Proactive Thermal-Aware Scheduling

by

Sankari Swaroop Anupindi

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 13, 2014

Keywords: thread scheduling, process scheduling, temperature aware,
regression and derivative

Approved by

Sanjeev Baskiyar, Chair, Associate Professor of Computer Science and Software Engineering
Cheryl Seals, Associate Professor of Computer Science and Software Engineering
James Cross, Professor of Computer Science and Software Engineering

Abstract

Modern CPU's cut-off operations when CPU temperature reaches a predetermined threshold making the CPU unavailable for all processes. Furthermore, operating the CPU for extended periods at temperatures close to, but slightly below, hardware cut-off, lowers reliability and lifetime of the CPU. In this dissertation, we develop proactive scheduling techniques to manage CPU temperatures by cutting off the major heat dissipating processes rather than the entire CPU. Such proactive scheduling promotes better component life, lower cooling fan usage, improved battery life and better availability. The techniques can be implemented over existing dynamic voltage and frequency scaling, dynamic power management, leakage energy and location-based techniques.

Memory accesses and floating-point operations are two major heat-dissipating activities in many programs. The first proactive approach developed is called Proactive Thermal Aware Scheduler (PTAS). PTAS forms a temperature predictor using the regression of the time derivatives of the number of Floating-Point Operations per Second (FLOPS) and the current CPU temperature. The predictor is used to make proactive scheduling decisions to handle thermal emergency before the temperature reaches the hardware cut-off. If the value of the predictor for any process is above an empirically determined

cut off, it is deemed likely that in the near future, the CPU will reach the hardware cut-off temperature. Therefore, that process is moved to the sleep state for a short duration. We analyzed the performance of PTAS using Scimark benchmarks in lowering CPU temperature. The reductions in peak temperatures were 2-4°C for FFT, LU, SOR, and Sparse (small) components of the Scimark benchmark runs respectively. For the larger versions of the aforementioned benchmark component runs, the reductions were 2-4°C respectively. The reductions in peak/average temperature on a laptop were 3-5/5°C. The corresponding penalties in schedule lengths were between 15-30%.

The second approach is called Proactive Thermal Aware Scheduling with Floating-Points and Memory access rates (PTFM). In this approach, a future temperature impact predictor (TIP) for any process is formed using a regression of the time derivatives of FLOPS, memory accesses and current CPU temperature. If the TIP for any process goes above a predetermined threshold, that process is put to sleep for a short duration. We evaluated the scheduler on small and large components of FFT, LU, SOR and Sparse within the Scimark benchmark suite. We found decrease in peak/average CPU temperatures: 3-6°C/6°C for small benchmarks and 3-6°C/5°C for large benchmarks. The schedule length penalties were less than 2-10%. The corresponding results in peak/average temperature on a laptop were 3-6/6˚C. We compared our results against other threshold based cut-off approaches: simple temperature, simple time derivative based cut-off strategies and PTAS. We found PTFM outperformed these strategies.

ACKNOWLEDGMENTS

I acknowledge Dr. Sanjeev Baskiyar for his valuable suggestions, ideas and directions. I thank my committee members Dr. Cheryl Seals, Dr. James Cross and Dr. Adit Singh for giving critical comments during the review process. I sincerely express gratitude for Dr. David Umphress and Dr. Xiao Qin for guiding me in the initial stages. I am grateful for all the support my spouse Kalyani has given to me. My love for my son Aarush made me stronger. I appreciate my family members for wonderful support. In addition, I recognize my research group members Vibudh, Adarsh, Matt, Brad, and James who gave valuable insights to make this research successful. Finally, I respect God for inspiration and making this research successful.

TABLE OF CONTENTS

ABSTRACT II

ACKNOWLEDGMENTS IV

LIST OF TABLES VII

LIST OF FIGURES IX

LIST OF ABBREVIATIONS XII

CHAPTER 1 INTRODUCTION 13

CHAPTER 2 BACKGROUND 15

CHAPTER 3 PROACTIVE THERMAL AWARE SCHEDULING 21

3.1. RELATED WORK 21
3.2. APPROACH 27
3.3. EXPERIMENTS SETUP 31
3.4. RESULTS AND DISCUSSION 34
3.5. CONCLUSION 41

CHAPTER 4 PROACTIVE THERMAL MANAGEMENT USING FLOPS VIA MEMORY RATES 43

4.1. RELATED WORK 43
4.2. APPROACH 48
4.3. EXPERIMENTS SETUP 50
4.4. RESULTS AND DISCUSSION 53
4.5. CONCLUSION 59

CHAPTER 5 CONCLUSION 81

BIBLIOGRAPHY 82

APPENDIX.. 85

LIST OF TABLES

Table 1 Peak FLOPS and peak CPU temperature for all benchmarks when running on desktop..... 35

Table 2 Normalized STDEV of cut-off FLOPS and cut-off temperature over a single experiment with PTAS..... 39

Table 3 Symbols and description 50

Table 4 MFLOPS vs. time 62

Table 5 Temperature vs. time 62

Table 6 CPU temperature without benchmarks and with benchmarks running 63

Table 7 Execution time of different benchmarks for the desktop and laptop 63

Table 8 Peak Temperature using PTAS in Desktop and laptop for various benchmarks 64

Table 9 Average CPU temperature using PTAS in desktop and laptop for various benchmarks..... 64

Table 10 Peak CPU Temperature of smaller benchmarks (FFT, LU, SOR and Sparse) when executed together..... 65

Table 11 Peak CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 10 ms..... 66

Table 12 Average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 10 ms..... 66

Table 13 Peak CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 500 ms..... 67

Table 14 Average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 500 ms..... 67

Table 15 Peak CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 1 s 67

Table 16 Average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 1 s 68

LIST OF FIGURES

Fig. 1 Algorithm PTAS 29

Fig. 2 Peak temperatures of ISA adapter and cores with and without execution of all benchmarks 32

Fig. 3 Execution time of different benchmarks for the desktop and laptop 36

Fig. 4 MFLOPS vs. time when benchmarks are executed in sequence (FFT, LU, SOR, and Sparse) 37

Fig. 5 CPU temperature during execution when small benchmarks(FFT, LU, Sor and Sparse) are executed together 37

Fig. 6 Peak temperature using PTAS in desktop and laptop for various benchmarks 39

Fig. 7 Average CPU temperature using PTAS in desktop and laptop for various benchmarks 40

Fig. 8 Algorithm PTFM 47

Fig. 9 CPU Temperature for smaller benchmarks when executed successively 55

Fig. 10 Peak CPU temperatures with a sleep time of 10 ms 56

Fig. 11 Average temperature for a sleep time of 10 ms 56

Fig. 12 Peak temperature for a sleep time of 500 ms 57

Fig. 13 Average temperature for a sleep time of 500 ms 57

Fig. 14 Peak temperature for a sleep time of 1 s 58

Fig. 15 Average temperature for a sleep time of 1 s 58

Fig. 16 Peak CPU Temperature comparison of PTFM on a desktop and laptop for a sleep time of 50 ms 59

Fig. 17 Average CPU Temperature comparison of PTFM on a desktop and laptop for a sleep time of 50 ms 60

Fig. 18 Peak CPU Temperature when all benchmarks are executed together 61

Fig. 19 Average CPU Temperature when all benchmarks are executed together 61

Fig. 20 Peak CPU Temperature due to FFT for 10 ms 69

Fig. 21 Average CPU Temperature due to FFT for 10 ms 69

Fig. 22 Peak CPU Temperature due to LU for 10 ms 70

Fig. 23 Average CPU Temperature due to LU for 10 ms 70

Fig. 24 Peak CPU Temperature due to SOR for 10 ms 71

Fig. 25 Average CPU Temperature due to SOR for 10 ms 71

Fig. 26 Peak CPU Temperature due to Sparse for 10 ms 72

Fig. 27 Average CPU Temperature due to Sparse for 10 ms 72

Fig. 28 Peak CPU Temperature due to FFT for 500 ms 73

Fig. 29 Average CPU Temperature due to FFT for 500 ms 73

Fig. 30 Peak CPU Temperature due to LU for 500 ms 74

Fig. 31 Average CPU Temperature due to LU for 500 ms 74

Fig. 32 Peak CPU Temperature due to SOR for 500 ms 75

Fig. 33 Average CPU Temperature due to SOR for 500 ms 75

Fig. 34 Peak CPU Temperature due to Sparse for 500 ms 76

Fig. 35 Average CPU Temperature due to Sparse for 500 ms 76

Fig. 36 Peak CPU Temperature due to FFT for 1 s 77

Fig. 37 Average CPU Temperature due to FFT for 1 s 78

Fig. 38 Peak CPU Temperature due to LU for 1 s 78

Fig. 39 Average CPU Temperature due to LU for 1 s 79

Fig. 40 Peak CPU Temperature due to SOR for 1 s 79

Fig. 41 Average CPU Temperature due to SOR for 1 s 80

LIST OF ABBREVIATIONS

DVFS Dynamic voltage and frequency scaling

DPM Dynamic process management

DTM Dynamic thermal management

FLOPS Floating point operations per second

Chapter 1 Introduction

CPU overheating is a major problem that can occur due to various CPU characteristics. This includes chip material, the characteristics of CPU processes, and thermal decisions taken by the CPU. Overheating results in permanent damage to the CPU, and manufacturers face this problem year after year. When the CPU temperature increases beyond a certain threshold, it decreases chip reliability and increases the CPU's cooling costs. Decreasing CPU chip temperature has become a major challenge. Every year, thousands of data centers spend millions of dollars to mitigate this problem, and every year, data loss occurs in computers due to severe thermal problems in the CPU. The CPU's energy consumption also increases with rising temperatures as manufacturers use several cooling techniques to control CPU temperature. Therefore, there is a need to control thermal-related CPU damage. We hypothesize that the rise in CPU temperature can be reduced by proactively scheduling processes using process characteristics. In our approach, we predict the gradient of the process by measuring FLOPS and memory accesses, and cut off the process with a higher gradient. We optimize the sleep time, gradient, and priority of the process.

Researchers have used Dynamic Voltage and Frequency Scaling (DVFS), Dynamic Power Management (DPM), leakage energy reduction, and variability-aware thermal

management, or a combination of these approaches [1], [2]. In addition, CPUs employ fan(s) and thermal cut-off to control chip temperature. In this research, we use a predictive and proactive process scheduling approach to manage chip temperatures.

The major contributions of this work are:

1. Review of state of the art approaches related to Proactive Thermal-Aware CPU Scheduling.
2. Use of time derivatives of temperature, FLOPS and memory access rates to predict the temperature.
3. Use of regression consisting of the above to predict the future temperature and proactively put the processes to sleep.
4. Evaluation of the approach and its outcomes as related to peak and average CPU temperature, with simulations using SciMark benchmarks.
5. The reductions in CPU temperatures [3] [4] are important as they result in increased component life of the CPU.

Chapter 2 Background

Our goal in the temperature-aware scheduling approach was to reduce CPU temperature. We can achieve this goal by scheduling processes in the CPU. By scheduling these processes in FIFO order, we make thermal decisions to reduce CPU temperature. There are several strategies to reduce CPU temperature. Foremost is the Dynamic Voltage and Frequency Scaling strategy, which varies voltage and frequency levels of the CPU to control its temperature. This is a popular and effective strategy for controlling CPU temperature; however, it is a reactive strategy. A proactive strategy to decrease CPU temperature and prevent thermal emergencies is necessary. A reactive strategy waits until the temperature reaches its threshold and slowly cuts off the processes, which can cause permanent CPU damage. We can prevent this damage by cutting off the process in advance, thus increasing CPU reliability and reducing cooling costs.

Another strategy is Dynamic Thermal Management (DTM), or Dynamic Process Management (DPM). In this strategy, we make thermal decisions dynamically to reduce temperature. We can implement this strategy at a software level, at a hardware level, or both. This is a reactive strategy, which allows CPU temperatures to scale up and bring

down the CPU temperature by effectively scheduling processes. Similar to DVFS, a proactive variant of this strategy would effectively reduce CPU temperature. It also increases CPU performance and energy consumption. The proactive strategy would also increase battery lifetime.

Another strategy aims to decrease CPU leakage energy to reduce its temperature. This strategy reduces CPU temperature and saves energy. We can use this strategy in conjunction with DVFS and DTM. We gain thermal improvements by implementing a proactive variant of this strategy.

A final strategy aims at allocating tasks by taking the variability factor of the CPU. We consider this factor to gain significant thermal improvements. This strategy has improvements above and beyond DVFS and DPM. It aims at increasing chip reliability and preventing thermal damage. It gives good improvement with thermal emergencies. However, it is not an entirely proactive strategy. A proactive improvement of this strategy would be a good approach.

Researchers have implemented scheduling techniques at an OS level. The goal of operating system (OS) scheduling is to increase CPU performance and maximize CPU utilization. It has to give better user response. There are algorithms with complexity of O (n^2) and O (n). However, algorithms with O (1) complexity effectively achieve the above goals. In such algorithms, time taken by the scheduler is short, irrespective of the

input size. In the quadratic and linear time algorithms, tasks take much longer to finish, and thus make the scheduler less scalable.

Previous versions of the Linux scheduler used a run queue in a symmetric multiprocessor, resulting in load balancing of the tasks. However, it resulted in bad memory caches. In addition, this queue locked the processes, which made processes take longer to execute. Finally, if some tasks took longer than others did, preemption was not possible.

Ingo Molnar developed Linux scheduler [5] with O (1) complexity. He developed a scheduler for wakeup, context switch, and time slicing. In addition, he used Java Virtual Machine (JVM) to reduce the overhead caused by thread execution in the O (n) scheduler. In this algorithm, OS used First In First Out (FIFO) with 140 priority lists in the run queue. Each task had a time slicing that let the scheduler decide the duration of scheduling tasks. The OS reserved the first 100 priorities for real time tasks, and reserved the next 40 tasks for user tasks (see Fig. 1). In addition to the OS's active queue, there was also an expired run queue. It placed expired tasks on an expired queue and active tasks on an active queue. If the tasks on the active queue were empty, it swapped tasks on the expired queue. The scheduler executed the highest priority task. It also used a bitmap to determine when the tasks were on a high priority list. Since the time to execute the tasks was dependent on priorities rather than task input, the scheduler's complexity was O (1). Thus, the Linux scheduler is a deterministic scheduler.

The Hadoop scheduling algorithm [6] uses fair scheduler and capacity scheduler. The Hadoop scheduler has a job tracker and a task tracker to schedule tasks on a cluster or a grid. The core of Hadoop architecture consists of master and slave nodes. The name node is a master node that controls filenames and clients. It distributes jobs to slave nodes. Slave nodes include the task tracker, which completes jobs and notifies the job tracker. Data node is a storage node, which represents the distributed file system. Both task tracker and data node are slaves in Hadoop architecture.

Hadoop is a fault-tolerant architecture, meaning that it operates when nodes fail and restart. Hadoop runs these nodes in isolation mode, where they do not have access to other nodes. Hadoop uses pluggable schedulers. First among them is the FIFO scheduler, which schedules jobs on a first-come first-served basis. It executes these jobs in the order they arrive, and gives higher priority to a job that has to be scheduled earlier. Another scheduler is the fair scheduler, which assigns equal share to each task. On average, each task gets an equal share of time. This helps the scheduler to spend equal time on all tasks, thus increasing its response time. In addition, Hadoop uses a job pool to assign jobs, and it shares these pools among tasks. It also gives equal share to each pool. Third is a capacity scheduler, which estimates cluster capacity and schedules tasks accordingly. The Hadoop server load balances the tasks so that it runs the scheduler with less response time. It executes high load levels without any change in the schedule.

SciMark benchmarks are Java benchmarks for making numerical calculations in scientific and engineering applications. They include Fast Fourier Transforms (FFT), Gauss-Seidel Relaxation, Sparse Matrix Multiplication, Monte Carlo Integration, and Dense LU Factorization. There are two versions of this benchmark. The smaller version of the benchmark focuses on CPU issues, while the larger version of the benchmark addresses the memory subsystem and out-of-cache problem sizes.

Fast Fourier Transform (FFT) consists of a 1-D forward transform of 4k complex numbers. In addition, this kernel performs complex arithmetic, shuffling, and non-constant memory references. It consists of two versions: one performs bit reversal, while the second version performs Nlog(N) computations. Jacobi Successive Over-Relaxation (SOR) does grid averaging on memory patterns in finite difference applications. This kernel exerts access patterns on a 100x100 grid. Sparse Matrix Multiplication uses indirection addressing and non-regular memory references in an unstructured matrix. Dense LU Matrix Factorization calculates the LU factorization of a dense 100x100 matrix using partial pivoting.

PAPI is a low-level interface for measuring the performance counters of hardware in most major microprocessors. PAPI measures the relation between performance and processor events. PAPI has components that measure real time software and hardware performance in most major processors. We can use PAPI in a real time scheduler to measure floating points of the processes in constant time. It gives a good predictor of the rate of change of floating points and memory accesses. We can install PAPI as a

library on the system, and use an event driven model for floating-point calculations. It gives lower overhead when used in programs.

Regression is normally used to predict a possible relationship between a dependent variable and a set of independent variables. We reviewed current state of the art thermal management implementations in the industry. A majority of these implementations are hardware techniques. Intel uses the PID controller to reduce maximum temperature in CPU chips. The PID controller is a reactive method that reduces residual error using a combination of integral and derivatives. In addition, thermal monitoring using fan sink and fan speed is used to reduce CPU chip temperatures. By increasing fan speed, reducing thermal noise, and setting higher and lower threshold points, Intel schedulers decrease CPU temperature. Finally, airflow and clock modulation with bit encoding help to reduce maximum temperature in CPU chips.

Chapter 3 Proactive thermal aware scheduling

CPU overheating is a major problem that is dependent on factors such as chip material and the rapidity of power dissipation. High temperatures reduce chip reliability and decrease its lifetime. Chip manufacturers use hardware, software, and hybrid approaches for CPU temperature reduction.

The remainder of this chapter is organized as follows: In Section 2, we discuss related work; in Section 3, we introduce PTAS and our experimental methodology; in Section 4, we discuss the results of our work, and in Section 5, we make concluding remarks. The related work section contains a review of academic papers as well as contemporary products in the industry.

3.1 Related Work

The majority of techniques used to reduce chip temperature are reactive, meaning chip temperature is allowed to rise to threshold levels, and then steps are taken to bring down the temperature [1] [2]. Such techniques have used strategies such as Dynamic Voltage Frequency Scaling (DVFS), Dynamic Power Management (DPM), Dynamic Thermal Management (DTM), leakage energy, and variability-aware thermal management. However, some of these approaches result in thermal emergencies [1] [2] due to sudden

spikes in chip temperature, which could cause irreversible damage to the chip. A 10°C reduction in CPU temperature below the hardware threshold cut-off produces a 20% increase in chip lifetime and reliability [3], [4].

Xiuyi et al. [1] identified temperature correlation among vertically adjacent layers in 3-D chips. They used OS-level task scheduling to minimize peak temperatures by identifying such sources of heat. Using Dynamic voltage and frequency scaling in OS scheduler, they reduced hardware DTMS in CPUs by 54% and improved CPU performance by 7.2%.

Coskun et al. [2] used an integer linear programming approach in task scheduling to reduce thermal hotspots and temperature gradients in CPUs. Kumar et al. [7] developed a system-level framework using DTM with proactive temperature estimates using integer linear programming and hardware sensor measurements. The scheduler overhead on execution time was 24% when using just the reactive hardware measurements approach. However, upon using the framework consisting of both the software and hardware approaches, the overhead improved to 10%.

Chuan et al. [8] identified leakage energy to be a factor in the increase of peak chip temperature. They developed energy-efficient scheduling to reduce leakage energy in CPUs. This consisted of a patterns-based approach that divided schedule length into active and dormant windows. Leakage power consumption was reduced by increasing

the speed of tasks in active mode, and allowing the CPU temperature to cool in dormant mode.

Wei and Nannarelli [9] discovered that the recurrence digit in floating-point computations was one of the main causes of heat increase in caches. They used Fused Multiply-Add (FMA) to reduce the peak temperatures in caches. They placed power-efficient drivers between FMA and cache block that reduced leakage energy by 12%, average temperature in caches by 5°C, and power consumption by 8.4%.

