129, 30, 29, 131, 84, 33].

2.3.3 Thermal-Aware Scheduling

Based on the DTM techniques introduced in the previous section, extensive researches have been published in the recent years. Those research works can be categorized based on different criteria. For example, some researches have been tremendously conducted for soft real-time systems [132] and hard real time systems [19, 125]. They focused on how to dynamically adjust the supply voltage and frequency level of a processor so that the tasks can be executed before the deadline while other design objective can be achieved as well [63, 127, 22, 53, 45].

In addition, thermal and power scheduling algorithms have been proposed for different platforms, such as single-core [125, 136], multi-core [41, 42], or even 3D multi-core platforms [73]). Based on task models, they can be classified into stochastic [72] and deterministic workload [17, 134]. Both of the online approaches [132, 77] and offline approaches [46, 136] have been proposed to solve the throughput maximization problem under a given constraint [19, 52, 132, 9, 17, 134]. They can be classified into two categories, which are reactive and proactive scheduling algorithms.

Reactive scheduling algorithm

The reactive thermal management approaches are relying on the temperature information, which is obtained either from thermal sensors or simplified thermal models, to trigger the thermal management operations [101]. For example, Pedram et al. [88] proposed two reactive DTM techniques based on DVFS and pipeline throttling techniques to remove the aggressive heat. Chen et al. [25] proposed a reactive workload balancing algorithm based on the task migration technique to reduce the overall processor temperature. Xiaorui et al. [122] developed a feedback control loop to optimize the system throughput, and the temperature information from thermal sensors are used as the feedback signal. However, the thermal sensor lacks accuracy in the following ways. First, there are only a limited number of sensors on the multiprocessor and the thermal hotspot on the processor migrates during the execution, so the sensor may not always be placed at the hottest spot. Second, the sensor also has manufacture variations, and the temperature from thermal sensors is not always reliable. In addition, it still takes time for the DTM techniques to take effect after changing frequency or migrating tasks. Thus, with the above limitations, the reactive approach cannot effectively keep up with a rapid temperature change.

Proactive scheduling algorithm

To address the problems of the reactive approach, the proactive thermal management algorithms with temperature prediction technique has been proposed. It can accurately detect the thermal emergency before the temperature exceeding the temperature limit and leave enough time for the thermal management algorithm to react to the temperature change. Thus, an accurate temperature prediction technique is highly in demand. Otherwise, it can cause the DTM algorithms to overreact with the temperature changes, and further degrade system performance. A lot of research has been proposed to develop the temperature prediction techniques [137, 131, 99, 106].

Meanwhile, numerous proactive researches with prediction techniques have been proposed. For instance, Yeo et al. [130] proposed a proactive DTM algorithm that utilize a regression-based temperature prediction technique to reduce the peak temperature. Coskun et al. [29] developed a proactive schedule to optimize the system throughput by balancing the workload. Also, Kumar et al. proposed a coordinated thermal management algorithm, which combines both hardware and software DTM techniques. In conclusion, the proactive DTM techniques can outperform the reactive approaches, and achieve better system performance.

The rest of the work is organized as follow, We first built up a practical hardware platform in Chapter 3. Then, we evaluated the cooling efficiency of the dynamic thermal management technique by comparing it with the traditional cooling methods, and validated two widely used thermal management principles in Chapter 4. After analyzing the limitations of the theoretical work on our practical hardware platform, the reactive algorithm, (EADTM) for the single-core platform has been proposed in Chapter 5. To extend our research to the multicore platform, we developed a proactive algorithm (NADTM) in Chapter 6. Finally, we conclude this dissertation in Chapter 7.

CHAPTER 3
THE INTEL BASED PRACTICAL HARDWARE PLATFORM

3.1 Related Work

Since power and thermal issue have already become the major design constraints for the development of the computing system, many theoretical works have been conducted based on dynamic thermal management techniques in the recent years. For example, several DTM techniques [9, 125, 44] have been proposed that take the temperature and leakage interdependency into consideration to optimize the total energy consumption. A few other approaches [20, 62] seek to minimize the peak temperature at run-time. There are also some researches that studied the thermal aware performance maximization problems [18, 134, 136]. Most of these theoretical research results are based on simplified models and idealized assumptions. For example, most of the above researches employ the lumped first order RC thermal model to simulate temperature dynamics. Also, as shown later in this chapter, some of the assumptions such as the exact knowledge of the actual temperature may not always be possible. Although theoretical researches simplify the research problem and help to uncover the fundamental principles in practical scenarios, practical applications must deal with some important details in the real-life environment.

To this end, a lot of practical hardware platforms have been developed to test theoretical works in the real-life environment [15, 24, 98, 102]. Tongquan et al. [123] developed a multicore real-life testbed using a dual-core Intel T2500 processor with the hard real-time scheduler adapted from the Linux operating system. Canturk et al. [48] proposed a processor power measurement and estimation methodology based on the Intel Pentium 4 processors, and the real total power measurement has been combined with performance counter value to get per-unit power estimation. Simone et al. [28] proposed a novel on-line temperature measurement methodology based on a quad-core Intel i7-820QM processor. Phillip et al. [55] developed a thermal management system that continuously monitors the temperature of the FPGA and reprograms the device if the temperature approaches the outer limits of safe operating conditions.

Also, a number of researches were conducted based on more practical computing systems [121, 109, 3, 122, 93, 61]. For example, Ahn et al. [3] developed and validated a heuristic algorithm to reduce the power consumption and control the temperature on an Intel Centrino Duo and an ARM-11 MPCore platform. Erven et al. [93] proposed an algorithm to maintain the system temperature under a pre-defined threshold by adjusting the utilization of the CPU in a Pentium-II desktop. Wang et al. [122] developed a feedback control algorithm based on-line temperature control method and implemented their novel algorithm on an Intel Xeon desktop. Amit et al. [61] incorporated hardware and software thermal management technologies and proposed a hybrid thermal management algorithm to optimize the heat dissipation in a Pentium-4 system. Fabrizio et al. [83] presented a lightweight thermal balancing policy, which could minimize the on-chip temperature gradients by using the task migration technique. This algorithm was tested on an actual three-core MPSoC platform. Kim et al. [54] proposed a new application-oriented learning-based dynamic thermal management technique for a multi-core system based on an Intel Dual Core.

3.2 The Intel i5 Based Hardware Platform

As we discussed in Section 3.1, it is very important to validate the theoretical works on the practical environment to get the reliable experiment results. Thus, a practical hardware platform, which has the flexibility to be configured to adopt different theoretical algorithms, is highly in demand. To satisfy the above constraints, we developed a practical hardware testbed based on a Dell Precision T1500 desktop workstation (i.e. the detailed system specification is shown in Table 3.1).

In addition, The overall system platform is depicted in Figure 3.1. The central component of this platform is an Intel i5-750 quad-core processor, running the Ubuntu 10.04.1 Linux operating system with kernel version 2.6.32. The Intel i5 processor has been integrated with the DVFS capability, which can adjust the working frequency of each core individually. Meanwhile, each of the cores on Intel i5 processor is integrated with an on-chip thermal sensor, which can continuously provide the temperature information during the execution. The cooling factor of the computing system can be adjusted by changing the fan speed. We can also fix the fan speed to get a constant cooling environment, which is a very important factor when testing the theoretical work. To test the thermal management algorithm, we used the well-known academic benchmark suite, SPEC CPU2000, including both integer and floating point operations to get reliable and comparable experiment results. Each benchmark can be assigned to any particular core before being executed. Furthermore, the task can be migrated between processors based on the thermal management demands. The detailed system implementation of each component of the platform will be discussed in the following sections.

Table 3.1: Dell Precision T1500 Technical Specifications

Dell Precision T1500 Desktop	
Processors	Quad-Core Intel i5 Processor, 64-bit
Operating System	Ubuntu 10.04.1, 64-bit , Linux Kernel 2.6.32
Chipset	Intel H57
Networking	Integrated Broadcom 57780 Gigabit Ethernet controller
Memory	8GB 1066MHz
Graphics	NVIDIA Quadro FX580
Hard Drives	SATA 3.0Gb/s: 7200RPM with 8MB DataBurst Cache, 1.0TB

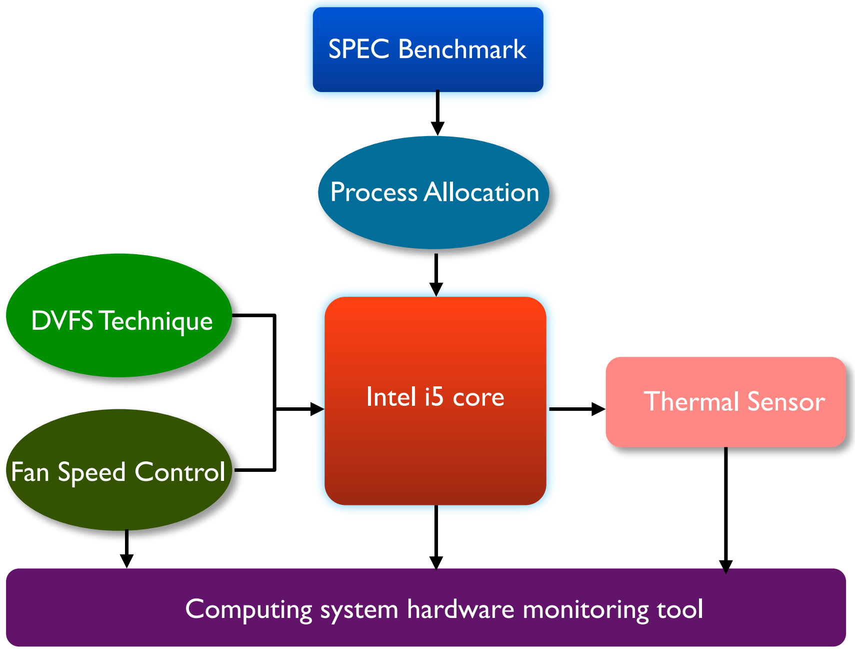


Figure 3.1: Our test bed based on an Intel Desktop computer

3.3 Dynamic Thermal Management Techniques

Our customized hardware platform is able to adopt the majority of the dynamic thermal management technique, which is introduced in Section 3.1. Such as, clock gating, idle period injection and system level task allocation. However, the two of the most effective approaches are the process migration technique and the dynamic voltage and frequency scaling technique. Thus, in this section, we are focusing on introducing the implementation detail of those two techniques on our platform.

3.3.1 DVFS Technique

The dynamic voltage and frequency scaling (DVFS) technique is a widely used approach, which can simultaneously adjust voltage and frequency level of the computing system either based on the workload demand or the temperature constraint during the execution. The modern chip multi-processor (CMP), Intel i5 (i.e. detailed specification is shown in Table 3.2), was integrated with the *Enhanced Intel SpeedStep*

Technology (EIST) [107]. Other than changing both the voltage and frequency between high and low levels, the Enhanced Intel SpeedStep Technology can adjust the supply voltage in a small increment separately from frequency changes. As a result, the system unavailability can be significantly reduced. In addition, it also allows the bus clock to continue running during the state transition, which keeps the logic of computing system to remain active. The Intel i5 microprocessor supports 12 different working frequency levels ranging from 1.2GHz to 2.66GHz, as shown in Table 3.3.

Table 3.2: Intel Core i5-750 Processor Specifications

Essentials	
Processors number	i5-750
Number of core	4
Clock speed	2.66 GHz
Max turbo frequency	3.2GHz
Cache	8MB
Instruction set	64-bit
Voltage Range	0.65V-1.5V
Memory Specifications	
Max memory size	16GB
Max memory Bandwidth	21GB/s
Physical address extension	36-bit
Package Specifications	
T_{CASE}	72.7^oC
Package size	$37.5mm \times 37.5mm$
Processing die size	$296mm^2$
Number of transistors	774 million

Besides the Enhanced Intel SpeedStep Technology hardware support as mentioned before, the *CPUfreq* Linux kernel subsystem has been used to implement the dynamic voltage and frequency technique features on the Linux operating system. Figure 3.2 shows the Cpufreq infrastructure at a high level. The Cpufreq module provides an interface between the user level frequency controlling policy and underlying mechanisms. The CPU-specific drivers include various CPU frequency-changing technologies, such as the Enhanced Intel SpeedStep Technology. The proper driver

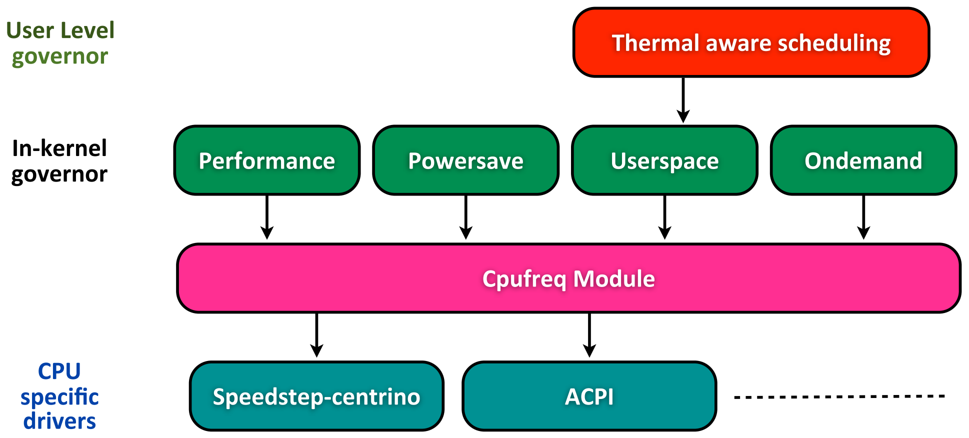


Figure 3.2: High-level Cpufreq module infrastructure overview

must be loaded for the platform to perform efficient frequency changes. The low-level CPU-specific drivers *acpi* and *speedstep-centrino* are used to implement the Enhanced Intel SpeedStep Technology-enabled CPUs. The cpufreq module allows different frequency-changing policy governors, which can change the CPU frequency based on different demand. For example, *Performance governor* execute the task with the highest possible frequency to achieve the maximal performance. *Powersave governor* keeps the CPU at the lowest possible frequency to save energy. *Ondemand governor* automatically adjust the working frequency based on the current system workload to save energy. And, *Userspace governor* exports the available frequency information to the user level, and permits user-space control of the CPU frequency. In this work, we use the Userspace governor to implement most of the thermal management algorithms. Core frequency is modified by updating the system file at */sys/devices/system/cpu/cpu(num)/cpufreq/scaling_setspeed*, where *num* is the index of the cores in the processor. In our experiments, whenever we changed the frequencies, we changed the supply voltages simultaneously.

Table 3.3: Intel i5 available frequency levels (GHz)

1	2	3	4	5	6
2.667	2.533	2.4	2.267	2.133	2.0
7	8	9	10	11	12
1.867	1.733	1.6	1.467	1.333	1.2

3.3.2 Task Allocation and Migration Technique

Task allocation is implemented by using the system call named *CPU affinity* on the practical computing system. The CPU affinity can be used to bound one or more processes to one or more processors. The original purpose of using the CPU affinity is optimizing cache performance. The operating system tries to keep the tasks on the same processor as long as possible. That way, the task's data can be kept in only one processor's cache at a time for the multiprocessing computing system. Otherwise, it could create a data synchronization problem, which can increase the cache miss rate and degrade the overall system performance (i.e. the operating system tries to find out which process has the most up-to-date copy of the memory). In addition, constantly migrating tasks from one core to another can increase context switch overhead, which means the computing system will spend extra energy to copy the data from one processor's cache to another. It can cause system delay, and also generate extra amount of heat. However, in this work, we only use the CPU affinity feature to bind a task based on our scheduling algorithm's demand.

In addition, the task migration technique can also be implemented by using the CPU affinity system call (i.e. *sched_setaffinity*). As long as the objective task PID is obtained, we are able to migrate the task by using the affinity bitmask. Each bit of the mask represents whether the given task is bound to the corresponding processor

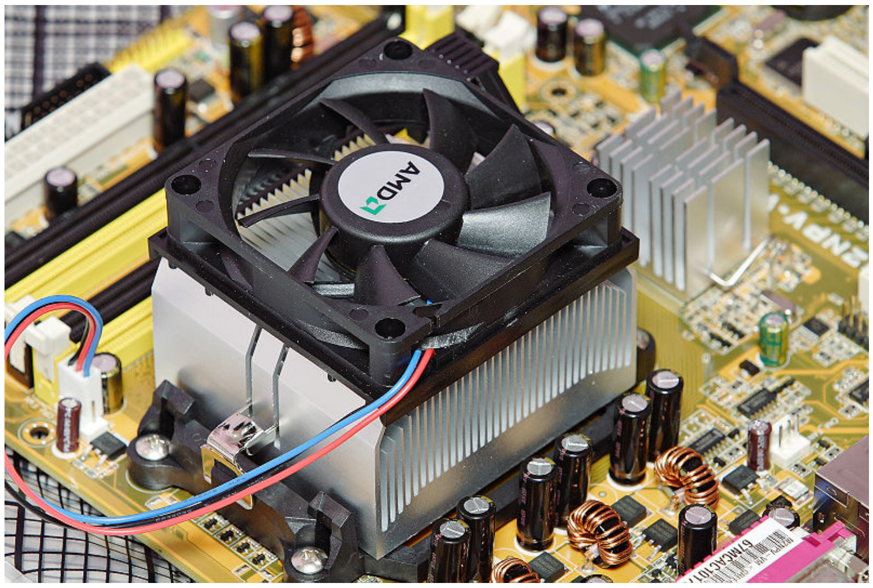


Figure 3.3: Conventional air-cooling with heatsink

3.3.3 Computing Cooling System Configuration

The computer cooling system is designed to remove the wasted heat generated by the computer components, and keep the computing system operating within a permissible temperature limits. Until now, a lot of cooling method have been proposed and adopted in the modern computing system. They can be classified into two categories. The first approach removes heat from conduction, such as the heatsink, which is made of metal with high thermal conductivity and with a large surface. The heatsink is attached directly on top of the microprocessor, thus the heat can be transferred from chip to the heatsink, and then heat is carried away from the heatsink by airflow. Another example could be the liquid cooling system that uses liquid to take away the heat from the processor. It has the advantage of less impact with the ambient temperature and low noise comparing with the air-cooling. The second category removes heat from the convection, such as the air-cooling. It is the most famous cooling solution for the computing system. Fans are most commonly used for air-cooling when natural convection is insufficient.

The Dell Precision desktop workstation has two cooling components: the heat sink and cooling fans as shown in Figure 3.3. The cooling efficiency of the heat sink depends on its physical characteristic. This static thermal characteristic will not change during the computing system operation. On the other hand, users can dynamically control the cooling fans. In our work, we control the fan speed using a simple shell script named *fancontrol*. The fan speed can be adjusted from the maximal of 4500RPM (round per minute) to the minimal of 1500RPM. To ensure our experiments were conducted under the same cooling condition we fixed the fan speed at the minimal speed (i.e. 1500RPM) unless otherwise specified. This helped to overrule the fan control by operating system when executing programs.

3.4 Power and Temperature Measurement

Since we are focusing on developing a hardware platform to analyze the thermal management algorithm, obtaining the computing system temperature and power information timely and accurately is the major topic. In this section, we introduce the detailed implementation of collecting temperature information from thermal sensors and the technique used to measure the total system power consumption. We introduce the hardware monitoring tool at the end.

3.4.1 Power Measurement

Since the computing system power dissipation keeps increasing rapidly each year, the power issue becomes a more and more critical research topic. Measuring the power dissipation of the computing system at runtime becomes more and more crucial for the system developer. In this section, we give detailed introduction on how to implement the power measurement technique. It was carried out based on a well know research work [48].

