**Temperature Aware Scheduling in Multicore Systems**

by

Vibudh Mishra

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 4, 2018

Keywords: Scheduling, HotSpot, Quilt, Processor, Multicore



Approved by

Sanjeev Baskiyar, Chair,
Professor of Computer Science and Software Engineering
Cheryl Seals, Associate Professor of Computer Science and Software Engineering
Dean Hendrix, Associate Professor of Computer Science and Software Engineering

Abstract

Power density in microprocessors is increasing rapidly, which results in higher operating temperature, and systems are prone to overheating. Dynamic thermal management has been used to dissipate heat, reduce the operating temperature to avoid thermal emergencies but is not aware of the application behavior. Several techniques like thread migration, Dynamic Voltage Scaling (DVS), Dynamic Voltage Frequency Scaling (DVFS), clock gating etc. are used to resolve the problem of thermal emergencies but are reactive to the increased chip temperature. In this thesis, we propose a proactive dynamic thermal management method that is based on grouping of the applications by thermal behavior. In the experiments, offline data is used as the primary resource for the categorization of running application. The temperature of a core is predicted by the rate of temperature change at the core proportional to the difference in current and steady state temperature. We evaluate this method in a multicore system running several benchmarks. The experiment results show a decrease of 1.2 -1.6C in the average peak temperature of the CPU with a little performance overhead as compared to Linux standard scheduler.

Acknowledgements

I would like to take this opportunity to express my sincere gratitude to each and every one who has been a part of my journey of pursuing my M.S dream. First and foremost I would thank my advisor Dr. Sanjeev Baskiyar, without his continuous guidance and support this thesis would have never been possible. His timely inputs, mentoring and immense knowledge throughout my research has made all efforts fruitful. Dr. Baskiyar helped me out in the most difficult personal situations and supported me when I needed most.

Furthermore, I would also extend my heartfelt gratitude to Dr. Cheryl Seals and Dr. Dean Hendrix, for being a part of my advisory committee and giving me valuable advice during the course of my studies.

I also owe much gratitude to Ms. Carol Lovvorn, Ms. Michelle Wheeles, Ms. Jo Ann Lauraitis and Ms. Angela Evans for helping me keep my immigration and school paper work in order. I would like to thank my lab mates Rakshith Venkatesh and Adarsh Jain for their suggestions and help. Also, my special thanks to my friends Aravind Sridhar, Sameer Shah, Rahul Potghan, Hiren Adesara, Prashant Dubey, Priyanka Boocha, Mitesh Soni, Radhen Mathuria, Vikas Yadav, Mradul Sangal, Mahendra Harsha, Shrey Sanadhaya, Shashi Reddy, Rahul Kumar and Digvijay Gholap at Auburn for being a helping hand and encouraging me throughout.

I am also deeply indebted to Mrs. Niranjana Nayak, Ms. Heather Sparks and Dr. Sushil Bhavnani for ensuring that I never miss the warmth and affection of my family here.

I would like to thank my parents Dr. Arun Mishra and Mrs. Seema Mishra, Dr. Anil Vajpayee and Mrs. Prabha Vajpayee who are my pillars of strength and love. They have always shown me the right path and given me the freedom to follow my dreams. I would also like to extend sincere thankfulness to Aashi Mishra, Arvinth Selvaraj and Preeti Shastri who have been sending me best wishes, affection and emotional support. I dedicate this work to my loving grandparents, Late. Mr. S.S. Mishra and Late. Chandan Devi Mishra and I seek your blessings always.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

| | |
|---|---|
| DVFS | Dynamic Voltage Frequency Scaling |
| DTM | Dynamic Thermal Management |
| CMP | Chip Microprocessor |
| FIFO | First In First Out |
| FCFS | First Come First Serve |
| RR | Round Robin |
| EARI | Energy Aware Reservation Infrastructure |
| ASP | Allocation and Scheduling Procedure |
| MST | Maximum Scheduling Threshold |
| LSS | Linux Standard Scheduler |
| TAS | Temperature Aware Scheduler |

# Chapter 1

## Introduction

Temperature awareness is an important aspect in multicore processor systems because the rising temperature of processing elements such as microprocessors is the issue of major concern. Research shows in order to run an individual 300W server for a year costs $338, and more significantly, it can emit close to 1,300 kg $CO_2$, without even considering the cooling expenses [2]. In the latest reports it is concluded that the datacenters in the United States consumes approximately 2.0% of the total electricity consumption in 2014 and this number has been estimated to approach double in the coming decade [2, 4]. It is also estimated that the costs to run the servers will surpass the costs to purchase server hardware. As we know the energy used by a microprocessor converts into heat, this exponential rise in the heat causes temperature rise which further causes problems in maintenance, consistency and manufacturing prices. The design of a microprocessor must remove heat and hence control the temperature of the cores but unfortunately most of the designs which consumes low-power have become very expensive.

The existing techniques to prevent thermal emergencies in the area of thermal management in multicore processor systems are reactive to the increased chip temperature. There are many dynamic thermal management (DTM) techniques like dynamic voltage and frequency scaling (DVFS) and clock gating that had been used in modern day's multicore processor systems. In the dynamic voltage category, the supply voltages are scaled down using the operating clock and thus the thermal management is achieved. The power consumption from this

technique is substantially reduced and at the same time it depends on the hardware components and configurations to perform all scaling tasks. On the other hand, dynamic frequency management promises better thermal awareness, as it confirms close to zero electricity usage by being turned off servers. The above-mentioned hardware based DTM techniques are reactive to the increased chip temperature and hence the demand of efficient proactive DTM techniques is prevailing in the thermal management research area. There have been always search for a model, which can provide thermal fairness and determine which core would be best for migration in a multicore system environment.

### 1.1 Motivation and Problem Definition

Dynamic thermal management has been used to dissipate heat, reduce the operating temperature to prevent thermal difficulties but is not aware of the application performance and behavior. The temperature difference between different applications in a multicore processor system can be up to 9C (ranging from 1C to 9C). It is also found that the temperature difference between on-chip modules can be up to 10-15C [14].

Dynamic thermal management schemes that have been used were broadly classified into temporal and spatial. Temporal techniques like DVFS and clock gating are reactive and have high performance overhead. Spatial techniques like thread migration where it is very difficult to determine the hot and cool threads at runtime because of the different thermal profiles of different application. In migration process the main overhead comes from task suspension and resumption [18, 20].

There is a significant difference in the temperature characteristics of different applications on different cores in the same multicore chip [20]. Therefore, it is very important to

study the behavior and thermal profile of different applications. Also, it is critical to use an efficient scheduling technique which should consider the application's thermal behavior and profile.

## 1.2 Proposed Solution

We have proposed a proactive dynamic thermal management method based on the classification of the application's thermal behavior. The application benchmarks classified and grouped based on the steady state temperature and thermal profile. The future temperature of the core is predicted by considering the rate of change of temperature at real time where the steady state temperature of any application is known.

The grouped applications are scheduled using the coolest core algorithm where each application group has its own user defined threshold temperature. The threshold temperature of all the application groups is kept below to the average of grouped applications' steady state temperature that is calculated offline.

## 1.3 Report Organization

This report is organized as follows. Chapter 1 discusses the introduction with the problem statement and proposed solution. Chapter 2 presents a detailed background study of different scheduling techniques. Chapter 3 presents the literature survey and review where we discussed different temperature aware scheduling techniques and results. In the Chapter 4 we discussed the research overview and methodology. Also, in the same chapter we explained the assumptions, experiment design and validation of the prediction method used in the experiments. In Chapter 5

we discussed the graphs and results of all the benchmarks. In Chapter 6 we conclude the report and discuss about the future work.

# Chapter 2

# Background

In this chapter, we take a look into the various basic terms used in the scheduling of a process. We start with the basic background about scheduling and different basic scheduling algorithm. Also, since our research focuses on temperature aware scheduling, we would discuss few basic scheduling techniques used to schedule tasks. We also emphasize on the different techniques that has been used at various levels – hardware, operating systems and applications.

## 2.1 Scheduling Basics

Scheduling is a method used to allocate system resources like memory, processor time and communication bandwidth to achieve the target with a minimum performance overhead. Scheduling algorithm is required to achieve multitasking (executing numerous tasks at a time) and multiplexing (simultaneous multiple flow transmission). The basic terms that are used with the scheduling process and algorithms are discussed below.

- Turnaround time: The time between the process submission and its complete execution.
- Response time: It is defined as the amount of time scheduler takes when a request is placed and when the first reaction is made.
- CPU utilization: It defines the busyness of the CPU. Typically, it ranges from 40% to 90% depending on the requests made to the processor.

- Throughput: In the event when CPU is executing different processes that means some work had been done. The numbers of processes completely executed per time unit is called throughput. It may range from milliseconds to hours for long processes.
- Waiting time: It is the time spent by the process spent waiting for the resources to receive the resources required for its execution.

There are different kinds of operating system schedulers. These schedulers are responsible for making all the decisions from selecting next jobs to be admitted to next processes to execute. The different types are schedulers are discussed below.