Chaturvedi et al. [10] developed validation for scheduling techniques on architectural-level platforms that reduce peak temperature. This technique used m-oscillation (or DVFS) to reduce dynamic energy. Senju et al. [11] investigated the Particle Swarm Optimization (PSO) strategy to reduce peak temperatures in clusters and grids. Schedules are distributed onto clusters based on fitness values. Cluster-best and personal-best schedules, rather than global-best schedules, were selected using binary PSO. Komada et al. [12] used electro-thermal coupling to reduce thermal interference in CPUs. They investigated the accuracy of predicting the thermal behavior of silicon chips. The thermal model used in the strategy accurately predicted chip temperature. Fisher et al. [13] investigated global real time scheduling of homogeneous tasks on multi-core systems. When performing matrix computations using the mathematically computed preferred speed of each core, it reduced peak temperatures by 30-70°C lower than load balancing strategies.

Jin and Maskell [14] developed a thermal-aware model at event level. This event-driven thermal model was used to create a thermal map when high-level events occur in the CPU. Taking temperature increments of each core, a number of lookup tables were prebuilt offline. Afterward, a thermal map was updated online using the superposition principle.

Jiajia et al. [15] used thermal-aware mapping methods on 3-D torus chip to increase throughput and latency. Using this strategy and CPU execution cycles in FFT, matrix multiplication and radix sort were reduced by 6.78%, 5.77%, and 4.07% in one experiment, and 8.58%, 10.37%, and 21.28% in another experiment.

Yang et al. [16] optimized Energy and Performance-Delay Product (EDP) using helper threads. The EDP criterion helps trade energy with performance. Helper threads, which help cool the CPU, were added to optimize EDP. This strategy varied different data points, such as CPU count, thread count, and voltage/frequency level, to optimize EDP. They measured EDP using performance counters. EDP reduction of 60-80% was achieved for FFT and multi-grid benchmarks. In addition to EDP, this approach reduced thermal emergency.

Merkel and Bellosa [17] determined that task migration is better than throttling the CPU, except in worst-case situations. They created an energy-aware scheduling policy on a Linux machine using Dynamic Voltage and Frequency Scaling by modifying the task

data structure in a Linux scheduler. Migrating hot tasks to coolest core and balancing energy of the tasks generated significant energy savings. The cost of migration was small compared to throttling. In addition, when tasks were load balanced, throughput increased and overhead lessened on the scheduler.

Ayoub and Rosing [4] created a temperature predictor by using the bandwidth of the temperature frequency spectrum and workload characterization of the tasks. The workload characterization of the tasks was measured by finding the task fetch rate. They used both parameters to reduce the average temperature of the tasks. They created an experiment with SPEC CPU 2000 benchmarks and compared their approach with other reactive approaches. Using this scheme, they reduced the average temperature of hottest cores by 6-8°C. There was a performance improvement of 41% and 72%. They discovered that average CPU temperature was related to Mean Time to Failure (MTTF).

Weissel and Bellosa [18] used event performance counters to measure the run-time characterization of the tasks. They assigned weights to events and used performance counters temperature and a count of CPU cycles to measure the energy. They used processor throttling to save energy and reduce temperature. They framed the problem as a linear optimization equation.

Altet and Rubio [3] discovered that a 10°C reduction in temperature below the hardware threshold cut-off produces a 20% increase in lifetime. They studied component lifetime

and the lifetime impact model of chips by reducing temperature using scheduling approaches.

Bellosa [19] examined the impact of a coarse-grained approach versus a fine-grained approach on a scheduling decision. He found that lightweight threads reduced scheduling decisions and minimized context switching by the scheduler. A fine-grained approach should increase the efficiency and scalability of the scheduler in computer systems.

Intel uses a PID controller [20] to reduce maximum temperature in CPU chips. The PID controller reactively reduces residual error gain, integral gain, and derivative gain to control fan speeds. Intel sets an upper level of temperature, or the thermal threshold point, to ensure that the CPU runs below dangerous levels. The threshold point is far below the critical thermal point at which semiconductor hazards occur. The CPU chip was designed with airflow ventilation. Clock frequency modulation varies the frequency to reduce CPU temperature. AMD [21] uses task migration to coolest core along with coarse- and fine-grained controls to turn off registers and thus save energy. When the temperature rises, a multi-point control using sensors reduces CPU performance states (p-states). The p-states define the frequency and voltage of the CPU. Depending on usage, it also reduces temperatures by using dual dynamic power management to vary the voltage of the cores and the integrated memory controller independently. ARM uses kernel thermal framework [5] to register thermal zones, and cooling devices for reducing temperature. Kernel thermal framework sets the power gating and clock gating of the peripherals and components. Devices can configure which of the chip's thermal sensors

and cooling devices to use on specific platforms. Linux thermal management [22] uses thermal zones with active, passive, and critical cooling points. Thermal zones are different trip points after which temperature increases. These zones are used to differentiate temperature levels, with passive being a low temperature trip point, and critical being a dangerous trip point in the CPU. Linux implements generic thermal management architecture to control temperature, using drivers for the cooling device (fan), event framework, and thermal zones. A platform-independent Sysfs driver architecture has been used to interact with platform specific thermal management [23]. In advanced configurations, device reconfiguration is used for power management. A third-party tool, Coretemp [24], uses digital thermal sensors to monitor temperature on each CPU core. All major processor manufacturers utilize Coretemp.

3.2 Approach

In this study, we propose a proactive scheduler to reduce temperature. The approach attempts to put a process to sleep before it can cause the CPU to reach threshold temperature. We base our approach on the observation [9] that temperature is dependent on FLOPS. We also observed a rise in temperature for SciMark benchmark programs, which have substantial floating-point computations. Fig. 2 shows the temperature of the ISA adapter and the cores before and after all SciMark benchmarks were run together. It shows that the temperature rises about 20°C upon execution of all the benchmarks.

The motivation for using the time derivative of FLOPS as a parameter in regression is the observed higher temperatures during intensive floating-point operations. The

motivation for using the rate of rise of temperature as a regression parameter comes from the recognition that when the temperature increases by repeated execution of a certain part of code, it is likely to continue that path (or loop) until that part terminates.

We use a multiple regression[1] consisting of the partial time derivative of FLOPS, *F* (which is the independent variable in the regression), the application process, and the partial time derivative of temperature (the dependent variable) to predict the CPU temperature. In actual implementation, we can use Performance Application Programming Interface (PAPI) or hardware counter, and the lm-sensors the Application Programming Interface (API) calls to determine the FLOPS and temperature of the CPU. The scheduler PTAS predicts the impact of the application level process on CPU temperature by using a regression of the current CPU temperature and the FLOPS generated by the application-level process. If the prediction was above an empirically determined threshold, the application process was put to sleep for a short duration. The duration for which the processes remain in the sleep state is empirically determined for best results.

[1] Regression is used to predict a possible relationship between a dependent variable and a set of independent variables.

Procedure *PTAS (***int** *i) // i* is observation number

const ThresholdGradient, K=1000

static float F[K], T[K], M[K], Y[K]

Process p, Q[n] //Application processes

//n is the number of processes

1 **For each** $p \in Q$ **do** // in FIFO
2 F_i = FLOPS (p)
3 T_i = CPUTemperature() //From lmsensors
4 Wait for δt time
5 F_{i+1} = FLOPS (p)
6 T_{i+1} = CPUTemperature()
7 $(\delta F/\delta t)_i = (F_{i+1}-F_i)/\delta t$
8 $(\delta T/\delta t)_i = (T_{i+1}-T_i)/\delta t$
9 Wait for δt time
10 F_{i+2} = FLOPS (p)
11 T_{i+2} = CPUTemperature()
12 $(\delta F/\delta t)_{i+1} = (F_{i+2}-F_{i+1})/\delta t$
13 $(\delta T/\delta t)_{i+1} = (T_{i+2}-T_{i+1})/\delta t$
14 $Y_i = 1 + (\delta F/\delta t)_i * F_i + (\delta T/\delta t)_i * T_i + \sigma_i(Y)$
15 $Y_{i+1} = 1 + (\delta F/\delta t)_{i+1} * F_{i+1} + (\delta T/\delta t)_{i+1} * T_{i+1} + \sigma_{i+1}(Y)$
16 $\delta Y_i/\delta t = (Y_{i+1} - Y_i)/\delta t$
17 **if** $\delta Y_i/\delta t >$ *ThresholdGradient* **then**
18 Sleep(p)
19 **endif**
20 **endfor**
end *PTAS*

Fig. 1 Algorithm PTAS

PTAS schedules the application processes in a FIFO manner. The procedure PTAS is outlined in Fig. 1 further elaborated below. In Equation 1, F_i and T_i represent the FLOPS and temperature for the i^{th} observation. We name the regression intercept Y_i, which represents the temperature predictor immediately following the i^{th} observation. Using regression, we have the intercept Y_i as:

$$Y_i = \beta_0 + \beta_1 F_i + \beta_2 T_i + \in \quad (1)$$

where, $\beta_0 = 1$, $\beta_1 = \frac{\delta F}{\delta t}$, $\beta_2 = \frac{\delta T}{\delta t}$, and $\in = \sigma(Y)$ is the standard deviation.

After computing the gradient Y_i of all application processes, PTAS moves the processes with gradients higher than threshold to the sleep state, allowing the CPU to cool. The goal was to maintain the temperature below threshold level, and thereby avoid spikes in temperature. The threshold gradient is the value of dY_i/dt (empirically determined) at which the process cuts off.

Probing the temperature and FLOPS took constant time. The complexity of PTAS is *O(n)*, where *n* is the number of processes. In our implementation, we used a Java Process API call to select the application-level processes, and Java Apache commons math API[2] to determine Y_i, using the historical standard deviation. We used scheduling quanta [25] and divisible load approaches [26] to schedule tasks in FIFO order. In the above two approaches, the process is broken down into manageable threads. It reduces load on the scheduler to execute the processes fairly and efficiently.

All the processes considered in this experiment are data-intensive tasks that have substantial floating-point operations, which increases the temperature significantly. The collective run results are shown in Fig. 2.

[2] org.apache.commons.math3.stat.regression

As stated earlier, when implementing PTAS in a real system, PAPI [27] or hardware counters may be used to measure FLOPS. The benchmark process computes floating-point calculations. In order to estimate the overhead due to PAPI in such an implementation, we ran benchmark programs with PAPI. These benchmark programs were Inner Product, Matrix Vector Multiplication, and Matrix Multiplication. We performed calibration on PAPI with various matrix sizes ranging from 2x2 to 500x500 to estimate the overheads on latency and throughput using PAPI. We found the overheads to be negligible, thus suggesting the feasibility of such an implementation. PAPI computes percentage error overhead between theoretical and real time computation of floating points. We conducted experiments to empirically determine error overhead due to PAPI for matrix vector test, inner product test, and matrix multiplication, and found the normalized error overheads to be 0.0016, 0.0000, and 0.0017 respectively.

3.3 Experimental Setup

The experiments were conducted on a desktop and laptop machine with the Ubuntu operating system. We used Ubuntu 9.10 on a Dell Optiplex 9020 i5 @ 2.90 Ghz desktop with 4 GB RAM and 320 GB HDD. For the laptop, we used Lenovo Intel Dual Core @2.10 Ghz with 4 GB RAM and 302 GB HDD. The ambient room temperature was 70°F with central air in the lab. We developed the scheduler using Java, and used Java APIs to calculate the temperature gradient. We ran our experiments on laptops as well, and saw similar benefits. We used the desktop results using Ubuntu in the dissertation as the OS is freely available, making the experiments easily replicable.

The experiment was run on eight SciMark benchmarks: Fast Fourier Transform (FFT), Jacobi Successive Over-Relaxation (SOR), Dense Unit Factorization (LU),

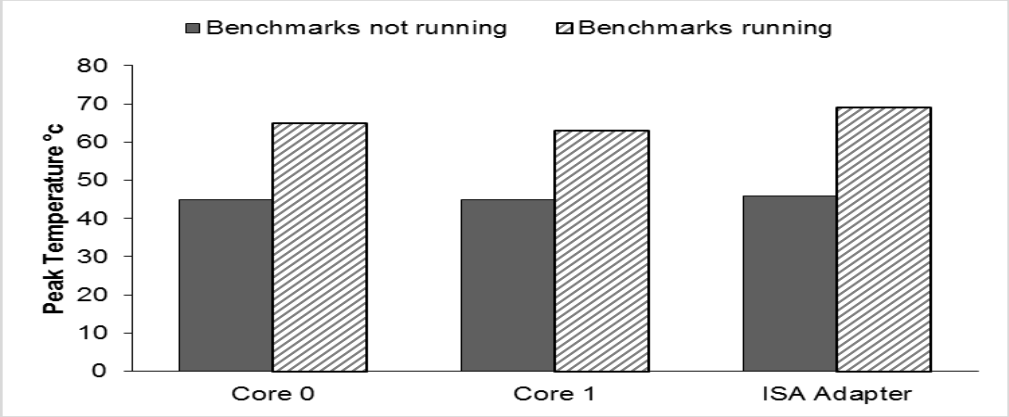

Fig. 2 Peak temperatures of ISA adapter and cores with and without execution of all benchmarks

Sparse Matrix Multiplication (Sparse), FFT-Large, SOR-Large, LU-Large, and Sparse-Large. When conducting the above experiments, the benchmark FFT does a 1-D transform of complex numbers. It uses complex arithmetic, shuffling, non-constant memory references, trigonometric, and bit reversal functions [28]. The benchmark SOR does a 100x100-matrix calculation on finite applications [28]. The benchmark LU uses pivoting methods on a 100x100 matrix to perform linear algebra kernels and dense matrix operations [28]. Sparse Matrix Multiplication computes 1000x1000 sparse matrix with uncompressed storage patterns [28].

The peak CPU temperature (and peak FLOPS) for a benchmark run was the maximum of all observed temperatures (and FLOPS) during the run of that benchmark. The average temperature (and average FLOPS) was a simple average of the observed values over time during the run of the benchmark.

We ran the benchmarks individually to estimate their specific impact on temperature. No processes other than those belonging to the OS were running on the computer on which the experiment was conducted. For each of the eight benchmarks mentioned above, we noted CPU temperatures, FLOPS, and schedule length with and without the PTAS scheduler. In addition, the CPU temperature was measured before and after each thread was sent to sleep to observe whether there was indeed a drop in CPU temperature. We ran each benchmark 10 times. The schedule length here represents the execution time. The results in the graphs shown in this dissertation are the average over these 10 runs.

Both sleep time and threshold gradient were empirically determined using several runs. The sleep time was chosen as the shortest time that can reduce CPU temperature below an empirically defined threshold temperature. In an implementation, these values can be user-programmable (such as in a data center) to achieve best outcomes of thermal and execution time savings.

To determine the best threshold gradient and sleep times, we ran the experiment many times with various threshold gradients and sleep durations. We kept the sleep duration short, such that the corresponding increase in the schedule length was minimal. We noted the threshold gradient and sleep duration for which there was a maximum decrease in temperature and used them for future experiments. In these experiments, the best values for threshold gradient and sleep time were found to be 0.24 units and 1msec respectively. We monitored δt by an electronic stopwatch. The duration for statements 5 and 6 in Fig. 1 were negligible compared to δt.

We probed the CPU temperature and FLOPS dynamically. The benchmarks provided a mechanism to probe the FLOPS. The benchmarks were modified to probe the CPU temperature using hardware sensors [29].

3.4 Results and Discussion

We found a favorable decrease in temperature for all eight benchmarks with PTAS. In FFT, LU, SOR, and Sparse benchmarks, temperature reductions were 2-4°C, whereas those in FFT-Large, LU-Large, SOR-Large, and Sparse-Large were 3-5°C. The reductions on a laptop were 3-5°C in FFT, LU, SOR, and Sparse, whereas those in FFT-Large, LU-Large, SOR-Large, and Sparse-Large were 3-6°C.

Table 1 shows the peak FLOPS and peak CPU temperature for all the benchmarks. The peak FLOPS using PTAS was smallest as the processes were put to sleep, when there were rapidly rising floating-point computations.

As expected, there was an increase in schedule length when using PTAS, because processes were put to sleep to reduce CPU temperature. Fig. 4 MFLOPS vs. time when benchmarks are executed in sequence (FFT, LU, SOR, and Sparse)shows the FLOPS versus time when all the small benchmarks (FFT, LU, SOR, and Sparse) are run together. This graph shows the FLOPS reduction with PTAS over the execution time of small benchmarks. The x-axis represents MFLOPS and the y-axis represents time. Fig. 4 MFLOPS vs. time when benchmarks are executed in sequence (FFT, LU, SOR, and Sparse)also shows the temperature versus time when small benchmarks are executed together. There is a decrease in CPU temperature with PTAS for the execution time of the small benchmarks.

Table 1 Peak FLOPS and peak CPU temperature for all benchmarks when running on desktop

	Peak FLOPS		Peak Temp (°c)	
Benchmark	Without PTAS	With PTAS	Without PTAS	With PTAS
LU Large	3503	2547	44	43
FFT Large	188	187	44	43
SOR Large	2032	2007	49	48
Sparse Large	1225	1162	45	43
LU	3510	3421	44	41
FFT	170	90	49	49
SOR	2377	2355	44	43
Sparse	1517	1280	49	48

In the smaller benchmark group (FFT, LU, Sparse, and SOR), the schedule length increases with PTAS were from 26-33%. The increase in schedule length of the Sparse benchmark was highest (33%). This could be attributed to the Sparse benchmark process causing a steep rise in peak CPU temperature due to large non-contiguous memory accesses.

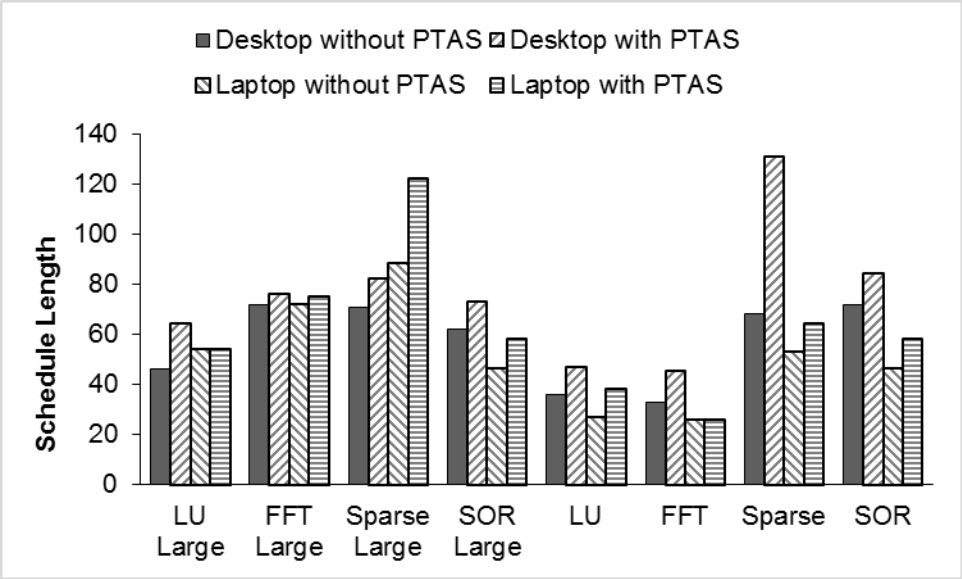

Fig. 3 Execution time of different benchmarks for the desktop and laptop

The peak CPU temperature of SOR without PTAS was highest. The maximum reduction of CPU temperature was in LU. This can be attributed to the higher FLOPS in LU compared to Sparse (see Fig. 6).