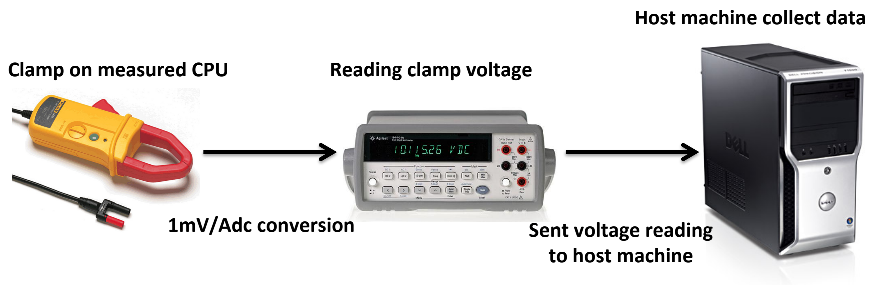

Clamp on measured CPU
Reading clamp voltage
Host machine collect data
1mV/Adc conversion
Sent voltage reading to host machine

Figure 3.4: Overview of the system interconnection

The overview of the power measurement system is shown in Figure 3.4. The major components include an *Fluke i410 AC/DC* current clamp, which is used to clamp directly to the power supply of the microprocessor on the motherboard. It can read the current passed through the CPU chip without cutting the wires. Then, the current information is converted to voltage and sent to the *Agilent Digital Multimeter 34401A*. The multi meter is connected with a hosting PC machine with an *Agilent 82357B USB/GPIB* cable. On the hosting machine, we installed the software and drivers to collect power data (i.e. *Agilent IO Libraries Suite 15.1*, *34401A Digital Multimeter IVI Instrument Drivers*, and *IntuiLink for Digital Multimeters Version 1.3.3*). During the power measurement process, the power consumption can be measured and recorded in the hosting machine simultaneously.

3.4.2 Temperature Capturing

Processor temperature should be carefully maintained within a proper working temperature limit to avoid hardware damage and performance degradation. A thermal sensor integrated on the microprocessor is the major component to obtain the system temperature information. It is used to guide the operating system to manage the cooling system. For example, the operating system reads the current temperature value from the thermal sensor, and based on the temperature, a new fan speed is calculated for the system configuration.

A Dell Precision T1500 desktop has two different kinds of thermal sensors. One is the external thermal sensor, located underneath the CPU chip and above the motherboard. It can provide the overall temperature of the chip. An alternative method is to read temperature value directly from the built-in digital thermal sensor integrated with each core, which can provide more accurate temperature information of the core. The temperature value is stored in the *Model Specific Register* (MSR), which can be accessed through the *Industry Standard Architecture* (ISA) ports or the *System Management Bus* (SMBus). To ease our implementation and tests, we simply adopted a Linux hardware monitoring tool called *Lm-sensors* [1] to capture the temperature, to set the fan speed, to vary supply voltage and working frequency. Through our experiments, we found that the time to access the on-chip thermal sensor is approximately $8ns$; however, the resolution of the on-chip thermal sensor is only 1^oC; and the minimal time for a temperature sensor to reflect a change in temperature is approximately 1 second.

3.4.3 Computing System Monitoring Tool

As we discussed before, our customized hardware platform has the flexibility to do a lot of modifications to adopt the thermal management techniques, such as, task

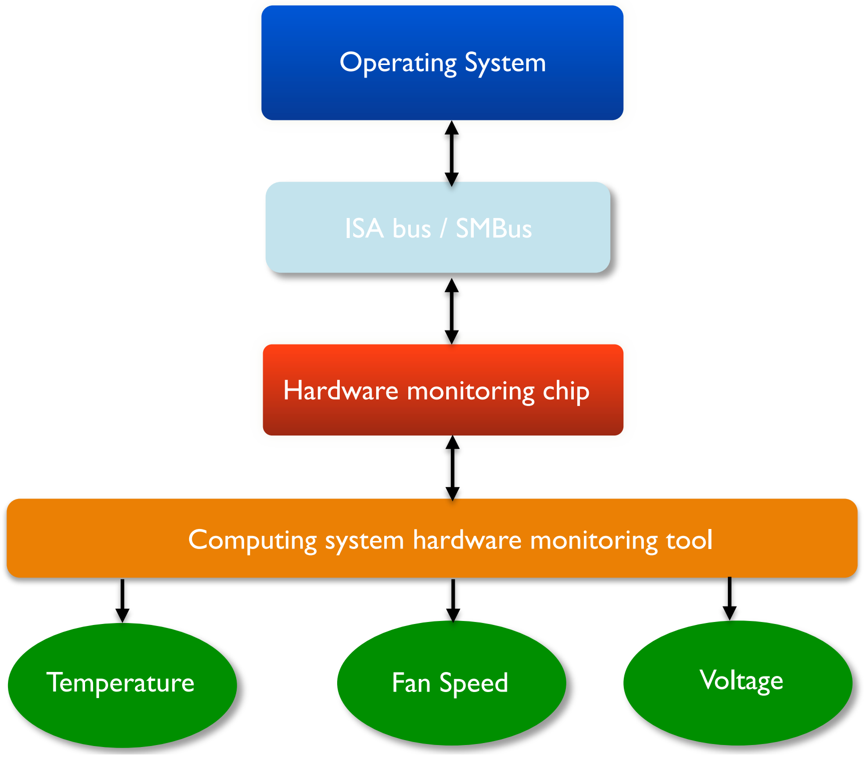


Figure 3.5: Lm-sensor monitor tool overflow

migration, DVFS, changing the cooling system, reading thermal sensor and measuring power consumption. However, quick changing system informations must be carefully monitored and presented to the developer simultaneously during the operation. In this work, we use the *Lm_sneosrs* Linux hardware monitoring tool to observe the system information. Most of the computers that have been built after the year of 1997, are integrated with a hardware health-monitoring chip. As shown in Figure 3.5, The computing system hardware monitoring tool could collect system information (i.e. the current processor information, supply voltage and frequency level, fan speed, as well as the CPU utilization of each individual core) from the hardware monitoring chip. And the operating system can access the information through the ISA/SMBus, and guide the thermal management algorithm to react with the temperature change.

CHAPTER 4
VALIDATING THERMAL MANAGEMENT PRINCIPLES ON PRACTICAL HARDWARE PLATFORM

In the previous section, we already gave a detailed introduction about our practical hardware platform. It is used to test and evaluate different DTM algorithms for the rest of this work. Since the exponentially increasing power consumption brings up the temperature of IC chips, how to deal with the heat generated by processors has become a major design constraint for computing systems. In the recent years, extensive theoretical research works on dynamic thermal aware computing have been published. However, there are not many experimental researches based on practical computing platforms. Thus, such theoretical works lack validation and are also not convincing. Validating the theoretical work in the practical environment is highly demanded. In this chapter, we first evaluate how effective the DTM techniques can be by comparing them with the conventional air-cooling method. Next, we introduce a number of widely used thermal aware scheduling algorithms, and then we implement each of them on our hardware platform to get practical experiment results. All of the algorithms are analyzed carefully and their performance are compared to evaluate their efficiency.

4.1 Related Work

The traditional cooling solutions, which include using heat spreaders, heat sinks, and cooling fans in a variety of configurations, have already been implemented in almost every computing system. However, it has been reported that only relying on the conventional cooling approach cannot effectively solve the more and more critical thermal challenges. As a result, numerous new cooling techniques and thermal materials have been developed in the recent years [11], and there has been increasing interest in both academy and industry in the computing side rather than the mechanical side. There has been extensive theoretical researches, which focus on dynamic thermal management for the single core microprocessor, which have been published in the literature [104, 79]. For example, Wang et al. proposed a scheduling algorithm that can exploit the flexibility of single core platform at low temperature by deriving an ideally preferred speed [35]. Zhang [135] et al. proposed an polynomial approximation algorithm for single core processor to maximize the performance under thermal constraint. Most of these theoretical researches are based on theoretically idealized models. While the theoretical model helps to abstract the problem essential characteristics, practical computing systems and applications must deal with other important details as well. Some of these details may be derived analytically, and some of them may not. It is therefore valuable and important to evaluate these techniques subjecting them to experimentation under a more practical scenario.

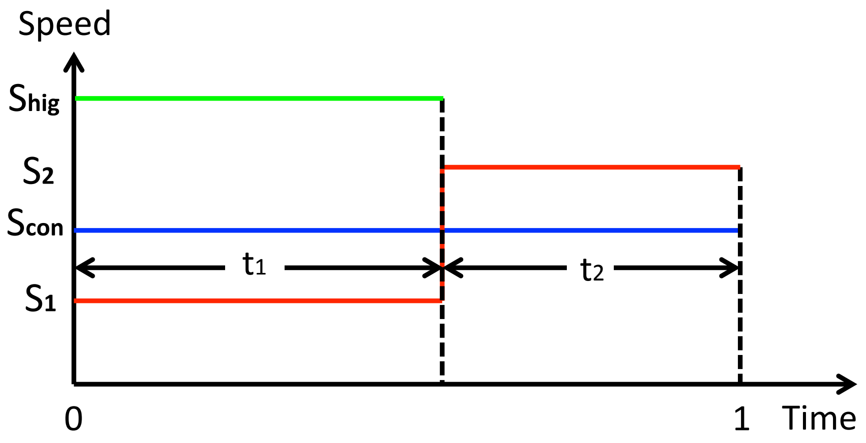
Some research works have been conducted to study the effectiveness of thermal management techniques based on real hardware platforms. Kumar [61] et al. proposed a hardware and software hybrid thermal management technique and tested their algorithm on an actual desktop machine with an Intel Pentium-4 processor and using the SPEC2000 benchmarks. Wang [122] et al. evaluated their newly developed feedback control thermal management technique based on a desktop with Intel Xeon X5365 Quad Core processor. Hanson [40] et al. investigated the thermal response of DVFS and other factors beyond voltage and frequency based on an Intel Pentium M system.

In this chapter, we investigate the effectiveness of several dynamic thermal management techniques based on our practical hardware platform. Through our research, we seek to understand how effective a dynamic thermal management technique, such as a thermal aware scheduling policy, can be to control the peak temperature when compared to the traditional air-cooling method. We also intend to use this platform to validate some principles that we found through our analytical study of this problem. To this end, we implemented and experimented a number of thermal aware scheduling techniques on this platform. The experimental results clearly demonstrated that the dynamic thermal management strategy is an effective method to control processor temperature. The rest of this chapter is organized as follows. We first analyze the cooling efficiency of the DTM algorithm by comparing it to the traditional air-cooling approach. Second, we introduce several power and thermal management algorithms. After comparing the DTM algorithm with the DPM algorithm, we conclude that although the power management algorithm can optimize energy and reduce the temperature as well, developing a thermal management algorithm to deal with special thermal characteristics is highly demanded. In addition, after testing the DTM algorithms under different cooling environments, we validated the fundamental thermal management principles still validate in the practical environment.

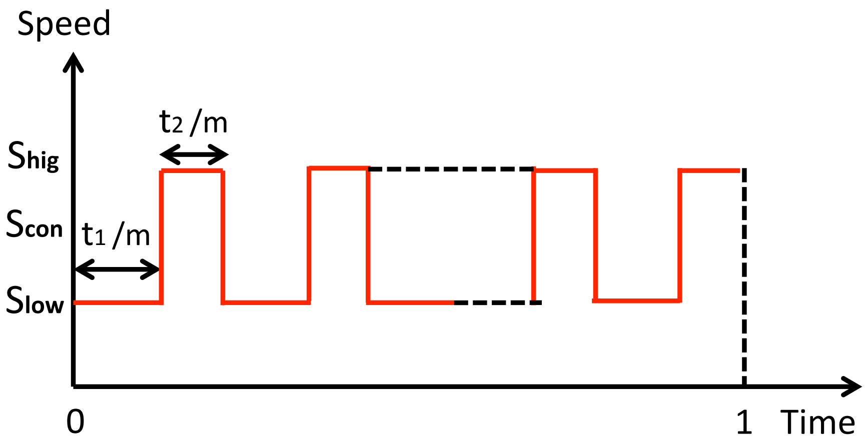
4.2 Preliminary

Due to the close relationship between power and temperature, both thermal and power management techniques have been proposed based on the DVFS technique. The DVFS technique controls the temperature by dynamically adjusting the processor working frequency and supply voltage based on the performance requirement and current workload. In this chapter, we study the four representative thermal and power management algorithms, as shown in Figure 4.1(a).

- The **ConstantSpeed** schedule (i.e. S_{cons}, the blue line plotted in Figure 4.1(a)). Given the throughput or latency requirement to execute a program or complete a workload, the ConstantSpeed schedule uses the minimum constant speed to finish the program or workload before its deadline. Note that, due to the limited



(a) The constant-speed schedule, the run-stop schedule and the two-speed schedule



(b) M-Oscilation schedule

Figure 4.1: Different thermal and power management algorithms

number of speed levels of a practical computing system, this constant speed is not always available.

- The **RunStop** schedule (i.e. S_{high}, the green line plotted in Figure 4.1(a)). With this schedule, the processor always uses a high speed S_{high} to execute the program and enters the power down mode after the workload is completed. Assuming a workload can be completed, use the lowest constant speed within time interval of 1, let the processor finish the same workload at time t_1 and shut down for interval of t_2. To keep the same throughput, we have $t_1 + t_2 = 1$, and

$$S_{cons} \times 1 = S_{high} \times t_1 \tag{4.1}$$

- The **TwoSpeed** schedule is a well-know DPM approach [50], which can minimize the processor energy consumption. Figure 4.1(a) illustrates a TwoSpeed schedule, that uses two available neighboring speeds to execute the process when the constant speed, S_{con}, is not available (i.e. $S_1 < S_{con} < S_2$). In this way, a fixed workload can be finished exactly within a given interval. This schedule uses a low speed S_1 to execute the workload for t_1 then finish the rest of the workload with a higher speed S_2 for t_2. To maintain the same throughput, we have $t_1 + t_2 = 1$ and

$$S_{cons} \times 1 = S_1 \times t_1 + S_2 \times t_2. \tag{4.2}$$

- The **m-Oscillation** schedule, as illustrated in Figure 4.1(b), is a DTM approach [20] that intends to minimize the peak operating temperature. Given a TwoSpeed schedule, the m-Oscillation scheduling algorithm divides the high speed interval and low speed interval evenly into m sections, and then executes the program using the high speed and low speed alternatively. It is not difficult to see that an *m-Oscillation* schedule can complete the same workload as its corresponding TwoSpeed schedule. More theoretic analysis on *m-Oscillation* schedule can be found in [20].

The above schedules are the basic and simplest thermal/power aware scheduling policies. Through some empirical simulations and formal analysis, Vivek et al [112] proposed several fundamental guidelines on peak temperature minimization. Specifically, they found that to minimize the peak temperature when the temperature reaches its stable status,

- Using the lowest constant processor speed that can guarantee the throughput is the optimal method to minimize the peak temperature in the stable state comparing with two speed schedule;
- If such a constant speed is not available and two different processor speeds have to be used, then using the two closest neighboring speeds is the optimal solution in peak temperature reduction.

In the next section, we first studied how effective the dynamic thermal management techniques can be when compared to the traditional air-cooling method. Next, we implement the thermal and power management algorithms described above on our platform to validate the thermal management principles under a more practical test environment.

4.3 System Implementation

Our customized hardware platform is equipped with DVFS capability and can adjust the working frequency for each core individually. We also have the full control of the fan speed in our hardware platform. This allows us to study the cooling efficiency at different fan speeds. To get credible and comparable experiment results, we select both integer and floating point operations from the SPEC CPU2000 benchmarks suit. Our experimental results showed that the output results for different benchmark files are very similar. Therefore, we present only one set of such experiment results generated by the *galgel* benchmark.

Furthermore, the accuracy of the temperature sensor reading depends heavily on its relative location. Since Intel i5 is a quad core processor, temperature reading of the chip can vary considerably depending if it is placed closer to a cooler core than a hot core. To overcome this problem, we implement the process allocation technique to generate four identical processes and assign them to each of the four cores. In this case, the temperature for each of the core is the same. The sensor reading is more accurate. Other than the sensor placement, too many sensor readings can also affect the experiment accuracy, since it will cost extra energy, which can bring up the overall system temperature. Based on the reference [1], the resolution of the thermal sensor is 1 second. This means that if the system reads sensor value faster than one second, the temperature value is always the same. Thus, to address the above problem, we limited our temperature sensor reading frequency to no more than one time per second. In addition, our experiment results also show that the execution time to read the temperature from a sensor is about 2 milliseconds, which is negligible comparing to the execution time of the benchmark. As a result, the sensor-reading overhead can be safely ignored.

Compared to the mathematical thermal simulation, our experiment platform has two major advantages to make our experiment result more accurate. First, the initial temperature and ambient temperature have been assumed to be a constant value. In fact, the initial temperature not only depends on the current room temperature, it also depends on the current fan speed and current working frequency. Based on our experiment results, the lowest initial temperature is 32^oC when the fan speed is 1500RPM and the lowest initial temperature for 4500RPM is 28^oC. Furthermore, the ambient temperature increases along with the chip temperature increases. It will heat up the processor and affect the peak temperature. As a result, the thermal characteristic of ambient temperature cannot be ignored.

There is another inaccurate assumption made in [21] to simplify their thermal model (i.e. the cooling constant is a fixed value). However, in practical computing system, the cooling constant is dynamic. The fan speed and heat sink are two major cooling components. The cooling constant of heat sink depends on its static physical material. In addition, the fan speed of a practical computing system is controlled by the operating system, which fetches the current chip temperature from thermal sensor and calculates a new fan speed based on the configuration. Specifically, when then processor executes an intense job, the chip temperature will increase dramatically in a short time. The operating system needs to increase the cooling constant to cool down the processor and to control the temperature under threshold. In our hardware platform, we have the flexibility to modify the cooling system by changing the configuration file. For example, the maximal and minimal fan speed or the threshold temperature can be adjusted. Also, we are able to fix the fan speed to provide a constant cooling environment.

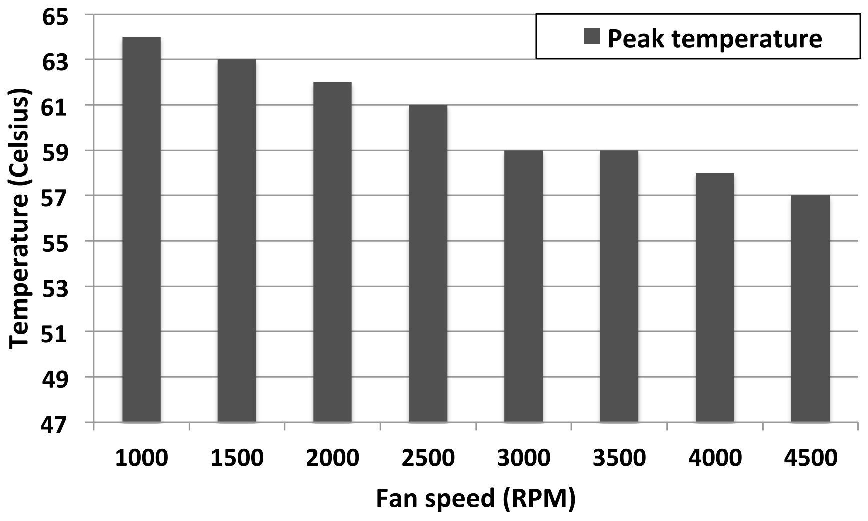
4.4 Experimental Results

In this section, we first evaluate the efficiency of the thermal management technique by comparing it with the traditional air-cooling method. Then each of the thermal management algorithms introduced in the previous section will be analyzed on our platform under different cooling conditions. Temperature information of each approach will be collected to evaluate their performance.

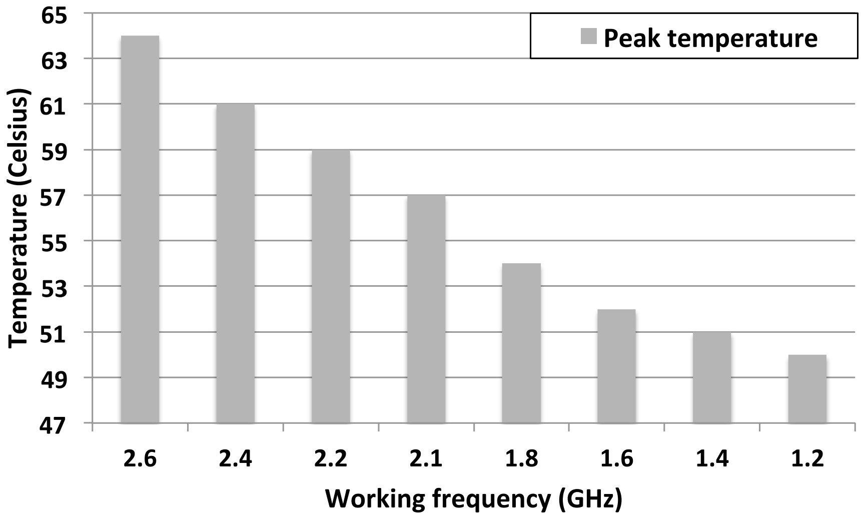
4.4.1 Air-Cooling VS Thermal Management Technique

To evaluate the cooling efficiency of the traditional air-cooling method, we analyze how different fan speeds affect the processor peak temperature. Since the fan on our platform can be adjusted to any speed between the maximum of 4500RPM and minimum of 1500RPM, we select six sampling points in 500RPM intervals. Then, each of the benchmarks is executed at the maximal frequency (i.e. 2.6GHz) respectively. The temperature traces under different cooling conditions are plotted in Figure 4.2(a). The experimental results clearly show that as the fan speed gradually increases, the peak temperature of the processor rapidly decreases. By changing the fan speed from 1500RPM to 4500RPM, the processor peak temperature at the stable state can be reduced by as much as 7^oC (i.e. 12.2% decrement from 64^oC to 57^oC). Furthermore, from our experimental results we can see that the air-cooling approach is more effective to reduce the peak temperature at the lower speed comparing with the high speed. (i.e. 3^oC temperature decrement from 1000RPM to 2500RPM, 2^oC temperature decrement from 3000RPM to 4500 RPM).