- Long-term scheduler: It is also called as the admission scheduler and is responsible to select the next job to be considered in the ready queue (main memory). When a process is requested to execute, the denial and authorization is decided by the long term scheduler. It is also responsible to control the extent of multiprogramming in the processor. It plays an important role in real time operations where it makes sure that the real time process gets sufficient CPU time to complete the execution.
- Medium-term scheduler: In some cases scheduler removes the processes in main memory and then these processes are moved to secondary memory. It is also called as swapping-in and swapping-out. The medium-term scheduler is responsible for making decisions on processes to swap-out based on certain conditions. In case if the process is of low priority, process which is not active, when a process is page faulting or when the process requires a lot of space in the main memory.
- Short-term scheduler: It is also known as the CPU scheduler. It decides which processes are ready in-memory and are ready to be executed after an I/O interrupt, clock interrupt

or a system call. The scheduling decisions by short-term scheduler more frequent and more as compared to long-term and medium-term scheduler.

- Dispatcher: Dispatcher is the element in CPU scheduling that gives access of the CPU to a process which is selected by a short-term scheduler [21]. This involves the following functions-
    - Context switching
    - Switching to the user mode
    - Jumping to the location in the program to restart that program

  It should be very fast because it is invoked in every process switch. The time lag between stopping one process and starting another running is called dispatch latency [21].

Scheduling Algorithms is responsible for making the decision on the allocation of processes which are already in the ready queue. In this section we describe different scheduling algorithms and how they minimize resource starvation.

## 2.2 First-Come, First-Served Scheduling

In this algorithm CPU is allocated to the process which has requested it first. The implementation of this algorithm is easily managed by the FIFO (First-In First-Out) queue. The process enters the ready queue and when the CPU is free it is allocated at the head of the queue. The average waiting time in FCFS algorithm is long as compared to other priority driven algorithms [21].

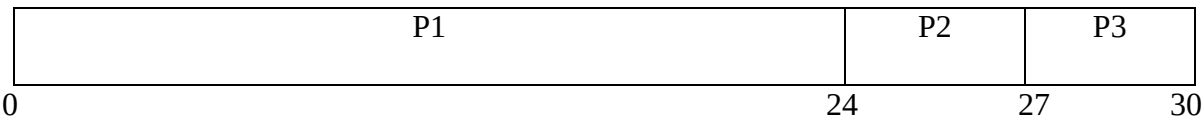

Figure 2.1: Gantt chart with Processes in Order

As we can see the in the above figure, processes are arriving at time 0 with the length of the CPU bursts given in seconds. The waiting time for the process P1 is 0 milliseconds, for process P2 it is 24 milliseconds and for process P3 it is 27 milliseconds. Now if we calculate the average waiting time of the three processes, it is 17 milliseconds. If the processes arrive in different order say, P2, P3, P1, the average waiting time is 3 milliseconds. The waiting time difference is substantial in FCFS policy and it is a non-preemptive algorithm.

## 2.3 Shortest-Job-First Scheduling

In this policy the length of the process next CPU burst is associated with the process and is used to allocate CPU to a particular process. It can also be explained as when CPU is open to accept process, it is assigned to the one which has smallest next CPU burst. In case if the next CPU burst is same for two or more processes, then FCFS algorithm is used to prevent a deadlock.

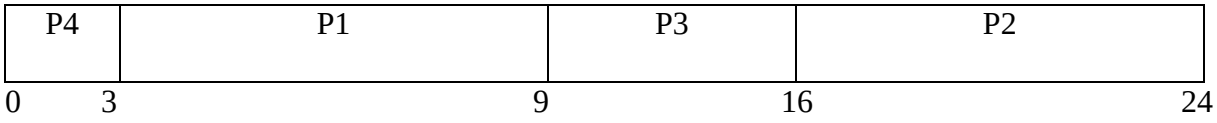


Figure 2.2: Shortest Job First Gantt chart

From the figure [21] which shows the scheduling of different processes according to shortest job first algorithm. The waiting time of process P1 is 3 milliseconds, process P2 waiting time is 16 milliseconds, 9 milliseconds for process P3 and 0 milliseconds for process P4. The average waiting time is 7 milliseconds which is better than FCFS where the waiting time in this scenario would have been 10.25 milliseconds. This algorithm is proved better than FCFS in

terms of average waiting time. The real difficulty to implement this algorithm is in the prediction of the processes having next smallest CPU burst. Also, starvation is possible especially when we have too many small processes running in the system.

## 2.4 Priority Scheduling

The priority scheduling is defined as the scheduling technique in which priority is assigned to each process and the process with the highest priority is scheduled first. In this policy the processes with equal priority are handled by the FCFS algorithm. In this technique the priorities can be assigned both internally and externally. Internal priorities depend on certain factors like, memory requirements, number of open files and time limits. External factors can be importance of the process, department sponsoring the work and others.

The priority scheduling can be both preemptive and non-preemptive. In an event when a process arrives in the ready queue, the priority of the process is weighted and compared to the other process currently executing. If the priority of the new process is higher than the one running and the process in the CPU is replaced by the new process, this will be preemptive priority scheduling. In non-preemptive scheduling the new task had to wait at the head of the ready queue till the current process is finished.

Starvation is one of the biggest problem with priority scheduling. In heavy loaded system with many processes with high priority, most of the processes of lower priority had to wait indefinitely and hence they are blocked for very long.

A solution to this indefinite blocking of lower priority processes is called aging. In this technique the priority of the processes that are waiting for a long time gradually increases with time.

## 2.5 Round-Robin Scheduling

Round-Robin scheduling is a preemptive algorithm. In this scheduling policy a time slice is defined and it is used for time-sharing systems. The ready queue behaves as a circular queue where CPU is allocated to each process for a pre-defined interval of time (time-slice). The implementation of the RR scheduling is done in a FIFO style where any new process is added to the end of the ready queue [21].

In RR scheduling, typically there are two cases, first where the time slice for a given process is larger than required and the process releases the CPU voluntarily to assign it to the process located at the very first place in the ready queue. In second case, if the time required to complete for a process is larger than the time-slice, the timer will go off, cause an interrupt to the operating system and execute context switch to put the process at the tail of the ready queue.

The performance of a RR scheduling algorithm depends on the size of the time slice or time quantum. If the time slice is extremely large, then the policy is very similar to FCFS and if the time slice is very small then RR policy is called processor sharing where it appear like n processes has its own processor.

In the software level context switching also plays an important role in the performance of the RR algorithm. If we have only one process of 13 time units and time slice is 15 time units, the process finishes in less than one time slice or time quantum. If the time slice is 7 time units, then the process will take 2 slices and a context switch to complete.

If the time slice is 1 time unit there is 11 context switches resulting in slowing the overall execution of the process. It can be concluded that turn-around time also depends on the size of

the time slice. Time slice should always be larger than the context switch time to avoid slow processing [21].

### 2.5.1 Linux Process Scheduling

In Linux based system a priority based scheduling is used. It depends on the ranking of the processes depending on their need for the processor time. Higher priority tasks are scheduled before the lower priority tasks and the tasks with same priority are scheduled as per round robin algorithm. Higher the priority of the task longer the processor time-slice the process will receive. In Linux scheduling, priority of the tasks are defined by both, user and the system. Initially, every process has a priority called nice value, which ranges from -20 to 19. With 19 being the lowest and -20 is the highest priority, default value is always considered as zero. This is also known as static priority as it cannot be changed by the user [10].

The process scheduler makes decision based on the dynamic priority. The dynamic priority is based on the function of static priority and process's interactivity. The scheduler computes the priority by either penalizing or giving bonus to process by +5 to -5 range. In this way the priority of the process is defined and scheduled in a Linux based system.

# Chapter 3

## Literature Survey and Review

In this chapter, we take a look into the various research techniques in the areas of temperature aware scheduling in single and multicore processor systems. Also, since our research focuses on temperature aware scheduling, we would discuss few reactive as well as proactive techniques used to schedule tasks with temperature control. We also emphasize on the different algorithms that has been used at various levels – hardware, operating systems and applications.

### 3.1 Green Strategies in Large Scale Distributed Systems

The green strategies in large scale distributed systems explains the temperature and power saving problem in electronic devices. For the experiment Orgerie, Lefevre and Gelas [3] considered a Grid 5000 test bed which has 3400 processors. Here users can reserve the resources in advance. By analyzing it is concluded that the real percentage of work time is 50.57%. Energy consumption can be reduced by following methods.

- Unused nodes switched off
- Nodes usage is predicted based on what is required in the near future
- Frequent ON/OFF cycles are avoided by aggregated reservations

To do this an Energy Aware Reservation Infrastructure (EARI) is implemented using an ON/OFF algorithm where resources is managed by the scheduler. The functions of the scheduler are (a) Scheduler gives resource access to users who have made reservations on scheduler program. (b) Energy parameters from the resources are monitored by energy senors. (c) Green advices data is sent to users back in order to let users choose the resources. (d) Infrastructure computes the consumption diagrams of past reservations and sends it back to the users. (e) This also makes decision over the resources ON/OFF state.

Energy monitoring is done for the different resources. Nodes at boot time consume 300W to 400W. The consumption of processor when it is idle takes 190W and quantity of power required while shutting down is 20W. The power consumed by the disk access application is 10W and consumption due to high performance network communication is 20W-22W. CPU intensive application consumes power up to 20W-25W.

Resource Managing Algorithm uses R as a tuple (l, n, and $t_0$) where l is the length of seconds of reservations, n is the required number of resources and $t_0$ is the wished start time. To support a request it should have $n < N$, $t_0 > t$ where t is the actual time and $L > 1$. Every past and the future reservation write down into agenda. $P_{IDLE}$ and $P_{OFF}$ is the power consumption when it is idle and off. $E_{ON}$ to $E_{OFF}$ is the energy required to switch between ON and OFF. The algorithm is defined into two parts described as follows.