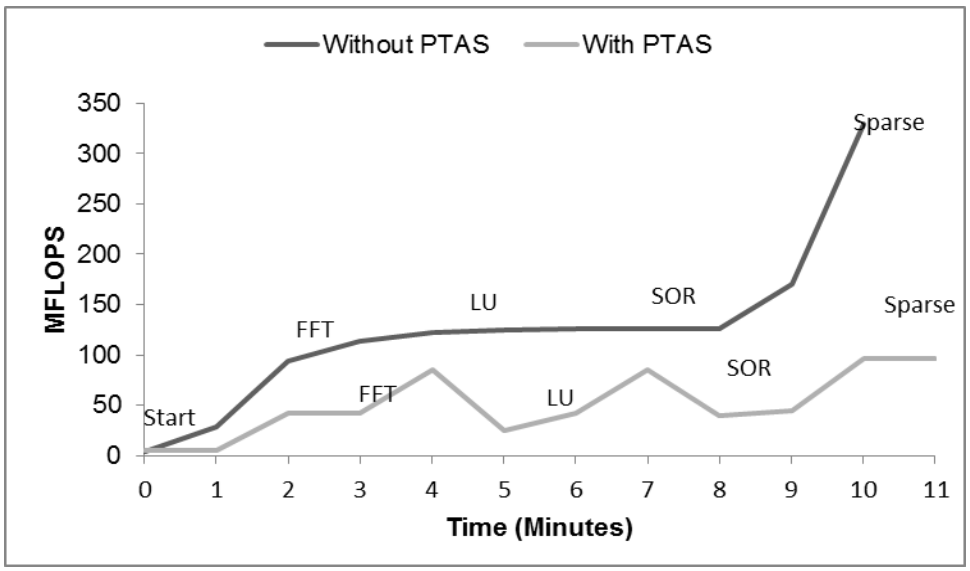

Fig. 4 MFLOPS vs. time when benchmarks are executed in sequence (FFT, LU, SOR, and Sparse)

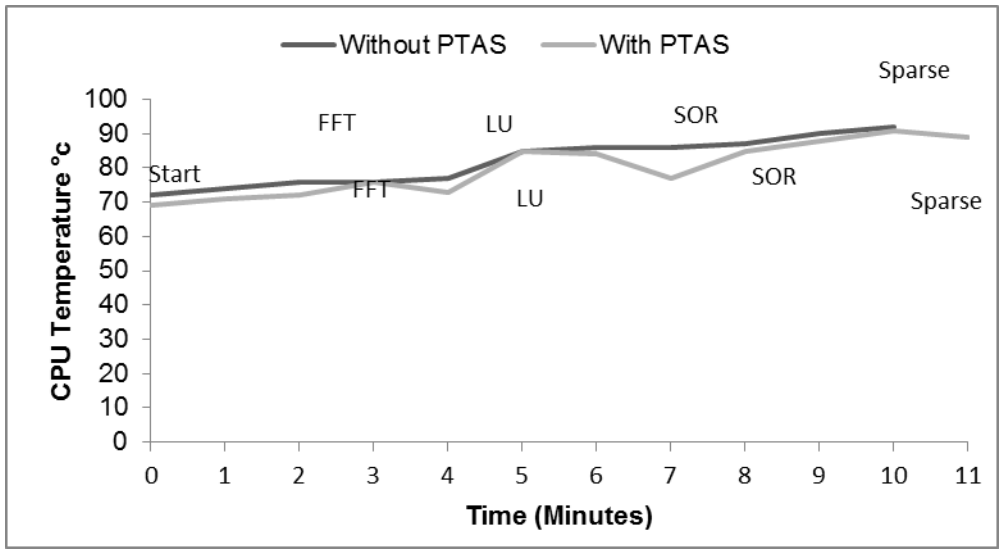

Fig. 5 CPU temperature during execution when small benchmarks(FFT, LU, Sor and Sparse) are executed together

In the larger benchmarks group (FFT-Large, LU-Large, Sparse-Large, and SOR-Large), increase in schedule length with PTAS was between 15-25% (see Fig. 3).

For all benchmarks, as seen in Fig. 7, the average peak CPU temperature decreased with the application of PTAS. We also determined the median results. We found that median readings with PTAS with and without PTAS were similar to average readings.

We noted that we could decrease the schedule length by decreasing the sleep time, at the cost of temperature. A carefully selected sleep time can reduce both the schedule length and temperature. Finally, we investigated the deviation of the readings to provide the level of confidence on the experiments. Table 2 lists the normalized standard error in our readings in a single experiment, where there are several crests of FLOPS at which the processes were put to sleep (cut-off points). The data represents the deviation of the cut-off values of the FLOPS at many crests in a single experiment. The normalized values were computed by dividing the standard deviation of the FLOPS at which the processes cut off. The corresponding deviations of temperature are also shown. We infer that in the experiment, the cut off occurred at similar points, suggesting confidence in experimental observations.

Table 2 Normalized STDEV of cut-off FLOPS and cut-off temperature over a single experiment with PTAS

Benchmark	Cut-off FLOPS	Cut-Off Temp (°c)
FFT	0	0.02
LU	0	0.01
SOR	0	0.02
Sparse	0	0.05
FFT-Large	0	0
LU-Large	0	0.02
SOR-Large	0	0.02
Sparse-Large	0	0.02

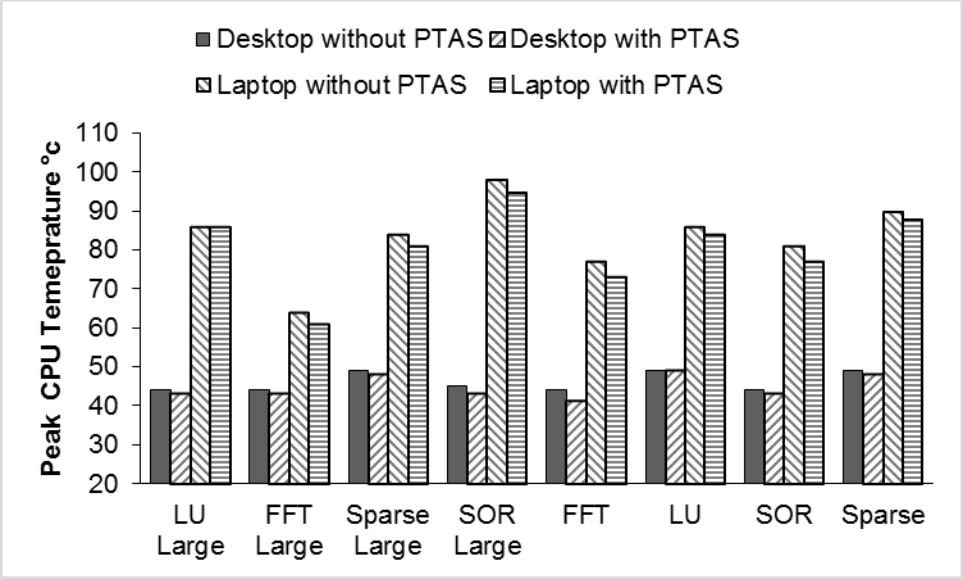

Fig. 6 Peak temperature using PTAS in desktop and laptop for various benchmarks

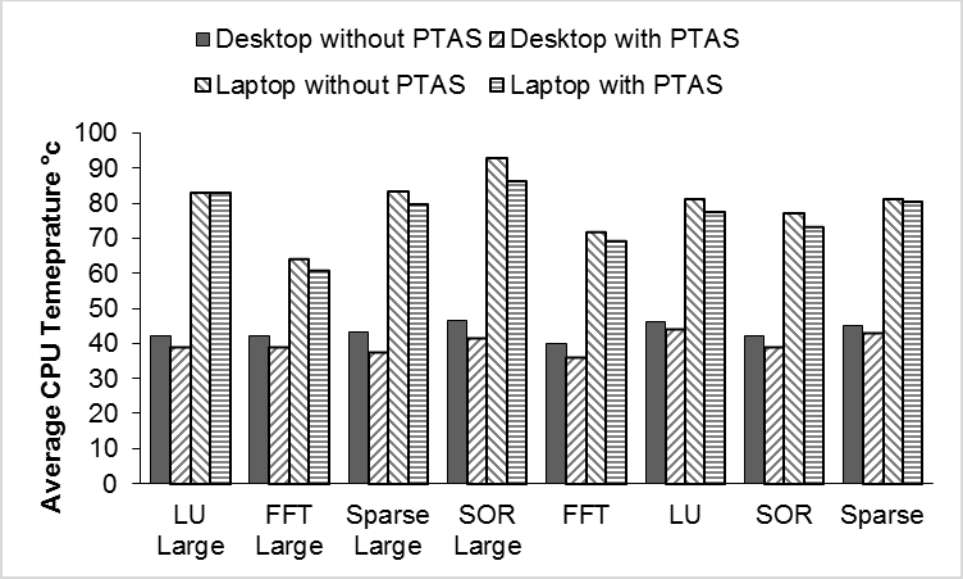

Fig. 7 Average CPU temperature using PTAS in desktop and laptop for various benchmarks

The peak CPU temperature for all the benchmarks decreased when we ran PTAS (see Fig. 6). There was also a drop in average CPU temperature for all the benchmarks (see Fig. 7). The temperature reduction comes from cooling during sleep times, and a more even redistribution of floating-point operations over time.

We also obtained temperature improvements on a battery-powered laptop. Fig. 6 and Fig. 7 show the comparison of peak CPU temperature and average CPU temperature on a desktop and laptop for FFT, LU, SOR, and Sparse small benchmarks.

Peak temperatures reduced for SciMark benchmarks, such as FFT, SOR, and Sparse, which are either integer arithmetic intensive or memory intensive. This shows that PTAS also has predictive power for non-FP applications.

The benefits of PTAS are over and above hardware-based approaches, such as DVFS, DTM, leakage energy, and system throttling. The experiment machine had Thermal Design Power (TDP). This strategy can also be implemented on a web browser or mobile device, as they have floating-point calculations.

3.5 Conclusion

In this chapter, a Proactive Temperature-Aware Scheduler was developed to cut off a process and put it to sleep if its predicted gradient was high. Such a strategy stabilizes CPU temperature and prevents temperature surges. Thus, PTAS would increase chip reliability by reducing thermal damage to the chip, and reduce related costs. In FFT, LU, SOR, and Sparse benchmarks, temperature reductions were 2-4°C, whereas those in FFT-Large, LU-Large, SOR-Large, and Sparse-Large were 3-5°C. The reductions on a laptop were 3-6°C in FFT, LU, SOR, and Sparse, whereas those in FFT-Large, LU-Large, SOR-Large, and Sparse-Large were 3-6°C.

As the regression uses the time derivative of the temperature, the effectiveness goes beyond FP intensive applications. Future work should add the rate of memory accesses as a predictor, which we have also found helpful in reducing CPU temperatures in our preliminary work. As pointed out in [5], these are the major causes of high temperatures. Our future work will explore the impact of this approach on web browsing and mobile devices. We expect similar benefits, as preliminary work with rate of memory change is encouraging.

As discussed previously, in our future work we will account for non-uniform memory accesses in the prediction. Preliminary results show a reduction in the schedule length penalty by employing memory access in the prediction. Future research will also focus on a detailed study of the relationship of threshold gradient to the duration of sleep time, schedule length, and multi-core temperature-aware scheduling.

Chapter 4 Proactive thermal management using flops via memory rates

After carefully reviewing literature, we found our approach is novel. The remainder of this paper is organized as follows: in Section 2, we discuss related work, Section 3 the PTFM approach, Section 4 the experiment section, Section 5 the results and discussion and finally, we discuss conclusion.

4.1 Related Work

There are several approaches to reducing CPU temperature. Temperature and energy aware strategies comprise of dynamic voltage and frequency scaling (DVFS), dynamic power management (DPM) or dynamic thermal management (DTM), leakage energy and variability thermal management or location based management.

Coskun et al. [2] implemented a workload scheduler using DVFS and DPM strategies that reduces CPU temperature by identifying spatial variations in workloads. This approach reduced hotspots by 35%, spatial gradients by 85% and thermal cycles (i.e. hot cool cycles) by 60%. Chatrurvedi et al. [10] developed m-oscillation (DVFS) thermal management strategy to reduce the chip temperature where *m* represents the speed of the CPU core. They used lower speed amongst two frequencies to complete tasks, which resulted in peak CPU temperature reduction of 2°*C.*

Xiuyi et al. [1] implemented DPM using operating system (OS) scheduler to minimize peak CPU temperatures in 3-d torus chips. In this research, there was 7.2% performance improvement in speedup of the processor using this scheduler. They discovered that hotspots around vertically adjacent layers in 3-d torus chip were the main reason for the increase in the 3-d torus chip temperature.

Kumar et al. [7] reduced CPU temperature by using DTM in an OS scheduler. This scheduling approach managed CPU temperatures effectively with average performance overhead of 10.4% (20.1% maximum). Chuan et al. [8] used leakage energy using active and passive modes of the OS or CPU to schedule the tasks. Their approach increased context switching of the tasks, which increased energy consumption. This research has small scheduling overhead and 5% to 6% energy savings.

Wei and Nannarelli [9] discovered that floating-point operations increased heat in caches. They used fused multiply add to optimize the division of the floating points which This approach reduced average temperature in caches by 5% and reduced leakage energy by 12%. In addition, this approach reduced power consumption by 8.4%.

Jin and Maskell [14] studied thermal management at an event level. They built a thermal map of events that reduced CPU temperature. By using an offline lookup table, they built a low complexity scheduler, which can be integrated into the kernel.

Fisher et al. [13] used speed scheduling that identifies ideal speed for each core of the

CPU to reduce CPU temperature. In this research they reduced peak CPU temperatures are reduced by 30-70 °C when compared to load balancing strategies.

Homogeneous scheduling in 3-d torus chips by Jiajia et al. [15] controls peak CPU temperatures. In the scheduling technique there was a speedup of 1.06, 1.05 and 1.04 in FFT, matrix multiply and radix sort.

Yang et al. deployed helper threads [16] to reduce peak temperatures in CPUs. Helper threads (which are cool threads) execute tasks in parallel thereby giving CPU thermal improvements and energy savings. The results were 66.3% and 83.3% savings in energy delay product (EDP) for FFT and Multigrid. This research reduced CPU thermal emergency.

Anupindi and Baskiyar [30] developed a proactive and predictive approach using derivative of floating points (FLOPS) and temperature to reduce CPU temperature. They evaluated the performance of PTAS using the various small and large FFT, LU, SOR and Sparse components of the Scimark benchmarks. The reductions in CPU temperatures on a desktop machine were between 7-13°C with the corresponding percentage reductions between 21-39%. Those for the desktop were between 3-6°C with the corresponding improvements between 4-9%. The corresponding penalties in schedule lengths in desktop were between 15-30% and 5-10% in laptop and there was 3-5% energy savings in both the desktop and laptop. The work reported in this paper is a substantial improvement over the previous work by including memory rates in the

predictor and varying sleep times and threshold cut-offs to conduct a more rigorous study.

Merkel and Bellosa [17] discovered that migration could be better than system throttling. They created an energy aware scheduling policy on a Linux scheduler by creating a task data structure. The scheduler moved hot tasks to the coolest core and balanced energy to get energy savings. They computed energy readings at a task level using event performance counters. The cost of migration was smaller than throttling.

Ayoub and Rosing [4] implemented a proactive thermal management strategy using a predictor formed by historical bandwidth of signals where temperature was modeled as a RC network. They discovered that Mean time to failure (MTTF) was related to average and peak CPU temperatures. The workload characterization of the tasks can be computed by finding the fetch rate of tasks. They measured the CPU temperatures due to reactive and proactive approaches using SPEC 2000 benchmarks. Using their proactive strategy, they decreased the average temperature of the hottest cores by 6-8 °C with a performance penalty between 40-60%.

Weissel and Bellosa [18] implemented event performance counters to measure tasks characteristics. They put weights to task events and measured CPU energy and temperature using performance counters. They measured CPU cycles to find energy consumption. Using static and dynamic parts of a linear optimization equation, they reduced temperature and saved energy. Scimark numerical benchmarks [28] were used for the experiment.

Procedure *PTFM (***int** *i) // i* is observation number

const ThresholdGradient, K=1000

static float F[K], T[K], M[K], Y[K]

Process p, Q[n] //Application processes

//n is the number of processes

1 **For each** $p \in Q$ **do** // in FIFO
2 F_i = FLOPS (p)
3 T_i = CPUTemperature() //From lmsensors
4 M_i = MemoryAccessRate(p)
5 Wait for δt time
6 F_{i+1} = FLOPS (p)
7 T_{i+1} = CPUTemperature()
8 M_{i+1} = MemoryAccessRate (p)
9 $(\delta F/\delta t)_i = (F_{i+1}-F_i)/\delta t$
10 $(\delta T/\delta t)_i = (T_{i+1}-T_i)/\delta t$
11 $(\delta M/\delta t)_i = (M_{i+1}-M_i)/\delta t$
12 Wait for δt time
13 F_{i+2} = FLOPS (p)
14 T_{i+2} = CPUTemperature()
15 M_{i+2} = MemoryAccessRate (p)
16 $(\delta F/\delta t)_{i+1} = (F_{i+2}-F_{i+1})/\delta t$
17 $(\delta T/\delta t)_{i+1} = (T_{i+2}-T_{i+1})/\delta t$
18 $(\delta M/\delta t)_{i+1} = (M_{i+2}-M_{i+1})/\delta t$
19 $Y_i = 1 + (\delta F/\delta t)_i * F_i + (\delta T/\delta t)_i * T_i + (\delta M/\delta t)_i * M_i + \sigma_i(Y)$
20 $Y_{i+1} = 1 + (\delta F/\delta t)_{i+1} * F_{i+1} + (\delta T/\delta t)_{i+1} * T_{i+1} + (\delta M/\delta t)_{i+1} * M_{i+1} + \sigma_{i+1}(Y)$
21 $\delta Y_i/\delta t = (Y_{i+1} - Y_i)/\delta t$
22 **if** $\delta Y_i/\delta t >$ *ThresholdGradient* **then**
23 Sleep(p)
24 **endif**
25 **endfor**
end *PTFM*

Fig. 8 Algorithm PTFM

Lenovo [31] uses four schemes for energy management in laptops. They are energy star, high performance, balanced and super energy saver. Energy star uses sleep, hibernation, hard disk rotation and CPU speed in a/c and battery mode to get energy savings. This scheme optimizes battery health. In addition, smart power savings for CD-ROM, hard disk, CPU and screen refresh rates are gained. Using smart sensing

approaches for ambient light sensing, ambient keyboard sensing and ambient panel light significant energy savings are obtained.

Dell [32] uses energy smart architecture for high power efficiency and intelligent power management. It uses a power supply unit with common form factor for a/c or battery. In addition, it uses Intel node manager firmware for power monitoring capabilities to the PSU and subsystem, processor, I/O, memory, storage and fan. Using it, a sample for ten seconds can be obtained to measure the accuracy and efficiency. Finally, energy controlling and reporting capabilities are provided in dell servers.

IBM [33] uses intelligent power management features in POWER7 processors. Using energy scale, this processor intelligently monitors power trending, power-saving capping of maximum power that allows the server to set system policies for energy-efficient servers. It maintains energy feeds and sets the upper limit of energy for a server in a data center. The core of IBM energy management is to eliminate hotspots in the CPUs. In addition, it eliminates wasteful cool spots, which decrease energy savings.

Linux [34] uses CPUfreq subsystem to control processor energy savings varying the frequency dynamically for different workloads.

4.2 Approach

We developed a proactive scheduler that reduces CPU temperature. This work is based on our previous work [30] in which we developed a proactive scheduler by predicting the impact of any process on CPU temperature using the time derivatives of its floating point instruction execution rate and current CPU temperature. This strategy was called,

PTAS, which used regression of FLOPS and temperature gradients to determine CPU temperature of a process in a CPU. This strategy successfully reduced CPU temperature.

In this research we hypothesize that using a memory derivative gradient in the temperature impact predictor could provide additional CPU temperature reductions. We call the resulting scheduler Proactive Thermal manager using Floating point rates and Memory rates (PTFM) which uses the time derivatives of FLOPS, memory access rates and current temperature to predict and proactively reduce CPU temperature. The temperature impact predictor of a process can be formulated as a regression as follows:

$$Y_i = \beta_0 + \beta_1 F_i + \beta_2 M_{i+} \beta_3 T_{i+} \in \quad (1)$$

where the co-efficient are $\beta_0 = 1$, $\beta_1 = \delta F / \delta t$, $\beta_2 = \delta M / \delta t$, and $\beta_3 = \delta T / \delta t$ and $\sigma(y)$ is standard deviation

Fig. 8 gives the algorithm for PTFM. The algorithm computes the rate of FLOPS for a process, the rate of CPU temperature change, and the rate of memory accesses of any process. In the experiment, these values were computed by determining the total FLOPS and total memory accesses separated by delays δt (among i^{th}, $i+1^{th}$,$i+2^{th}$ observation points) and then dividing them by δt. The temperatures were also observed at the i^{th}, $i+1^{th}$,$i+2^{th}$ observation points and the differences were divided by δt to get the rates of temperature. The term $\sigma_i(Y)$ represents the standard deviation of values from 0^{th} to i^{th} observation. Using the above values a regression predictor $(\delta Y/\delta t)_i$ was formed. If the value of the predictor is greater than the predefined threshold gradient, the process is temporarily put to sleep. In order to get better thermal improvements, we

experimented with different sleep times and cut-off gradients. The cut-off gradient values that we experimented with are: 0.22, 0.23, 0.24 and 0.25. We used 10 ms, 50 ms, 250 ms, 500 ms and 1 s sleep times for our strategies. The threshold cut-off gradient of 0.22 gave the best thermal improvements. The graphs shown in this paper correspond to this threshold value.