Conversely, to analyze the cooling efficiency of the DVFS thermal management technique, we reset the fan speed to the minimal speed and execute the same benchmark with different working frequencies (i.e. range from 2.6GHz to 1.2GHz with a step size of 0.2GHz). The temperature trace for each running case was recorded. From our experimental results in Figure 4.2(b), we can conclude that the DFVS technique can control temperature more effectively than the traditional air-cooling methods. When changing the processor frequency level, the peak temperature can be significantly reduced by as much as 14^oC (i.e. from 64^oC to 50^oC). However, the effectiveness of the DTM technique in controlling temperature is achieved by reducing the system performance. To quantify this trade-off, we compared the program execution time when achieving the same cooling effects by implementing both of the



(a)



(b)

Figure 4.2: (a) Cool down processor temperature with traditional *Air-cooling* solution. (b) Cool down processor temperature by using the *DVFS* technique.

cooling methods. The experimental results were collected in Table 4.1. It quantitatively shows the trade-offs of DTM. While DTM can control temperature within a larger range than air-cooling method, it is at the cost of degraded performance. As shown in Table 4.1, to achieve the same cooling effects as air-cooling, DVFS strategy can cause the performance to degrade as much as 32.3%.

Table 4.1: Execution times for different cooling methods

	Execution Time			
Peak Temperature	64^oC	61^oC	59^oC	57^oC
Air-cooling	$18.621sec$	$18.861sec$	$18.885sec$	$19.009sec$
DVFS	$18.621sec$	$20.577sec$	$21.913sec$	$25.15sec$
Relative Difference	0%	9%	16%	32.3%

4.4.2 DPM VS DTM Technique

In the previous subsection, we already proved that the thermal management technique is more effective in managing the processor temperature than traditional air-cooling method on our practical platform. To further evaluate the efficiency of the thermal management technique, we compare the thermal and power management algorithms (i.e. m-Oscillation and TwoSpeed schedule) on how effective them can reduce the processor peak temperature. We try to prove that although the power management algorithm can effectively optimize the power consumption, it may not necessarily optimize the temperature as well. Developing a thermal management algorithm to deal with the special thermal characteristic is highly demanded.

In our experiments, we set $S_{con} = 2.1GHz, S_{low} = 1.8GHz, S_{hig} = 2.2GHz$. For each benchmark, we collected its execution time using S_{con}, and then constructed the TwoSpeed schedule and the m-Oscillation scheduling accordingly with $m = 5000$. We then ran the two schedules separately on our test bed and collected the peak temperature at each ran. We also compared the execution time of the two schedules,

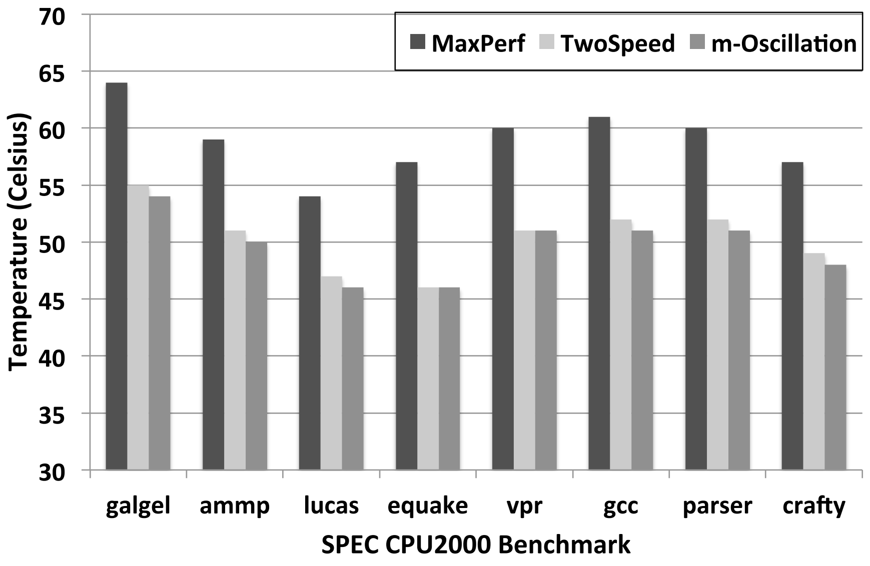


Figure 4.3: The DTM algorithm cooling efficiency comparison

and the difference is less than 2%. For the sake of comparison, we also ran the benchmark at the highest processor speed and also collected the corresponding peak temperature. The peak temperature results are plotted in Figure 4.3. Specifically, *MaxPerf* refers to the schedule with the maximal processor speed used.

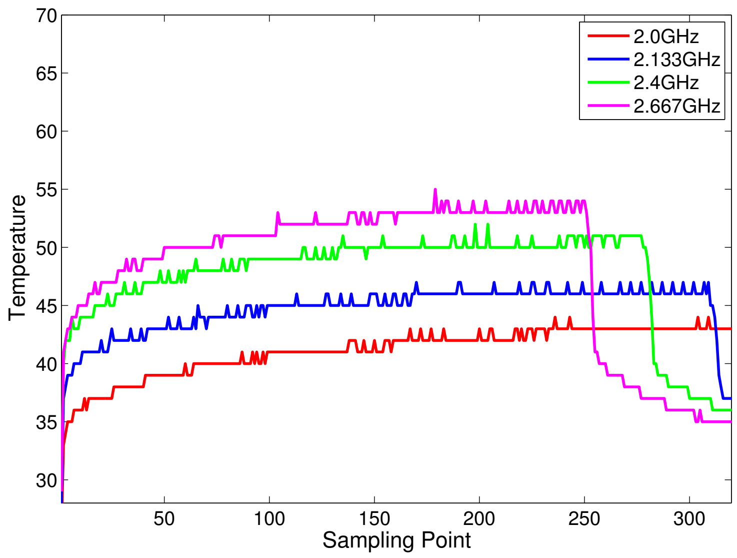
From Figure 4.3, we can see that, comparing with *MaxPerf*, both the TwoSpeed schedule and the m-Oscillation schedule can significantly reduce the peak temperature by 9^oC. This is because both the TwoSpeed schedule and m-Oscillation schedule have a lower power consumption than *MaxPerf*. Furthermore, Figure 4.3 also implies that, a thermal aware schedule, i.e. the m-Oscillation schedule, can better reduce the peak temperature than a power aware schedule, i.e. the TwoSpeed schedule even though the temperature difference in our experiments are relative small (i.e. 0.75^oC in average). However, our experiments verify that even though low power design methods help to reduce the temperature, an optimal power efficient approach is not necessarily the most effective way to deal with thermal issues.

4.4.3 Scheduling Algorithm Evaluation

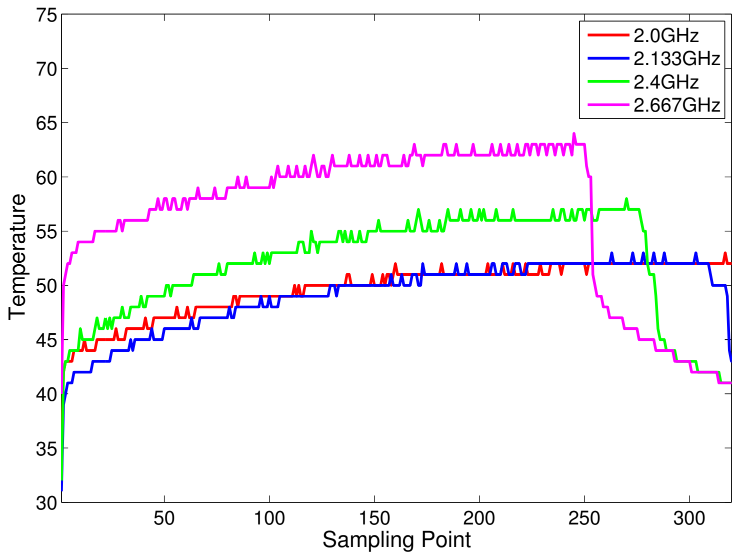
To compare the efficiency of different thermal aware scheduling algorithms, all the approaches are executed with the same initial condition like we mentioned in Section 4.3. For each set of comparisons the same benchmarks are used to guarantee every algorithm is assigned with the same workloads. To ensure the accuracy, both the integer benchmarks and floating-point benchmarks have been tested for each comparison and they all give the same results. In order to save space, we only plot the experimental results by using benchmark *galgel*. A constant speed of 2.0GHz is selected as the baseline speed to get the standard throughput. Since we already proved that the cooling constant could affect the peak temperature in the previous section, each set of experiment is implemented under the maximal cooling constant (i.e. fan speed, 4500RPM) and the minimal cooling constant (i.e. fan speed, 1500RPM) to cover all the possibility.

RunStop schedule analysis

Three higher frequencies, which are higher than the baseline speed, have been selected to implement the RunStop schedule. The temperature traces obtained by each of the frequencies have been plotted in Figure 4.4. From this experimental result we can see that the highest frequency can always finish the job earlier than the other frequencies and leave more time to cool down the chip temperature to a lower level in the second state. However, it also causes a corresponding high peak temperature. Since we are only focusing on the peak temperature minimization, the lowest neighboring speed plotted in blue curve is the best option to reduce the peak temperature when implementing the RunStop schedule. Then we further analyze how different cooling constants can affect the performance. The detailed temperature information has been collected in Table 4.2 to compare with the optimal ConstantSpeed approach. Even



(a) Thermal profile at maximal fan speed 4500RPM



(b) Thermal profile at minimal fan speed 1500RPM

Figure 4.4: The temperature profile of running RunStop schedule with different cooling conditions

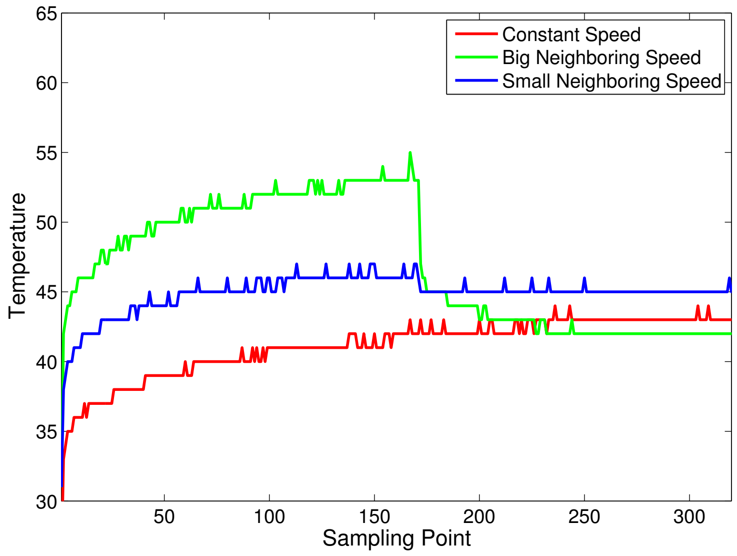
using the same working frequency, the peak temperatures are different. For example, the RunStop schedule has a temperature difference of 3^oC under the maximal fan speed, as well as a temperature difference of 1^oC under the minimal fan speed. From this result, we can reach the conclusion that the cooling condition does affect the performance of thermal management algorithm. Further more, the RunStop schedule could achieve a better performance with low fan speed.

Table 4.2: Peak temperatures under different fan speeds

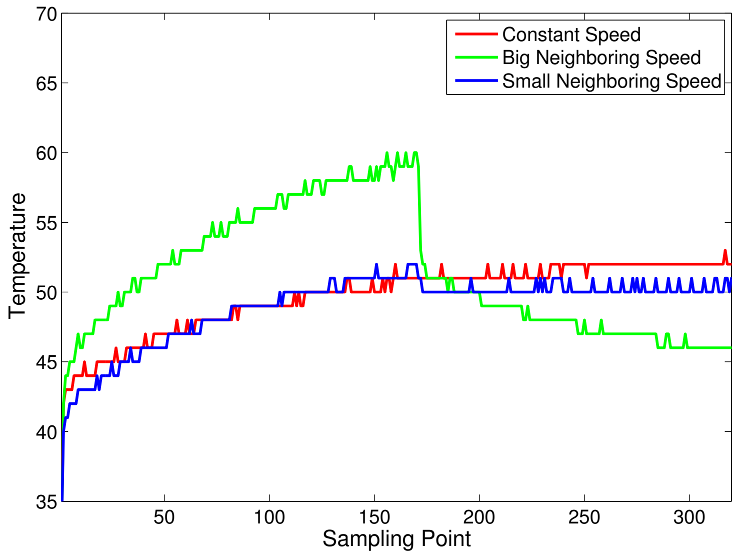
Algorithm	Max fan speed (4500RPM)	Min fan speed (1500RPM)
ConstantSpeed	44^oC	52^oC
RunStop	47^oC	53^oC
Difference	3^oC	1^oC

TwoSpeed schedule analysis

To implement the TowSpeed schedule, 6 available neighboring working frequencies are selected and divided into 3 speed groups as shown in Table 4.3. Each frequency group has been used to implement TowSpeed schedule and tested under different cooling conditions. To make the figure clearer, we only plot the thermal trace of big neighboring speed group and small neighboring speed group to cover all the scenarios. From the experimental results in Figure 4.5, we can see that the small speed group can always outperform the other two speed groups and achieve a lower peak temperature at each cooling condition. Thus, it validates the second principle mentioned in Section 4.2 still feasible in the practical hardware platform. In addition, the detailed temperature information are collected in Table 4.4. It shows that by implementing the TowSpeed schedule, the peak temperature can be controlled as low as 47^oC. Therefore, the TwoSpeed schedule is as good as using the constant speed at minimal fan speed. It proves that the TwoSpeed schedule is more effective under the minimal cooling condition.



(a) Thermal profile at maximal fan speed 4500RPM



(b) Thermal profile at minimal fan speed 1500RPM

Figure 4.5: The temperature profile with the TwoSpeed schedule under different cooling conditions

Table 4.3: Different speed group

Frequency group	Frequency levels (GHz)	
Big neighboring speed	2.667	1.333
Mid neighboring speed	2.4	1.6
Small neighboring speed	2.133	1.867

Table 4.4: TwoSpeed schedule under different Cooling condition

Algorithm	Max fan speed	Min fan speed
OneSpeed	44^oC	52^oC
Small neighboring speed	47^oC	52^oC
Big neighboring speed	55^oC	60^oC

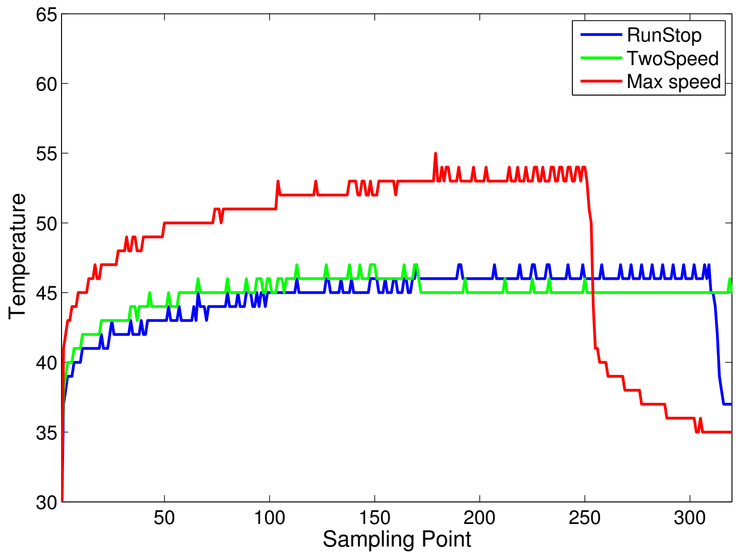
Scheduling algorithm comparison

After we analyzing the efficiency of each thermal aware schedule algorithm, we evaluate their performance by comparing the peak temperature with their best scenario. From the experimental result in Figure 4.6, it shows that both the RunStop and TwoSpeed schedule can achieve almost the same results. However, from Table 4.5, we can see that the TwoSpeed schedule can outperform the RunStop schedule at the minimal fan speed by 1^oC. They have the same performance at the maximal fan speed.

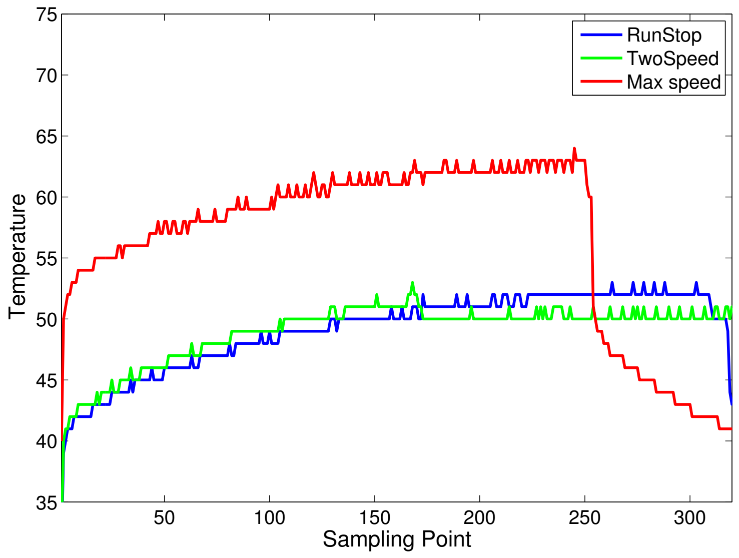
In addition, we also compare the DVFS based thermal-aware scheduling algorithms with the MaxPerf algorithm, which executes the process with the maximal working frequency. In Figure 4.6, the red line presents the thermal profile without using thermal management technique. Table 4.5 shows that by implementing the DVFS technique, the peak temperature can be decreased as much as 11^oC at the maximal fan speed, and 8^oC at the minimal fan speed.

Table 4.5: Temperature comparison under different fan speed

Algorithm	Max fan speed (4500RPM)	Min fan speed (1500RPM)
Max speed	55^oC	63^oC
Run stop	47^oC	53^oC
Two speed	47^oC	52^oC



(a) Thermal profile at maximal fan speed 4500RPM



(b) Thermal profile at minimal fan speed 1500RPM

Figure 4.6: Thermal aware schedules comparison at different fans peed

4.5 Summary

Power consumption and heat dissipation are the major design constraints. To solve this issue a lot of research works have been published based on mathematical simulation. However, many important practical parameters like the cooling constant and ambient temperature are underestimated, their experimental results are not accurate. Thus it is very necessary to test those thermal management techniques on a practical hardware platform. In this chapter, we first proved the DVFS based thermal management technique is an effective approach to control processor temperature by comparing it with the traditional air-cooling method. Then we implemented different thermal aware scheduling algorithms on our customized hardware platform and analyzed their efficiency under different cooling conditions. From the experimental results, we proved that the two widely used theoretical principles are achievable on the practical hardware platform.

CHAPTER 5
A PRACTICAL ON-LINE DYNAMIC THERMAL MANAGEMENT ALGORITHM ON SINGLE CORE MICROPROCESSOR

In the previous chapter, we studied the effectiveness of the dynamic thermal management techniques in controlling the peak temperature on the practical computing system. To further study the importance of the dynamic thermal management algorithm, in this chapter, we first analyze the limitations of the existing theoretical work. Then we proposed a reactive DTM algorithm to maximize the throughput under a given peak temperature constraint.