When reservation is submitted, R (l, $n_0$, $t_0$), we estimate different amount of energy consumption by R if it starts at

i. at $t_0$ (if not possible, $t_1$ is the next possible start time)
ii. Just after the next possible end time of the reservation $t_{end}$
iii. L seconds before the next possible start time $t_{start}$

iv. During the slack period $t_{stack}$

Our goal is to aggregate reservation to avoid booting and turning off which consumes energy and raises temperature. The scheduler makes resource allocation by choosing resources with smallest power coefficient. Lower the energy consumed bigger will be the power coefficient and all this information is provided to the user. Also, the resource allocation uses the imminent reservation where the reservation will start in less than $T_s$ seconds in relation to present time. $T_s$ is the minimum time which ensures energy saving if we turn off the resource during this time. Using this algorithm they compared results in Figure 3.1 [3] for surprise reservation which is reduced substantially by EARI algorithm.

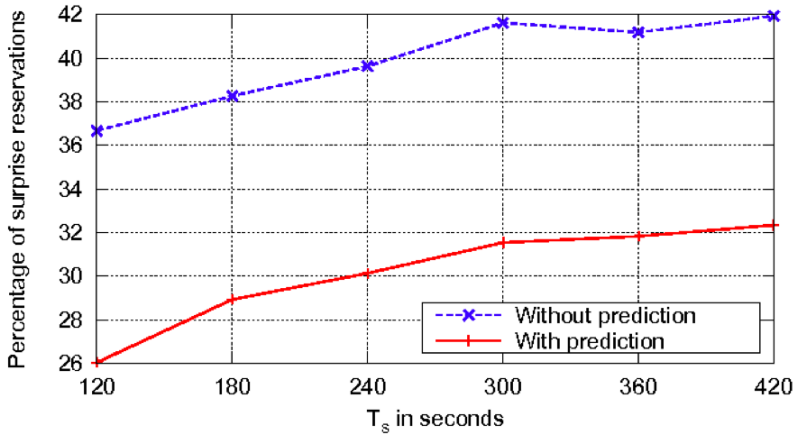


Figure 3.1: Percentage of surprise reservations in relation to total reservation number

## 3.2 Thermal Aware Task Allocation and Scheduling for Embedded Systems

In this section we look forward to an idea of task allocation and scheduling algorithm called as the thermal-aware approach for embedded structures. It is implemented by reducing peak heat ups in the circuitry and uniform thermal dispersal. W.L Hung et al. [6] proposed that

power, area and performance triangle in VLSI, says that optimization in one is achieved at the cost of the other two parameters. With the advent of VLSI technology, the miniaturizing of circuits has led to larger number of circuits to be mounted in one single chip. This hints rising power consumption, resulting heating up of the chip. High temperatures not only mar the device operation by affecting the delays in the circuit, electron level failures and leakage currents. This leads to additional expenses of cooling the chip by supplementary external hardware. The outcome of all above is the degraded reliability of the system. The power-aware design alone is inadequate to tackle the temperature glitch and so are various low power designs. The low power designs do not include spatial and temporal behavior. Though thermal-aware scheme being a subset of power-aware scheme, it proves to be a better solution as it marks maximal and average temperature reduction. The allocation and scheduling procedure (ASP) works with the task graph and architecture with target library as input. It outputs task mapping and scheduling on the final design. The worst-case power consumption and worst case execution times are saved for different implemented Processing elements.

The static and dynamic criticalities are computed. The distance between the present and last task is called the Static Criticalities (SC). The dynamic criticalities (DC) depend on a number of factors. The conventional ASP is applicable only in few areas but it ignores the temperature influence. To take care of this they proposed power/energy aware ASP and thermal-aware ASP. The thermal-aware accounts for the temperature in the system. The heat generation of an embedded system depends on various factors like the placement of the section and the power used up by various components. Computer aided tools are also implemented for sketching heat generation patterns. The temperature outline is obtained using a simulator named Hotspot. This tool considers the overall placement of sub circuits and their power and approximates almost

correct temperatures for each unit. The cumulative power used by each PE and overall power is passed as inputs to the ASP. The outputs from Hotspot are aggregated and DC is worked out. Using this approach the overall temperature of the embedded systems test bed is substantially reduced. As we can see from the Table 3.1 [6] below, temperature aware approach to schedule the tasks is proved to be better than power aware approach. The overall temperature and power is reduced in the experiment reading of thermal aware architecture technique.

| | **Power Aware Arch.** | | | **Temperature Aware Arch.** | | |
|---|---|---|---|---|---|---|
| **Benchmarks** | **Total Pow** | **Max. Temp.** | **Avg. Temp.** | **Total Pow** | **Max. Temp.** | **Avg. Temp.** |
| Bm1 | 10.90 | 85.88 | 75.5 | 6.37 | 65.8 | 61.5 |
| Bm2 | 24.09 | 106.3 | 97.4 | 22.37 | 96.7 | 93.6 |
| Bm3 | 25.70 | 103.5 | 94.6 | 24.98 | 102.5 | 94.1 |

Table 3.1: Temperature Comparison of Power and Thermal Aware Approach

### 3.3 Thermal Aware Scheduling in Multiprocessors (Operating System Level)

In this section we take a look into thermal aware scheduling in multicore processor at operating system level. Current microprocessor suffers decrease in performance due to more complexity and operating frequency. Due to multiple cores on common chip, results in more heat dissipation and hence affects the reliability and performance of processor. In this section a technique is presented to minimize those problems at operating system level. Stavrao and Transcoso [7] introduced dynamic thermal management in a multicore processor by combining coolest core and maximum scheduling threshold techniques.

Thermal-Aware Scheduling (TAS) helps in minimizing problems by considering temperature into process scheduling. In this technique temperature history is used by the operating system to determine the core on which a new and existing processes run. Dynamic Thermal Management (DTM) technique is used by most to avoid the workloads when temperature is increased. Lower temperature helps in cooling solutions when using DTM but high operation temperature can affect performance and hence reliability becomes the issue as it is based on operating temperature. In chip multiprocessors architecture per-core cooling capabilities are less efficient then single chip multiprocessors.

There can be two states in operating system that it can be idle or busy in executing some process and hence comes the need of scheduling for multiprocessor operating system. By implementing thermal aware scheduling at the system level (OS level) helps chip multiprocessors to avoid the modification of micro architecture. In this experiment Thermal Scheduling Simulator (TSIC) is used where different numbers of cores are modeled using different parameters including scheduling algorithm and maximum allowed chip temperature. Error recorded was less than 1 degree Celsius and thermal model was also validated using Hotspot simulator. Coolest Core algorithm is used in experiment. Using this algorithm process is scheduled on coolest available core. Maximum Scheduling threshold (MST) restricts scheduling on a core if threshold temperature is reached. If threshold is reached process is migrated to other core.

Results from the experiment indicates improved performance. First Experiment results shows inefficiencies increase when thermal awareness is done without scheduling. Algorithms

such as Coolest Core combined with MST heuristic result in decrease of number of migration and performance loss is reduced with temperature awareness.

### 3.4 Thermal Aware fully loaded Process Scheduling

According to the Arrhenius equation, temperature increase of 10C results in 50% decrease in the reliability of an electronic device. In situations when we use DVFS to reduce the temperature rise caused due to processes, it may degrade the overall performance of the system. The technique of process migration provides support in these situations but there is no solution currently that can solve thermal problems when all cores or threads are active. In this section we take a look at the solution to this problem proposed by Dong Li et al. which is independent of the architecture and operates at the user application level without any change in the operating system kernel. To implement they used AMD 64bit Athlon processor and AMD K8 embedded temperature sensors. When the temperature of the processor is above the specified value, the process scheduling algorithm is initiated which resets the process priority to control rising temperature.

The process scheduling algorithm is described in the Linux 2.6 kernel on which the whole framework is based. In a typical Linux environment process with higher priority is scheduled before the less priority process and if the priority is same it will be accommodated in round robin. The process with higher priority receives longer time slice in Linux. All the processes have an initial priority which called nice value and its values ranges from -20 with highest priority to +19 of lowest priority. It doesn't change with the user specification as it is defined as static priority. The actual or dynamic priority depends on the static priority as well as task's interactivity which is defined by the scheduler and computes as a bonus or penalty of -5 to

+5. The thermal management system assigns nice value to each process depending on the relative resource intensity which allows reduction of the time slice for the higher priority hot processes when the temperature is high. In this way hot and cold processes are interleaved to reduce the heat produced due to high intensity processes.