The approach taken by PTFM does not cut-off all the user processes in CPU whereas hardware based cut-off suspends all user processes, making the CPU temporarily unavailable for a short duration. Therefore, in PTFM approach the CPU remains usable for other processes. Table 3 shows the symbols used by PTFM.

4.3 Experimental Setup

In order to evaluate PTFM, we used the Scimark benchmarks. We selected Scimark benchmarks since they performed numerical calculations. The benchmarks used Fast Fourier transform, Jacobi successive over-relaxation, dense unit factorization and

Table 3 Symbols and description

Symbol	Description
Δt	Delay time between successive observations
$d\mathrm{Y}/\,dt$	Temperature gradient predictor
Y_i	TIP of the process at the i^{th} observation
β	Regression co-efficient
F_i	FLOPS at the i^{th} observation
M_i	Memory accesses at the i^{th} observation
T_i	Temperature at the i^{th} observation

Sparse matrix multiply. These numerical benchmarks perform various functions like bit reversal, matrix multiply and memory accesses. These benchmark processes (FFT, SOR, LU and Sparse) are floating point intensive, memory intensive and integer intensive.

We conducted the experiment with Ubuntu 9.10 on a Dell OptiPlex 9020 i5 @ 2.90 Hz desktop with 4 GB RAM and 320 GB HDD. We also conducted the experiments on a Lenovo Intel Dual Core @2.10 Hz with 4 GB RAM and 302 GB HDD laptop running Ubuntu 9.10 operating system. The ambient room temperature was 70 °F.

The algorithm was written in Java and used apache commons API for mathematical calculations. We modified the code in the Scimark benchmarks to calculate the values of FLOPS and memory access rates. In a real time scheduler implementation, we can measure FLOPS and memory access rates using PAPI. In order to see whether in a real implementation what sort of overheads PAPI could cause, we performed calibration tests on PAPI for Matrix Multiply, Inner Product and Matrix Vector Multiply benchmarks. We found overheads for FLOPS and memory access rates for matrix multiplication to be negligible (0.0017). This shows that PAPI can be used in a real implementation without incurring significant overhead.

We used hardware temperature sensors [29] in the simulator to measure CPU temperature. We measured the temperature readings from these sensors and calculated.

Using the memory access rates, FLOPS and current temperature of the CPU, we compute Temperature Impact Predictor for any process. The regression for any process can also be computed by using the apache commons library [35].

When the value of the gradient of the predictor goes above a threshold, we proactively put the process to sleep for a small duration to reduce CPU temperature.

We evaluated the effect of four different strategies on CPU temperature: PTAS, PTFM, STD (Simple Time derivative) and Threshold. STD strategy used a derivative of CPU temperature (d/dt). Using this derivative a cut-off was employed if the derivative exceeds a threshold value to reduce CPU temperature. The simple derivative value used was 0.24—we chose this value after a few experiments to provide the best CPU temperature reduction, and schedule length penalty. Finally, Threshold strategy does not use a derivative or regression of FLOPS, memory and temperature but used a direct cutoff. It cuts off a process, which exceeds a given Threshold. In all these strategies, the threshold used was computed empirically after many experiments. The threshold value used for comparison was the minimum threshold at which there was maximum CPU temperature reduction. The threshold temperature used for Threshold strategy was 45 *°C* for desktop and 53 *°C* for laptop. We compared the results of these four different strategies for peak CPU temperature, average CPU temperature and performance.

4.4 Results and Discussion

We recorded the readings of our experiment and plotted the results. The average CPU temperature is an average of several readings during the entire execution of the specific component of the benchmark. The peak CPU temperature is the peak temperature reached during the entire execution of the specific component of the benchmark. Fig. 9 shows CPU temperature readings for the small benchmarks in Celsius when they are all run successively. The crests in the graph indicate rise in CPU temperature with the benchmarks whereas troughs in the graph indicate process cut-offs due to PTFM. For these benchmarks, there was an overall reduction of CPU temperature of 3-6°C with PTFM than without. The reduction of CPU temperature was due to reducing FLOPS and memory accesses. There were thermal improvements with large benchmarks too. For large benchmarks, there was 3-6°C of average CPU temperature reduction.

The schedule length with PTFM extends from 2-10% whereas with PTAS it extends by 15-25%. We were able to keep the schedule length tight for PTFM by employing optimization of sleep time, the additional memory rate parameter in the regression, and experimenting with different predictor threshold cut-offs.

Fig. 10 shows peak CPU temperature comparison for Threshold, STD, PTAS and PTFM PTAS and PTFM strategies with a sleep time of 10 ms. We found that peak CPU temperature reduction for PTFM was greater than PTAS, STD and Threshold for FFT. In PTFM strategy, the FFT benchmark had greater peak CPU temperature reduction than due to LU, SOR and Sparse. In all the strategies, with Sparse benchmark there was minimal peak CPU temperature reduction due to non-uniform accesses. Perhaps the

lack of locality causes memory accesses, which causes increase in CPU temperature [36]. The peak CPU temperature of the threshold strategy was greater than No Strategy for SOR and Sparse benchmarks because the temperature continues to rise during the cut-off for the entire execution of the run.

We found that average CPU temperature reduction for PTFM (see Fig. 11) was higher than PTAS at 10ms sleep time. In addition, as expected, we observed that average CPU temperature reductions were higher than peak CPU temperature reductions for all the strategies, for all benchmarks and all sleep times.

We set the cut-off slope of the predicted gradient as 0.22 as we found best results at this value. This was determined after several experiments. In a real time scheduler, the cut-off slope can be varied using a feedback control loop to get further CPU temperature improvements. For all the processes, priority was set as normal.

Fig. 12 shows peak CPU temperature comparison for four different strategies at 500 m seconds sleep time**.** We found that the peak CPU temperature reduction for all the benchmarks. The PTFM strategy was better for FFT, LU Large and Sparse Large. The PTAS strategy had a better run for LU, Sparse and FFT Large. For Sparse and SOR threshold strategy was better.

Fig. 13 shows average CPU temperature comparison for 500 ms sleep time. The PTFM strategy outperforms for SOR, FFT Large and SOR Large. The threshold strategy shows

better results for LU and Sparse. We found PTAS improvements are similar to PTFM. Fig. 14 shows peak CPU temperature comparison for 1 s. We found PTFM was superior for FFT, LU, SOR Large. The threshold strategy shows improvement for SOR and Sparse Large. The PTAS strategy gives good results for FFT large. Fig. 15 shows average CPU temperature comparison for 1 s sleep time. The PTFM strategy had thermal improvements for FFT, LU and SOR large. For Sparse and SOR Large PTAS gives thermal savings. The threshold strategy had improvements for SOR and Sparse Large. The thermal behavior was similar to smaller sleep time but the process goes to sleep for a longer duration.

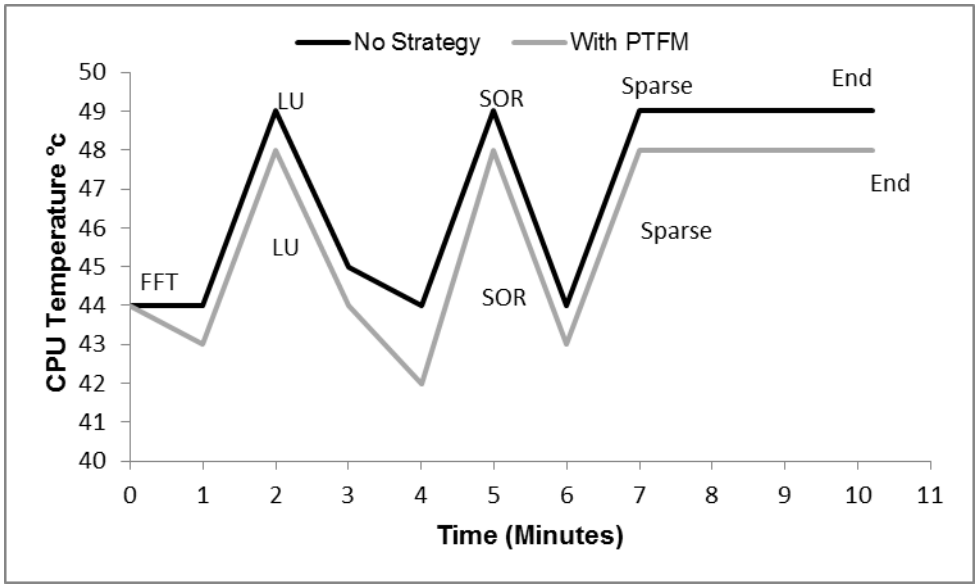

Fig. 9 CPU Temperature for smaller benchmarks when executed successively

The PTFM strategy used memory rates in addition to FLOPS, which was instrumental in giving the best outcome when compared to the other strategies.

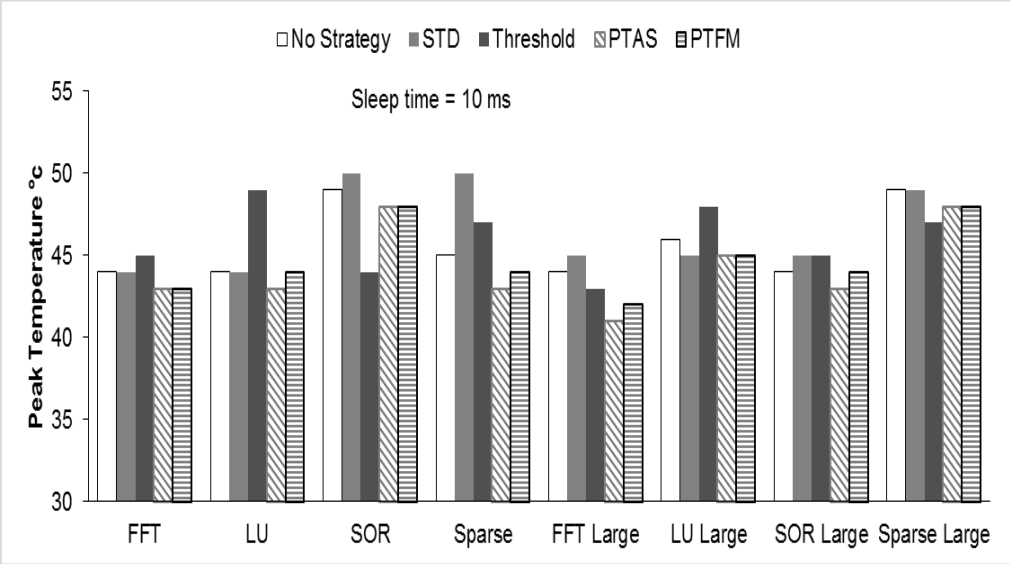

Fig. 10 Peak CPU temperatures with a sleep time of 10 ms

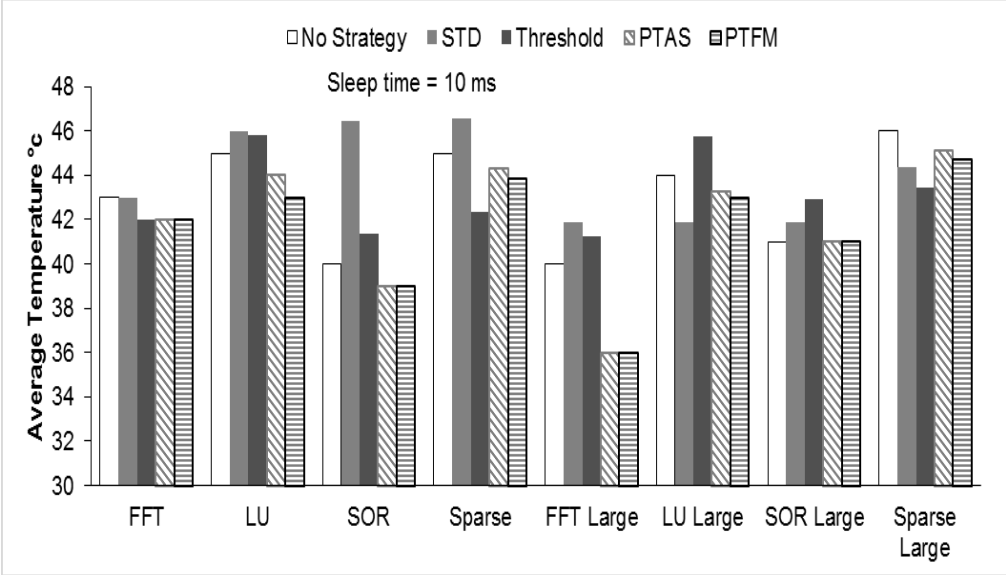

Fig. 11 Average temperature for a sleep time of 10 ms

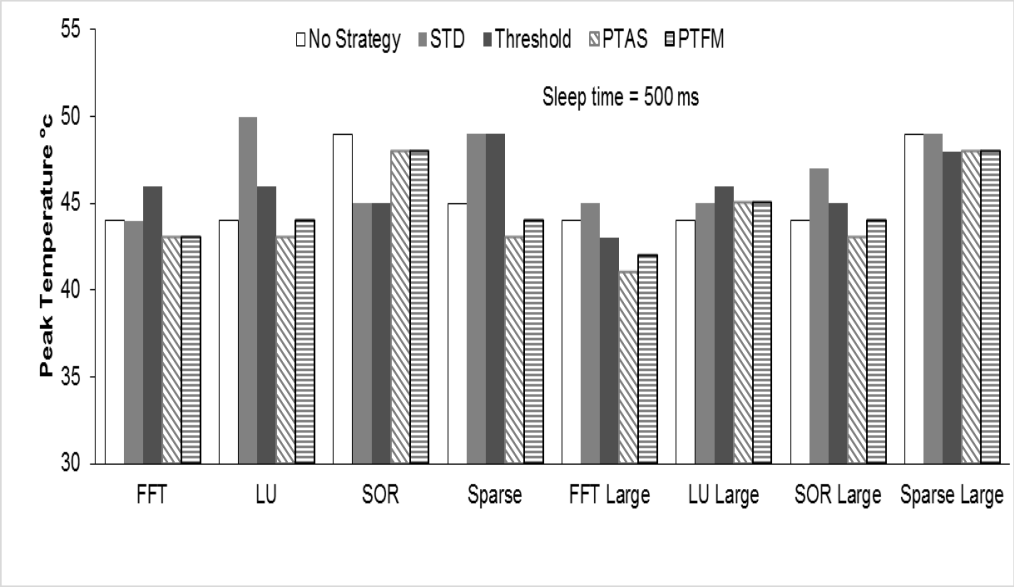

Fig. 12 Peak temperature for a sleep time of 500 ms

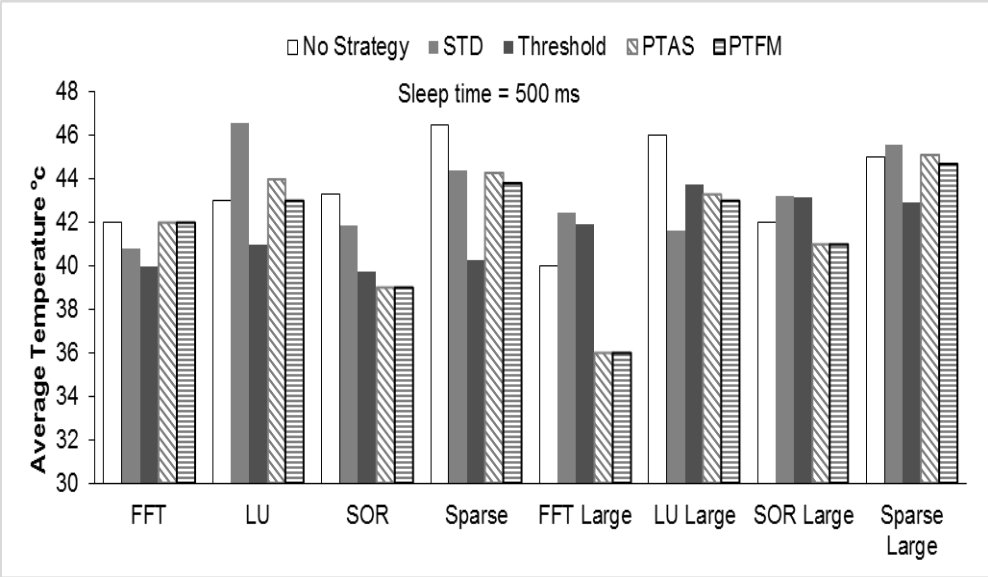

Fig. 13 Average temperature for a sleep time of 500 ms

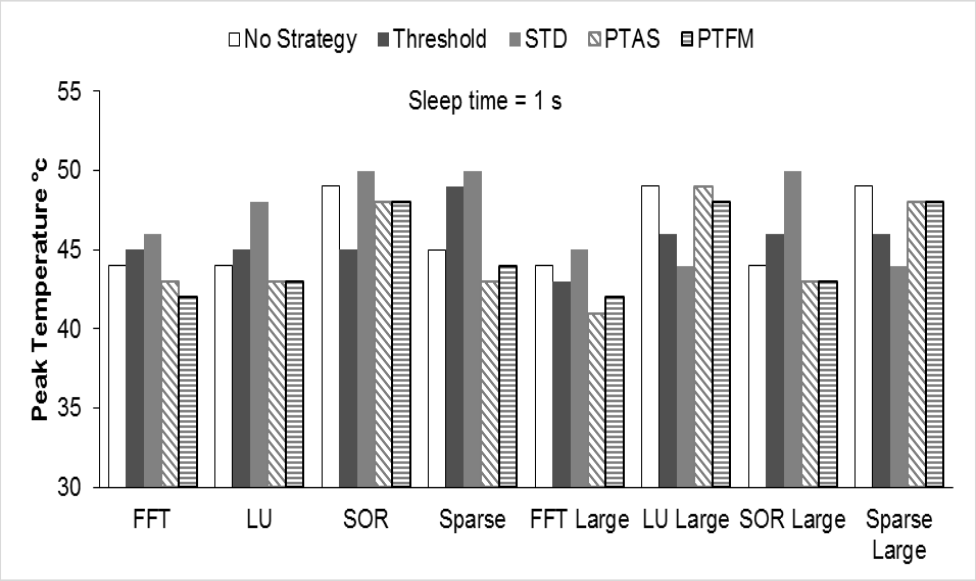

Fig. 14 Peak temperature for a sleep time of 1 s

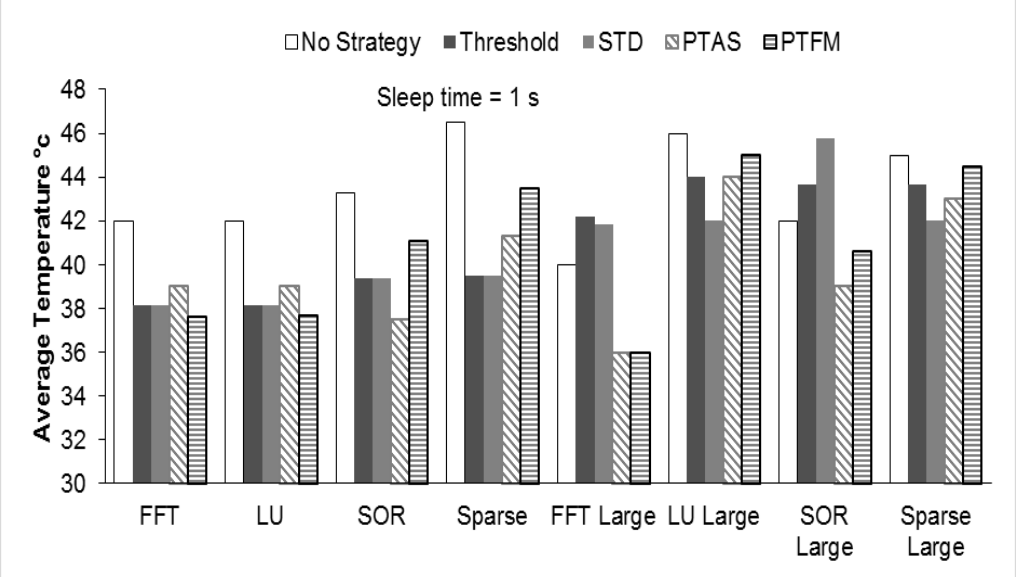

Fig. 15 Average temperature for a sleep time of 1 s

Fig. 16 shows the peak CPU temperature comparison for a desktop and laptop whereas Fig. 17 shows average CPU temperature comparison for a desktop and laptop. We got better thermal improvements (3-6°C) on the laptop. The peak CPU temperature is the peak CPU temperature the CPU reaches during the entire execution of specific program

4.5 Conclusion

In this chapter, we developed a proactive CPU thermal management strategy, which reduce CPU temperature by predicting the higher temperature gradient of a process using rates of change of No current CPU temperature, floating point access rates and memory access rates. We varied sleep time, cut-off gradient of any process to provide the best temperature and execution times. We compared our strategy with PTFM, STD and Threshold strategies. We found PTFM outperformed other three strategies. We found around 3-6°C/6°C reduction in peak/average CPU temperatures due to small benchmarks (FFT, LU, SOR and Sparse) and 3-6°C/6°C for large benchmarks (FFT-Large, SOR-Large, LU-Large and Sparse Large). Preliminary results on Spec suite gave similar improvements. We compared our strategy with PTFM, STD and Threshold strategies. We found PTFM outperformed other three strategies.