5.1 Related Work

Thermal aware scheduling, as one of the most effective dynamic thermal management techniques, has been researched extensively in recent years. Some researches [9, 125, 44] take the temperature and leakage interdependency into consideration to optimize the total energy consumption. Some other approaches [20, 62] seek to minimize the peak temperature at run time. There are also some other researches that studied the thermal-aware performance maximization problems [18]. Modern processors have a thermal aware self-protect mechanism, which automatically shut down a processor to avoid physical damage [93] when its temperature exceeds a certain threshold. Thus, the processor temperature needs to be carefully monitored and managed to avoid sudden performance disruption. To this end, Zhang and Chatha [134] presented a pseudo-polynomial time speed assigning algorithm based on dynamic programming to minimize the total execution latency. They further developed several heuristics [136] to maximize the throughput of a real-time system by sequencing the execution of a task set consisting of tasks with different power and thermal characteristics for processors with and without DVFS capabilities. Chantem et al. [18] proposed to run real-time tasks by frequently switching two neighboring speeds other than a constant speed under a given peak temperature limit.

Most theoretical thermal aware scheduling algorithms are based on simplified models and idealized assumptions [18, 100, 126]. For example, some of them employ the lumped first order RC thermal model to simulate the temperature dynamics. However, most of these approaches require a detailed knowledge of processes running on the platform, such as the exact number of processes and their execution times, which is not always available on a general computing platform. Some other information such as the thermal resistance and thermal capacitance, that are essential to build the thermal model for the processor, are also not immediately available. In addition, some common assumptions, which are used in these approaches, may not be applicable in the practical scenario. For example, it has been a common practice to assume [134] that power consumption of a processor remains constant as long as it is executed using a constant speed. However, in reality, power consumption varies with not only temperature but also applications. Also, it is not surprising that a process consumes different power consumptions at different execution stages. It has been reported [131] that the processor temperature changes rapidly even though the workload is static. Although theoretical work can simplify research problems and help to uncover the fundamental principles in practical scenarios, practical applications must deal with some important details in the real-life environment. In the following section, we implement several theoretical works on our platform and analyze their limitations.

5.2 Practical Hardware Limitations Analysis

To develop a general thermal aware scheduling algorithm for the desktop computing platform, the less information is required regarding the processes and platform, the more effective the algorithm can be. In this regard, the *reactive two speed* (RTS) scheduling approach, proposed by Wang [119]et al. seems to be a good choice. According to this algorithm, the processor runs at the maximum speed until it reaches the temperature limit, and then it will run at a constant speed, so called the *equilibrium speed*, to maintain the temperature without exceeding the limit. It has been theoretically proved [27] this is the optimal approach to maximize the workload under a peak temperature constraint.

However, there are a number of problems with this approach. First, since most processors support only discrete levels of working frequencies. The ideal equilibrium speed may not always exist for any given peak temperature constraints. Furthermore, the power consumption of the processor varies with not only the processor speed, but also other factors such as the types of processes, operating temperature, etc. In fact, a processor running a single process may have different power consumptions at different times. Therefore, *the equilibrium speed* is not unique and constant at all, and it is simply not possible to simply set a processor to a constant speed once and for all to maintain a constant temperature.

To deal with these problems, one reactive thermal management algorithm [93], as shown in Figure 5.1, has been proposed. It seems to be more flexible compared to the reactive two speed approach. This approach assumes that no a priori knowledge of the applications running on the computing system is known at all. It monitors the chip temperature regularly and adjusts the processor performance dynamically. At the first stage of the execution, the processor is assigned with the maximal working frequency to guarantee the maximal throughput until the temperature reaches the temperature threshold. After that, at each temperature sampling point, if the current temperature does not reach the threshold, the processor speed is elevated to one level higher. Otherwise, if the current temperature equals or exceeds the temperature limit,

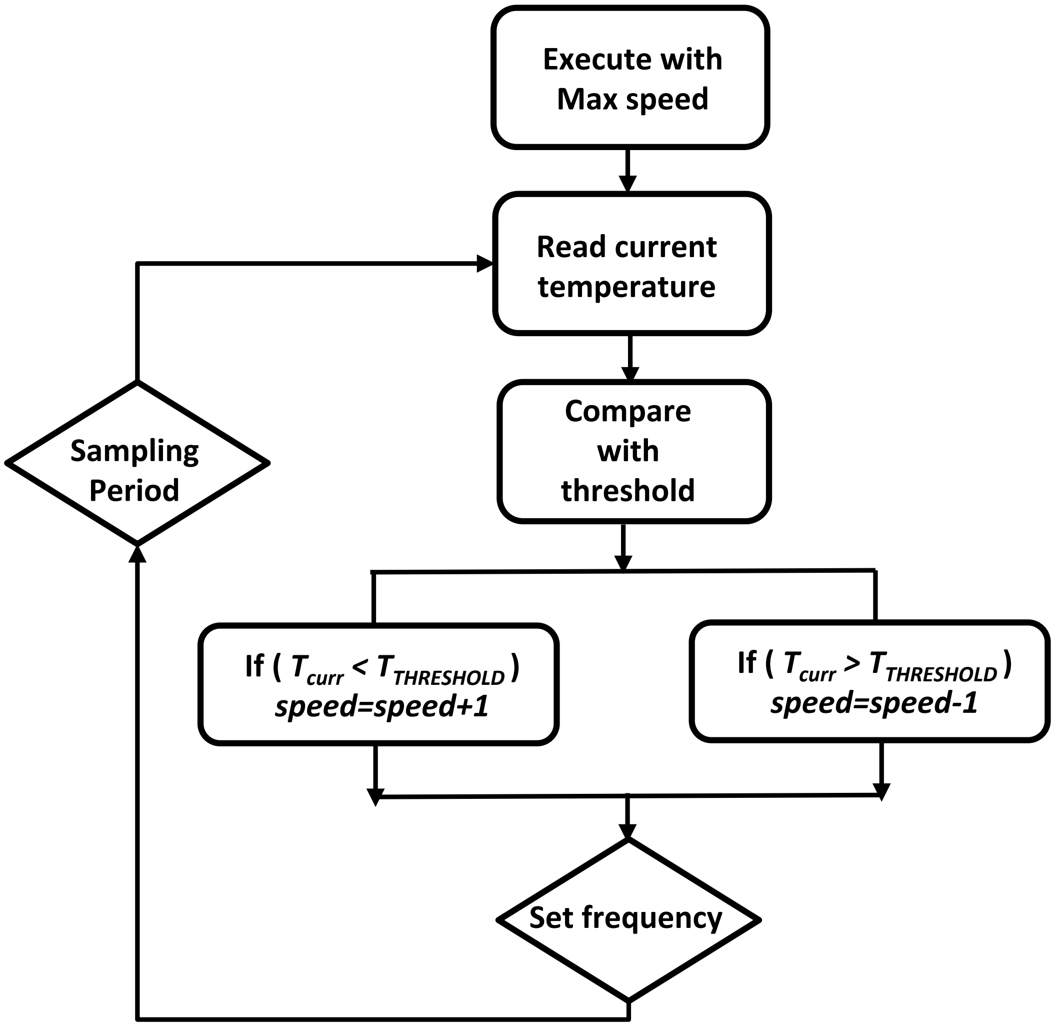


Figure 5.1: The conventional dynamic thermal management algorithm (CDTM).

the processor speed is changed to one level lower to cool down the temperature.

At first glance, it seems that this approach can solve all the problems mentioned above. It naturally assumes that the processor has discrete working frequency levels. And it does not assume any a priori knowledge of the programs running on the processor either. However, there are still a few problems that make this approach less effective in a practical desktop environment. Because there are two important conditions that have to be satisfied to guarantee the accuracy of this approach. First, this approach assumes that the instant temperature information is available immediately and accurately. Second, updating the frequency one level at a time must be quick enough to respond to the temperature change and meet the temperature constraint. But, in a practical scenario, those assumptions cannot be true.

We use a simple example to explain why those two assumptions are not practical. First, the thermal sensor cannot keep up with the rapid temperature changes. As shown in Figure 5.2, recall that it takes about 1 second for the computing system to reflect a temperature change. It is possible that even though the system temperature has already reached or surpassed the temperature threshold at t_1, the sensor reading may still be lower than the temperature limit. In this case, if the computing system continues to increase the frequency, the temperature will keep raising and eventually exceed the peak temperature constraint. Moreover, even though this algorithm can detect the thermal emergency at t_1, it has to adjust frequency one level at a time. Thus, it may not be able to reduce the temperature fast enough (i.e. the frequency switching overhead for computing system is around $10ms$ [59]). The temperature will continue to increase and eventually overstep the peak temperature constraint. To solve those problems, we develop a new dynamic thermal management algorithm that can maintain the processor temperature under a limit while maximizing the system performance.

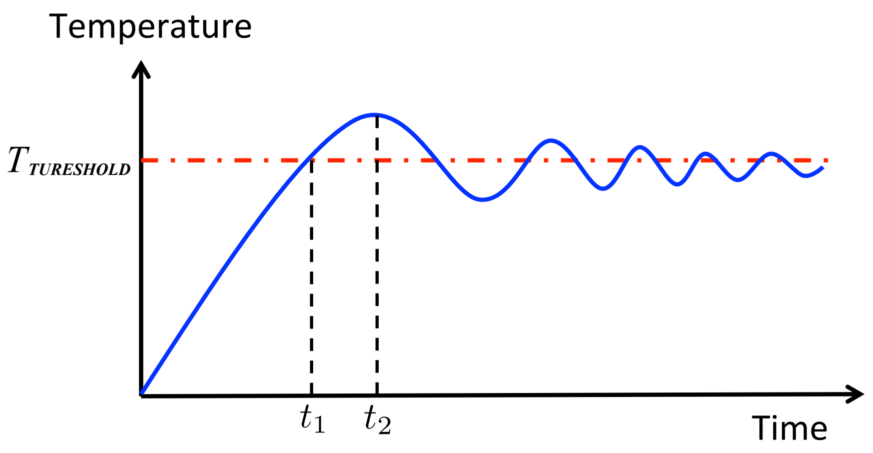


Figure 5.2: An example of temperature trace.

5.3 An Enhanced Dynamic Thermal Management Algorithm

Being able to monitor the temperature change timely and accurately is one of the most critical issues for our approach. Theoretically, it is possible to use the interrupt mechanism to monitor thermal sensor readings, and it will be our future work to study the effectiveness and efficiency of using interrupts for this purpose. In this work [93], we use the simple polling method to monitor thermal sensors for the temperature variations. As a result, defining the appropriate sampling period becomes critical.

5.3.1 Non-Constant Sampling Interval

One intuitive idea to define the sampling period is to set the period as small as possible. So it can detect the temperature change quickly, and only generate negligible accumulated overhead. Unfortunately, using a very small sampling period can increase the possibility of temperature violations. Since the sampling period has been set very small, it will overreact with temperature changes (e.g. rapidly increase the processor working frequency within a small interval). Because it does not give enough time for the adjusted frequency to reach its stable temperature, it can mislead the scheduling algorithm. On the other hand, setting the sampling period too large will

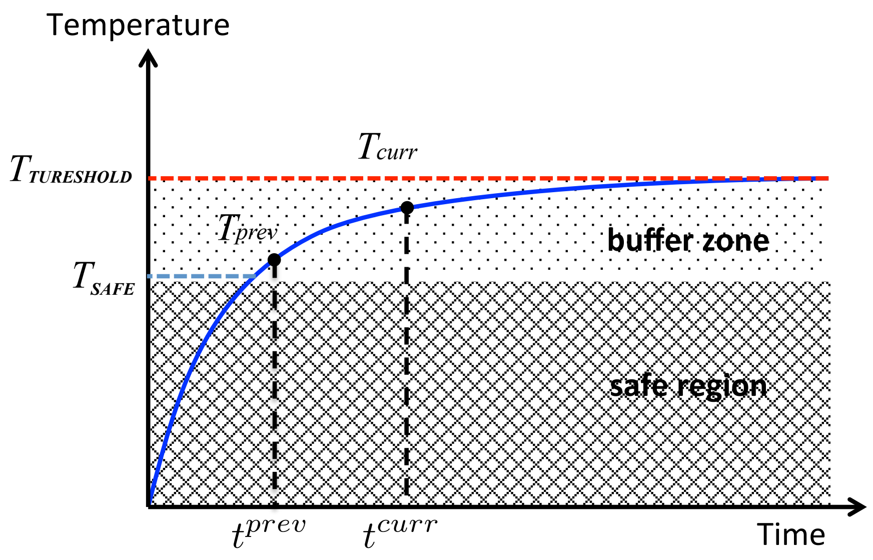


Figure 5.3: The definition of *safe region* and *buffer zone*

lead to not catching temperature timely. Thus, the sampling period should be carefully defined based on the special characteristic of the practical platform.

Given the limitations of the temperature sensor in our platform, in our approach, we set the sampling period to be equal to the minimal response time of the thermal sensor for temperature change. To identify the minimal temperature response time, we ran different benchmarks at different speeds with different sample periods. The minimal interval, for which the temperature sensor has the same reading, is set to be the sampling period P_{sample}. From our empirical work, we found the minimal temperature response time to be 0.98 seconds.

The sampling period defined above can be more effective in avoiding the mishandling of a temperature change. However, in the worst-case scenario, it does not always take 1 second to identify temperature change, (e.g. when the thermal reading changes exactly after one sampling point). To further improve the performance, we develop a more accurate sampling technique, which uses a non-constant sampling periods to detect temperature change. It consists of two sampling periods, P_{regu} and P_{smal} (i.e. $P_{regu} >> P_{smal}$), which are *regular sampling period* and *small sampling* *period* respectively. In our case, $P_{regu} = 0.98$ seconds and $P_{smal} = 0.1$ seconds.

The DTM algorithm with variable sampling frequency (VS-DTM) is described in Algorithm 1. First, the temperature information is collected from on-chip thermal sensors (i.e. line 2). If a temperature change is not detected, the processor speed remains the same. However, it is very possible that the temperature change can occur very soon. Thus, the small sampling period P_{smal} is selected to detect the potential temperature change for the next sampling period (i.e. line 3-4). On the other hand, if a temperature change is detected after the regular sampling period P_{regu}, the next temperature change cannot be detected before another 0.98 seconds due to the delay of the temperature sensor, we can safely set the sampling period as P_{regu}. In addition, the processor working frequency is adjusted based on the comparison between T_{curr} and $T_{THRESHOLD}$ (i.e. line 6- 11). In comparison with the algorithm using a constant sampling period, this approach catches temperature changes faster and responds to them more timely.

Algorithm 1 DTM WITH VARIABLE SAMPLING FREQUENCIES (VS-DTM)

```
1:  while Process is running do
2:     Read current temperature $T_{curr}$;
3:     if $T_{curr} = T_{prev}$ then
4:        Set $P_{sample}$= $P_{smal}$;
5:     else
6:        if $T_{curr} < T_{THRESHOLD}$ then
7:           Increase processor speed by one level;
8:        else
9:           Decrease processor speed by one level;
10:       end if
11:       Set $P_{sample}$= $P_{regu}$;
12:    end if
13: end while
```

5.3.2 Offline Thermal Analysis for Safe Speed

Even though VS-DTM (Algorithm 1) can effectively detect a temperature change, it still adjust the processor speed one level at a time. Thus, when temperature is really close to its limit, it is simply not fast enough to slow down the processor speed to cool down the temperature in time. On the other hand, when the temperature is much lower than the temperature threshold, the processor speed is not increased fast enough to maximize the throughput.

To solve these problems, we first introduce a concept called the *buffer zone* as shown in Figure 5.3. Given a temperature threshold $T_{THRESHOLD}$, the temperature buffer zone is defined as the interval of $[T_{SAFE}, T_{THRESHOLD}]$, where T_{SAFE} is determined by the following equation

$$T_{SAFE} = T_{THRESHOLD} - \triangle T, \tag{5.1}$$

where $\triangle T$ is the maximum possible temperature increment within one sampling period. $\triangle T$ can be determined empirically. Using SPEC2000 benchmark, we found that $\triangle T = 4^oC$. When the temperature is lower than T_{SAFE}, we say that the temperature is in the *safe region*. When the processor temperature is within the safe region, we can safely use the highest possible speed to maximize the throughput before temperature enters into the buffer zone. Thus, the problem becomes how to define the safe speed to run the task and ensure the temperature does not exceed the threshold after entering the buffer zone.

To solve this problem, we can conduct an offline thermal profiling analysis to identify the *safe speed* (S_{safe}) when running a task in its buffer zone. Consider the commonly used thermal model as follows

$$\frac{dT(t)}{dt} = aP(s) - bT(t), \tag{5.2}$$

where $T(t)$ is the temperature at time t, $P(s)$ is the power consumption with processor speed of S, and a, b are the cooling constants. To ensure that temperature does not exceed $T_{THRESHOLD}$, we only need to make sure when temperature is located in the buffer zone, we have

$$\frac{dT(t)}{dt} \, |_{T(t) \in [T_{SAFE}, T_{THRESHOLD}]} = 0, \tag{5.3}$$

By combining equation (5.2) and (5.3), analytically we can solve for processor speed S. However, it can be extremely challenging to determine the analytical function of $P(s)$ and the cooling constants a, b. Hence, we determine the safe speed empirically from the SPEC2000 benchmark. Specifically, given a task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, $T_{stable}(\tau_i, s_j)$ denotes the stable temperature when running τ_i using processor speed s_j. Let S_i be the speed such that

$$S_i = \max\{s_j \text{ such that } T_{stable}(\tau_i, s_j). \leq T^{THRESHOLD}\}. \tag{5.4}$$

And the safe speed S_{safe} is determined as follows.

$$S_{safe} = \min_{\tau_i \in \mathcal{T}} S_i. \tag{5.5}$$

Each benchmark is executed by using all the available CPU speeds and the corresponding peak temperatures are recorded to build up a look up table to guide the scheduling algorithm making thermal management decision.

With the safe speed available for each task, we are ready to present our *enhanced reactive dynamic thermal management* (ERDTM) algorithm, which is depicted in Algorithm 2. It is developed based on the VS-DTM algorithm. When the processor is executing a task and its temperature is within the safe region, The highest processor working frequency S_{max} is used to guarantee the maximal throughput (line 5-6). Otherwise, it selects the safe speed S_{safe} to make sure the temperature constraint is not violated (line 7-9). If the running task sets are known a priori, we can further improve the performance of our algorithm by building up a lookup table. The lookup table lists the tasks and their specific safe speeds under different temperature constraints, as defined in equation (5.4). In that case, we can use the corresponding safe speed depending on the current running process to further maximize its throughput.

Algorithm 2 ENHANCED REACTIVE DYNAMIC THERMAL MANAGEMENT (ERDTM)

```
1: while Process is running do
2:   if $T_{curr} = T_{prev}$ then
3:     Set $P_{sample} = P_{smal}$;
4:   else
5:     if $T_{curr} \leq T_{SAFE}$ then
6:       Set processor speed to the $S_{max}$;
7:     else
8:       Set processor speed to $S_{safe}$.
9:     end if
10:    Set $P_{sample} = P_{regu}$;
11:  end if
12: end while
```

5.4 Experimental Results

In this section, we first evaluate each thermal-aware throughput maximization algorithm (i.e. CDTM, VS-DTM and ERDTM) by comparing their ability to control the processor temperature under a temperature constraint. Then, we analyze their performance by analyze the throughput, which is obtained with each thermal management algorithm.

5.4.1 Experiment Setup

All experiments were carried out with the same ambient temperature and initial chip temperature to get credible and comparable experiment result. Eight benchmarks, which include both integer operation and floating point operation, have been selected

Table 5.1: Lookup table for SPEC 2000 benchmarks

	Frequency levels (GHz)				
Benchmarks	2.6GHz	2.4GHz	2.2GHz	2.1GHz	1.8GHz
galgel	64^oC	61^oC	59^oC	57^oC	54^oC
ammp	59^oC	55^oC	53^oC	49^oC	48^oC
lucas	54^oC	51^oC	48^oC	46^oC	44^oC
equake	57^oC	53^oC	49^oC	45^oC	43^oC
vpr	60^oC	57^oC	53^oC	50^oC	48^oC
gcc	61^oC	58^oC	54^oC	51^oC	47^oC
parser	60^oC	57^oC	$55oC$	51^oC	47^oC
crafty	57^oC	55^oC	51^oC	48^oC	45^oC

from the SPEC CPU2000 benchmark suit (i.e. $vpr, gcc, parser, crafty$ from integer operation and $galgel, ammp, lucas, equake$ from floating point operation). In our work, the temperature threshold has been selected at 55^oC.

We first built up the lookup table by running each benchmark with different working frequencies. The corresponding peak temperatures and speed levels are recorded as shown in Table 5.1. As discussed in section 5.3, Table 5.1 shows running different applications can result in distinctly different stable temperatures even with the exactly the same speeds. This clearly demonstrates the limitations of many theoretical results.