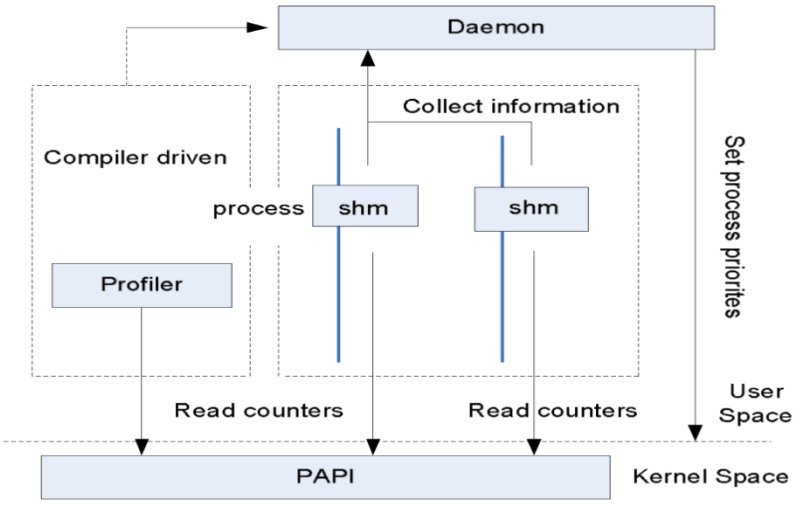


Figure 3.2: Thermal Management System Design

The thermal management framework shown in the above Figure 3.2 [10] has a process profile and a control daemon. The process information and reading of temperature sensors is done by the control daemon. Profiler is used to keep record of resource access counts. The scheduling algorithm works by initializing the process set as empty and reading the configuration file to access the threshold temperature and process priority range. The current CPU frequency is recorded.

Now the segment ID store is scanned to remove the finished processes and register the new ones in the process set. The current temperature is accessed through the sensor and compared if it is greater than the threshold and emergency temperature. In case it is more than

emergency temperature, CPU frequency is set to lower by DVFS. On the other hand if the temperature is lower than the threshold temperature, the event number of the process is accessed from the shared memory and its priority is decreased by 5. The advantage of using this algorithm is, it doesn't compromise the system throughput if compared with Linux. The finishing time is within 4% of what achieved by Linux scheduling. The scheduling algorithm allows cold processes to finish up earlier than Linux scheduling and it is due to higher priority is assigned to cold processes to prevent the temperature rise. After the cold processes finish, the hot processes are immediately allocated more time slices and thus it can run faster. This helps to compensate time which was used by cold processes. By the end of all processes the final temperature is much lower when process scheduling algorithm is used.

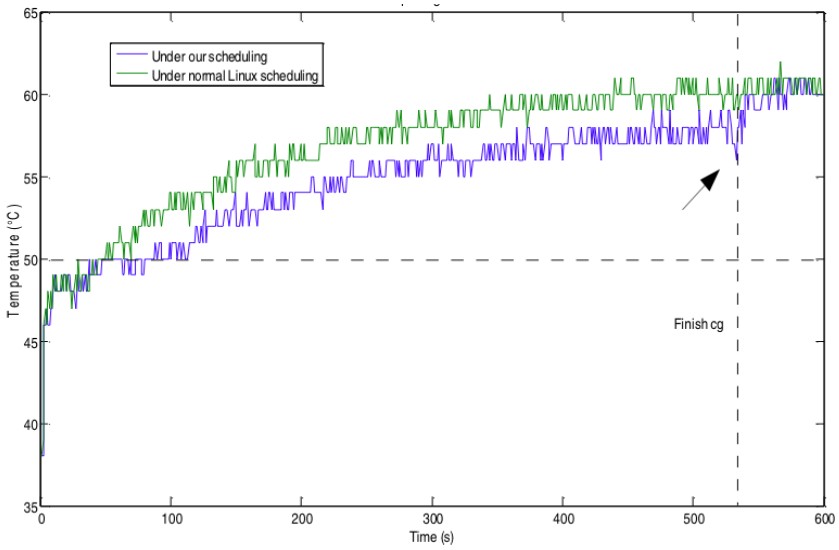


Figure 3.3: Comparison of both Scheduling Schemes

From the Figure 3.3 [10] it can be concluded that the processor temperature is lower when they have implemented the process scheduling algorithm than under normal Linux scheduling.

## 3.5 Other Temperature Aware Scheduling Techniques

Most of the current processors manufactured by Intel, AMD, and IBM uses dynamic voltage scaling in which clock speed is controlled by varying the supply voltage in the processor. This technique gets the maximum work done and hence is considered even when the objective is complex. In any case when the objective is complex or simple, the best strategy is always that makes more work done. D. Rajan and P.S. Yu [14] explains the difference in the decisions comprising job selection and system operating load. The concerns are with scheduling decisions with the decision of processing time to the active tasks as compared to the other task scheduling techniques.

It is proved that under many scenarios, simple system-throttling rules are sufficient to guarantee the maximum amount of work done. In this case they considered an alternate strategy, which is referred as the Zig-Zag operating policy. In this strategy, the scheduler operates between alternate stages of cooling and heating, in an attempt to increase the amount of work done while ensuring that the maximum temperature is below the threshold and emergency temperature.

B. Shirazi et al. [1] explained capable subdivision and scheduling of parallel programs on CPU and distributed computer systems which are considered challenging and significant issues in parallel processing. The parallel programs tasks are portioned into clusters. Once the parallel programs are portioned into clusters it can be represented as the directed acyclic graph.

Algorithms use properties of the input directed acyclic graph for determining the priorities, where it classifies the existing scheduling algorithms into four categories according to the properties used. These properties are node weights, distances, critical path, and few

combinations of these parameters. Some the algorithms used by implementing this technique are HNF (Heavy Node First), LC (Linear Clustering), DSC (Dominant Sequence Clustering), CPND (Critical Path Node Dominant) algorithms. In [20], an attempt has been made to study the nature of the application executing at the core using different kind of parameters. Wang et al. [20] have used OCIP (Off-Chip Instruction proportion), Cycle per instruction (CIP) and Off Chip rate (OCR) to determine the thermal characteristics of the given task and further made use of this information in the decision making and time-slice scaling implemented on the temperature aware scheduler.

In another study [11], Inchoon Yeo el al. presented another way to study the thermal profile of the application. They have used SPEC 2006 benchmarks to determine the steady state temperature of the overall multicore processor. By classifying different application into several categories, grouping is done using k-method. This grouping of different applications using k-method from steady state temperature helped to put different applications in the same group with their thermal profiles. The applications in the same group is assumed to have similar thermal pattern.

Using this information and application grouping the thermal scheduling algorithm is designed to migrate the task in the multicore processor environment to the core which will take maximum time to reach the threshold temperature during the execution of a particular application, grouped in a specific category. Also, they have used the slope value of the thermal patterns of the application to determine the group of that application at run time. In this method, access the current running temperature of the application. The thermal pattern can be divided into two major regions which are steep and flat region. In the steep region, the temperature of the running application suite will increase quickly before it reaches flat region where after sometime

the temperature reaches close to the steady state temperature and that region of the pattern becomes flat because the temperature in this region remains more or less the same in the range of ±5C.

To show the regions and method to determine the group of the application suite, following graph can be used.

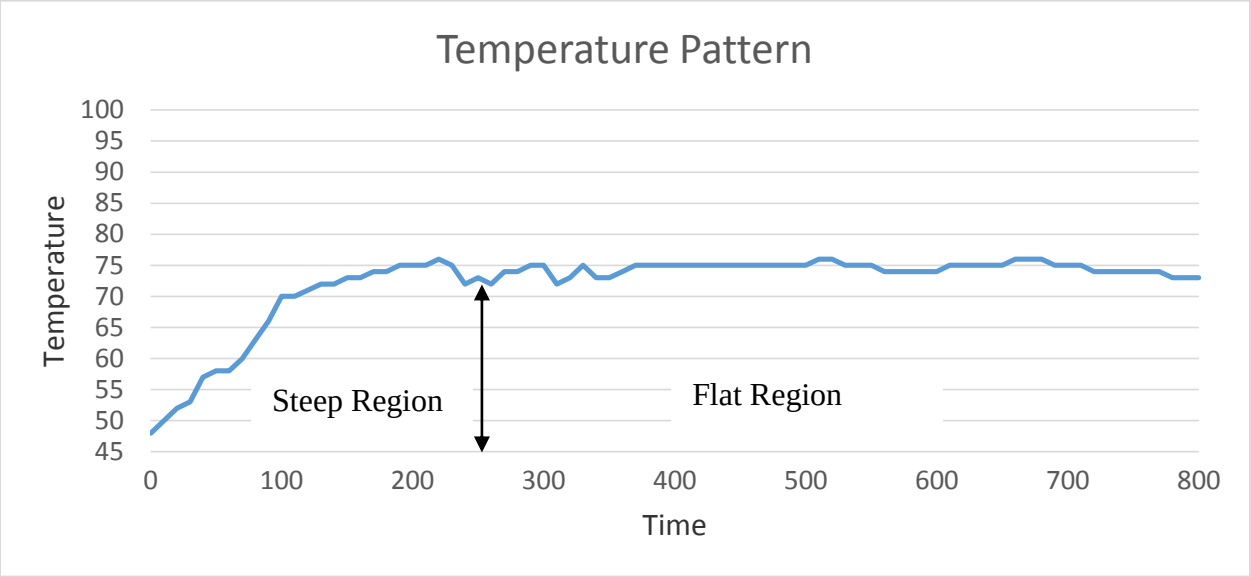


Figure 3.4: Thermal Pattern of an Application

As we can see from the Figure 3.4 that the thermal pattern of an application is divided into two parts, i.e. steep and flat region. Also it clearly shows that the temperature of the system rises till 250 seconds and after that period of time it tries to achieve a steady state temperature and maintains its temperature within the range of ±2C. To determine the group of the application first we have calculated the slope of the groups of SPEC CPU applications which were grouped based on the steady state temperature. In order to calculate the slope following equation is used.

$$S_r = \frac{T(r + \Delta t) - T_r}{\Delta t} \quad (3.1)$$

In the above equation (3.1), $S_r$ is the slope of the thermal pattern of any application in the r region. R region can be either steep or flat depending on the value of the slope. $T(r + \Delta t)$ is the current temperature, $T_r$ is the previous temperature and $\Delta t$ is the predefined time interval. Using this equation group of the running application is determined. In the case of new application the current temperature of the running application suite is accessed twice in a predefined time interval. Using above equation to compute the slope value, any given running application can be matched with the slope value of the group in a particular region. Also, the slope value of all the four groups is known in both steep and flat regions to match with the running application slope. In our experiments we have used the same assumptions but the classification of applications is done based on their steady state temperature.