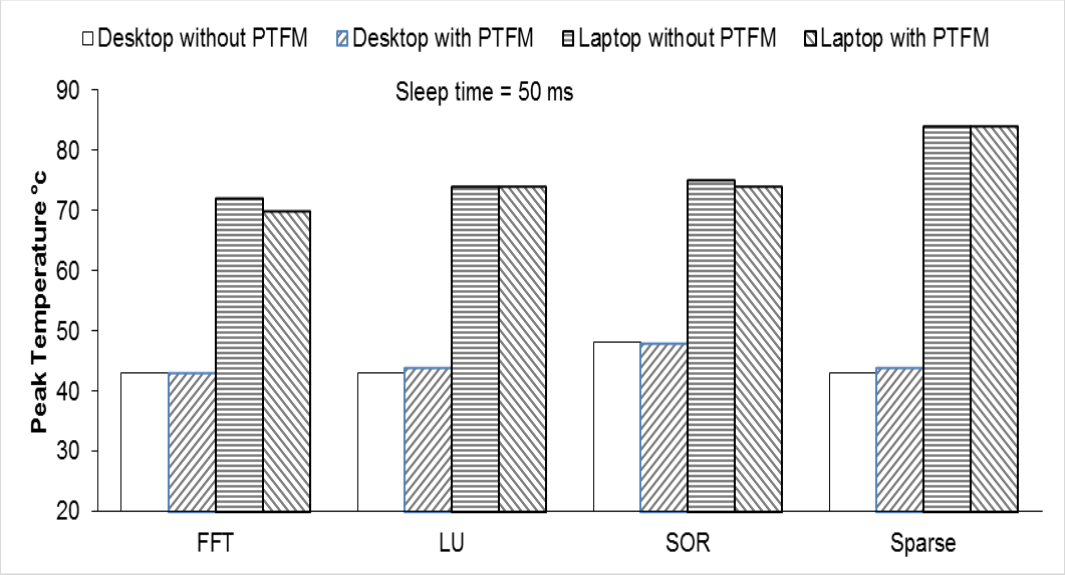

Fig. 16 Peak CPU Temperature comparison of PTFM on a desktop and laptop for a sleep time of 50 ms

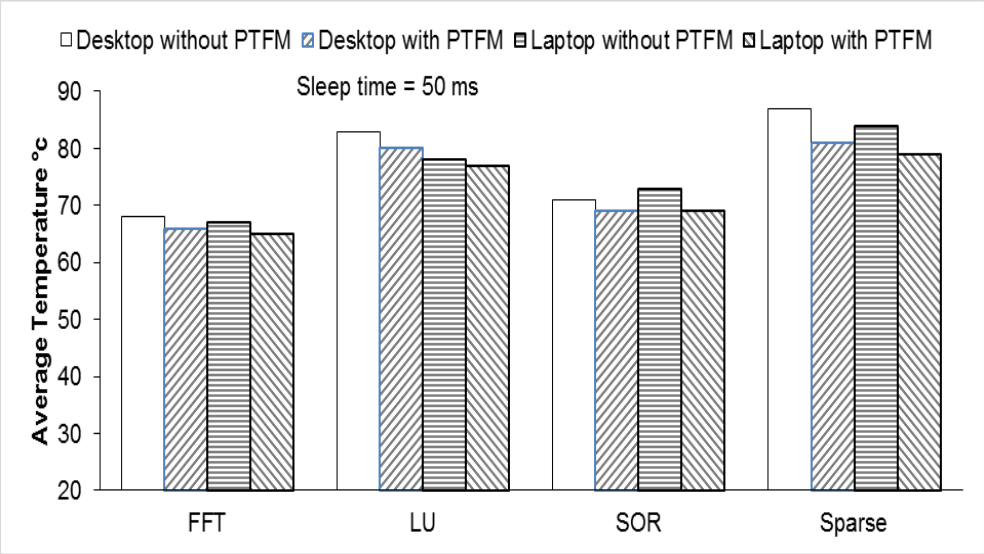

Fig. 17 Average CPU Temperature comparison of PTFM on a desktop and laptop for a sleep time of 50 ms

This strategy can be applies over and above the hardware approaches such as DVFS, DPM and leakage energy. In the future, we aim to conduct these experiments on other mobile devices and on cloud environments. Based on this research we can implement a real time scheduler on multi cores with different scheduling policies. **Fig. 18** shows the peak CPU temperature for different slopes values when all the benchmarks are executed together whereas **Fig. 19** shows the average CPU temperature of different slope values when all benchmarks are executed together. The peak CPU temperature reduction for a slope of 0.21 and 0.22 was higher than other slope values (0.23 and 0.24). Similarly, the average CPU temperature reduction was high at 0.21 and 0.22 slope values.

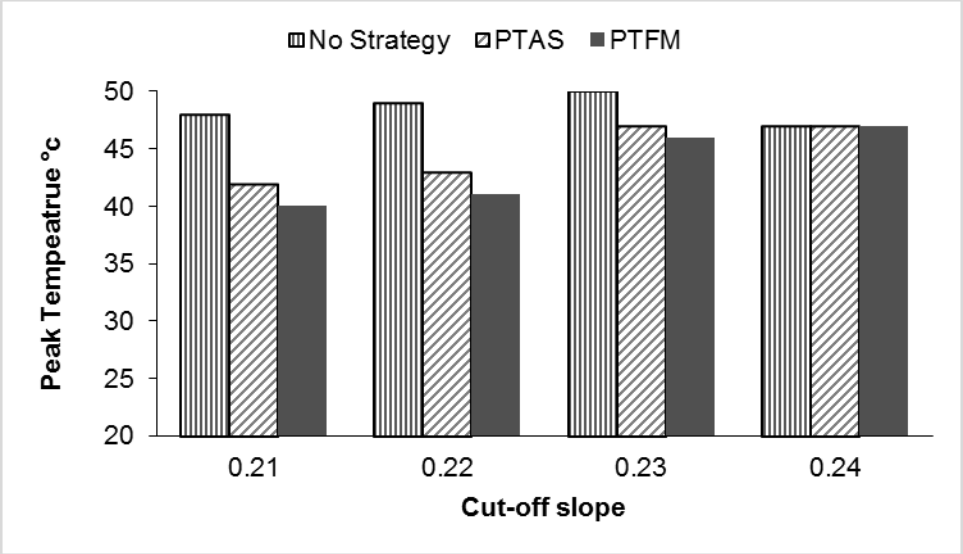

Fig. 18 Peak CPU Temperature when all benchmarks are executed together

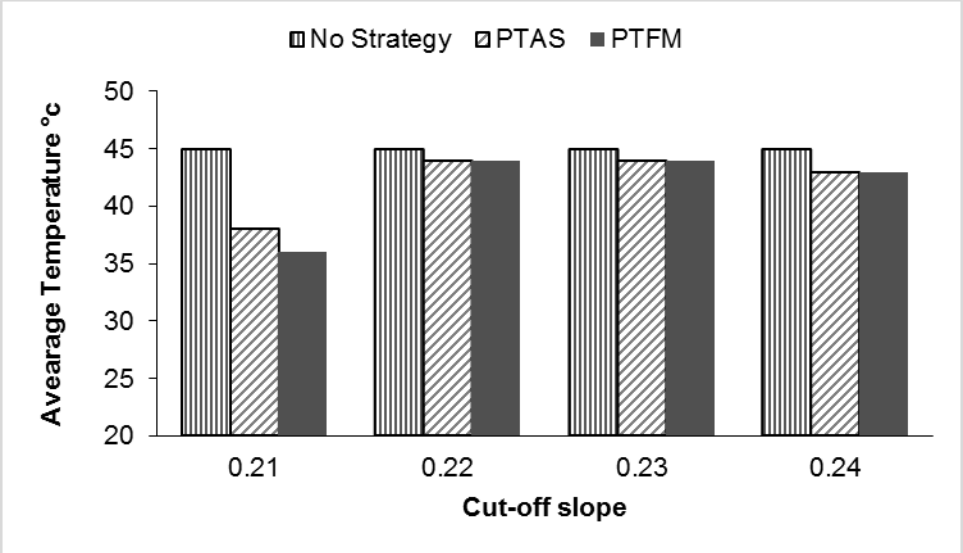

Fig. 19 Average CPU Temperature when all benchmarks are executed together

Table 4 shows the time relationship between different floating-point values (MFLOPS). We see a drop in MFLOPS with our strategy. Similarly, in Table 5 we see a drop in CPU temperature.

Table 6 depicts the CPU temperature without benchmarks running and with benchmarks running for core 0, core1 and ISA adapter. It shows our motivation that running floating-point intensive workloads increase CPU temperature. These benchmarks are floating-point intensive and memory intensive.

Table 4 MFLOPS vs. time

Time	Without PTAS	With PTAS
0	4	5
1	28	5
2	94	43
3	113	43
4	123	85
5	124	25
6	126	43
7	126	85
8	126	40
9	170	45
10	329	96

Table 5 Temperature vs. time

Time	Without PTAS	With PTAS
0	72	69
1	74	71
2	76	72
3	76	76
4	77	73
5	85	85
6	86	84
7	86	77
8	87	85
9	90	88
10	92	91
	94	89

Table 6 CPU temperature without benchmarks and with benchmarks running

	Benchmarks not running	Benchmarks running
Core 0	45	65
Core 1	45	63
ISA Adapter	46	69

Table 7 displays the execution times of different benchmarks on a desktop and laptop for different benchmarks. We see that there was an increase in schedule length by 10 - 15%. Table 8 depicts the peak CPU temperature using PTAS in a desktop and laptop for different benchmarks. We see a considerable drop in peak CPU temperature with PTAS.

Table 7 Execution time of different benchmarks for the desktop and laptop

	Desktop without PTAS	Desktop with PTAS	Laptop without PTAS	Laptop with PTAS
LU Large	46	64	54	54
FFT Large	72	76	72	75
Sparse Large	71	82	88	122
SOR Large	62	73	46	58
LU	36	47	27	38
FFT	33	45	26	26
Sparse	68	131	53	64
SOR	72	84	46	58

Table 8 Peak Temperature using PTAS in Desktop and laptop for various benchmarks

	Desktop without PTAS	Desktop with PTAS	Laptop without PTAS	Laptop with PTAS
LU Large	44	43	86	86
FFT Large	44	43	64	61
Sparse Large	49	48	84	81
SOR Large	45	43	98	95
FFT	44	41	77	73
LU	49	49	86	84
SOR	44	43	81	77
Sparse	49	48	90	88

Table 9 shows the average CPU temperature using PTAS in desktop and laptop. We were able to get a clear reduction in CPU temperature in all the benchmarks using desktop and laptop.

Table 9 Average CPU temperature using PTAS in desktop and laptop for various benchmarks

	Desktop without PTAS	Desktop with PTAS	Laptop without PTAS	Laptop with PTAS
LU Large	42	39	83	83
FFT Large	42	39	64	61
Sparse Large	43.3	37.5	83.4	80
SOR Large	46.5	41.3	93.2	86.3
FFT	40	36	71.9	69.4
LU	46	44	81.4	77.6
SOR	42	39	77.3	73.4
Sparse	45	43	81.2	80.5

Table 10 gives peak CPU temperature without PTFM and with PTFM for smaller benchmarks for a sleep time of 10 ms. We see that PTFM has best reductions at a sleep time of 10 ms .We are able to reduce peak CPU temperature using PTFM whereas Table 11 shows peak CPU temperature comparison of PTAS and PTFM. We see PTFM is better than PTAS as we use FLOPS and memory rates to predict a process and put it to sleep.

Table 10 Peak CPU Temperature of smaller benchmarks (FFT, LU, SOR and Sparse) when executed together

	Without PTFM	With PTFM
0	52	52
1	52	49
2	57	54
3	60	58
4	63	51
5	63	56
6	63	54
7	63	62
8	63	62
9	63	57
10	63	57

Table 11 Peak CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 10 ms

	FFT	LU	SOR	Sparse	FFT Large	LU Large	SOR Large	Sparse Large
No Strategy	44	44	49	45	44	46	44	49
Threshold	43	43	48	43	41	45	43	48
STD	43	44	48	44	42	45	44	48
PTAS	44	44	50	50	45	45	45	49
PTFM	45	49	44	47	43	48	45	47

Table 12 gives average CPU temperature for No Strategy, Threshold, STD, PTAS and PTFM on smaller benchmarks for a sleep time of 10 ms. As expected, the average CPU temperature drop was lower than peak CPU temperature. Table 13, Table 14, Table 15 and Table 16 show peak and average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM for all the benchmarks for a sleep time of 500 ms and 1 s respectively. The thermal behavior was similar for other sleep times.

Table 12 Average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 10 ms

	FFT	LU	SOR	Sparse	FFT Large	LU Large	SOR Large	Sparse Large
No Strategy	43	45	40	45	40	44	41	46
PTAS	42	44	39	44	36	43	41	45
PTFM	42	43	39	44	36	43	41	45
STD	40	41	46	47	42	42	42	44
Threshold	39	46	41	42	41	46	43	43

Table 13 Peak CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 500 ms

	FFT	LU	SOR	Sparse	FFT Large	LU Large	SOR Large	Sparse Large
No Strategy	44	44	49	45	44	44	44	49
PTAS	43	43	48	43	41	45	43	48
PTFM	43	44	48	44	42	45	44	48
STD	46	45	50	50	45	45	47	49
Threshold	46	46	45	49	43	46	45	48

Table 14 Average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 500 ms

	FFT	LU	SOR	Sparse	FFT Large	LU Large	SOR Large	Sparse Large
No Strategy	42	42	43	47	40	46	42	45
PTAS	42	44	39	44	36	43	41	45
PTFM	42	43	39	44	36	43	41	45
STD	39	39	40	40	42	42	43	46
Threshold	39	39	40	40	42	44	43	43

Table 15 Peak CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 1 s

	FFT	LU	SOR	Sparse	FFT Large	LU Large	SOR Large	Sparse Large
No Strategy	44	44	49	45	44	49	44	49
PTAS	43	43	48	43	41	49	43	48
PTFM	42	43	48	44	42	48	43	48
STD	46	48	50	50	45	44	50	44
Threshold	45	45	45	49	43	46	46	46

Table 16 Average CPU temperature of No Strategy, Threshold, STD, PTAS and PTFM. strategies for a sleep time of 1 s

	FFT	LU	SOR	Sparse	FFT Large	LU Large	SOR Large	Sparse Large
No Strategy	42	42	43	47	40	46	42	45
PTAS	39	39	38	41	36	44	39	43
PTFM	38	38	41	44	36	45	41	45
STD	38	38	39	40	42	42	46	42
Threshold	38	38	39	40	42	44	44	44

The Fig. 20 shows the peak CPU temperature of FFT for 10 ms whereas Fig. 21 shows average CPU temperature of FFT for 10 ms. In both cases, there was a drop in CPU temperature. The average CPU temperature drop was higher than peak CPU temperature. Fig. 22 shows the peak CPU temperature of LU for 10 ms and Fig. 23 shows the average CPU temperature of LU for 10 ms. The LU benchmark gave better thermal improvements in average CPU temperature and peak CPU temperature at 10 ms. Fig. 24 shows the peak CPU temperature of SOR for 10 ms. and Fig. 25 depicts shows the average CPU temperature of SOR for 10 ms. The PTFM strategy outperformed other three strategies. Finally, Fig. 26 and Fig. 27 shows the peak and average CPU temperature of Sparse for 10 ms. The PTAS strategy was better than other three strategies. We see that CPU temperature drop in Sparse was lower than other benchmarks. As discussed, before it could be attributed to non-uniform accesses by Sparse.