5.4.2 DTM Algorithm Performance Evaluation

As we mentioned before, when temperature exceeds the critical temperature limits, it could cause serious hardware damage, and even crush the entire computing system. Thus, effectively maintaining the processor temperature under the temperature threshold is the highest priority for the thermal-aware scheduling algorithm. To evaluate how effectively each algorithm can control the temperature, we ran our benchmarks with all three approaches on our hardware platform. Processor temperature exceeding the threshold is defined as a temperature violation. Then the total number

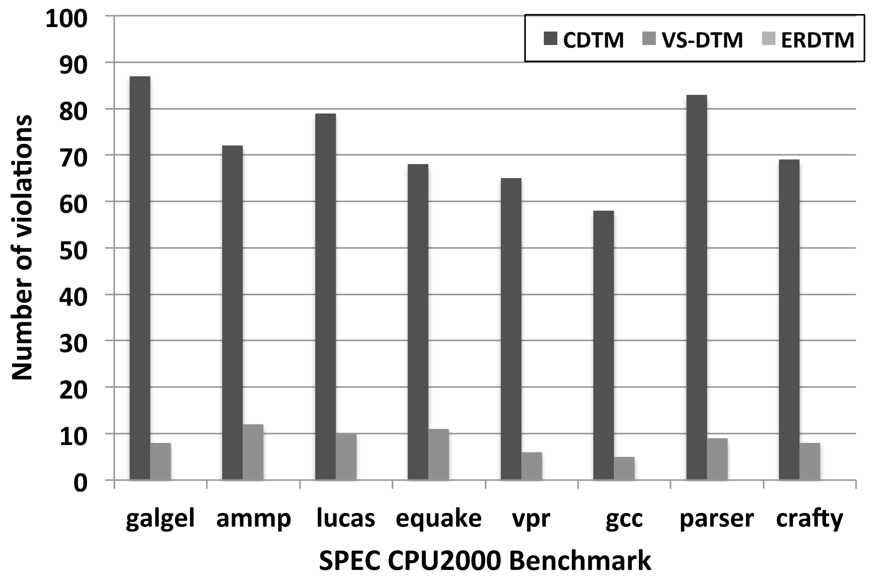


Figure 5.4: The number of temperature violations after implementing different thermal management algorithms

of temperature violations derived from each thermal-aware schedule (i.e. CDTM, VS-DTM and ERDTM) are plotted in Figure 5.4. The experiment result shows that after implementing the CDTM algorithm, the average number of violations is 72 during the whole process. However, after we implemented the VS-DTM scheduling algorithm, the number of temperature violations can be reduced as much by as 88% in average. Another important observation is that our ERDTM algorithm can effectively eliminate the temperature violation and keep the temperature under the threshold.

To further study the details of how ERDTM and VS-DTM can outperform CDTM in reducing the number of temperature violations, we plotted the temperature trace when running benchmark *galgel* with each of the three algorithms. From the experimental result in Figure 5.5, running the benchmark with the CDTM algorithm can cause excessive temperature violations. This is mainly due to two reasons: first, because of the hardware limitations, the thermal sensor cannot keep up with the rapid temperature changes timely. Thus, the temperature continues to rise even if it is already close to the temperature limit; Second, the CDTM algorithm constantly decreases or increases the processor speed level, no matter whether the temperature changes or not. To this end, the CDTM algorithm does not provide enough time for the temperature to reach a stable state before changing it to a new speed level. On the other hand, as shown in Figure 5.6, the VS-DTM algorithm can significantly reduce the number of temperature violations, simply because it can detect the temperature change more timely. The experiment results show that the temperature oscillation range has also been reduced by as much as 50% after implementing the VS-DTM algorithm. However, there are still several spikes (i.e. temperature violations) as shown in Figure 5.6. This is due to the fact that VS-DTM has to adjust the working frequency one level at a time, and sometimes the frequency adjustment cannot keep up with the rapid temperature changes. In contrast, Figure 5.7 shows that our proposed ERDTM algorithm can perfectly maintain the temperature under the threshold without any temperature violations. After doing the offline thermal profile analysis, the temperature lookup table can perfectly guide the ERDTM algorithm to manage the processor temperature.

5.4.3 DTM algorithm Throughput Analysis

In the previous subsection, we already evaluated the efficiency of each algorithm on controlling the temperature under the threshold. We further analyze those thermal-aware schedules with the throughput maximization perspective. We execute the same benchmarks with all of the algorithms, the execution time from each individual approach are recorded for comparison. The execution time for each benchmark with different algorithms are normalized and plotted in Figure 5.8. It clearly shows that the CDTM algorithm always take the longest time to finish a benchmark. It is simply because of two reasons: first, as discussed in the previous subsection, the CDTM algorithm lacks accuracy in maintaining the temperature under the threshold. Thus,

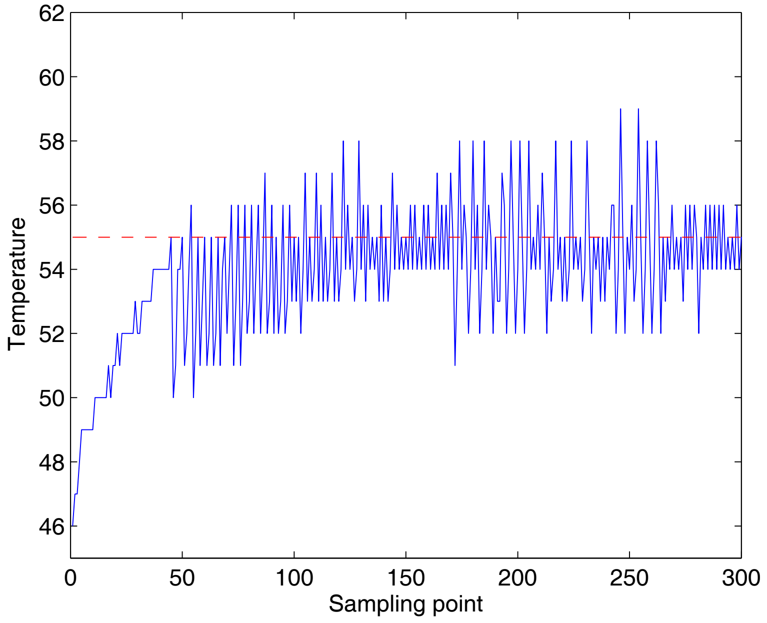


Figure 5.5: Massive temperature violations occur with the CDTM approach

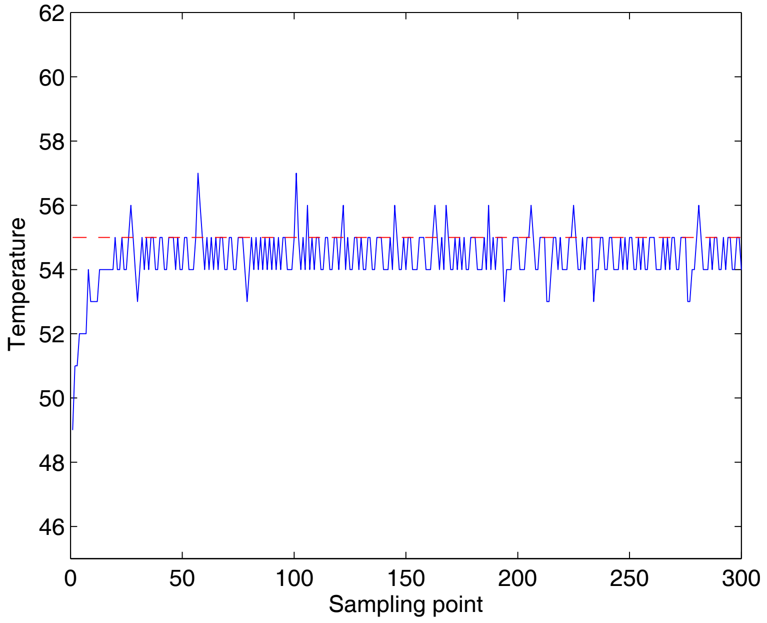


Figure 5.6: Temperature violations has been significantly reduced with the VS-DTM technique

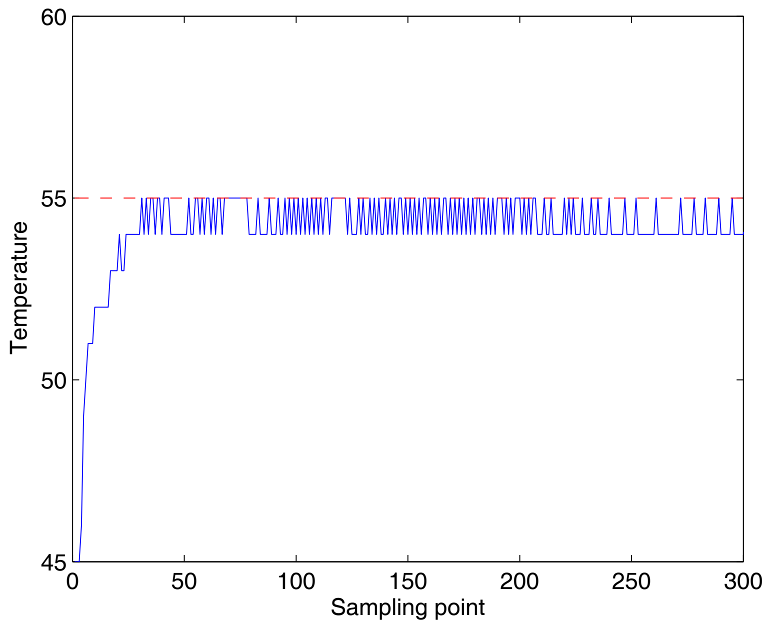


Figure 5.7: ERDTM algorithm completely eliminates peak temperature violation

the processor speed is over reduced to avoid temperature violations. As a result, the throughput is decreased. Our experiments show that the processor temperature can be lower than the threshold by as much as 4^oC. Second, it adjusts the processor frequency constantly. The massive frequency-switching overhead caused by the DVFS technique increased the overall execution time.

However, after implementing the VS-DTM algorithm, the system throughput has been improved by as much as 2.4%. Compared to the CDTM algorithm, the VS-DTM algorithm can respond to the temperature changes more quickly. Because of the close correlation between the temperature and throughput, the VS-DTM can achieve a better throughput performance. Figure 5.8 shows that the ERDTM algorithm has the best throughput performance. The experimental result shows that it improves the execution time of CDTM by 8.1% in average. The improvement comes from the fact that we use the maximum speed to run the benchmark when the temperature is in the safe region, which outperforms the cases when the processor speed has to

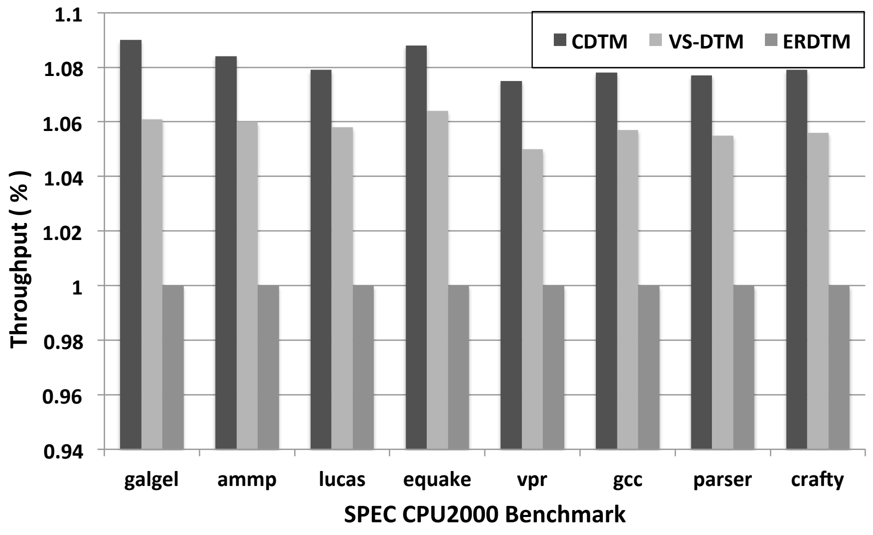


Figure 5.8: Throughput comparison with different thermal management algorithms

be increased one level at a time for every sampling period. Also, when the temperature enters the temperature buffer region, we use the safe speed to further maximize its throughput and ensure the temperature constraint at the same time. Overall, the experiment results show that the ERDTM algorithm not only can precisely control the processor temperature under a pre-defined temperature limit but it also can significantly improve the throughput for the computing system.

5.5 Summary

The dynamic thermal management is becoming one of the most effective techniques to address the thermal issue for the computing system. Carefully maintaining the processor temperature under a critical temperature limit to avoid fatal hardware damage has already become an important research problem. In this chapter, we first analyzed the limitations of the theoretical thermal management algorithms on the practical hardware platform. To further address those limitations, we proposed our DVFS-based thermal aware scheduling algorithms ERDTM to maximize the throughput under a given peak temperature constraint.

CHAPTER 6
NEIGHBOR-AWARE DYNAMIC THERMAL MANAGEMENT FOR MULTI-CORE PLATFORMS

In the previous chapter, we proposed a reactive thermal-aware throughput maximization algorithm for the single core microprocessor platform. However, the reactive thermal management techniques heavily reply on the processor temperature information, that is obtained either from the thermal model or the thermal sensors, to trigger the thermal management operation. This special characteristic can cause a series of problems and directly affect the overall system performance. For instance, because of the manufacture variance and the sensor placement, the on chip thermal sensor lacks accuracy. In addition, even though the sensor can detect the thermal emergency timely, it still takes time for the algorithm to respond to the temperature change. To address the problems above and extend our research to the multicore platform, we first developed a temperature prediction technique, which can take the heat transfer from the neighboring cores into consideration to accurately detect the temperature emergency. Based on this temperature prediction technique, we propose a thermal management algorithm, which can significantly optimize the computing system throughput without exceeding the temperature limit.

6.1 Related Work

Fueled by the market need for high computation capability, the size of transistors is continuously shrinking, and more and more transistors are integrated into a single chip to build up more complicated circuit architectures, i.e. chip multiprocessors (CMPs). As a result, the power density within the chip and the heat generated by transistors increase rapidly in CMPs. Thus, power and thermal issues become major challenges for the further improvement of computing performance on CMPs.

The rapidly growing heat generation greatly increases the packaging and cooling costs, and adversely affects the life-span, performance, and reliability of a computing system. The increased heat dissipation can cause thermal failures, even permanent physical damage to the processor. Therefore, developing an effective thermal management solution is highly desirable, not only to balance the chip's temperature but also to enable the computing system to operate at a high computing performance without exceeding its temperature limit.

The dynamic thermal management technique (DTM) is one of the most effective approaches to address the power and thermal design problems. Many theoretical works have been done by using the dynamic voltage and frequency scaling (DVFS) technique [70, 104, 59, 58], which can control the temperature by dynamically adjusting the processor speed based on the workload. For example, Chantem [18] et al. proposed an algorithm to run real-time tasks under temperature constraint by switching two available speeds neighboring of the ideal speed. However, DVFS techniques sacrifice the performance to cool down the temperature. Task migration is an alternative technique to manage the temperature by balancing the workload among CPU cores without slowing down the processing speed [131, 58, 36, 68]. For example, Gomaa [37] et al. proposed a reactive task migration algorithm, that migrates the task away from overheated core to the coolest core. However, most theoretical thermal management algorithms are based on simplified models and assumptions, such as the assumption that the accurate temperatures of processors are readily available, which is not necessarily true on a real-life platform.

When DTM techniques are applied for real applications, they must deal with important practical details in the practical environment. To this end, many researches have been carried out based on practical hardware platforms [69, 110, 7, 38, 123, 49]. For example, Yefu [122] et al. proposed a chip-level power management algorithm by using control theory and implemented their algorithm on an Intel Xeon desktop. Ahn [4] et al. developed and validated a heuristic to reduce the power consumption and control the temperature on the Intel Centrino Duo and ARM-11 MPCore platforms. The algorithms above rely on the thermal sensor reading to trigger their DTM actions. Since the thermal sensor lacks accuracy due to their placement location and long latency, the effectiveness of the DTM techniques can be severely degraded. Even if the thermal sensor can accurately detect a thermal emergency when the temperature reaches the threshold, it still takes 100 to 200 millisecond for the DTM manager to decrease the frequency or migrate the hot task to a different processor [59]. As a result, the temperature would exceed the threshold before the algorithm takes effect. To this end, predicting the potential thermal emergency before thermal failure occurring is a very important feature for the DTM algorithm [36]. In response to this, Inchoon [131] et al. proposed a temperature prediction algorithm that takes the application's thermal behavior into consideration. Khan [58] et al. developed an alternative thermal management schedule that combined temperature history based prediction and task migration techniques to efficiently control the CPU temperature under threshold. However, they assume that at each sampling point, the temperature will increase at the same rate until it reaches the threshold, which is not true in a real-life scenario.

In this Chapter, we develop an on-line predictive thermal management algorithm to maximize the throughput on multi-core systems while satisfying the peak temperature constraint. Compared with the previous work, we make three major contributions in this work:

- We develop a temperature prediction method, which can predict the temperature of a core more accurately by taking its temperature as well as the neighboring impacts into consideration.

- We develop a new task migration strategy. While it has been a common approach to migrate tasks from the hottest to the coolest core, our approach chooses the destination core differently. We choose the destination core not only by its current temperature, but also by the temperature trends as well as the neighboring impacts as well.

- We validate our algorithm on a practical hardware test bed, i.e a desktop workstation with an Intel i5 750 quad core microprocessor. The experimental results show that our proposed algorithm can significantly outperform the conventional approach.

6.2 Preliminary

We first use a motivation example to illustrate the the importance of heat transfer from neighboring processor, then we formulate our research problem.

6.2.1 Motivation Example

Before we propose our neighbor-aware temperature prediction technique, we first use a motivation example to study how significant the temperature from neighboring processors can affect processor temperature and why we should take this factor into consideration when developing temperature prediction algorithm. As we mentioned in the previous chapter, the processor heat dissipation comes mainly from the power consumed by the processor. However, there is another important heat resource, which comes from the neighboring processor cannot be ignored. Since the number of transistors and cores that are integrated into the CMPs chip, the power density rapidly increases. Each processor also receives heat transferred from its surrounding neighbor. This heat can also heat up a processor, even though it is not running at the high power status.

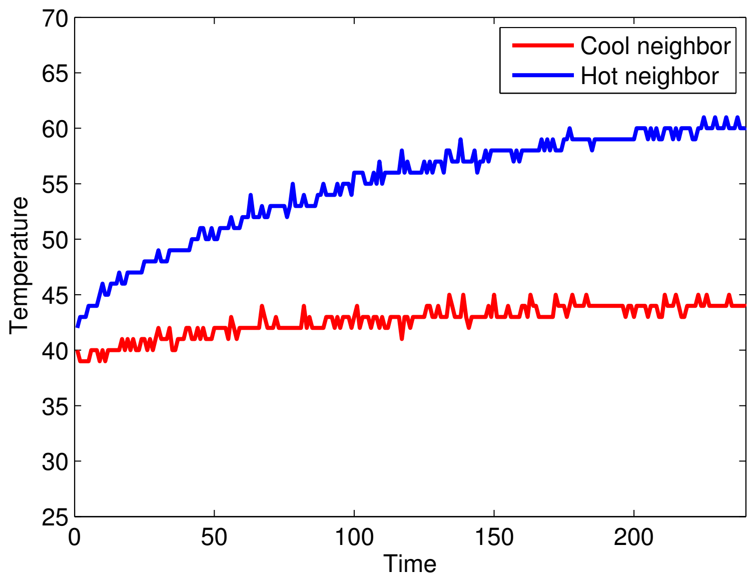


Figure 6.1: Temperature trace with Hot and Cool neighbor processors

To illustrate this scenario, we executed one set of experiments to study how different neighbor environments can affect the processor temperature. First, we selected one core of our multi-core platform, for which its working frequency was set to the minimal speed level without executing any benchmark. Then, we collected the temperature trace of this idle processor with two different neighbor environments. With the **Hot neighbor** environment, the surrounding processors have been assigned with the highest working frequency and executing a *hot* process to create a high temperature neighbor environment. On the other hand, with **Cool neighbor** environment, all the neighbor processors have been assigned with low working frequency running a *cool* task. The temperature information of the idle core with two different neighbor environments are collected and plotted in Figure 6.1. The experimental result clearly shows that even when the idle processor does not execute any process, the heat transfer from the neighboring processor can also heat up its temperature by as much as 18^oC (i.e. 61^oC at the stable state). In contrast, the idle core temperature only increased 5^oC (i.e. 45^oC at the stable state) with the cool neighbor environment.