We have done a rigorous survey and had gone through many different techniques and scheduling algorithms which helps in reducing the power and temperature at different levels i.e. application, operating systems and hardware. These different research articles has given a lot of directions for the future work in the field of power and temperature aware scheduling and computing. These techniques presented in this chapter helps decreasing the temperature of the cores of the processor as well as total energy consumption. In this way we are still in search of more feasible and cheaper technique that can be easily applicable. Based upon these studies we evaluated our temperature aware scheduling algorithm using different CPU intensive benchmark applications. Also, most of the techniques discussed so far has not considered the behavior of different applications. In the next chapter we have explained the experiment setup, assumptions, simulators and algorithm used towards the completion of our experiment and research thesis.

# Chapter 4

# Research Overview and Methodology

In last chapters we have discussed the motivation behind the research and where most of the techniques discussed so far has not considered the behavior of different applications at the processor level. In this chapter we have explained the research overview, experiment setup, assumptions, simulators used and algorithm used towards the completion of our experiment and research thesis.

## 4.1 Research Description

In the previous few chapters we have discussed about the different dynamic thermal management techniques and concluded that most of the techniques are not aware of the type of application. It is discussed that there are different functional units in applications which affects the operating temperature and hence the temperature difference between different types of applications can be up to 9C [11].

In this research project we propose a simple accurate prediction method that is used to profile the thermal behavior of different applications and then use that data to classify these applications into different groups. The main contributions of the project is divided into following parts.

- We have classified applications into different groups by their thermal behavior which is based on the steady state temperature. Steady state temperature is defined as the temperature reached at the processor level when an application is executed infinitely.
- We have implemented coolest core temperature aware scheduler in the multicore processor system and we demonstrate that our technique has successfully decreased the peak and overall temperature of the processor. We have compared our proposed method with the Linux Standard Scheduler (LSS) and had achieved better results. Also, in order to implement this technique no additional hardware is required.

## 4.2 System Characteristics

The system characteristics in which the experiment is performed are listed in the table below.

| **Parameters** | **Description** |
|---|---|
| CPU | Intel (R) Core (TM) 2 Quad Q6600 2.4 GHz |
| L2 Cache | 4096KB |
| Memory | 2GB |
| Storage | Intel SATA X25-M 80GB SSD |
| OS | Ubuntu 10.10 |
| gcc/g++ | v4.4.5 |
| emacs | v23.1.1 |

Table 4.1: System Environment

## 4.3 HotSpot and Quilt Tools

We have used HotSpot v 5.0 simulator and Quilt v1.0 tool for implementing our proactive temperature aware scheduling method and to design the floor plans for multicore processor system. HotSpot helps to provide the runtime responses which can be used to implement thermal aware algorithm that can change the processor's behavior and hence prevent thermal emergencies rather than depending on the expensive thermal packaging solutions. We have used Intel Quad Core Q6600 processor design to implement our proactive temperature aware method. HotSpot is written in C and it also have a very simple set of interfaces to study power aware aspects and can be used with other simulators to include more parameters in the experiment.

Quilt tool allows user to rapidly build floor-plans of integrated circuits providing both visual aid and input to the HotSpot simulator. It stands for Quick Utility for Integrated circuit Layout and Temperature modeling. This tool is platform independent and is written in Java.

## 4.4 Assumptions

We have proposed a method to predict the future temperature at any point of time during the execution of a specific kind of application. This method depends on two proven assumptions.

According to the first assumption, the rate of change of temperature during the execution of any application depends on the difference between current temperature and the steady state temperature of the application [11] [14]. This assumption gives a heat transfer equation which can be used to determine the future temperature in a core at any given point of time.

By solving the differential equations considering initial and infinite time, the future temperature of a core executing any application can be estimated. The heat transfer equation is derived as follows.

$$dT/dt = c \times (T_{ss} - T) \qquad (4.1)$$

where c is the processor constant, Tss is the steady state temperature and T is the current temperature.

At the initial and infinite time, $T(0) = T_{init}$ and $T(\infty) = T_{ss}$ (4.2)

Solving these two equations with the differential equation (4.1) is done as follows,

Equation (4.1) can be written as,

$$dT/dt + bT = cT_{ss} \qquad (4.3)$$

the above equation is in the form of $dy/dx + P(y) = Q$

Now the integrating factor is $e^{\int P.dt}$ and the integrating factor in the above equation would be,

$$I.F = e^{\int c.dt} = e^{cT}$$

The complete solution of the equation is computed as follows,

$$T \times I.F = \int Q\,(I.F)\,dt + C \qquad (4.4)$$

Putting values of equation 4.3 into the complete solution equation (4.4) we get,

$$T\,e^{cT} = c\,T_{ss} \times \int e^{ct}\,dt + C$$

Solving this further, $T\,e^{ct} = T_{ss}\,e^{ct} + C$ (4.5)

Considering the initial and infinite time and putting those values in equation (4.5) we get,

$$T_{init} = T_{ss} + C$$

And, $C = T_{init} - T_{ss}$ at T(0)

Now putting the value of C in equation (4.5) to get a complete solution we have,

$$T\,e^{ct} = T_{ss}\,e^{ct} + (T_{init} - T_{ss}) \qquad (4.6)$$

Rearranging equation (4.6) we get our final desired equation as follows,

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-ct} \qquad (4.7)$$

Equation (4.7) is the final equation that can be used to predict the future temperature of the core executing any specific kind of application. It is also used to validate the groupings of the application discussed in next section.

The second assumption we have considered in this method is the applications in the same thermal groups have similar thermal patterns. We have also considered and measured the slopes of the different thermal patterns in steep and flat regions and concluded that the thermal pattern of the applications belonging to the same groups have similar slopes in the particular region. The regions of the slopes considered are steep and flat regions.

## 4.5 Grouping of the Applications

We have done the classification of all the applications by the steady state temperature. In our experiments we have used SPEC CPU 2006 benchmark applications, as most of the applications are CPU intensive with almost 100% utilization. We have also calculated by running several benchmarks that the value of hardware constant b is almost equal to 0.0064 when the workload is 100%. We run the entire SPEC CPU 2006 benchmark application suite on the processor to get the steady state temperature of applications, which is required to group them into different category groups. We accessed the current temperature of the processor cores at runtime and hence calculated the hardware constant b for each core in the processor.

We measured the steady state temperature of each application to identify the thermal group. We run SPEC CPU 2006 application suite on HotSpot to calculate the steady state

temperature and then calculating the average temperature of the chip microprocessor. The suite is supposed to run until the temperature doesn't change further. In this way steady state temperature for each application suite can be calculated.

The table below shows the different SPEC CPU 2006 application groups with steady state temperature.

| SPEC CPU Applications | AVG. Temperature | Group |
|---|---|---|
| 400.perlbench | 78C | 2 |
| 401.bzip2 | 77.4C | 2 |
| 403.gcc | 77.7C | 2 |
| 429.mcf | 81.2C | 3 |
| 433.milc | 73.5C | 1 |
| 435.gromacs | 84.7C | 4 |
| 445.gobmk | 74.5C | 1 |
| 454.calculix | 81C | 3 |
| 456.hmmer | 80.5C | 3 |
| 458.sjeng | 73C | 1 |
| 462.libquantum | 86.5C | 4 |
| 464.h264ref | 75.1C | 1 |

Table 4.2: Grouping SPEC 2006 CPU Application Benchmarks

As we can see from the Table 4.2 that all the application suites are categorized into groups using their steady state temperature. There are four groups and all applications belonging to the same thermal groups have same thermal patterns. In the next section we discussed about the thermal patterns and prediction of the application at the runtime and how it would be used to determine the group of a particular application.

## 4.6 Classification of Application at Run Time

In this section we demonstrate the method to predict the temperature of any executing application suite. To determine the group of any running application we will use the data of the last section which is calculated offline while creating different groups.

We aimed to maintain the overall temperature of the processor below the threshold and to achieve it, our model consider the equation (4.7), where the current temperature of an application can be derived if the offline data about the group and its steady state temperature is known. Here we rearrange the equation to come up with another equation, which is mentioned as follows.

$$Tss = ((T_t) - T_{init}\, e^{-ct})/(1 - e^{-ct}) \qquad (4.8)$$

Using the above equation (4.8) steady state temperature of an application at runtime can be calculated. We run the application for a definite amount of time which can be said as t and then access the temperature of the core in multicore processor. Once we get the values of current temperature, initial temperature and hardware processor constant c, the steady state temperature of the running application can be calculated.

This predicted steady state temperature can be related to the application groups which are known and calculated offline. In this way a running application suite can be categorized in the particular group. The application groups used to categorize the SPEC 2006 benchmarks are shown in the Table 4.2. Also, to map a running application into a group can be determined by the scheduler based on the data available offline and matching the steady state temperature of each group.

## 4.7 Validation of the Prediction Method

In the previous section we determined the group of the running application to predict the future steady state temperature. This helps to determine the behavior and thermal pattern of the application. In this section we discuss the correctness of the prediction method and validate the data calculated.