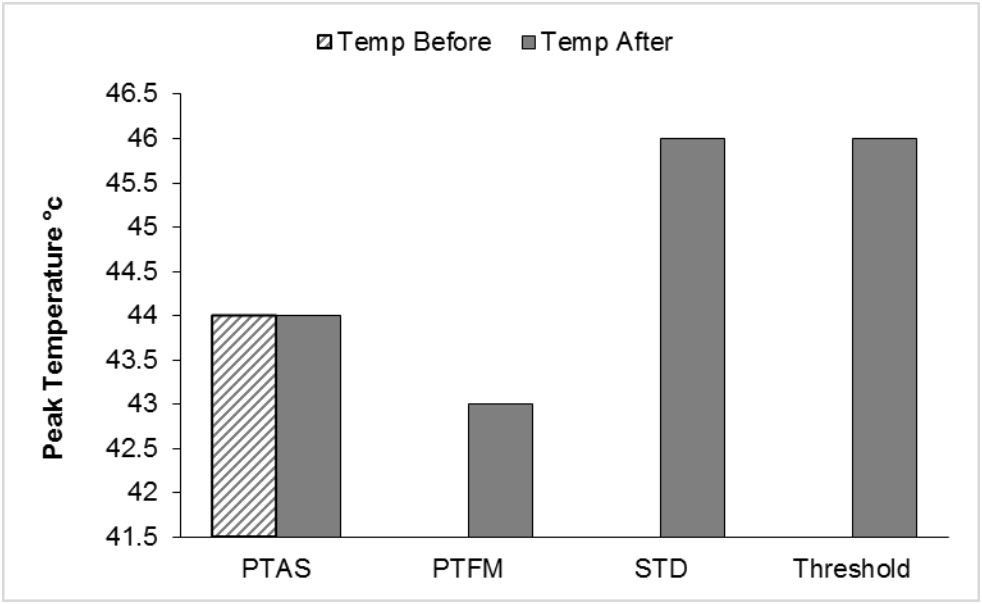

Fig. 20 Peak CPU Temperature due to FFT for 10 ms

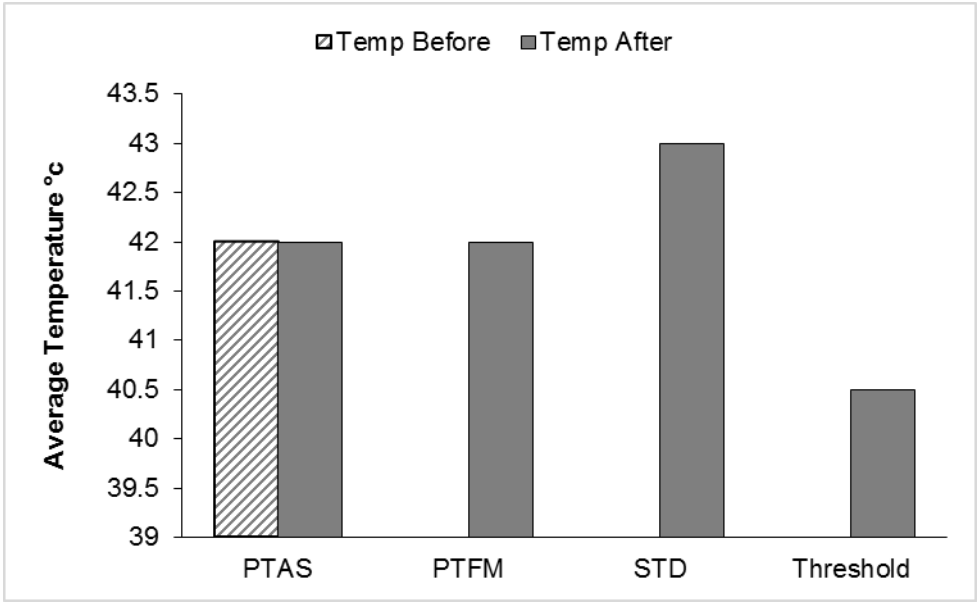

Fig. 21 Average CPU Temperature due to FFT for 10 ms

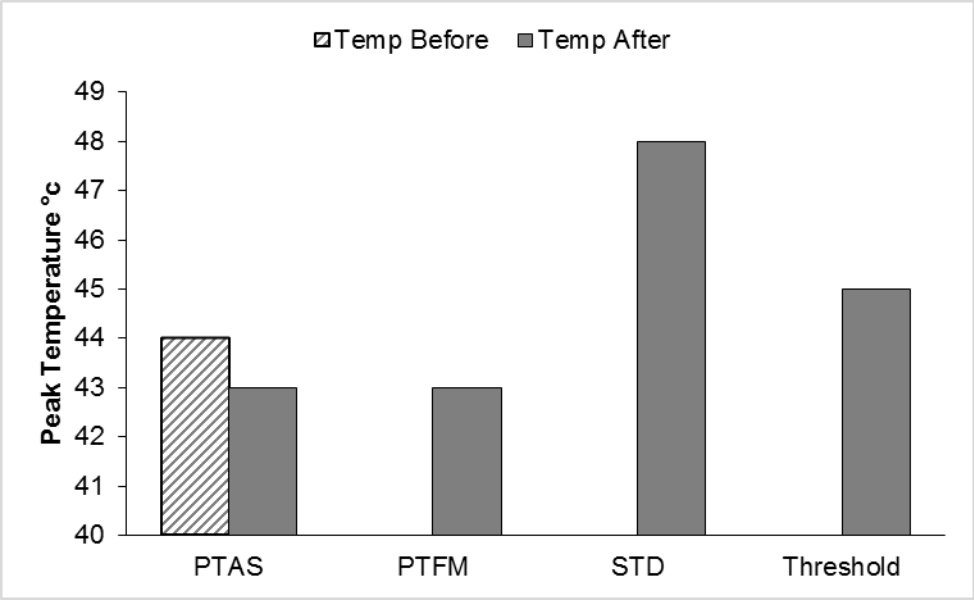

Fig. 22 Peak CPU Temperature due to LU for 10 ms

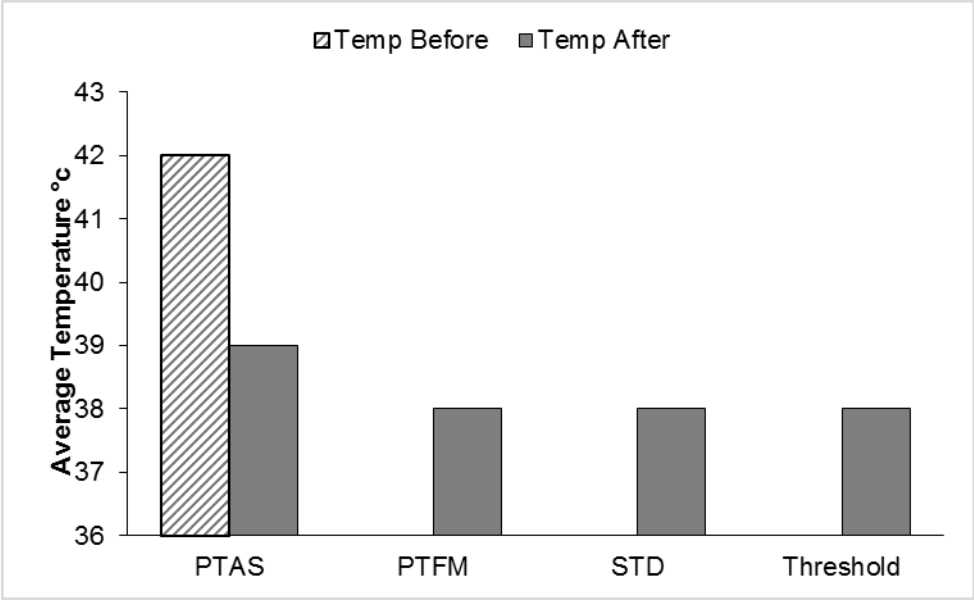

Fig. 23 Average CPU Temperature due to LU for 10 ms

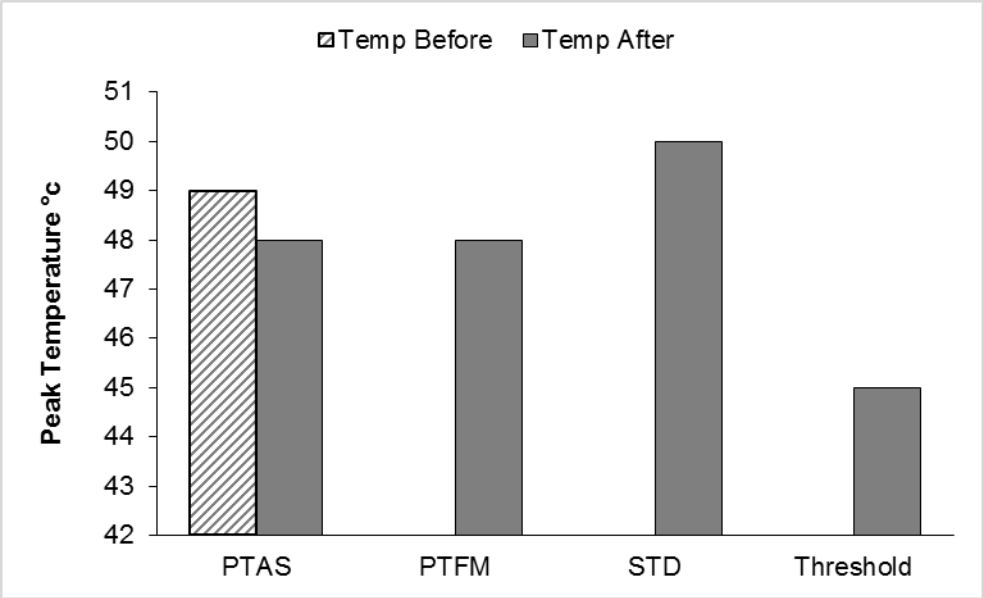

Fig. 24 Peak CPU Temperature due to SOR for 10 ms

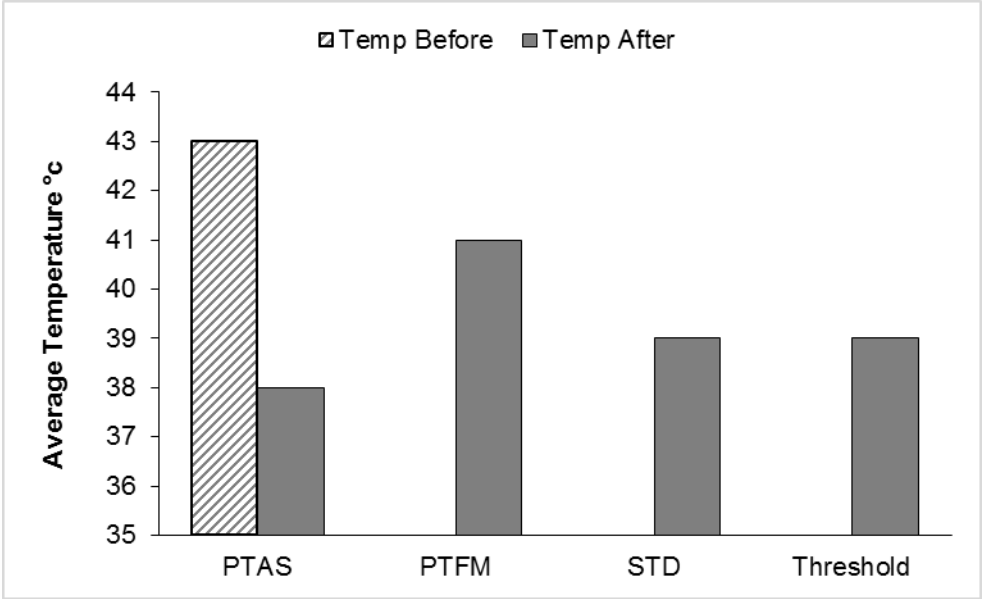

Fig. 25 Average CPU Temperature due to SOR for 10 ms

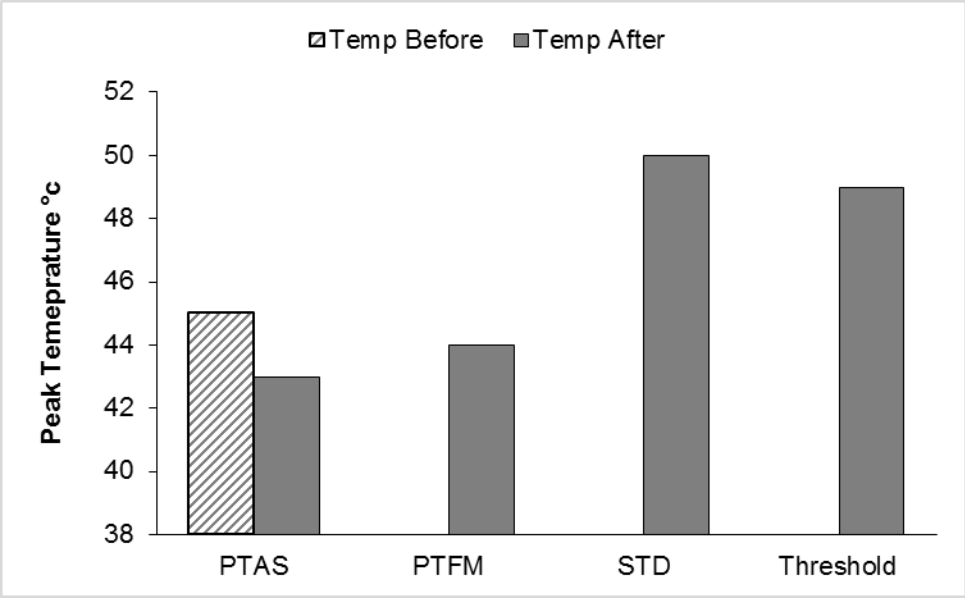

Fig. 26 Peak CPU Temperature due to Sparse for 10 ms

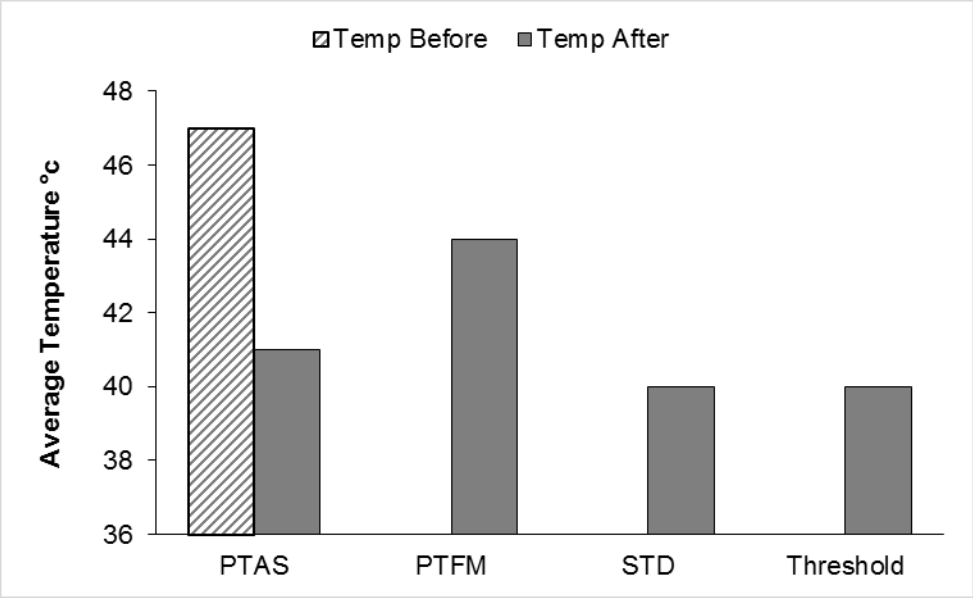

Fig. 27 Average CPU Temperature due to Sparse for 10 ms

Fig. 28 shows the peak CPU temperature of FFT for 500 ms whereas Fig. 29 shows average CPU temperature of FFT for 500 ms. The Fig. 30 shows the peak CPU temperature of LU for 500 ms and Fig. 31 shows the average CPU temperature of LU for 500 ms. In this figure, the LU benchmark has better readings for threshold strategy.

Fig. 32 shows the peak CPU temperature of SOR for 500 ms. and Fig. 33 depicts the average CPU temperature of SOR for 500 ms. PTFM and PTAS outperformed other two strategies. Fig. 34 and Fig. 35 shows the peak and average CPU temperature of Sparse for 500 ms. We see similar thermal behavior for SOR and Sparse.