The above motivation example clearly demonstrated how significant the neighbor environment could affect the processor temperature. To this end, the heat transfer from the surrounding processors must be taken into consideration when developing the thermal management algorithms.

6.2.2 Problem Description

To address the research problems, the system considered in this work consists of N tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ and M identical processors, denoted as $\mathcal{P} = \{P_1, P_2, ..., P_M\}$. The problem discussed in this chapter is how to manipulate the scheduler such that the throughput of the system can be maximized and the processor temperature can be maintained under peak temperature constraint, $T^{THRESHOLD}$. The formal description of the problem is represented below.

***Problem Description*:** Given a task set Γ and a multi-core system $\mathcal{P}$, maximize the throughput of the system under the peak temperature constraint.

For processor P_i, we use a tuple (T_i, t_i) to represent the temperature of P_i at a certain time point t_i. To be more specific, we use T_i^{curr} and T_i^{prev} to denote P_i's current temperature and previous temperature respectively, while t_i^{curr} and t_i^{prev} are the corresponding time.

In this chapter, we developed a heuristic to solve the above problem based on task migration and DVFS technique. We first introduce two temperature prediction methods, which can predict the future temperature of a processor core by considering both local temperature history and neighbors' effect. Next, once a potential risk is detected under our temperature prediction model (i.e. the predicted temperature is over the threshold), we dynamically manage the executions of corresponding tasks on that core by either migrating the hot task to other candidate cores or changing the corresponding working frequency on that core based on the temperature status.

By considering the neighbors' current temperature and their temperature changing trends, we can select a processor among all available candidates to improve the total system performance from a global and long-term perspective.

6.3 Neighbor-Aware Temperature Prediction

As we discussed in Section 6.1, the reactive approach might not precisely react with the temperature change due to the latency cased by the dynamic thermal management techniques, such as reading the thermal sensor, changing processor working frequency or migrating the task from hot core the a cooler core. Thus, an effective temperature prediction heuristic, which can accurately detect the temperature emergency, is highly demanded. In this section, we introduce our neighbor-aware temperature prediction techniques.

6.3.1 Temperature Prediction Model

In this subsection, we introduce the temperature prediction model, which takes the heat transfer from the neighboring processors into consideration. It can accurately predict the future temperature of a core as well as its future trend. First, we introduce the following definitions to represent the future local temperature increment of each processor individually.

Definition 6.3.1. *Given processor P_i, the* local increment *factor of P_i, denoted as $\mathcal{I}_i^{in}$, is defined as*

$$\mathcal{I}_i^{in} = T_i^{curr} - T_i^{prev}. \tag{6.1}$$

This local temperature increment will be used to predict the future temperature at the next sampling point. Figure 6.2 shows an example of the temperature trace of a running process. We make the assumption that the temperature will keep increasing at the same rate if the sampling interval is extremely small (i.e. Δt is the sampling

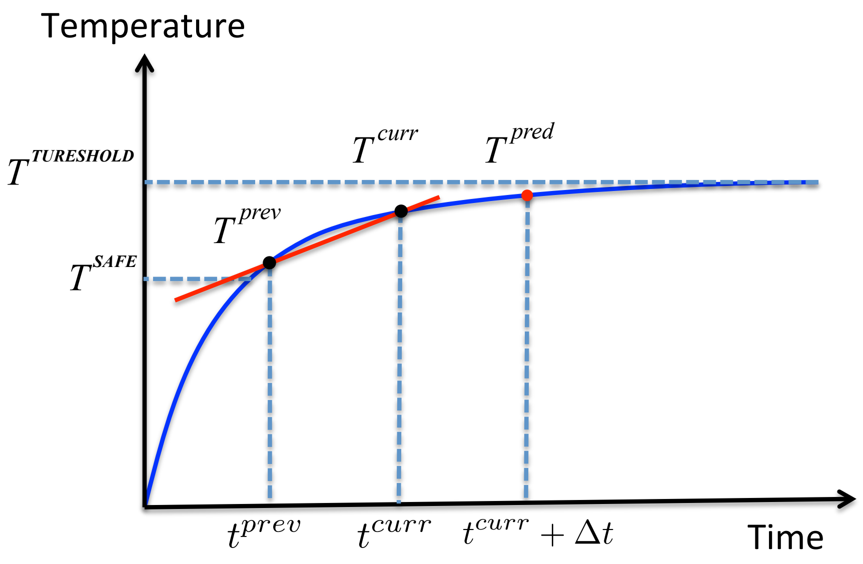


Figure 6.2: Temperature history based prediction

period). The current temperature T^{curr} is obtained from the thermal sensor. It has been used to predict the temperature at the next sampling point T^{pred} with the most recent temperature history T^{prev}. In our work, the sampling period has been set to 1 second, since this is approximately how long it takes for the thermal sensor to reflect a temperature change [1].

As we discussed in Section 6.2.2, besides the heat generated by the processor itself, its temperature is also affected by other processors on the same chip. In this work, we define the *neighbor processors* of a processor P_i, denoted as $\mathcal{P}_i^{NB}$, as the cores which are adjacent to P_i. When predicting the temperature of a processor, we only consider the heat transfer impacts from its neighboring processors to simplify our algorithm. By considering the effect of neighbor processors, we define the following two concepts to represent the neighbors' thermal effect for a given processor. The first concept, i.e. ***neighbor average*** factor, represents the average temperature of all neighbors. The second concept, i.e. ***neighbor increment*** factor, represents the temperature increment trend of all neighbors. Two concepts are formally defined as follows.

Definition 6.3.2. *Given any processor P_i, the* neighbor average *factor of P_i, denoted* *as $\mathcal{A}(P_i)$, is defined as*

$$\mathcal{A}(P_i) = \frac{\sum_{P_j \in \mathcal{P}_i^{NB}} T_j^{curr}}{|\mathcal{P}_i^{NB}|} \tag{6.2}$$

Definition 6.3.3. *Given any processor P_i, the* neighbor increment *factor of P_i, denoted as $\mathcal{I}(P_i)$, is defined as*

$$\mathcal{I}(P_i) = \frac{\sum_{P_j \in \mathcal{P}_i^{NB}} (T_j^{curr} - T_j^{prev})}{|\mathcal{P}_i^{NB}|} \tag{6.3}$$

According to Definition 6.3.3, $\mathcal{I}(P_i)$ represents the average temperature increment of P_i's neighboring processors. In other words, the neighbor increment factor describes the temperature increment speed for each processor's neighbors.

Consider processor P_i, the temperature increment caused by P_i's neighbors can be calculated as following

$$I_i^{nb} = \gamma_1 \cdot \mathcal{A}(P_i) + \gamma_2 \cdot \mathcal{I}(P_i), \tag{6.4}$$

where γ_1 and γ_2 are the weights of $\mathcal{A}(P_i)$ and $\mathcal{I}(P_i)$, respectively, which can be obtained from doing off-line analysis. The detailed implementation will be introduced in the later section.

With the above definitions, we are now ready to introduce our temperature prediction model. Let T_i^{pred} denote the predicted temperature for P_i. We formulate T_i^{pred} as a linear function of its current temperature T_i^{curr}, its local temperature increment rate I_i^{in}, and also its neighbor effect factor I_i^{nb}, as shown below:

$$T_i^{pred} = \alpha_i \cdot T_i^{curr} + \beta_i \cdot \mathcal{I}_i^{in} + \gamma_i \cdot \mathcal{I}_i^{nb}, \tag{6.5}$$

where α_i, β_i and γ_i are weight parameters for P_i.

In addition, to make our prediction model more accurate, we take different pro-

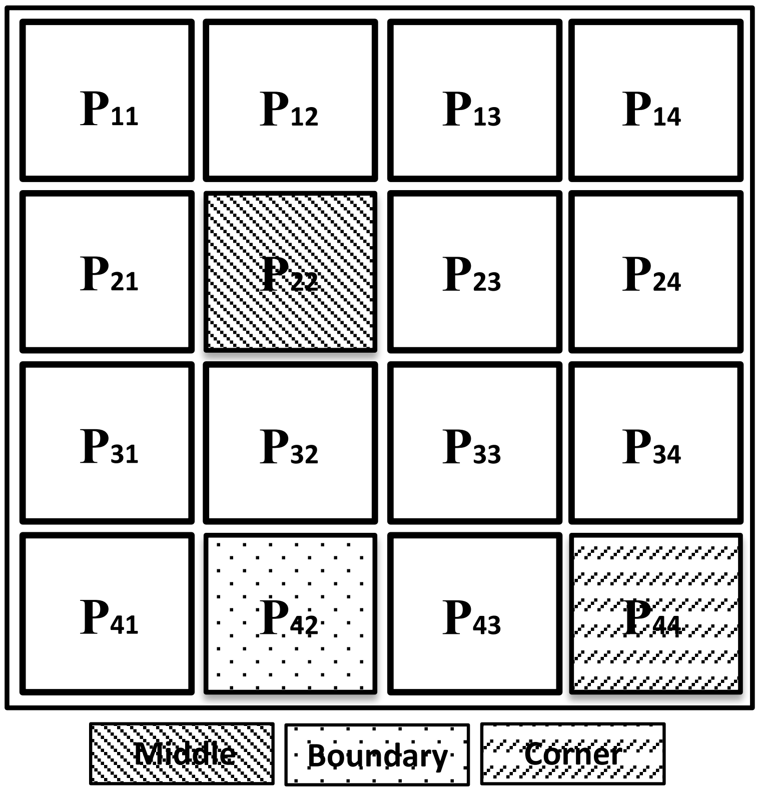


Figure 6.3: Different processor location scenarios

cessor location scenarios into consideration. Each processor with a different number of neighboring cores has different neighbor effects as shown in Figure 6.3. Thus, the temperature prediction for a task τ_i can be categorized into three cases: 1) τ_i runs on a corner processor; 2) τ_i runs on a boundary processor; 3) τ_i runs on a middle processor. Then we discuss the neighbor effect for τ_i by using matrix.

Temperature prediction base $\hat{T}_i^B$ is a 3×1 vector:

$$\hat{T}_i^B = [T_i^{curr},\ I_i^{in},\ I_i^{nb}]^T.$$

Based on the different cases of processor position (i.e. corner, boundary and middle), temperature prediction weights for different scenario can be expressed as following.

- weights for corner scenario:

$$w_i^c = [\alpha_i^c,\ \beta_i^c,\ \gamma_i^c].$$

- weights for boundary scenario:

$$w_i^b = [\alpha_i^b,\ \beta_i^b,\ \gamma_i^b].$$

- weights for middle scenario:

$$w_i^m = [\alpha_i^m,\ \beta_i^m,\ \gamma_i^m].$$

Combine all three scenarios of τ_i together, we have

$$W_{i\ 3\times 3} = [w_i^c,\ w_i^b,\ w_i^m]^T.$$

The above temperature prediction model already took the processor neighbor effect, as well as the different task locations into consideration. Thus, based on this accurate model, we proposed two temperature prediction algorithms in the following section.

6.3.2 Neighbor-Different Prediction

In this subsection, we introduce a *neighbor-different temperature prediction* (NDTP) algorithm, which considers all the different scenarios of neighbor processor condition as discussed in the previous subsection. We conduct the temperature prediction matrix, i.e. $\hat{T_i}$ to represent the temperature prediction result for task τ_i.

$$\hat{T_i}_{\,3\times1} = [T_i^c,\ T_i^b,\ T_i^m]^T,$$

where T_i^c, T_i^b and T_i^m are the temperature prediction results for corner processor, boundary processor and middle processor, respectively.

For each item of $\hat{T_i}$, i.e. T_i^x, $x \in [c, b, m]$, the temperature can be calculated by

$$T_i^x \ = [\alpha_i^x,\ \beta_i^x,\ \gamma_i^x] \times [T_i^{curr},\ \Delta T_i^{in},\ \Delta T_i^{nb}]^T \ = w_i^x \times B_i, \tag{6.6}$$

thus, we have

$$\begin{bmatrix} T_i^c \\ T_i^b \\ T_i^m \end{bmatrix} = \begin{bmatrix} \alpha_i^c & \beta_i^c & \gamma_i^c \\ \alpha_i^b & \beta_i^b & \gamma_i^b \\ \alpha_i^m & \beta_i^m & \gamma_i^m \end{bmatrix} \times \begin{bmatrix} T_i^{curr} \\ \Delta T_i^{in} \\ \Delta T_i^{nb} \end{bmatrix} \tag{6.7}$$

or

$$\hat{T_i} \ = W_i \times B_i. \tag{6.8}$$

Since we can get the weight matrix W_i off-line, the predicted temperature of τ_i can be obtained on-line by determining the host processor position of τ_i.

The detail flow of *NDTP* algorithm is presented in Algorithm 3. For any processor P_i, the current temperature obtained from the thermal sensor, and the most recent temperature are stored in the temperature history table. They are denoted as T_i^{curr} and T_i^{prev} respectively (i.e. line 1 and line 2). Then, based on the assumption that the temperature will keep changing at the same rate after the next sampling interval, we are able to calculate the local temperature increment from the current and previous temperatures (i.e. line 3). Moreover, the schedule takes the heat transfer from P_i's neighbor processor into consideration by calculating the neighbor effect from equation 6.4 (i.e. line 4). The weight factors can be determined by identifying the processor's location (i.e. line5 and line 6). Then the temperature at the next sampling point can be predicted by using *NDTP* algorithm (i.e. line 7).

Algorithm 3 Neighbor-Different Temperature Prediction

```
1: $T_i^{curr}$ := the current temperature of processor $P_i$;
2: $T_i^{prev}$ := the previous temperature of processor $P_i$;
3: calculate the local temperature increment of $P_i$ after $\Delta t$ by
       $I_i^{in} = \frac{T_i^{curr} - T_i^{prev}}{t_i^{curr} - t_i^{prev}} \cdot \Delta t$;
4: calculate $P_i$'s neighbors effect $I_i^{nb}$ based on equation (6.4)
5: $x$ = determine the location of processor, corner, boundary or middle;
6: determine the weight of $\tau_i$ under mode $x$ such that
       $w_i^x = [\alpha_i^x\ \beta_i^x\ \gamma_i^x]$;
7: predict the future temperature of $P_i$ by
       $T_i^{pred} = w_i^x \cdot B_i$
   where $B_i = [T_i^{curr}\ I_i^{in}\ I_i^{nb}]$;
```

6.3.3 Neighbor-Normalized Prediction

Instead of categorizing the processors into three categories and generating three different groups of weight, we propose a *neighbor-normalized temperature prediction* (NNTP) algorithm to reduce the complexity for temperature prediction by applying the least-square estimation [131] to derive one uniform and normal weight matrix for all three different neighbor cases.

For any task τ_i, from equation (6.5), we know that the temperature prediction problem is formulated by

$$T_i^{pred} = \alpha_i \cdot T_i^{curr} + \beta_i \cdot \mathcal{I}_i^{in} + \gamma_i \cdot \mathcal{I}_i^{nb}.$$

To map the above temperature prediction problem into a general least-square problem, we construct a linear model for the output T^{pred} by the following linear parameterized expression

$$T^{pred}(t) = \alpha \cdot T^{curr}(t) + \beta \cdot I^{in}(t) + \gamma \cdot I^{nb}(t),$$

where $t = [t_1, t_2, t_3]$ is the model's input vector, $T^{curr}(t)$, $I^{in}(t)$ and $I^{nb}(t)$ are known functions of t, and α, β and γ are unknown parameters to be estimated. Let $\hat{T}$ represent $[T^{curr}(t), I^{in}(t), I^{nb}(t)]$, and $\hat{W}$ represent $[\alpha,\ \beta,\ \gamma]$. In our model, let t be time units, and can be chosen from three different scenarios with respect of neighbor processor condition, i.e. $t \in [t^c, t^b, t^m]$, where t^c, t^b, t^m represent the scenarios for corner, boundary and middle processor respectively.

To identify the unknown parameters, $\hat{W}$, experiments usually have to obtain a training data set $(T_j^{pred}(t); T_j^{curr}(t), I_j^{in}(t), I_j^{nb}(t))$, where $j = 1, ..., n$. Expressed in matrix notation, the following equation can be obtained:

$$\hat{T}^{pred} = \hat{T} \ \times \hat{W} \ ,$$

where $\hat{T}$ is a 3×3 matrix:

$$\hat{T} = \begin{bmatrix} T^{curr}(t^c) & I^{in}(t^c) & I^{nb}(t^c) \\ T^{curr}(t^b) & I^{in}(t^b) & I^{nb}(t^b) \\ T^{curr}(t^m) & I^{in}(t^m) & I^{nb}(t^m) \end{bmatrix} \tag{6.9}$$

$\hat{W}$ is a 3×1 unknown weight parameter vector:

$$\hat{W} = [\alpha,\ \beta,\ \gamma]^T, \tag{6.10}$$

and $\hat{T}^{pred}$ is a 3×1 output vector:

$$\hat{T}^{pred} = [T^c,\ T^b,\ T^m]^T. \tag{6.11}$$

If $(\hat{T}^{pred})^T \hat{T}^{pred}$ is nonsingular, the least square estimator can be derived as

$$\hat{W} = (\hat{T}^T \hat{T})^{-1} \hat{T}^T \hat{T}^{pred}. \tag{6.12}$$

Eventually, we predict the future temperature by applying equation (6.5), with the corresponding task-based weight parameter obtained by equation (6.12).

Algorithm 4 Neighbor-Normalized Temperature Prediction

```
1: $T_i^{curr}$ := the current temperature of processor $P_i$;
2: $T_i^{prev}$ := the previous temperature of processor $P_i$;
3: calculate the local temperature increment of $P_i$ after $\Delta t$ by
        $I_i^{in} = \frac{T_i^{curr} - T_i^{prev}}{t_i^{curr} - t_i^{prev}} \cdot \Delta t$;
4: calculate $P_i$'s neighbors effect $I_i^{nb}$ based on equation (6.4)
5: get the weight parameter $w_i$ for current task, $w_i^x = [\alpha_i^x, \beta_i^x, \gamma_i^x]$;
6: predict the future temperature of $P_i$ by
        $T_i^{pred} = w_i \cdot B_i$
   where $B_i = [T_i^{current}, I_i^{in}, I_i^{nb}]$;
```

The *NNTP* prediction algorithm could be described in the similar expression as algorithm 3. First the schedule calculates the local temperature increment and the neighbor effect (i.e. line1-3). Next, instead of identifying the task location, the *NNTP* prediction algorithm use a least-square estimation method to calculate the weight parameters from equation (6.4) (i.e. line 4). Then the expected temperature at the next sampling point can be predicted (i.e. line 5-6).

6.4 Proactive Algorithm

With the temperature prediction algorithms, which are proposed in the previous section, we are able to detect the thermal emergency in advance and leave enough time for the computing system to react to the temperature change. Thus, in this section, we first introduce the algorithm that we used to select the candidate processor to implement the task migration. Then, we give a detailed introduction of our neighbor-aware dynamic thermal management (NADTM) algorithm.

6.4.1 Candidate Processor for Migration

When the thermal emergency is detected by the temperature prediction technique, one solution is to migrate the task away from the hot processor to bring down the temperature. To identify the appropriate destination, one common approach [37] is to migrate the task to the processor with the lowest current temperature. However, selecting the coolest processor is not always the best decision. Due to the sudden neighboring processor temperature change or the potential of the big temperature increasing rate by itself, the coolest core can rapidly become a hotspot after the next sampling interval. Thus, to address this problem in our approach, besides the current temperature of the candidate processor, we consider its neighboring temperatures, as well as its temperature changing rate to make the migration decision.

We first introduce a concept, *heat index*, to quantify how hot a candidate processor (i.e. P_k) is.

Definition 6.4.1. *Given processor P_k, the* heat index *of P_k, denoted as $\mathcal{H}(P_k)$, is defined as*

$$\mathcal{H}(P_k) = \frac{\sum_{P_j \in \mathcal{P}_k^{NB} \bigcup \{P_k\}} T_j}{|\mathcal{P}_k^{NB} \bigcup \{P_k\}|}. \tag{6.13}$$

Intuitively, the smaller the heat index of a processor is, the better the candidate processor it can be.

Besides the heat index of a processor, we also consider the temperature changing rates of itself as well as its neighbors. We present the following definition, i.e. the *heat index increasing factor* of a processor P_k, to capture this concept.