To achieve the validation of the data and prediction method we run a few of the SPEC 2006 benchmarks and predict its group using the method explained in the previous section. Once we obtain the predicted steady state temperature we actually match it with the available offline data which we obtained in runs. The offline run data has steady state temperature based on what the pre-determined applications' groups were made. In order to make the validation process rigorous we tested all the application benchmarks to measure the error in the peak and average overall temperature before it reaches the steady state temperature. By studying the thermal patterns we observed the points of maximum diversion and calculated the temperature error value obtained by prediction method as compared to the offline. We calculated the average temperature error value of 0.18C.

In the next chapter we have discussed the results obtained by running different application suites using our prediction method and had compared it with the standard Linux scheduler runs. We have observed and had shown how our method is better as compared to other traditional methods. The graphs in the next chapter shows the cooler runs of the application suite and how it has affected the peak as well as overall temperature.

# Chapter 5

## Results and Graphs

The temperature aware scheduling using application behavior is implemented with the coolest core algorithm. In this chapter we show the different benchmark runs in graphs using our temperature aware predictive technique in comparison to the Linux standard scheduler. We compare our results with the one obtained by running the same benchmarks using Linux standard scheduler. In the experiment we have used Hotspot to run different application benchmarks. We started by running the benchmarks and getting all the offline data. In this we run SPEC 2006 application suite individually until the temperature doesn't change further which is known as steady state temperature. We analyzed the values of steady state temperature obtained offline that helps to categorize the benchmark applications into groups. As we discussed in the last chapter section 4.4 and 4.5 the application groups is the part of the prediction model. Also our experiments are based on the assumptions we discussed in the previous chapter. We used the heat transfer equations to derive the relation between steady state temperature with the current temperature of an executing application and initial temperature of the processor. The equation can be used to group the application during execution. The group of the running application helps us predict the future steady state temperature. This predicted steady state temperature of the application can be used in scheduling the application and migrate the task before it reaches the threshold temperature. The threshold temperature of the group can be defined at the user end. We defined the threshold temperature of the groups lower than the actual steady state

temperature of the applications' group. The applications are then scheduled using coolest core algorithm. In the results we have found that our technique and prediction method reduces the average peak temperature as compared to the standard Linux scheduler.

### 5.1 Result Graphs

In this section we have graphs of the results obtained by running experiments on different SPEC 2006 benchmarks. All the graphs are compared as per our temperature aware scheduling with Linux standard scheduler. In the following graphs we have clearly showed how our technique helps reduce the peak operating temperature at run time. In the graphs, average temperature of all four cores is considered and it shows how temperature aware scheduler (TAS) runs cooler as compared to Linux standard scheduler (LSS).

The graphs also shows the steady state temperature attained by each application suite and we have categorized each application into groups.

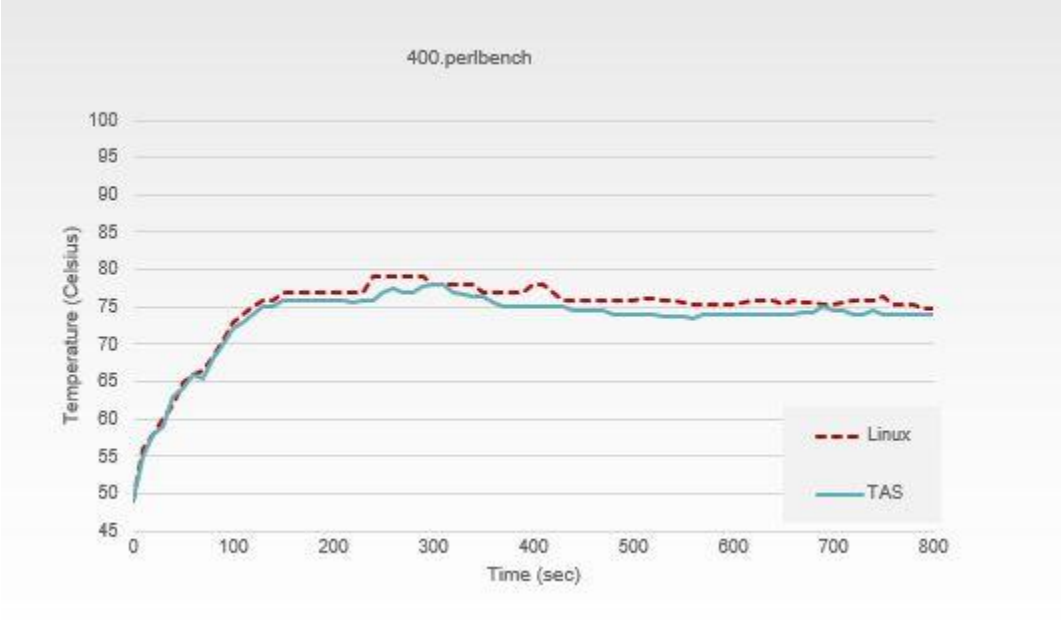


Figure 5.1: Temperature pattern comparison of 400.perlbench benchmark run using TAS and LSS.

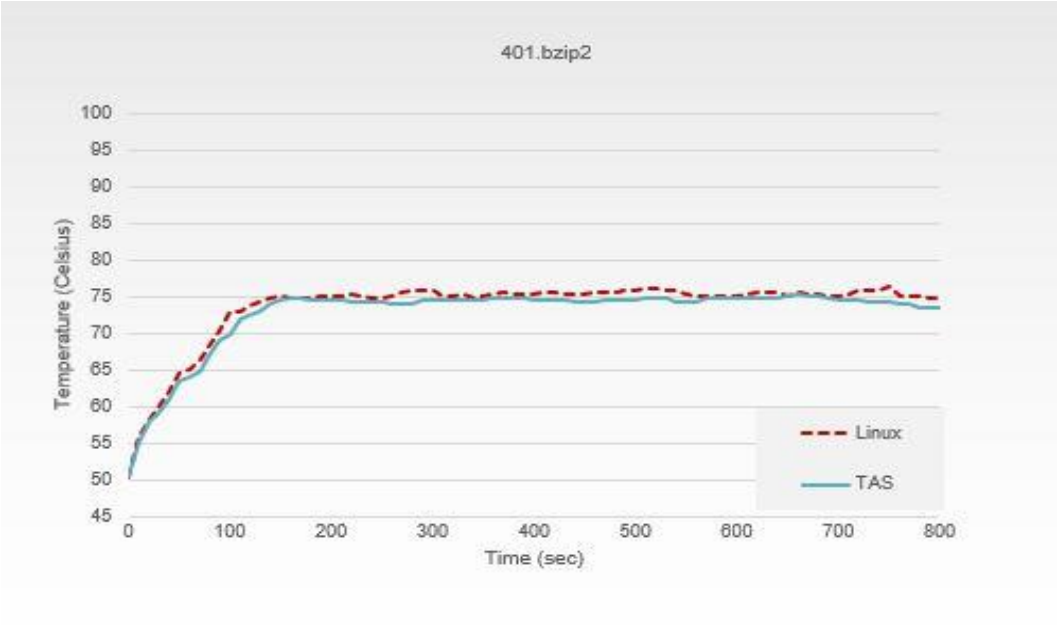


Figure 5.2: Temperature pattern comparison of 401.bzip2 benchmark run using TAS and LSS.

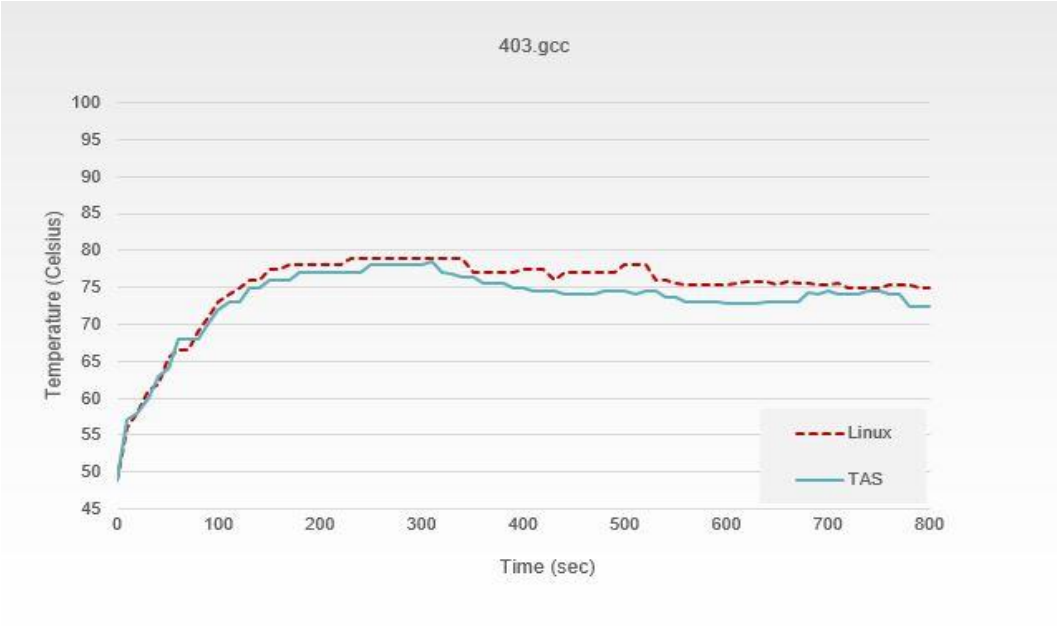


Figure 5.3: Temperature pattern comparison of 403.gcc benchmark run using TAS and LSS.