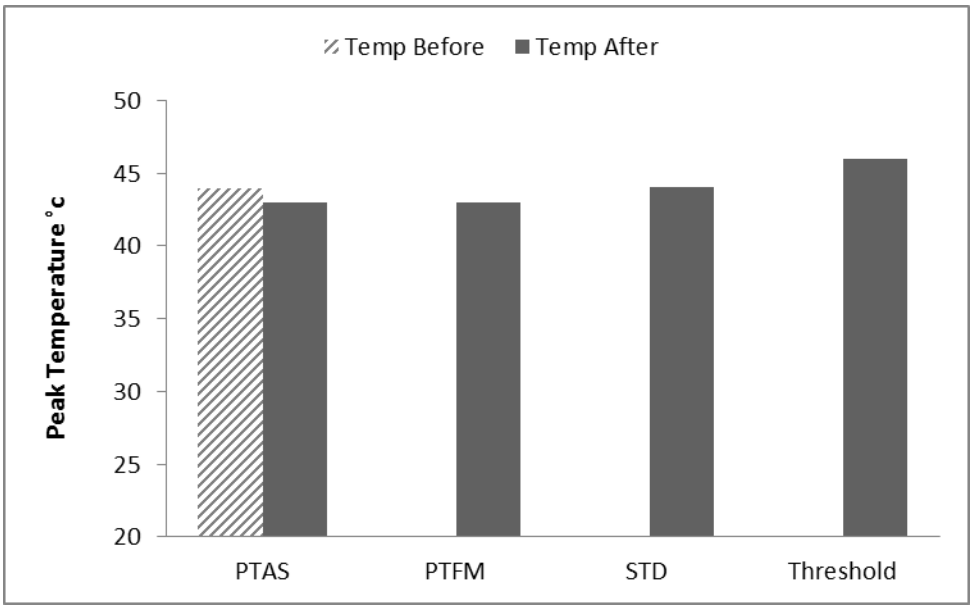

Fig. 28 Peak CPU Temperature due to FFT for 500 ms

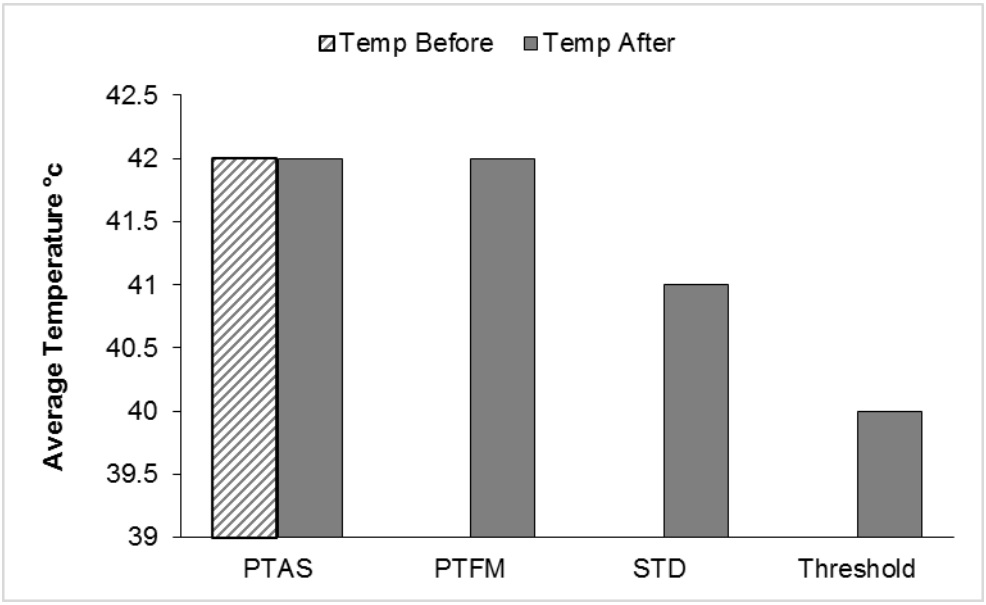

Fig. 29 Average CPU Temperature due to FFT for 500 ms

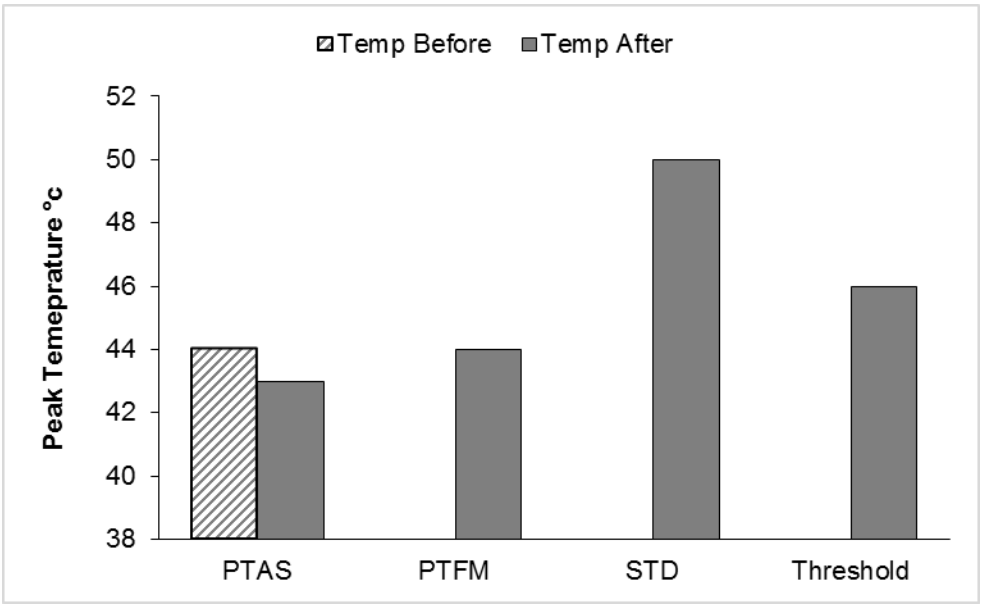

Fig. 30 Peak CPU Temperature due to LU for 500 ms

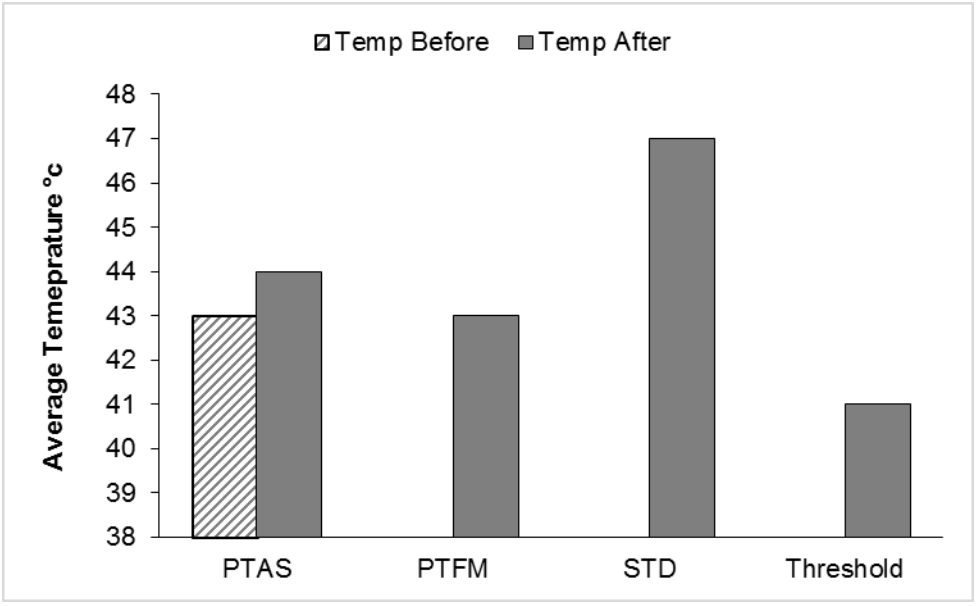

Fig. 31 Average CPU Temperature due to LU for 500 ms

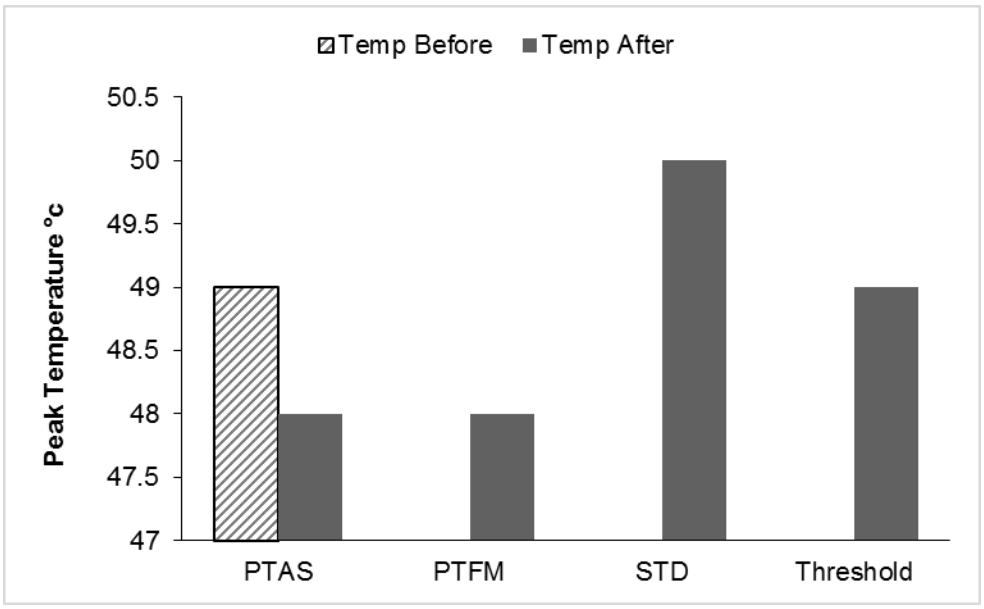

Fig. 32 Peak CPU Temperature due to SOR for 500 ms

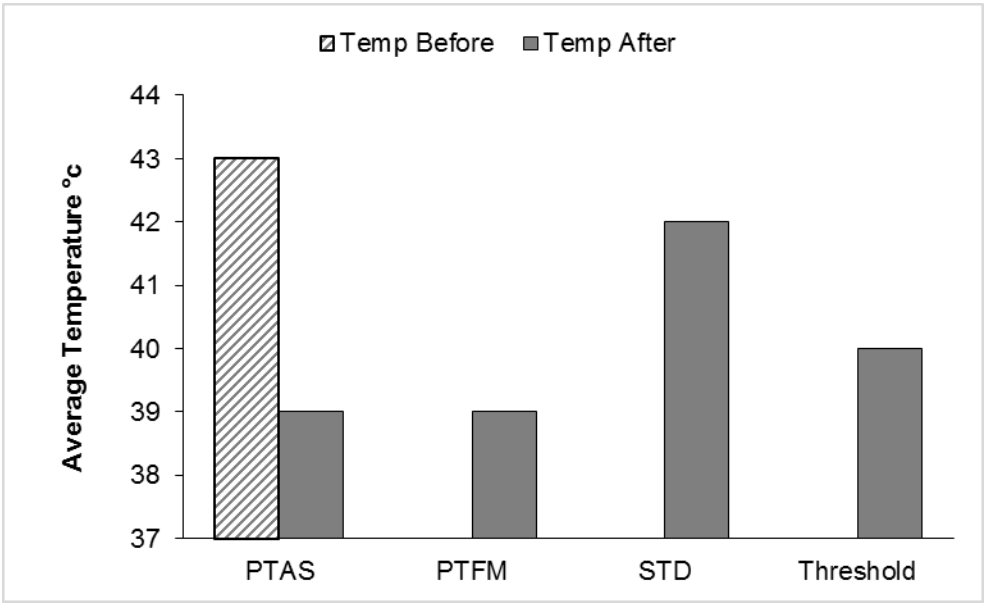

Fig. 33 Average CPU Temperature due to SOR for 500 ms

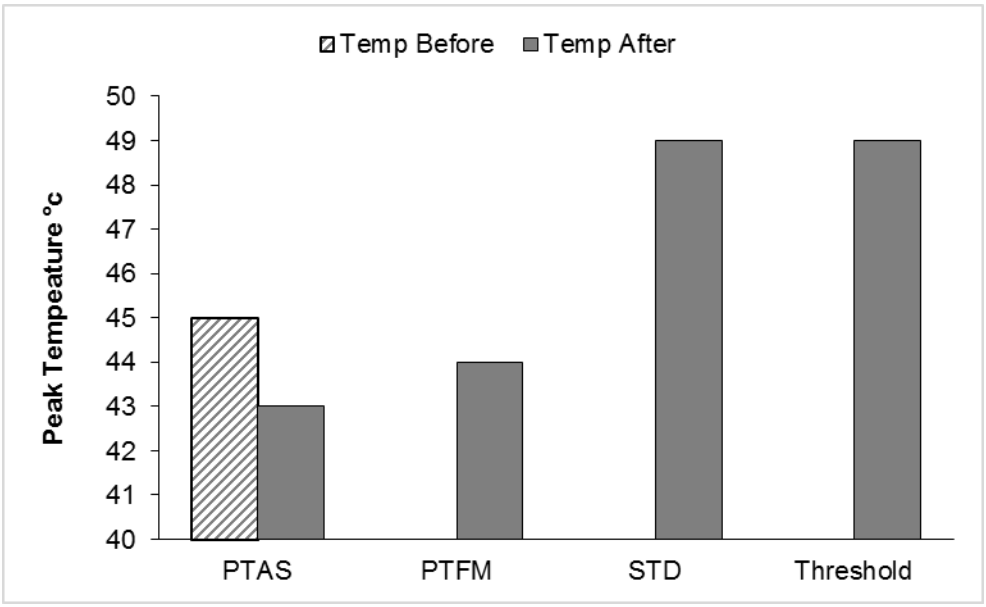

Fig. 34 Peak CPU Temperature due to Sparse for 500 ms

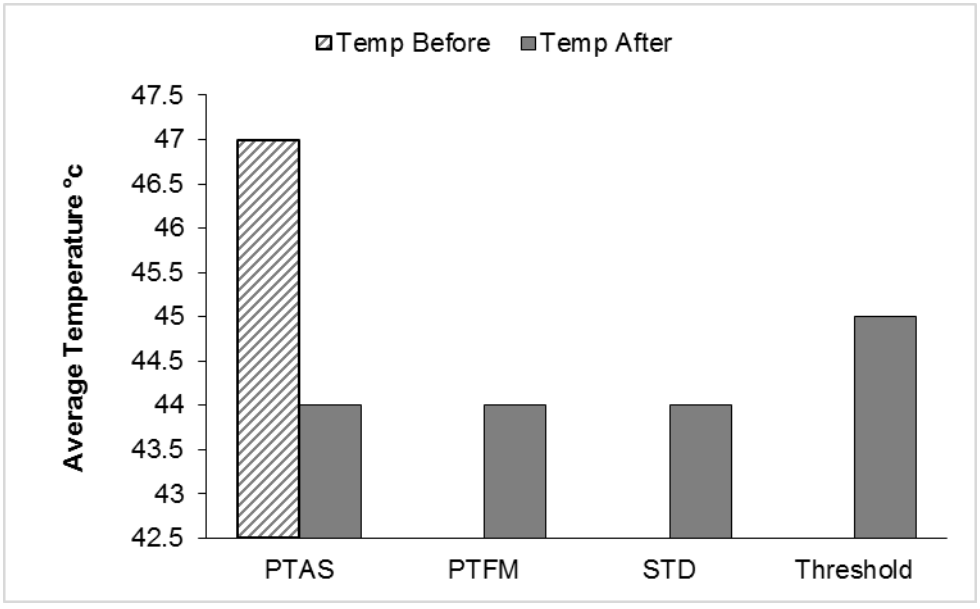

Fig. 35 Average CPU Temperature due to Sparse for 500 ms

Fig. 36 shows the peak CPU temperature of FFT for 1 s whereas Fig. 37 shows average CPU temperature of FFT for 1 s. The PTFM strategy outperformed other three

strategies at 1 s. We see that putting a process to sleep for a larger time might decrease the performance as process takes a longer duration to complete. The Fig. 38 shows the peak CPU temperature of LU for 500 ms and Fig. 39 shows the average CPU temperature of LU for 500 ms. The Fig. 40 shows the peak CPU temperature of SOR for 1 s and Fig. 41 average CPU temperature of SOR for 1 ms. We see similar thermal behavior for LU and SOR.

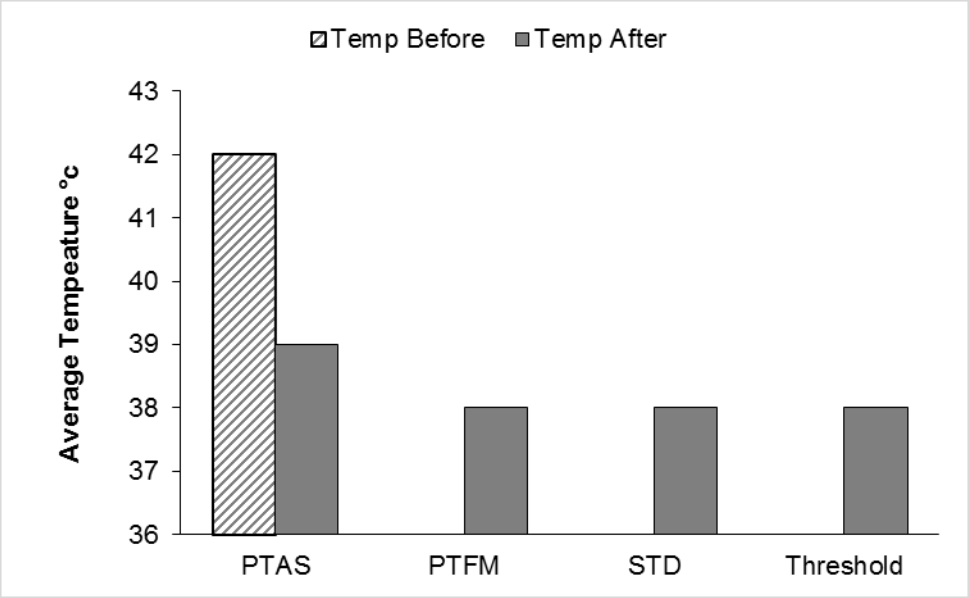

Fig. 36 Peak CPU Temperature due to FFT for 1 s

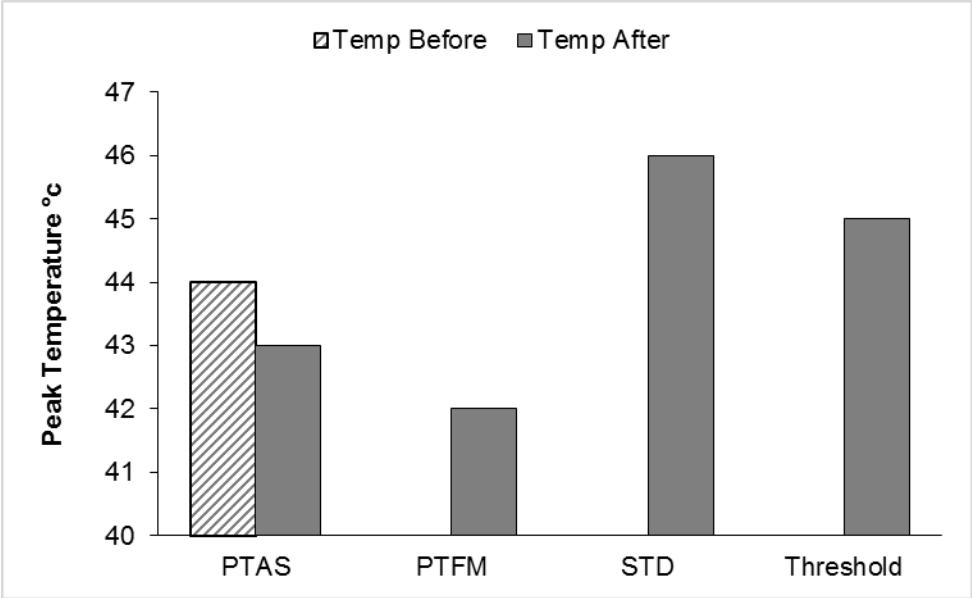

Fig. 37 Average CPU Temperature due to FFT for 1 s

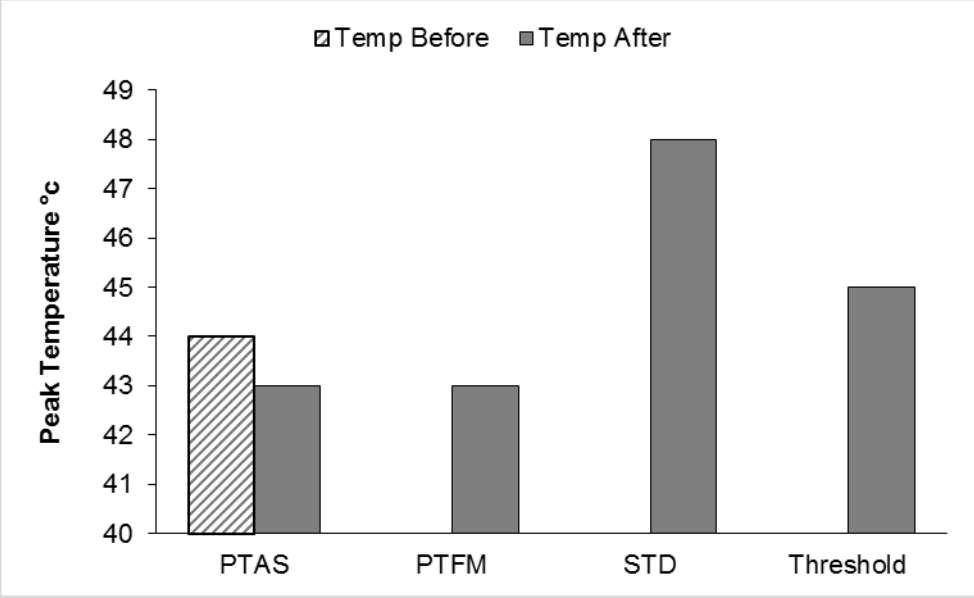

Fig. 38 Peak CPU Temperature due to LU for 1 s

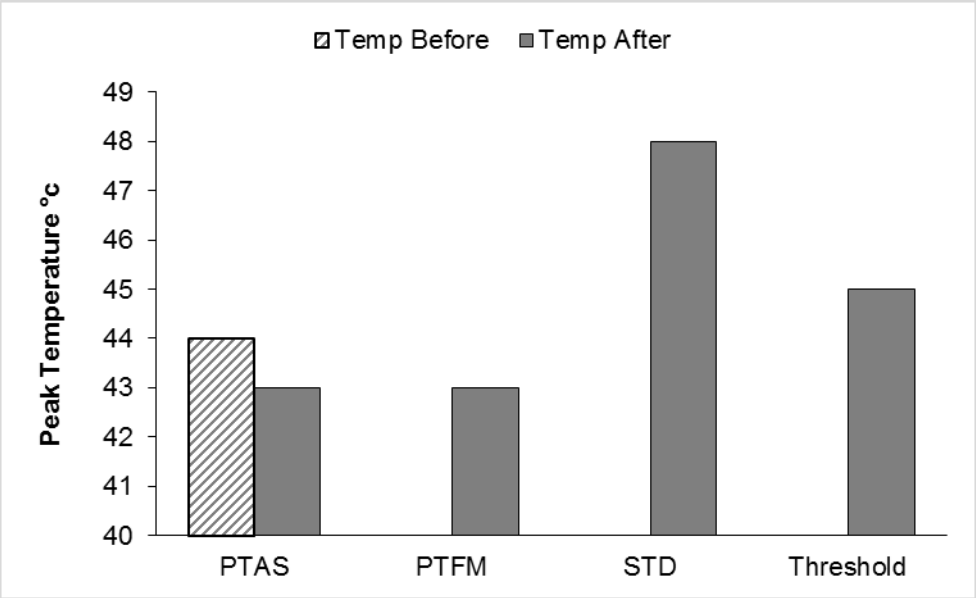

Fig. 39 Average CPU Temperature due to LU for 1 s

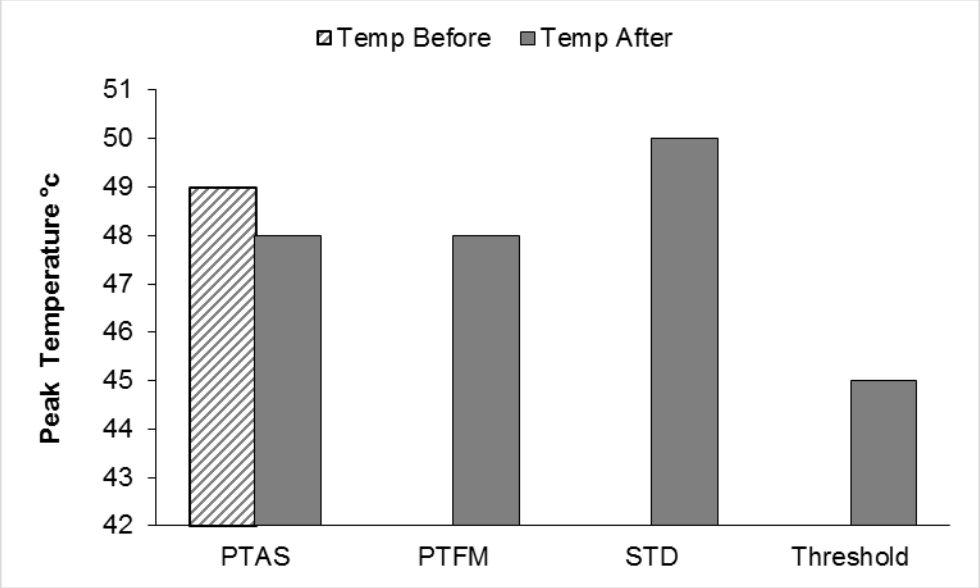

Fig. 40 Peak CPU Temperature due to SOR for 1 s

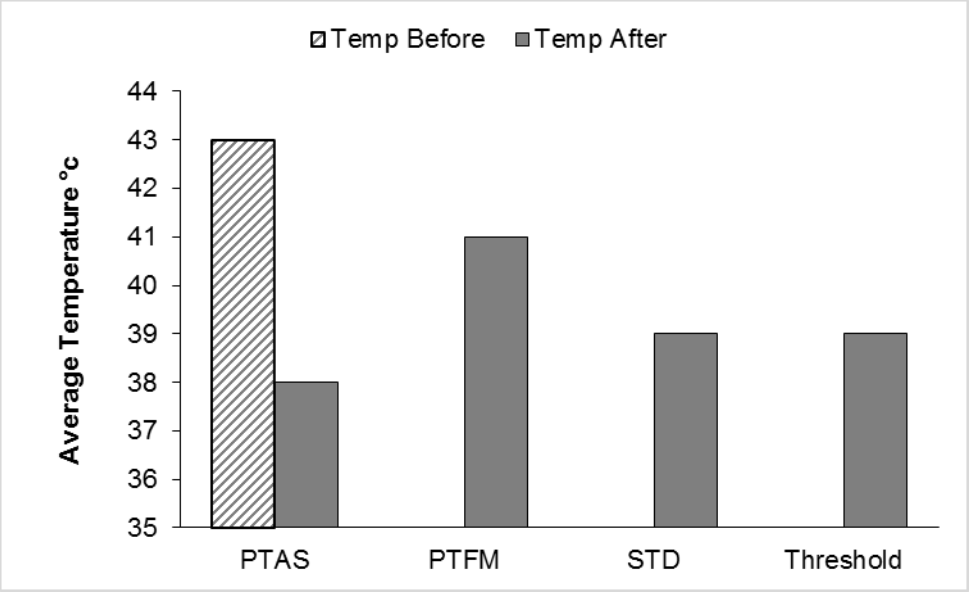

Fig. 41 Average CPU Temperature due to SOR for 1 s

Chapter 5 Conclusion

In this dissertation, we developed two proactive thermal aware approaches PTAS and PTFM, which reduces CPU temperature by predicting the higher temperature gradient of a process using rates of change of current CPU temperature, floating point access rates and memory access rates. We formed a regression predictor formed by FLOPS, memory and CPU temperature and cut-off a process when the predicted gradient goes beyond threshold gradient. We do not cut-off all processes but we put a process to sleep when its predicted gradient exceeds threshold gradient. We found thermal improvements in both the strategies. We varied sleep time and gradient of the process to optimize PTFM. We compared our strategies with PTAS, STD and Threshold strategies and we found that PTFM outperformed other approaches. PTAS and PTFM can be applied along with the Intel PID controller technique, DVFS, DTM, and leakage energy strategies, making the benefits of this approach far exceed those strategies. Since it is a software scheduling strategy, it can be readily applied to all CPUs, including those in mobile devices. PTAS and PTFM successfully lower the CPU temperature using a prediction based on FLOPS and memory. The benefits are beyond fp-applications (floating point) and these two strategies can be implemented for non fp-applications. Preliminary results on Spec suite benchmarks gave similar results.