Definition 6.4.2. *Given processor P_k, the* heat index increasing factor *of P_k, denoted as $\mathcal{I}(P_k)$, is defined as*

$$\mathcal{I}(P_k) = \frac{\sum_{P_j \in \mathcal{P}_k^{NB} \bigcup \{P_k\}} \frac{T_j^{curr} - T_j^{prev}}{t_j^{curr} - t_j^{prev}}}{|\mathcal{P}_k^{NB} \bigcup \{P_k\}|}. \tag{6.14}$$

According to Definition 6.4.2, $\mathcal{I}(P_k)$ indicates how fast the temperature at P_k and its neighbors can increase in average. Thus, the smaller the heat index increasing factor, the better the candidate processor can be. From equation (6.13) and (6.14), we choose the migration candidate as the one that minimizes

$$\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t, \tag{6.15}$$

where Δt is the length of the sampling interval.

Note that task migration is not always effective in dealing with a thermal emergency, especially when the workload is heavy. Given a processor P_k in thermal emergency, it does not help much if the selected target processor (e.g. P_k) for migration has a temperature very close to the peak temperature limit, even if the $\mathcal{H}(P_k)+\mathcal{I}(P_k)\cdot\Delta t$ is minimum among all other processors. Besides, too many unnecessary task migrations may cause redundant context switch overhead, which could degrade throughput performance. To avoid this scenario, in our approach, the tasks on processor P_k are only allowed to migrate to processor P_k if

$$\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t \leq T^{THRESHOLD}, \tag{6.16}$$

where $T^{THRESHOLD}$ is the given temperature constraint. Otherwise, we can adopt an alternative solution to cool down the processor. Such as selecting a safe working speed for the processor P_k by using the same offline thermal profiling analysis approach.

6.4.2 Thermal Management Algorithm

In this subsection, we introduce our proposed thermal management algorithm, the NADTM algorithm, to maximize the throughput of a multi-core system while keeping the temperature under a predefined peak temperature limit.

Algorithm 5 Neighbor-Aware Dynamic Thermal Management (NADTM) Algorithm

```
1: $T_i^{prev} := T_i^{curr}$ // the temperature at previous sampling point ;
2: $T_i^{curr}$ := the temperature of $P_i$ from temperature sensor;
3: $T_i^{pred}$ := predicted temperature of $P_i$ at next sampling point based on equation
   (6.5);
4: if $T_i^{pred} > T^{THRESHOLD}$ then
5:     $P_k$ := the processor from $\mathcal{P}$ such that $\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t$ is minimum;
6:     if $\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t \leq T^{THRESHOLD}$ then
7:         migrate current running tasks on $P_i$ to $P_k$;
8:     else
9:         degrade the performance of $P_i$ by setting its speed to the pre-defined safe
           speed (i.e $S_i^{SAFE}$);
10:    end if
11: end if
```

The NADTM algorithm is presented in Algorithm 5. For processor P_i, we read the temperature sensor to get its current temperature and then predict its temperature at the next sampling point based on the method described in section 6.3. If the predicted temperature exceeds the temperature constraint, we will search for a candidate processor that we can migrate the task to. The candidate processor is selected based on the method presented in section 6.4.1. If such a processor is not available, we select a safe speed from the thermal profile lookup table as discussed in previous chapter.

We assume that the weights in equation (6.5) have been identified off-line. The safe speed to run a processor is essentially the maximum processor speed for a processor such that its peak temperature will not exceed the temperature constraint. We also assume that this speed is obtained off line.

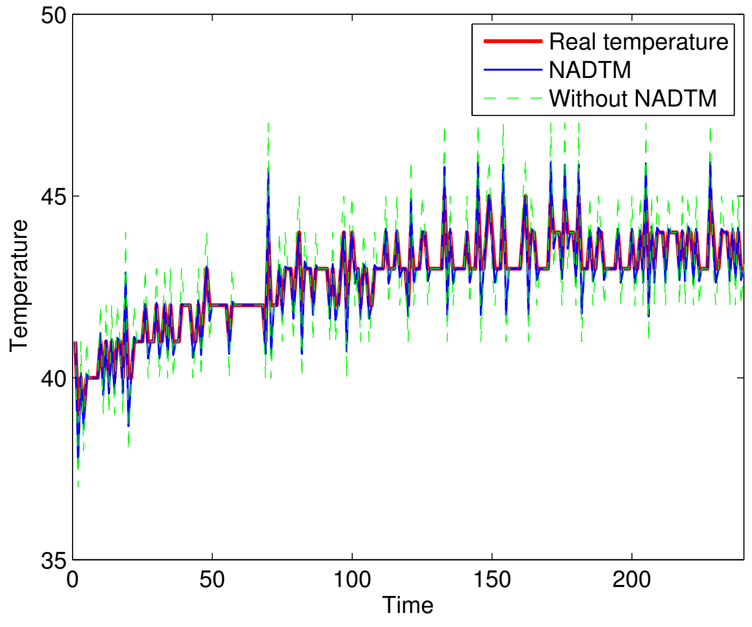


Figure 6.4: NADTM compares with the conventional prediction method, which does not take the neighbor effect into consideration

6.5 Experimental Results

In this section, we first introduce the experiment setup. Then we validate the accuracy of our predictive thermal management technique by comparing it with the enhanced reactive approach. At last, we verify the performance improvement of our algorithm by analyzing the efficiency of the neighbor-aware temperature prediction and migration, respectively.

6.5.1 Experiment Setup

All experiments were carried out with the same ambient temperature. We selected six benchmarks *galgel, parser, ammp, crafty, lucas* and *equake* from the well-known commercial benchmark SPEC CPU2000, including both integer and floating point operation to obtain credible and comparable experiment results. Those benchmarks have been grouped into three categories, which are *hot*, *warm*, and *cool*, based on their thermal characteristics. To build up the temperature lookup table, we conducted the off-line thermal profiling analysis by running each benchmark at different CPU speeds. The stable temperatures with their corresponding speed levels were stored in a lookup table. To ensure the schedule effectiveness, each benchmark was tested with the *hot* benchmark applications running on its neighboring processors. In the lookup table, the safe speed is the maximal speed corresponding to the stable temperature lower than the given temperature constraint.

6.5.2 Prediction Analysis

To evaluate the accuracy of our NADTM temperature prediction technique, we compared our heuristic with the conventional temperature prediction approach that uses the previous and current temperature values of a processor to predict the next temperature value without considering the heat transfer from the neighboring processors.

Figure 6.4 shows the temperature traces of running benchmark *galgel*, as well as the temperature prediction results based on our proposed temperature prediction method and the conventional one. From Figure 6.4, we can clearly see that the temperature prediction results of using the NADTM approach is much closer to the actual temperature value than the conventional approach. Also, the NADTM approach has a smaller maximum prediction error of 1^oC comparing with 3^oC by the conventional approach. The results shown in Figure 6.4 demonstrate that, by taking the heat transfer impacts from the neighboring processors in consideration the temperature prediction methods introduced in section 6.3 can achieve a higher accuracy than the traditional method.

To further validate this conclusion, we ran different benchmark programs on our test platform. First, temperature prediction results are collected and compared with the actual temperature value. Then the temperature prediction accuracy by using

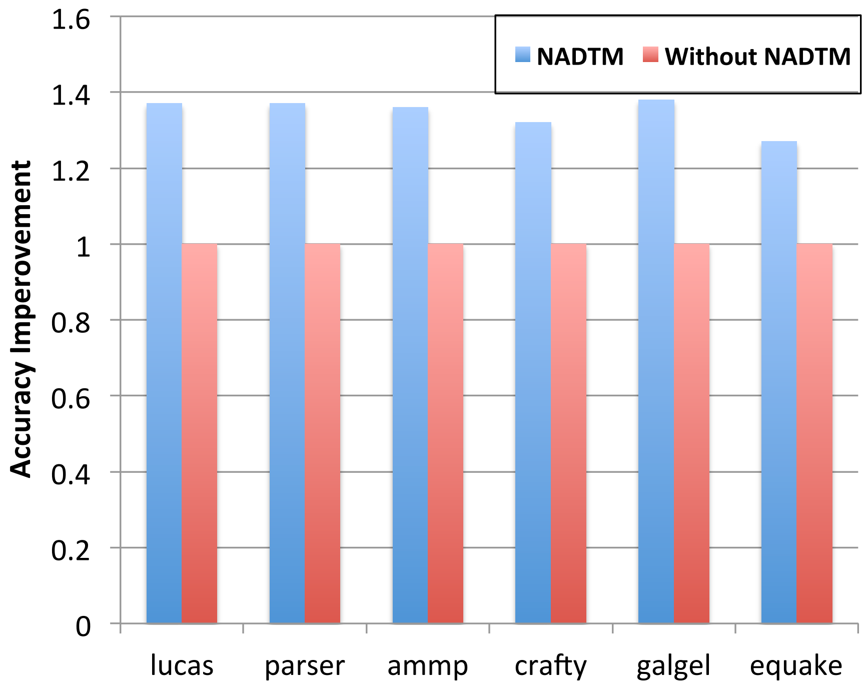


Figure 6.5: Prediction accuracy comparison with different benchmarks

two different prediction methods is plotted in Figure 6.5. The prediction accuracy is the number of accurate predictions over the total number of predictions. In order to compare the two approaches, both results are normalized to the approach without NADTM. From Figure 6.5 we can see that our NADTM approach can improve the temperature prediction accuracy by 38% in average compared to the conventional approach. Based on the above experiment results, our neighboring aware temperature prediction technique could effectively improve the prediction accuracy.

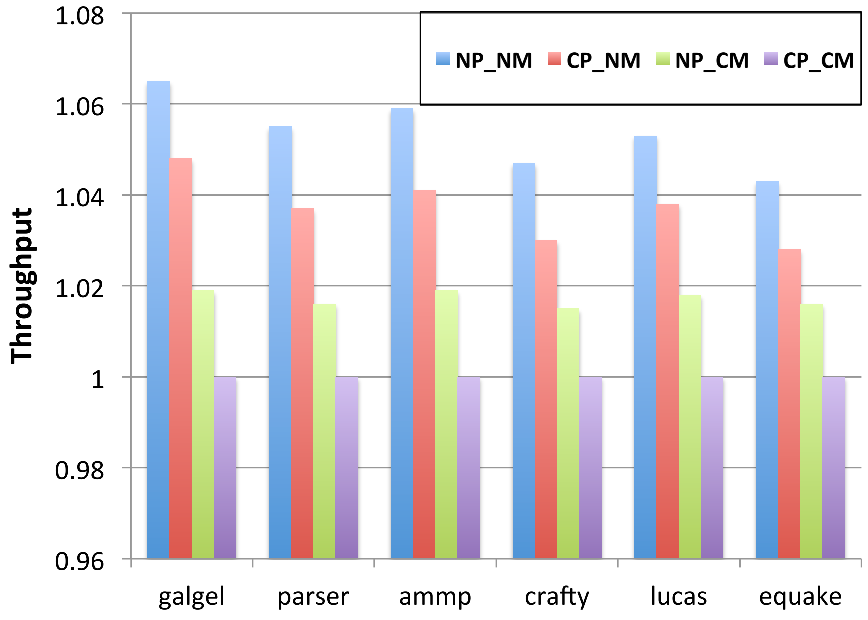
6.5.3 Throughput Analysis

To analyze the throughput of our NADTM algorithm, we only compare it with the proactive approaches. It is due to two reasons: first, in the previous subsection, we already proved that the reactive approach cannot effectively management the processor under the temperature constraint. And preventing thermal violation is the first priority for a thermal-aware scheduling algorithm, thus any thermal violation is not acceptable for a thermal-aware algorithm. Second, because of the close correlation between chip temperature and working speed, the reactive approach has a longer time to push the system temperature over the threshold, which will result in a higher throughput. Thus, it is not justified to compare our approach with the reactive approach, that cannot satisfy the temperature constraint.

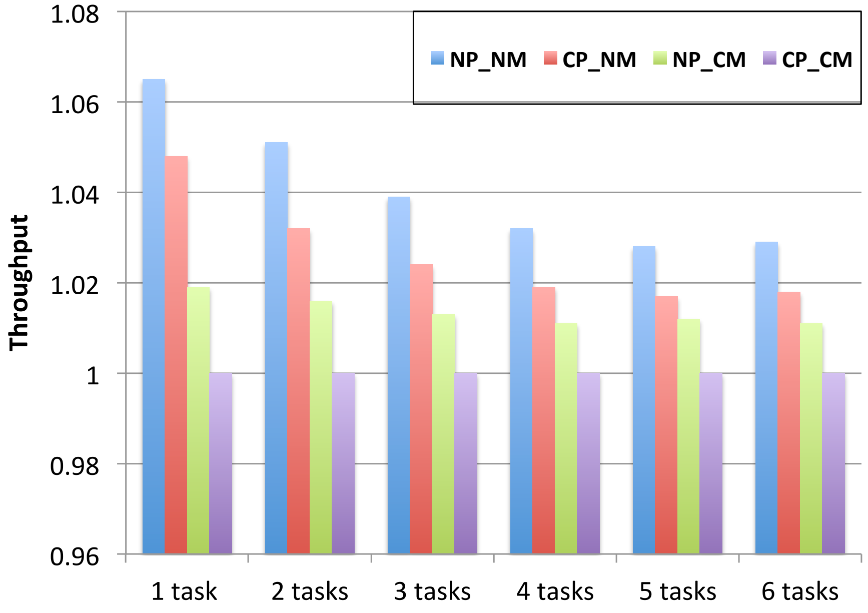
We use *NP, CP* to denote neighbor-aware prediction and conventional prediction, and *NM, CM* for neighbor-aware migration and conventional migration, respectively. The conventional temperature prediction approach refers to the one that predicts the future temperature solely based on its own temperature history. And the conventional migration approach refers to the approach that simply migrates the running tasks from the hottest core to the coolest core. As a result, we have four combinations, *i.e.* *CP_CM NP_CM, CP_NM* and *NP_NM*.

We first compare the throughput of each approach when running a single task on our hardware platform. In this experiment, six previously used benchmarks have been selected to provide reliable experiment results. The execution times by using different approaches have been recorded for comparison, those experiment results have been normalized and plotted in Figure 6.6(a). The results show that, the neighbor-aware prediction algorithm *i.e.* *NP_CM* can improve the throughput over *CP_CM* as much as 1.7% in average. Since our prediction technique is more accurate than the conventional approach as shown before, it helps make better scheduling decision and thus improves the performance. Another observation is that *CP_NM* improves the throughput over *CP_CM* as much as 3.6%. This is because *CP_NM* can find the appropriate migration candidate rather than simply locate the coolest core. By combining our proposed prediction and task migration algorithm together, *NP_NM* can achieve an average of 5.8% overall throughput improvement when compared to *CP_CM*

To further test our thermal management algorithm, we assigned multiple tasks



(a) Execution time comparison by using different single task



(b) Execution time comparison with multiple tasks running on the multiprocessor platform

Figure 6.6: Execution time comparison with four different approaches. *NP and CP represent the neighbor-aware and conventional prediction respectively. NM and CM represent neighbor-aware and conventional migration respectively*

to the multicore platform. By gradually increasing the number of tasks running on the multicore processor, their corresponding execution times have been recorded for comparison. The execution times have been normalized and plotted in Figure 6.6(b). As we can see from the experiment results, the overall throughput decreases as the number of tasks increases. Another important observation is that when the number of tasks is larger than the number of core (*i.e. the number of tasks is more than 4*), the throughput drops significantly. The experiment results show that the throughput for the *NP_CM* decreased by 0.9% while the tasks increased from 1 to 6. The throughput for *CP_NM* decreased by 3%. The throughput decreased by 3.6% for the overall NADTM algorithm. All these results show that the proposed algorithm works better with a lighter workload than a heavy workload.

6.6 Summary

In this chapter, we developed a predictive thermal-aware algorithm for the practical multi-core platform to maximize the system throughput under peak temperature constraint. Our proposed approach takes the neighbor effect into consideration to make a more accurate temperature prediction and to determine a better migration destination. The algorithm has been validated on our multi-core platform, the experiment results illustrate that our thermal management algorithm can significantly improve the system throughput while satisfying the temperature constraint.

CHAPTER 7
CONCLUSIONS

In this chapter, we summarize our research presented in this dissertation and discuss possible future work of this research.

7.1 Concluding Remarks

Due to the increasing demand for higher computation capability, more and more transistors and cores are integrated into a single processor chip. As a result, the power density of the IC chip exponentially increases and generates a large amount of heat. The rapidly growing heat generation greatly increases the packaging and cooling costs, and adversely affects the performance and reliability of a computing system. Besides, the increased heat generation may reduce the processor life span, and even force the computing system to completely shut down to prevent permanent physical damage to the processor. Therefore, developing effective thermal management solutions is highly desirable, not only to balance the chip's temperature but also to enable the computing system to operate at a high computing performance without exceeding its temperature limit.

In this dissertation, we are focusing on developing thermal-aware throughput maximization algorithms for the practical hardware platform. Compared to most of the related works that carried out their research based on simplified model or idealized assumptions, our work can obtain experimental results directly from the actual computing system. Thus, our research is more practical. We developed a real-life hardware platform based on an Intel i5-750 quad-core processor, running the Ubuntu 10.04.1 Linux operating system with kernel version 2.6.32. It has the flexibility to adopt most of the advance thermal management techniques. Thus, most of the theoretical work can be implemented and validated on our platform. Furthermore, we studied how effective the DTM technique can be in a practical hardware environment. Specifically, we investigated the cooling efficiency and computing performance tradeoffs when employing the DVFS technique on our hardware platform, and compared it with the traditional cooling method by running the same benchmark. Furthermore, by implementing different thermal management algorithms on our platform, we validated two widely used thermal management principles in reality. We also analyzed how the cooling solution can affect the performance of the thermal management algorithm by comparing those approaches under different cooling conditions.

We identified several limitations in the assumptions of the existing theoretical researches on our hardware platform. And then we proposed an ERDTM for a single-core processor to maximize the program throughput under a given temperature constraint. Compared to the conventional reactive approach, our ERDTM could detect temperature changes more accurately. In addition, by doing offline thermal analysis, we were able to built up a thermal profiling look-up table, which can guide the ERDTM algorithm by selecting the optimal working frequency to maximize the throughput. Our experimental results show that the ERDTM algorithm can significantly reduce the number of temperature violations by 88%. Also, by comparison with the conventional reactive approach, the overall system throughput can be improved by 8.1%. To further extend our research work from single-core to multicore platform, we proposed a proactive NADTM algorithm with a temperature prediction technique. Compared to the reactive DTM approaches, the proactive DTM algorithm can detect the thermal emergency in advance, and leave enough time for the DTM algorithm to react with the temperature change. Other than simply using the temperature history to predict temperature, our NADTM algorithm takes the heat transfer from neighboring processors into consideration. It can significantly improve the prediction accuracy by 38% compared to the temperature history based prediction method. And the prediction error is as small as 1^oC. In addition, we proved that simply migrating the task from the hottest core to the coolest core is not always the optimal solution. Thus, our NADTM algorithm includes a neighbor-aware task migration technique. The experimental results show that our NADTM can effectively maintain the temperature under the pre-defined temperature limit. The system throughput was improved by 5.8%.

7.2 Future Work

In this dissertation, we have done extensive research work on the dynamic thermal management analysis, especially focusing on developing the thermal-aware throughput maximization algorithm for the practical computing system. However, based on the close correlation between temperature and power. It is very logical for us to extend our research work to the computing system power management analysis.

As we discussed in this dissertation, the exponentially increasing heat dissipation significantly increases the total amount of power used to cool down the temperature. Meanwhile, our experimental shows that the DTM algorithm can outperform the air-cooling system on reducing the processor temperature, however, it has to sacrifice the system performance. On the other hand, the air-cooling does not affect the performance, but will cost extra power consumption. Thus, developing a scheduling algorithm for the practical computing system that would analyze the trade-offs between the cooling energy and the DTM algorithm is highly demanded. Based on the special characteristic of our hardware platform, we are able to measure the computing system power consumption, and we also have the flexibility to adjust the cooling system on the platform. It is very logical for us to extend our work to analyze the cooling energy optimization.

REFERENCES

[1] Lm-sensors linux hardware monitoring: http://www.lm-sensors.org.

[2] Texas instruments. In *CMOS Power Consumption and Cpd Calculation*, 1997.

[3] Y. Ahn, Y.-S. Hwang, and K.-S. Chung. Flexible framework for dynamic management of multi-core systems. In *SoC Design Conference (ISOCC), 2009 International*, pages 237 –240, nov. 2009.