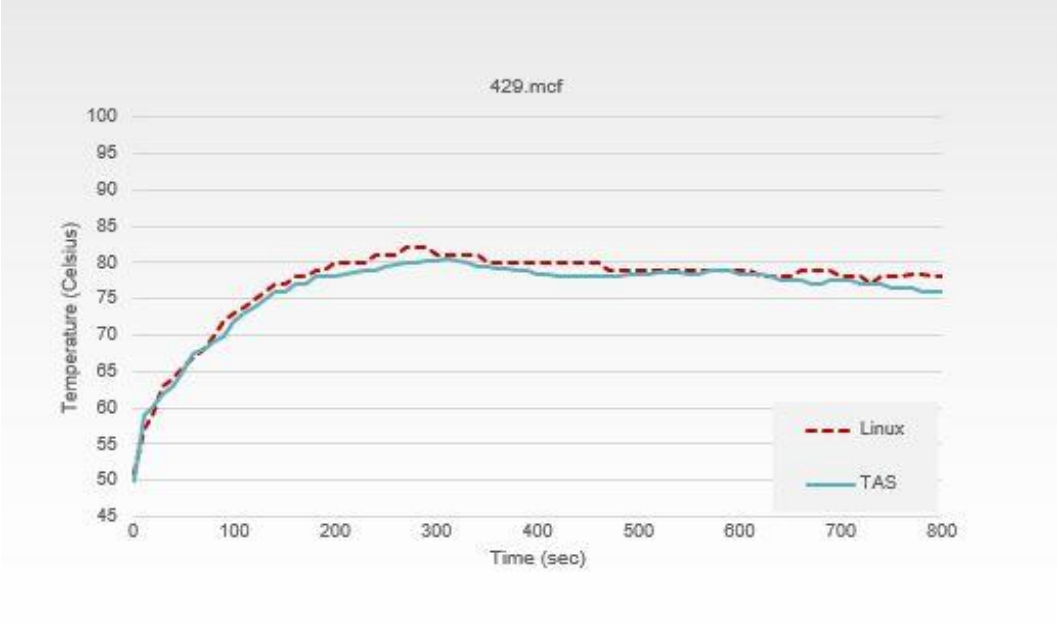


Figure 5.4: Temperature pattern comparison of 429.mcf benchmark run using TAS and LSS.

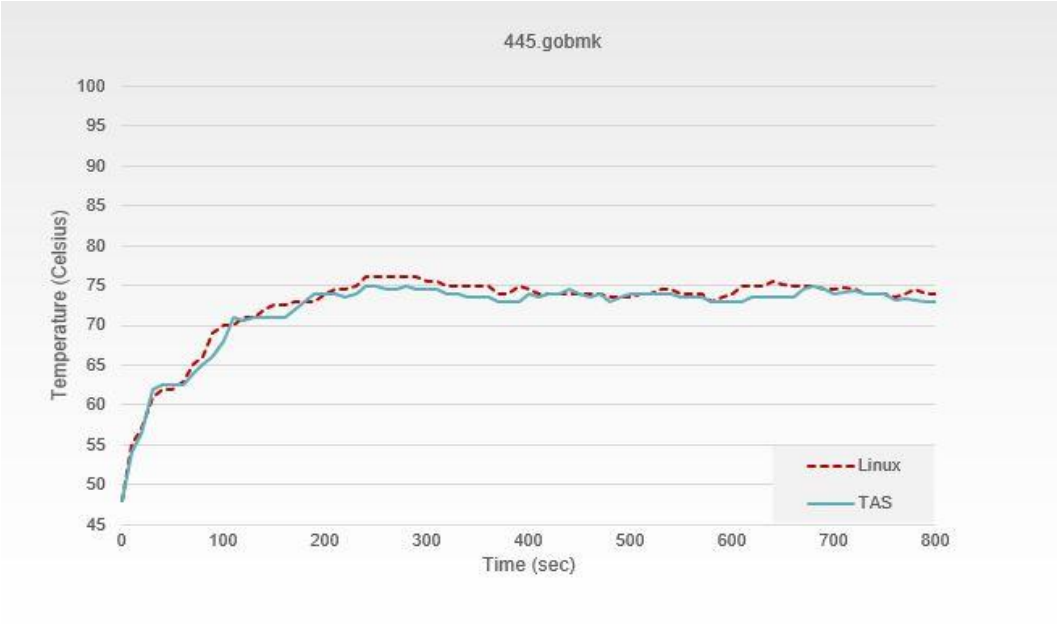


Figure 5.5: Temperature pattern comparison of 445.gobmk benchmark run using TAS and LSS.

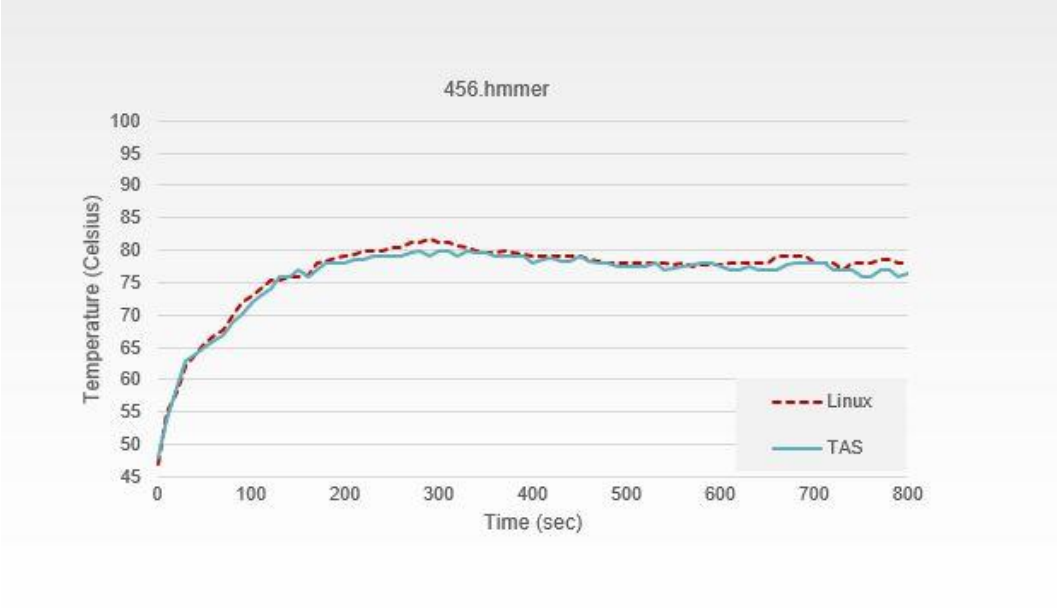


Figure 5.6: Temperature pattern comparison of 456.hmmer benchmark run using TAS and LSS.

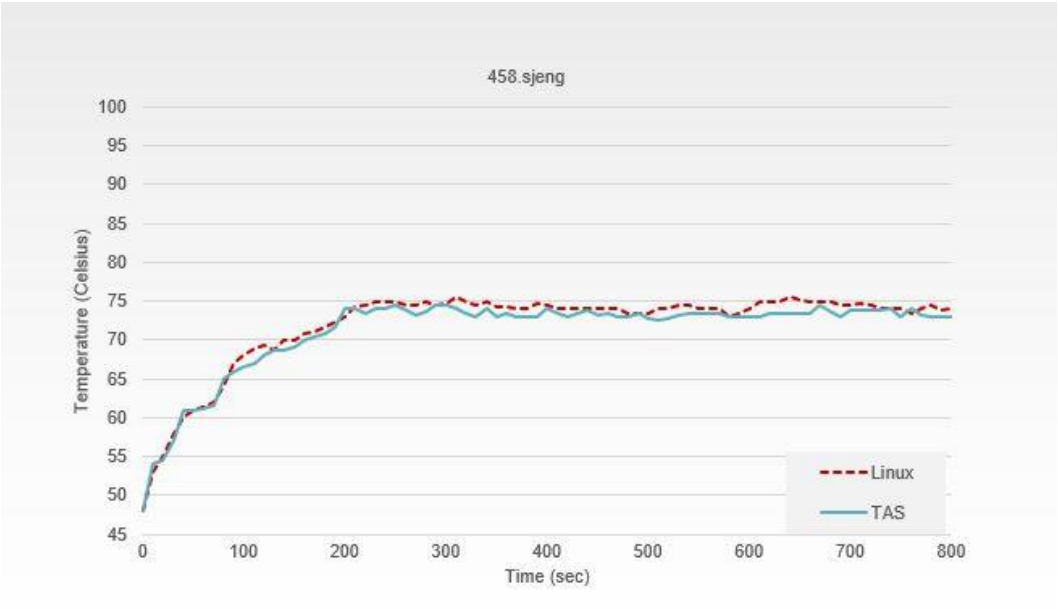


Figure 5.7: Temperature pattern comparison of 458.sjeng benchmark run using TAS and LSS.