BIBLIOGRAPHY

[1] Z. Xiuyi, Y. Jun, X. Yi, Z. Youtao and Z. Jianhua, "Thermal-Aware Task Scheduling for 3D Multicore Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 1, pp. 60-71, 2010.

[2] A. Coskun, T. Rosing, K. Whisnant and K. Gross, "Temperature-aware MPSoC scheduling for reducing hot spots and gradients," *in Proceedings of Design Automation Conference*, pp. 49-54, 2008.

[3] J. Altet and A. Rubio, in *Thermal Testing of Integrated Circuits*, Spain, Springer, 2002, pp. 1-21.

[4] A. Raid and R. Tajana, "Predict and Act: Dynamic Thermal Management for Multi-Core processors," in *ISLPED*, San Fransisco, 2009.

[5] Elinux, "Linux.org," Linux, [Online]. Available: http://elinux.org/images/2/2b/A_New_Simplified_Thermal_Framework_For_ARM_Platforms.pdf. [Accessed 25 Sep 2012].

[6] "Hadoop," [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html. [Accessed 12 Feb 2013].

[7] A. Kumar, S. Li, P. Li-Shiuan and N. Jha, "System-level dynamic thermal management for high-performance microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 96-108, 2008.

[8] Y. Chuan- Yue, C. Jian-Jia, L. Thiele and K. Tei-Wei, "Energy-efficient real-time task scheduling with temperature-dependent leakage," in *Design, Automation & Test in Europe Conference & Exhibition*, Dresden, Germany, 2010.

[9] L. Wei and A. Nannarelli, "Temperature aware power optimization for multicore floating-point units," in *Signals, Systems and Computers (ASILOMAR)*, Pacific grove, CA, 2010.

[10] V. Chaturvedi, P. Thanarungroj, L. Chen and Q. Gang, "Validation of scheduling techniques to reduce peak temperature on an architectural level platform set-up," in *Proceedings of IEEE Southeastcon*, Nashville, Tennessee, 2011.

[11] T. Senjyu, S. Chakraborty, A. Saber, H. Toyama, A. Yona and T. Funabashi, "Thermal generation scheduling strategy using binary clustered particle swarm optimization algorithm," in *Power and Energy Conference(PECon)*, Orlando, Florida, 2008.

[12] T. Kojima, Y. Yamada, Y. Nishibe and K. Torii, "RC Compact Thermal Model of HV Inverter Module for Electro-Thermal Coupling Simulation," in *PCC '07, Power Conversion Conference*, Nagoya, Japan, 2007.

[13] N. Fisher, C. Jain-Jia, W. Shengquan, L. Thiele and W. Shengquan, "Thermal-Aware Global Real-Time Scheduling on Multicore Systems," in *Real-Time and Embedded Technology and Applications Symposium(RTAS)*, San Fransisco, CA, 2009.

[14] C. Jin and D. L. Maskell, "High level event driven thermal estimation for thermal aware task allocation and scheduling," *15th Asia and South Pacific Design Automation Conference*, pp. 793-798, 2010.

[15] J. Jiajia, F. Yuzhuo, L. Ting, W. Han, H. Xing and W. Janfang, "Performance analysis and optimization for homogenous multi-core system based on 3D Torus Network on Chip," in *8th IEEE International NEWCAS Conference*, Boston, MA, 2010.

[16] D. Yang, M. Kandemir, P. Raghavan and M. Irwin, "A helper thread based EDP reduction scheme for adapting application execution in CMPs," in *Parallel and Distributed Processing(IPDPS)*, Shangai, China, 2008.

[17] A. Merkel and F. Bellosa, "Event–Driven Thermal Management in SMP Systems," in *Proceedings of the Second Workshop on Temperature-Aware Computer Systems*, Madison, USA, 2005.

[18] A. Weissel and F. Bellosa, "Dynamic thermal managment for distibuted systems," 2005. [Online]. Available: www.cs.virginia.edu/~skadron/tacs/weiss.pdf. [Accessed 15 Feb 2014].

[19] F. Bellosa, "Job Scheduling Strategies for Parallel Processing," *Lecture Notes in Computer Science*, vol. 1162, pp. 271-289, 1996.

[20] Intel, "Processor design," 2009 Jun. [Online]. Available: http://download.intel.com/design/processor/designex/317804.pdf. [Accessed 25 Sep 2012].

[21] AMD, "Energy features," [Online]. Available: http://www.amd.com/us/products/desktop/processors/athlon-ii-x2/Pages/athlon-ii-key-features.aspx. [Accessed 25 Sep 2012].

[22] Linux, "Operating system," Linux, [Online]. Available: www.linux.org. [Accessed 25 Sep 2012].

[23] Linux, "Kernel.org," [Online]. Available: http://kernel.org/doc/ols/2008/ols2008v2-pages-227-234.pdf. [Accessed 25 Sep 2012].

[24] Coretemp, "alcpu.com," [Online]. Available: http://www.alcpu.com/CoreTemp/. [Accessed 25 Dec 2012].

[25] Y. He, W. Hsu and C. Leiserson, "Provably Efficient Online Non-clairvoyant Adaptive Scheduling," in *International Parallel and Distributed Processing Symposium(IPDPS)*, Shenzen, China, 2007.

[26] V. Sivakumar, V. Bharadwaj and T. Robertazzi, "Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems," *IEEE Transacations on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1450-1461, 2007.

[27] PAPI, "Event driven interface," [Online]. Available: http://icl.cs.utk.edu/papi/. [Accessed 15 Sep 2012].

[28] B. Miller and R. Pozo, "Scimark 2.0," 31 March 2004. [Online]. Available: http://math.nist.gov/scimark2. [Accessed 19 August 2011].

[29] lmsensors, "Hardware sensors," [Online]. Available: www.lm-sensors.org. [Accessed 15 Sep 2012].

[30] S. S. Anupindi and S. Baskiyar, *Proactive Thermal Aware Scheduling*, Unpublished manuscript, 2014.

[31] Lenovo, "Laptop Energy," IBM, [Online]. Available: http://support.lenovo.com/en_AE/downloads/detail.page?DocID=HT034410. [Accessed 10 Jan 2014].

[32] Dell, "Power and Cooling," Dell, [Online]. Available: http://content.dell.com/us/en/enterprise/d/business~solutions~whitepapers~en/Documents~power-and-cooling-innovations.pdf.aspx. [Accessed 31 Jan 2013].

[33] IBM, "Power Systems Energy," [Online]. Available: http://www-

03.ibm.com/systems/power/software/energy/about.html. [Accessed 31 Jan 2013].

[34] Linux, "CPU Frequency," [Online]. Available: http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=%2Fliaai%2Fcpufreq%2Fliaai-cpufreq.htm. [Accessed 31 Jan 2013].

[35] Apache, “Maths Library,” Apache Commons, [Online]. Available: http://commons.apache.org/math/api-2.2/index.html. [Accessed 23 Jan 2014].

[36] V. Tiwari, S. Malik and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," in *Proceedings of the IEEE/ACM international conference on Computer-aided design*, Los Alamitos, CA, 1994.

APPENDIX

```
Initializing...
-------------------------------------------------------------------------------
PAPI Version            : 5.0.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Pentium(R) Dual-Core CPU       T4300  @ 2.10GHz (23)
CPU Revision            : 10.000000
CPUID Info              : Family: 6  Model: 23  Stepping: 10
CPU Max Megahertz        : 2100
CPU Min Megahertz        : 1200
Hdw Threads per core     : 1
Cores per Socket         : 2
NUMA Nodes               : 1
CPUs per Node            : 2
Total CPUs              : 2
Running in a VM          : no
Number Hardware Counters : 5
Max Multiplex Counters   : 64
-------------------------------------------------------------------------------


Inner Product Test:
i        papi       theory     diff   %error
---------------------------------------------------------------------------
1          2          2       0     0.0000
2          4          4       0     0.0000
3          6          6       0     0.0000
4          8          8       0     0.0000
5         10         10        0     0.0000
6         12         12        0     0.0000
7         14         14        0     0.0000
8         16         16        0     0.0000
9         18         18        0     0.0000
10         21          20        1     5.0000
11         22          22        0     0.0000
12         24          24        0     0.0000
13         26          26        0     0.0000
14 ……...28……28…… 0….0.0000
```

15	30	30	0	0.0000
16	32	32	0	0.0000
17	34	34	0	0.0000
18	36	36	0	0.0000
19	38	38	0	0.0000
20	40	40	0	0.0000
21	42	42	0	0.0000
22	44	44	0	0.0000
23	46	46	0	0.0000
24	48	48	0	0.0000
25	50	50	0	0.0000
26	52	52	0	0.0000
27	54	54	0	0.0000
28	57	56	1	1.7857
29	58	58	0	0.0000
30	60	60	0	0.0000
31	62	62	0	0.0000
32	64	64	0	0.0000
33	66	66	0	0.0000
34	68	68	0	0.0000
35	70	70	0	0.0000
36	72	72	0	0.0000
37	74	74	0	0.0000
38	76	76	0	0.0000
39	78	78	0	0.0000
40	80	80	0	0.0000
41	82	82	0	0.0000
42	84	84	0	0.0000
43	86	86	0	0.0000
44	88	88	0	0.0000
45	90	90	0	0.0000
46	92	92	0	0.0000
47	94	94	0	0.0000
48	97	96	1	1.0417
49	98	98	0	0.0000
50	100	100	0	0.0000
51	102	102	0	0.0000
52	104	104	0	0.0000
53	106	106	0	0.0000
54	108	108	0	0.0000
55	111	110	1	0.9091
56	112	112	0	0.0000
57	115	114	1	0.8772
58	116	116	0	0.0000
59	118	118	0	0.0000

60	120	120	0	0.0000
61	122	122	0	0.0000
62	124	124	0	0.0000
63	126	126	0	0.0000
64	128	128	0	0.0000
65	130	130	0	0.0000
66	132	132	0	0.0000
67	134	134	0	0.0000
68	136	136	0	0.0000
69	138	138	0	0.0000
70	141	140	1	0.7143
71	142	142	0	0.0000
72	144	144	0	0.0000
73	146	146	0	0.0000
74	149	148	1	0.6757
75	150	150	0	0.0000
76	152	152	0	0.0000
77	154	154	0	0.0000
78	156	156	0	0.0000
79	158	158	0	0.0000
80	160	160	0	0.0000
81	162	162	0	0.0000
82	164	164	0	0.0000
83	166	166	0	0.0000
84	168	168	0	0.0000
85	170	170	0	0.0000
86	172	172	0	0.0000
87	174	174	0	0.0000
88	177	176	1	0.5682
89	178	178	0	0.0000
90	180	180	0	0.0000
91	183	182	1	0.5495
92	184	184	0	0.0000
93	186	186	0	0.0000
94	188	188	0	0.0000
95	190	190	0	0.0000
96	193	192	1	0.5208
97	195	194	1	0.5155
98	196	196	0	0.0000
99	198	198	0	0.0000
100	200	200	0	0.0000
150	301	300	1	0.3333
200	400	400	0	0.0000
250	500	500	0	0.0000
300	600	600	0	0.0000
350	700	700	0	0.0000

```
400         801         800        1     0.1250
450         900         900        0     0.0000
500        1000        1000        0     0.0000
--------------------------------------------------------------------------------
PAPI Version             : 5.0.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Pentium(R) Dual-Core CPU       T4300  @ 2.10GHz (23)
CPU Revision             : 10.000000
CPUID Info               : Family: 6  Model: 23  Stepping: 10
CPU Max Megahertz        : 2100
CPU Min Megahertz        : 1200
Hdw Threads per core     : 1
Cores per Socket         : 2
NUMA Nodes               : 1
CPUs per Node            : 2
Total CPUs               : 2
Running in a VM          : no
Number Hardware Counters : 5
Max Multiplex Counters   : 64
--------------------------------------------------------------------------------


Matrix Vector Test:
i         papi       theory     diff   %error
---------------------------------------------------------------------------
1           2           2        0     0.0000
2           8           8        0     0.0000
3          18          18        0     0.0000
4          32          32        0     0.0000
5          50          50        0     0.0000
6          72          72        0     0.0000
7          98          98        0     0.0000
8         128         128        0     0.0000
9         162         162        0     0.0000
10          200         200        0     0.0000
11          242         242        0     0.0000
12          288         288        0     0.0000
13          338         338        0     0.0000
14          392         392        0     0.0000
15          450         450        0     0.0000
16          512         512        0     0.0000
17          578         578        0     0.0000
18          648         648        0     0.0000
19          722         722        0     0.0000
20          800         800        0     0.0000
21          882         882        0     0.0000
```

22	968	968	0	0.0000
23	1058	1058	0	0.0000
24	1152	1152	0	0.0000
25	1250	1250	0	0.0000
26	1352	1352	0	0.0000
27	1458	1458	0	0.0000
28	1568	1568	0	0.0000
29	1682	1682	0	0.0000
30	1800	1800	0	0.0000
31	1922	1922	0	0.0000
32	2048	2048	0	0.0000
33	2178	2178	0	0.0000
34	2312	2312	0	0.0000
35	2450	2450	0	0.0000
36	2592	2592	0	0.0000
37	2738	2738	0	0.0000
38	2888	2888	0	0.0000
39	3042	3042	0	0.0000
40	3200	3200	0	0.0000
41	3362	3362	0	0.0000
42	3528	3528	0	0.0000
43	3698	3698	0	0.0000
44	3872	3872	0	0.0000
45	4050	4050	0	0.0000
46	4232	4232	0	0.0000
47	4418	4418	0	0.0000
48	4608	4608	0	0.0000
49	4802	4802	0	0.0000
50	5000	5000	0	0.0000
51	5202	5202	0	0.0000
52	5408	5408	0	0.0000
53	5618	5618	0	0.0000
54	5832	5832	0	0.0000
55	6050	6050	0	0.0000
56	6272	6272	0	0.0000
57	6498	6498	0	0.0000
58	6728	6728	0	0.0000
59	6962	6962	0	0.0000
60	7200	7200	0	0.0000
61	7442	7442	0	0.0000
62	7688	7688	0	0.0000
63	7938	7938	0	0.0000
64	8192	8192	0	0.0000
65	8451	8450	1	0.0118
66	8712	8712	0	0.0000
67	8980	8978	2	0.0223

68	9248	9248	0	0.0000
69	9524	9522	2	0.0210
70	9800	9800	0	0.0000
71	10084	10082	2	0.0198
72	10368	10368	0	0.0000
73	10658	10658	0	0.0000
74	10953	10952	1	0.0091
75	11250	11250	0	0.0000
76	11557	11552	5	0.0433
77	11858	11858	0	0.0000
78	12168	12168	0	0.0000
79	12482	12482	0	0.0000
80	12803	12800	3	0.0234
81	13122	13122	0	0.0000
82	13450	13448	2	0.0149
83	13780	13778	2	0.0145
84	14113	14112	1	0.0071
85	14453	14450	3	0.0208
86	14793	14792	1	0.0068
87	15138	15138	0	0.0000
88	15488	15488	0	0.0000
89	15842	15842	0	0.0000
90	16201	16200	1	0.0062
91	16562	16562	0	0.0000
92	16929	16928	1	0.0059
93	17299	17298	1	0.0058
94	17674	17672	2	0.0113
95	18050	18050	0	0.0000
96	18432	18432	0	0.0000
97	18818	18818	0	0.0000
98	19208	19208	0	0.0000
99	19603	19602	1	0.0051
100	20001	20000	1	0.0050
150	45002	45000	2	0.0044
200	80001	80000	1	0.0012
250	125002	125000	2	0.0016
300	180005	180000	5	0.0028
350	245005	245000	5	0.0020
400	320003	320000	3	0.0009
450	405009	405000	9	0.0022
500	500008	500000	8	0.0016

PAPI Version : 5.0.0.0
Vendor string and code : GenuineIntel (1)
Model string and code : Pentium(R) Dual-Core CPU T4300 @ 2.10GHz (23)
CPU Revision : 10.000000

CPUID Info : Family: 6 Model: 23 Stepping: 10
CPU Max Megahertz : 2100
CPU Min Megahertz : 1200
Hdw Threads per core : 1
Cores per Socket : 2
NUMA Nodes : 1
CPUs per Node : 2
Total CPUs : 2
Running in a VM : no
Number Hardware Counters : 5
Max Multiplex Counters : 64

Matrix Multiply Test:

i	papi	theory	diff	%error
1	2	2	0	0.0000
2	16	16	0	0.0000
3	54	54	0	0.0000
4	128	128	0	0.0000
5	250	250	0	0.0000
6	432	432	0	0.0000
7	686	686	0	0.0000
8	1024	1024	0	0.0000
9	1458	1458	0	0.0000
10	2000	2000	0	0.0000
11	2662	2662	0	0.0000
12	3456	3456	0	0.0000
13	4394	4394	0	0.0000
14	5488	5488	0	0.0000
15	6750	6750	0	0.0000
16	8192	8192	0	0.0000
17	9826	9826	0	0.0000
18	11664	11664	0	0.0000
19	13718	13718	0	0.0000
20	16000	16000	0	0.0000
21	18522	18522	0	0.0000
22	21296	21296	0	0.0000
23	24334	24334	0	0.0000
24	27648	27648	0	0.0000
25	31250	31250	0	0.0000
26	35152	35152	0	0.0000
27	39366	39366	0	0.0000
28	43904	43904	0	0.0000

29	48778	48778	0	0.0000
30	54000	54000	0	0.0000
31	59582	59582	0	0.0000
32	65536	65536	0	0.0000
33	71874	71874	0	0.0000
34	78608	78608	0	0.0000
35	85750	85750	0	0.0000
36	93313	93312	1	0.0011
37	101306	101306	0	0.0000
38	109744	109744	0	0.0000
39	118638	118638	0	0.0000
40	128000	128000	0	0.0000
41	137842	137842	0	0.0000
42	148178	148176	2	0.0013
43	159014	159014	0	0.0000
44	170368	170368	0	0.0000
45	182250	182250	0	0.0000
46	194672	194672	0	0.0000
47	207646	207646	0	0.0000
48	221184	221184	0	0.0000
49	235298	235298	0	0.0000
50	250000	250000	0	0.0000
51	265302	265302	0	0.0000
52	281216	281216	0	0.0000
53	297754	297754	0	0.0000
54	314928	314928	0	0.0000
55	332750	332750	0	0.0000
56	351232	351232	0	0.0000
57	370386	370386	0	0.0000
58	390224	390224	0	0.0000
59	410758	410758	0	0.0000
60	432001	432000	1	0.0002
61	453962	453962	0	0.0000
62	476656	476656	0	0.0000
63	500094	500094	0	0.0000
64	524289	524288	1	0.0002
65	549252	549250	2	0.0004
66	574992	574992	0	0.0000
67	601529	601526	3	0.0005
68	628864	628864	0	0.0000
69	657020	657018	2	0.0003
70	686002	686000	2	0.0003
71	715822	715822	0	0.0000
72	746498	746496	2	0.0003
73	778035	778034	1	0.0001
74	810449	810448	1	0.0001

```
75      843752      843750      2     0.0002
76      877954      877952      2     0.0002
77      913066      913066      0     0.0000
78      949108      949104      4     0.0004
79      986078      986078      0     0.0000
80     1024002     1024000      2     0.0002
81     1062886     1062882      4     0.0004
82     1102740     1102736      4     0.0004
83     1143579     1143574      5     0.0004
84     1185408     1185408      0     0.0000
85     1228257     1228250      7     0.0006
86     1272113     1272112      1     0.0001
87     1317009     1317006      3     0.0002
88     1362977     1362944     33     0.0024
89     1409944     1409938      6     0.0004
90     1458000     1458000      0     0.0000
91     1507154     1507142     12     0.0008
92     1557384     1557376      8     0.0005
93     1608834     1608714    120     0.0075
94     1661193     1661168     25     0.0015
95     1714755     1714750      5     0.0003
96     1769477     1769472      5     0.0003
97     1825351     1825346      5     0.0003
98     1882386     1882384      2     0.0001
99     1940962     1940598    364     0.0188
100    2000003     2000000      3     0.0002
150    6750071     6750000     71     0.0011
200   16000275    16000000    275     0.0017
250   31251991    31250000   1991     0.0064
300   54005677    54000000   5677     0.0105
350   85750270    85750000    270     0.0003
400  128007477   128000000   7477     0.0058
450  182265956   182250000  15956     0.0088
500  250004275   250000000   4275     0.0017
calibrate.c                   PASSED
```