[4] Y. Ahn, Y.-S. Hwang, and K.-S. Chung. Flexible framework for dynamic management of multi-core systems. In *SoC Design Conference (ISOCC), 2009 International*, pages 237 –240, nov. 2009.

[5] AMD. Power and cooling in the data center. page http://enterprise.amd.com, 2007.

[6] P. Bailis, V. Reddi, S. Gandhi, D. Brooks, and M. Seltzer. Dimetrodon: Processor-level preventive thermal management via idle cycle injection. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 89 –94, june 2011.

[7] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer. Dimetrodon: Processor-level preventive thermal management via idle cycle injection. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 89 –94, june 2011.

[8] M. Bao, A. Andrei, P. Eles, and Z. Peng. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *Design Automation Conference*, pages 490–495, 2009.

[9] M. Bao, A. Andrei, P. Eles, and Z. Peng. Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 21 –26, march 2010.

[10] C. Belady. In the data center, power and cooling costs more than the it equipment it supports. *Electronics Cooling*, 23(1), 2007.

[11] Z. Bian and A. Shakouri. Cooling enhancement using inhomogeneous thermoelectric materials. In *Thermoelectrics, 2006. ICT '06. 25th International Conference on*, pages 264 –267, 2006.

[12] S. Borkar. Thousand core chips: a technology perspective. In *DAC*, pages 746–749, 2007.

[13] K. G. Brill. The invisible crisis in the data center: The economic meltdown of moores law. *Uptime Institute*, 2007.

[14] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 171, 2001.

[15] C. Chan, Y. Chang, H. Ho, and H. Chiueh. A thermal-aware power management soft-ip for platform-based soc designs. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, pages 181 – 184, nov. 2004.

[16] T. Chantem, R. P. Dick, and X. S. Hu. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. In *DATE*, pages 288–293, 2008.

[17] T. Chantem, X. S. Hu, and R. Dick. Online work maximization under a peak temperature constraint. In *ISLPED*, pages 105–110, 2009.

[18] T. Chantem, X. S. Hu, and R. P. Dick. Online work maximization under a peak temperature constraint. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, ISLPED '09, pages 105–110, New York, NY, USA, 2009. ACM.

[19] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximal temperature minimization for periodic hard real-time systems. In *ICESS*, pages 1802–1809.

[20] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximum temperature minimization for periodic hard real-time systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1802 –1809, 29 2010-july 1 2010.

[21] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximum temperature minimization for periodic hard real-time systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1802 –1809, 29 2010-july 1 2010.

[22] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS*, pages 408–417, 2006.

[23] J.-J. Chen, S. Wang, and L. Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. *RTAS*, 0:141–150, 2009.

[24] T. Cheng, W. Xiong, X. Luo, S. Huang, Z. Gan, and S. Liu. Thermal management of a multi-core master processing unit (mpu) for an ultrascalable computing platform. In *Electronic Packaging Technology High Density Packaging, 2008. ICEPT-HDP 2008. International Conference on*, pages 1 –3, july 2008.

[25] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. Thermal-aware task scheduling at the system software level. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 213 –218, aug. 2007.

[26] R. Chu, R. Simons, M. Ellsworth, R. Schmidt, and V. Cozzolino. Review of cooling technologies for computer products. *Device and Materials Reliability, IEEE Transactions on*, 4(4):568 – 585, dec. 2004.

[27] A. Cohen, F. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy. On estimating optimal performance of cpu dynamic thermal management. *IEEE Comput. Archit. Lett.*, 2:6–, January 2003.

[28] S. Corbetta and W. Fornaciari. Estimation of thermal status in multi-core systems. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, 2011.

[29] A. K. Coskun, T. S. Rosing, and K. C. Gross. Proactive temperature management in mpsocs. In *Proceeding of the 13th international symposium on Low power electronics and design*, ISLPED '08, pages 165–170, New York, NY, USA, 2008. ACM.

[30] A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature aware task scheduling in mpsocs. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 1659–1664, San Jose, CA, USA, 2007. EDA Consortium.

[31] A. K. Coskun, R. Strong, D. M. Tullsen, and T. Simunic Rosing. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09, pages 169–180, New York, NY, USA, 2009. ACM.

[32] M. Dellinger, P. Garyali, and B. Ravindran. Chronos linux: A best-effort real-time multiprocessor linux kernel. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 474 –479, june 2011.

[33] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. *SIGARCH Comput. Archit. News*, 34(2):78–88, 2006.

[34] D. Feitelson. The supercomputer industry in light of the top500 data. *Computing in Science Engineering*, 7(1):42 – 47, jan.-feb. 2005.

[35] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. *RTAS*, 0:131–140, 2009.

[36] Y. Ge, P. Malani, and Q. Qiu. Distributed task migration for thermal management in many-core systems. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 579 –584, june 2010.

[37] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. *SIGOPS Oper. Syst. Rev.*, 38:260–270, October 2004.

[38] L. Guanglei, Q. Gang, and Q. Meikang. Throughput maximization on intel desktop platform under the maximum temperature constraint. In *The 2011 IEEE/ACM International Conference on Green Computing and Communications*, August. 2011.

[39] S. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, 5(1), 2001.

[40] H. Hanson, S. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio. Thermal response to dvfs: analysis with an intel pentium m. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 219 –224, 2007.

[41] V. Hanumaiah, R. Rao, S. Vrudhula, and K. S. Chatha. Throughput optimal task allocation under thermal constraints for multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 776–781, New York, NY, USA, 2009. ACM.

[42] V. Hanumaiah, S. Vrudhula, and K. Chatha. Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 310 –313, nov. 2009.

[43] H. Huang and J. Fan. Multicore processor cluster based sleep transistor sizing considering delay profile. In *IEEE 8th International Conference on ASIC (ASICON)*, 2009.

[44] H. Huang and G. Quan. Leakage aware energy minimization for real-time systems under the maximum temperature constraint. In *DATE*, 2011.

[45] H. Huang, G. Quan, and J. Fan. Leakage temperature dependency modeling in system level analysis. In *ISQED*, pages 447–452, 2010.

[46] H. Huang, G. Quan, J. Fan, and M. Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In *DAC*, pages 363–368, 2011.

[47] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 23 –32, dec. 2009.

[48] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.

[49] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.

[50] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED*, pages 197–202, August 1998.

[51] ITRS. *International Technology Roadmap for Semiconductors.* International SEMATECH, Austin, TX., http://public.itrs.net/.

[52] R. Jayaseelan and T. Mitra. Temperature aware task sequencing and voltage scaling. In *ICCAD*, pages 618 – 623, 2008.

[53] R. Jejurikar, C. Pereira, and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. *DAC*, pages 111 – 116, 2005.

[54] W. jin kim, J.-W. Song, and K.-S. Chung. On-line learning based dynamic thermal management for multicore systems. In *SoC Design Conference, 2008. ISOCC '08. International*, volume 01, pages I–391 –I–394, nov. 2008.

[55] P. H. Jones, J. W. Lockwood, and Y. H. Cho. A thermal management and profiling method for reconfigurable hardware applications. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1 –7, aug. 2006.

[56] D.-C. Juan, H. Zhou, D. Marculescu, and X. Li. A learning-based autoregressive model for fast transient thermal analysis of chip-multiprocessors. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 597 –602, 30 2012-feb. 2 2012.

[57] S.-O. Jung, K.-W. Kim, and S.-M. Kang. Noise constrained transistor sizing and power optimization for dual v/sub t/ domino logic. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 10(5):532 –541, oct. 2002.

[58] O. Khan and S. Kundu. Hardware/software co-design architecture for thermal management of chip multiprocessors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 952–957, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[59] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123 –134, feb. 2008.

[60] A. Kumar, L. Shang, L.-S. Peh, and N. Jha. Hybdtm: a coordinated hardware-software approach for dynamic thermal management. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 548 –553, 0-0 2006.

[61] A. Kumar, L. Shang, L.-S. Peh, and N. Jha. System-level dynamic thermal management for high-performance microprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(1):96 –108, 2008.

[62] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 123 –128, jan. 2011.

[63] C.-H. Lee and K. Shin. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 319 – 335, 2004.

[64] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller. Energy management for commercial servers. *Computer*, 36(12):39 – 48, dec. 2003.

[65] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1042 – 1053, 2005.

[66] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(7):1042 – 1053, 2005.

[67] C. H. Lim, W. Daasch, and G. Cai. A thermal-aware superscalar microprocessor. In *Quality Electronic Design, 2002. Proceedings. International Symposium on*, pages 517 – 522, 2002.

[68] G. Liu, M. Fan, and G. Quan. Neighbor-aware dynamic thermal management for multi-core platform. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 187 –192, march 2012.

[69] G. Liu and G. Quan. Thermal aware scheduling on an intel desktop computer. In *Southeastcon, 2011 Proceedings of IEEE*, pages 79 –84, march 2011.

[70] G. Liu, G. Quan, and M. Qiu. Throughput maximization for intel desktop platform under the maximum temperature constraint. In *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*, pages 9 –15, aug. 2011.

[71] S. Liu and M. Qiu. Thermal aware scheduling for peak temperature reduction with stochastic workloads. *RTAS(WiP)*, 0:53–56, 2010.

[72] S. Liu, M. Qiu, W. Gao, X.-J. Tang, and B. Guo. Hybrid of job sequencing and DVFS for peak temperature reduction with nondeterministic applications. In *ICESS*, pages 1780–1787, 2010.

[73] S. Liu, J. Zhang, Q. Wu, and Q. Qiu. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 390 –398, march 2010.

[74] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *DATE*, pages 1526–1531, 2007.

[75] Y. Liu, H. Yang, R. Dick, H. Wang, and L. Shang. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 204 –209, 2007.

[76] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In *ISQED*, pages 204–209, 2007.

[77] C.-L. Lung, Y.-L. Ho, D.-M. Kwai, and S.-C. Chang. Thermal-aware on-line task allocation for 3d multi-core processor throughput optimization. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[78] R. Lyon and A. Bergles. Noise and cooling in electronics packages. In *Semiconductor Thermal Measurement and Management Symposium, 2004. Twentieth Annual IEEE*, pages 154 – 160, mar 2004.

[79] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey. A voltage reduction technique for digital systems. In *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pages 238 –239, 1990.

[80] H. Mahmoodi, V. Tirumalashetty, M. Cooke, and K. Roy. Ultra low-power clocking scheme using energy recovery and clock gating. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(1):33 –44, jan. 2009.

[81] H. Mizunuma, C.-L. Yang, and Y.-C. Lu. Thermal modeling for 3d-ics with integrated microchannel cooling. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 256 –263, nov. 2009.

[82] K. Mohanram and N. Touba. Lowering power consumption in concurrent checkers via input ordering. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(11):1234 –1243, nov. 2004.

[83] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli. Thermal balancing policy for multiprocessor stream computing platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(12):1870 –1882, dec. 2009.

[84] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, and D. Atienza. Thermal balancing policy for streaming computing on multiprocessor architectures. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 734–739, New York, NY, USA, 2008. ACM.

[85] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. D. Micheli. Temperature-aware processor frequency assignment for mpsocs using convex optimization. In *CODES+ISSS*, pages 111–116, 2007.

[86] H. Nakano. Technology and applications of bluegene supercomputer and cell broadband engine. In *VLSI Technology, Systems and Applications, 2007. VLSI-TSA 2007. International Symposium on*, pages 1 –4, april 2007.

[87] M. Patterson. The effect of data center temperature on energy efficiency. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on*, pages 1167 –1174, may 2008.

[88] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487 –1501, aug. 2006.

[89] M. Pedram and J. M. Rabaey. *Power Aware Design Methodolodies.* Kluwer Academic Publishers, 2002.

[90] G. Quan and Y. Zhang. Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks. *ECRTS*, pages 207–216, 2009.

[91] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective.* Prentice Hall, 2003.

[92] R. Rao and S. Vrudhula. Performance optimal processor throttling under thermal constraints. In *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '07, pages 257–266, New York, NY, USA, 2007. ACM.

[93] E. Rohou and M. D. Smith. Dynamically managing processor temperature and power. In *In 2nd Workshop on Feedback-Directed Optimization*, 1999.

[94] M. Santarini. Thermal integrity: a must for low-power-ic digital design. 2005.

[95] R. Sarikaya, C. Isci, and A. Buyuktosunoglu. Runtime workload behavior prediction using statistical metric modeling with application to dynamic power management. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1 –10, dec. 2010.

[96] D. Sekar, C. King, B. Dang, T. Spencer, H. Thacker, P. Joseph, M. Bakir, and J. Meindl. A 3d-ic technology with integrated microchannel cooling. In *Interconnect Technology Conference, 2008. IITC 2008. International*, pages 13 –15, june 2008.

[97] S. Sha, J. Zhou, C. Liu, and G. Quan. Power and energy analysis on intel single-chip cloud computer system. In *Southeastcon, 2012 Proceedings of IEEE*, pages 1 –6, march 2012.

[98] L. Shang, L. Peh, A. Kumar, and N. Jha. Thermal modeling, characterization and management of on-chip networks. *ISM*, pages 67–78, 2004.

[99] S. Sharifi, R. Ayoub, and T. Rosing. Tempomp: Integrated prediction and management of temperature in heterogeneous mpsocs. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 593 –598, march 2012.

[100] S. Sharifi, A. Coskun, and T. Rosing. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 873 –878, jan. 2010.

[101] S. Sharifi and T. Rosing. Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(10):1586 –1599, oct. 2010.

[102] H. Shen and Q. Qiu. An fpga-based distributed computing system with power and thermal management capabilities. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, 2011.

[103] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware computer systems: opportunities and challenges. *IEEE Micro*, 23(6):52–61, 2003.

[104] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 2 – 13, 2003.

[105] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. 3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 463 –470, nov. 2010.

[106] J. Srinivasan and S. V. Adve. Predictive dynamic thermal management for multimedia applications. *ICS*, pages 109–120, 2003.

[107] E. I. S. Technology. http://www.intel.com/support/processors/sb/cs-028855.htm.

[108] A. Telikepalli. *Designing for Power Budgets and Effective Thermal Management.* Xcell Journal, 2006.

[109] C. Tianzhou, H. Jiangwei, X. Lingxiang, and S. Qingsong. Dynamic power management framework for multi-core portable embedded system. In *Proceedings of the 1st international forum on Next-generation multicore*, 2008.

[110] C. Tianzhou, H. Jiangwei, X. Lingxiang, and S. Qingsong. Dynamic power management framework for multi-core portable embedded system. In *Proceedings of the 1st international forum on Next-generation multicore/manycore technologies*, IFMT '08, pages 1:1–1:4, New York, NY, USA, 2008. ACM.

[111] D. Tuckerman and R. Pease. High-performance heat sinking for vlsi. *Electron Device Letters, IEEE*, 2(5):126 –129, may 1981.

[112] G. Q. V. Chaturvedi. Leakage conscious dvs scheduling for peak temperature minimization. In *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2011.

[113] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, and et al. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 2011.

[114] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 98 –589, feb. 2007.

[115] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, 37(3):195–237, Sep 2005.

[116] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur, M. Group, and I. Corp. Thermal performance challenges from silicon to systems, 2000.

[117] H. Wang, S.-D. Tan, X.-X. Liu, and A. Gupta. Runtime power estimator calibration for high-performance microprocessors. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 352 –357, march 2012.

[118] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10 pp. –170, 0-0 2006.

[119] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10 pp. –170, 0-0 2006.

[120] T.-Y. Wang and C. C.-P. Chen. 3-d thermal-adi: a linear-time chip level transient thermal simulator. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(12):1434 – 1445, dec 2002.

[121] X. Wang, K. Ma, and Y. Wang. Adaptive power control with online model estimation for chip multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1, 2011.

[122] 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, pages 314–324, New York, NY, USA, 2009. ACM.

[123] T. Wei, X. Chen, and P. Mishra. Designing a multi-core hard real-time test bed for energy measurement experiments. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1998–1999, New York, NY, USA, 2009. ACM.

[124] C. Xu, L. Jiang, S. K. Kolluri, B. J. Rubin, A. Deutsch, H. Smith, and K. Banerjee. Fast 3-d thermal analysis of complex interconnect structures using electrical modeling and simulation methodologies. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, pages 658–665, New York, NY, USA, 2009. ACM.

[125] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 9 –14, march 2010.

[126] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin. Dynamic thermal management through task scheduling. In *International Symposium on Performance Analysis of Systems and Software*, pages 191–201, 2008.

[127] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

[128] L.-T. Yeh and R. C. Chu. Thermal management of microelectronic equipment: Heat transfer theory, analysis methods and design practices. In *ASME Press, New York, NY*, 2002.

[129] I. Yeo and E. J. Kim. Temperature-aware scheduler based on thermal behavior grouping in multicore systems. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 946 –951, april 2009.

[130] I. Yeo, C. C. Liu, and E. J. Kim. Predictive dynamic thermal management for multicore systems. In *DAC*, pages 734–739, 2008.

[131] I. Yeo, C. C. Liu, and E. J. Kim. Predictive dynamic thermal management for multicore systems. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 734 –739, june 2008.

[132] L. Yuan, S. Leventhal, and G. Qu. Temperature-aware leakage minimization technique for real-time systems. In *ICCAD*, pages 761–764, 2006.

[133] B. Yun, K. Shin, and S. Wang. Thermal-aware scheduling of critical applications using job migration and power-gating on multi-core chips. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 1083 –1090, nov. 2011.

[134] S. Zhang and K. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 281 –288, nov. 2007.

[135] S. Zhang and K. Chatha. Automated techniques for energy efficient scheduling on homogeneous and heterogeneous chip multi-processor architectures. In *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pages 61 –66, 2008.

[136] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 585–590, New York, NY, USA, 2010. ACM.

[137] Y. Zhang and A. Srivastava. Accurate temperature estimation using noisy thermal sensors. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 472 –477, july 2009.

VITA

GUANGLEI LIU

EDUCATION

Ph.D. Candidate in Electrical Engineering	08/2007-Present
Florida International University, Miami, FL	
Advisor: Dr. Gang Quan	
Bachelor of Science in Electrical Engineering	08/2002-07/2006
Harbin University, Harbin, China	

HONORS AND AWARDS

Graduate Teaching/Research Assistantship	Fall 2007 to Present
46th Design Automation Conference(DAC2009) Summer School Student TravelGrant	July 2009
Graduate Association Award for Conference Attending	March 2008

TEACHING EXPERIENCES

Instructor	Fall 2007 to Summer 2009
EEL3111L Circuit I Lab	

PUBLICATIONS

Journals

J3. Guanglei Liu, M. Fan, G. Quan, M. Qiu "On-Line Predictive Thermal Management under Peak Temperature Constraints for Practical Multi-core Platforms", Journal of Low Power Electronics (ASP). (under review), 2012.

J2. Guanglei Liu, G. Quan, M. Qiu "Practical Dynamic Thermal Management on An Intel Desktop Computer " , Embedded Software Design, Journal of Sustainable Computing (SUSCOM) (under review), 2012.

J1. H. Huang, V. Chaturvedi, Guanglei Liu, G. Quan, "Leakage Aware Scheduling On Maximum Temperature Minimization For Periodic Hard Real-Time Systems", Journal of Low Power Electronics (ASP), 2012.

Refereed Conferences

C5. Guanglei Liu, M. Fan, G. Quan, "Neighbor-Aware Dynamic Thermal Management for Multi-core Platform", The 15th Design, Automation, and Test in Europe (DATE 2012), Dresden, Germany, March 12-16, 2012.

C4. Guanglei Liu, G. Quan, M. Qiu, “The Practical On-line Scheduling for Throughput Maximization on Intel Desktop Platform under the Maximum Temperature Constraint“, The 2011 IEEE/ACM Green Computing and Communications (GreenCom 2011), Sichuan, China, August 4-5, 2011.

C3. Guanglei Liu, G. Quan, ”Thermal Aware Scheduling on an Intel Desktop Computer,” IEEE SouthEast Conference (SouthEast 2011), Nashville, Tennessee, March 17-20, 2011.

C2. Guanglei Liu, J. Fan, “Framework for Statistical Analysis of Homogeneous Multi-core Power Grid Networks“, IEEE 8th International Conference on ASIC (ASICON 2009), Changsha, China, October 20-23, 2009.

C1. C. Liu, J. Tan, R. Chen, Guanglei Liu, J. Fan, “Thermal Aware Clocktree Optimization in Nanometer VLSI Systems Considering Temperature Variations“, IEEE 40th Southeastern Symposium on System Theory (SSST 2008), New Orleans, LA, March 17-18, 2008.