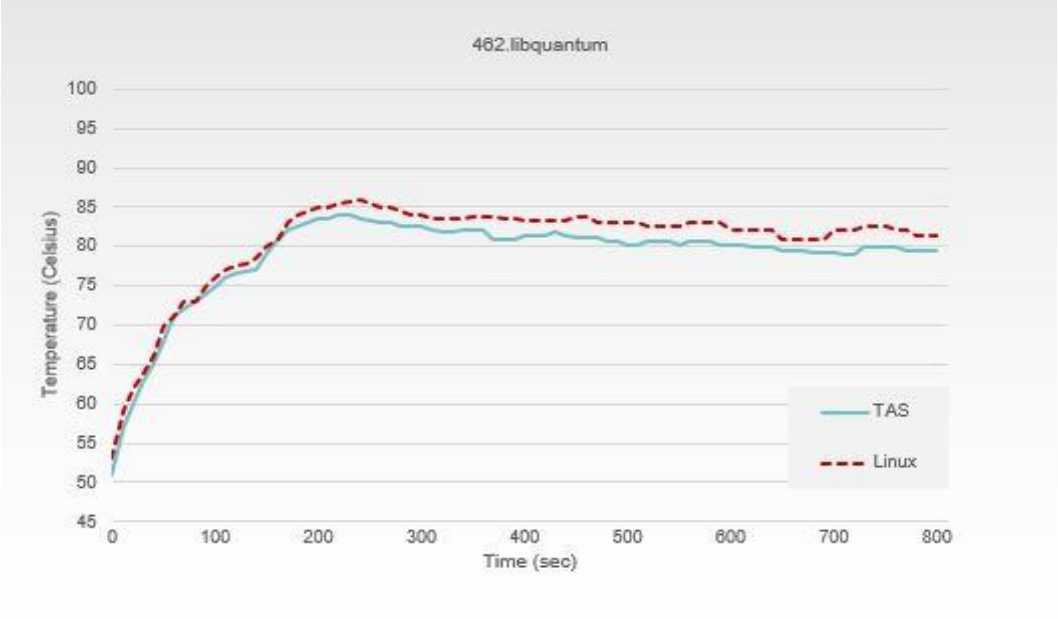


Figure 5.8: Temperature pattern comparison of 462.libquantum benchmark run using TAS and LSS.

As we can see in the above figures that the peak temperature has been reduced using our temperature aware scheduling using coolest core as compared to the Linux standard scheduler in all of the benchmark application suites. We have also observed the thermal temperature patterns of the different application suites and when steady sate temperature is attained. We have successfully categorized the applications based on the steady state temperature. The grouping of the applications is done at the run time and we have validated the data and groups by running each application offline until we get the steady state temperature.

In the next section we have discussed about the different groups and application falling in each of the category and it clearly shows the applications having same thermal groups reach similar steady state temperature range. We run experiments individually for each application benchmarks and allowed processor to cool down before initiating another run of the benchmark.

## 5.2 Applications' Grouping

In this section we define the groups of the applications based on the data, which is collected during the experiments offline and online. The temperature range for each thermal group is defined by considering the steady state temperature and nature of the application. We have also considered other parameters such as thermal pattern, complete execution time, peak and average temperature, into grouping of the application benchmarks. In the previous chapter we have described the groups of different applications. In Table 4.2, the groups of the individual applications are mentioned.

In the table below grouping of the applications made is described on the basis of temperature range. As per the data collected offline we considered and placed the applications into different groups. Please see the table below, which describes steady state temperature range for four different application groups.

| **SPEC CPU2006 Applications (Grouping for 100% Workload)** | **Steady State Temperature Range (Celsius)** |
|---|---|
| Group - 1 | 72 - 75 |
| Group - 2 | 75.1 - 78 |
| Group - 3 | 78.1 - 82 |
| Group – 4 | 82 - Above |

Table 5.1: Application Groups with Temperature Range

# Chapter 6

# Conclusion and Future Work

## 6.1 Conclusion

In this research we have tried to achieve the goal of reducing the overall and peak temperature in a processor and also avoid thermal emergencies during the execution of application. This research shows considerable improvements can be achieved when we consider the nature and behavior of the applications. Our experiments and results show how the temperature characteristics of different applications differ on the same multicore chip.

We categorized the same kind of applications in the same group according to their steady state temperature. Also, we have used the coolest core algorithm to schedule the tasks during run-time and the prediction method uses the offline data to categorize the new application and predict its group by accessing the operating temperature of the application in a pre-defined time frame. In the conclusion, our results show that the peak temperature of different applications has been reduced in the temperature range from 1.0 - 1.6C. The overall temperature of the four cores is reduced by 1.4C on an average for the SPEC 2006 application benchmarks. We have validated the prediction method using the offline runs of the benchmarks.

## 6.2 Future Work

Some of the future work on this research can be as follows:

- This prediction method can be applied to different application suite benchmarks and real time applications.
- We have used the coolest core algorithm to schedule the tasks but there is a good scope of implementing other temperature aware scheduling algorithms with this prediction method.
- In coming years, applications and workload will play major role in the design and development of dynamic thermal management systems and this prediction method can be implemented to different floor plans with 8/16/32 cores.

Bibliography


[1] Gyung-Leen Park, B. Shirazi, J. Marquis and Hyunseung Choo, "Decisive Path Scheduling: A new list of Scheduling method", in *International Conference on Parallel Processing*, 1997, pp. 472–480.

[2] D. Wang, "Meeting Green Computing Challenges", in *International symposium on High Density packaging and Microsystem Integration*, 2007, pp. 1–4.

[3] A.C. Orgerie, L. Lefevre and J.P. Gelas, "Save watts in your Grid: Green Strategies for Energy Aware Framework in Large Scale Distributed Systems" in *14th IEEE conference on Parallel & Distributed Systems*, 2008, pp. 171-178.

[4] R. Harmon, H. Demirkan, N. Auseklis and M. Reinoso, "From Green Computing to Sustainable IT: Developing a Sustainable Service Orientation", in *43rd International Conference on System Sciences (HICSS)*, pp. 1–10.

[5] Truong Vinh, Truong Duy, Y. Sato and Y. Inoguchi, "Performance Evaluation of a Green Scheduling Algorithm for Energy Savings in Cloud Computing" in *IEEE International Symposium on Parallel & Distributed Processing*, 2010, pp. 1–8.

[6] W.L. Hung, Y. Xie, N.Vijaykrishnan, M. Kandemir and M.J. Irwin, "Thermal-Aware Task Allocation and Scheduling for Embedded Systems" in *vol.2. of Design of Automation and Test in Europe*, 2005, pp.898-898.

[7] K. Stavrou and P. Trancoso, "Thermal-Aware Scheduling: A solution for Future Chip Multiprocessors Thermal Problems" in *9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, 2006, pp. 123–126.

[8] D.Niyato, S.Chaisiri and Lee Bu Sung, "Optimal Power Management for Server Farm to Support Green Computing" in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 84–91.

[9] J. Williams and L. Curtis, "Green: The new computing coat of arms" in *IT Professional, Vol.10, Issue.1*, 2008, pp.12-16.

[10] Dong Li, Hung-Ching Chang, H.K. Pyla and K.W. Cameron, "System Level, Thermal Aware, Fully Loaded Process Scheduling" in *International Symposium on Parallel & Distributed Processing*, 2008, pp.1-7.

[11] Inchoon Yeo and Eun Jung Kim, "Thermal Aware Scheduler Based in Thermal Behavior Grouping in Multicore Systems" in *Europe Conference and Exhibition on Design, Automation and Test,* 2009, pp- 946-951.

[12] G. von Laszewski, J. Dayal, Xi He, A.J. Younge and T.R. Furlani, "Towards Thermal Aware Workload Scheduling in a Data Center" in *10th International Symposium on Pervasive Systems, Algorithms and Networks*, 2009, pp- 116-122.

[13] Lizhe Wang, G. von Laszewski, J. Dayal and Fugang Wang, "Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS" in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing,* 2010, pp-368-377.

[14] D. Rajan and P.S. Yu, "Temperature Aware Scheduling: When is System-Throttling Good Enough?" in *9th International Conference on Web-Age Information Management,* 2008, pp-397-404.

[15] Donghwa Shin, Jihun Kim, Jinghang Choi, Sung Woo Chung and Eui- Young Chung, "Energy-Optimal Dynamic Thermal Management for Green Computing" in *IEEE/ACM Conference on Computer-Aided Design*, 2009, pp-652-657.

[16] J. Baliga, R.W.A. Ayre, K. Hinton, R.S. Tucker, "Green Cloud Computing: Balancing Energy in Processing, Storage and Transport" in *Proceedings of IEEE, Vol.PP, Issue.99,* 2010, pp-1-19.

[17] Kyungsu Kang, Jungsoo Kim, Sungjoo Yoo and Chong-Min Kyung, "Temperature-Aware Integrated DVFS and Power Gating for Executing Tasks with Runtime Distribution" in *IEEE Transactions, Vol.29, Issue.9*, 2010, pp-1381-1394.

[18] K. Skradon, M.R. Stan, Wei Huang, S. Velusamy, K. Sankaranarayanan and D. Tarjan, "Temperature Aware Computer Systems: Opportunities and Challenges" in *Micro IEEE, Vol.23, Issue.6,* 2003, pp-52-61.

[19] K. Skadron, M.Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Transactions on Architecture and Code Optimization (TACO), vol. 1, no. 1,* 2004.

[20] Jun Li and Zhongfei Wang, "An Application-Oriented Temperature Aware Scheduler in Linux Kernel" in *2nd IEEE Conference on Applied Robotics for the Power Industry*, 2012, pp-965-970.

[21] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, "Operating System Concepts, 8th Edition", 2008, Chapter 5.