Investigating the Effects of Task Scheduling on Thermal Behavior

Eren Kursun[1], Chen-Yong Cher, Alper Buyuktosunoglu, Pradip Bose

IBM T. J. Watson Research Center

Abstract — Thermal behavior is expected to be an important design consideration for the next generation of microprocessors. Most dynamic thermal management techniques are based on the principle of 'reactive throttling', where some form of performance throttling (such as voltage or frequency scaling) is invoked, after a pre-architected temperature threshold has been exceeded. As such, there is performance degradation with each such DTM technique. In some cases, if the temperature threshold is set to a conservative value for package/cooling cost considerations, the degradation can be quite severe. In this paper, we investigate the potential benefits of temperature-aware task scheduling with minimum or no performance overhead. We explored policies based on metrics such as average temperature; dynamic profiling of threads and localized heating, instead of IPC-based metrics. Our preliminary results indicate that thermal-aware task scheduling provides significant alleviation on chip temperatures. We observed up to 52% reduction in the number of cycles above the thermal threshold, compared to a thermally-oblivious policy. On average, our MinTemp policy yields less than 3% thermally critical cycles, even on a challenging benchmark set such as SPEC2000 - with 12 of the 25 benchmarks above 360K. As we employ already existing scheduling infrastructure, there was no considerable change in net performance.

1. Introduction

Thermal characteristics of modern microprocessors have presented numerous challenges in recent years. Continuous increases in power dissipation along with feature scaling caused the power density to double every *3* years *[1]*. The corresponding elevation in the on-chip temperatures has started to become a critical design constraint. Thermal characteristics affect vital aspects of the microprocessors: such as timing, reliability, fault-tolerance as well as increased packaging and cooling costs. As the reliability of an electronic circuit is exponentially dependent on the junction temperature: a *10-15^oC* rise in the operating temperature results in *~2x* reduction in the lifespan of the device *[7]*. Furthermore, effective operating speed decreases at higher temperatures due to the temperature dependence of carrier mobility.

Static power dissipation, which is expected to constitute over *50%* of the total power consumption *[8]* in post-90nm CMOS technologies, has exponential dependency on temperature. The positive feedback loop between leakage power and on-chip temperatures, also called *'Thermal Runaway'*, can cause serious damage if not carefully controlled through the package/cooling and dynamic thermal management techniques. A wide range of packaging and cooling solutions has been proposed to alleviate the thermal problems. However, their costs increase at a super-linear rate *(>$10* per Watt after *65$^oC$*, according to *[5]*) and package impedances are saturating to values that are challenging to manage cost-effectively. Due to the increases in the transistor count and clock frequency, it is no longer practical to design the processor packaging for the worst-case temperature; at least, with regard to the desktop and lower-end server market. Many chip producers use packaging that target a safe threshold below the absolute worst power/thermal behavior. For instance, Intel's Pentium4 is reported to utilize packaging that is designed for around *80%* of the worst-case power dissipation. Beyond this threshold the thermally challenging, yet less-frequent, workloads are handled with global clock throttling *[5]*.

Dynamic thermal management is commonly used to manage the gap between the absolute worst-case temperatures and packaging constraints. There has been a wide range of thermal management techniques proposed in recent years [2]-[9]. As the thermal problems exacerbate for the next generation microprocessor architectures, the need for efficient temperature management is also becoming more prominent. Most of the current DTM techniques use throttling of processor resources such as frequency, voltage or pipeline stages, in order to keep temperatures below a safe threshold. However, in some cases the corresponding performance degradation is significant, depending on the frequency and the severity of heating problems. For instance, activity migration is usually accompanied with considerable performance loss due to the overhead of moving processor state and the cold start effects [12].

[1] Eren Kursun was a summer intern (from UCLA) working at IBM T. J. Watson Research Center, when this work was done.

Instead, if one can leverage the existing context switching infrastructure supported by the system software (e.g. the operating system and hypervisor layers) and implement a temperature-aware task scheduling heuristics, temperature reduction without additional on-chip hardware complexity is possible. The goal of this study is to investigate the potential benefits of scheduling techniques for temperature management with minimum or no performance degradation. Our initial experimental analysis indicates that thermal-aware scheduling policies alleviate the on-chip temperatures effectively. The policies successfully manage to keep the temperatures within a safe range, even for challenging benchmark sets such as SPEC2000 on a Power4-like architecture, where *12* of the *25* benchmarks have temperatures above *360K*. The scheduling policies we investigated in this study are implemented on single core architecture. Yet the same concept can be applied to a multi-core processor. Moreover, the increase in the number of threads and processor cores is likely to reflect as an advantage that scheduling policies can utilize even further for multi-core processor architectures.

It is important to note that temperature-aware operating system scheduling is beyond the scope of this paper. Our goal is to probe the limits of thermal-aware task scheduling through the use of architectural tools and techniques. The complete implementation of temperature-aware scheduling policies in the operating system kernel is a part of our future work.

The rest of the paper is organized as follows: In Section 2 we discuss the most relevant related work; Section 3 presents a motivating example; preliminaries on temperature modeling are in Section 4, followed by a discussion of experimental methodology in Section 5. Section 6 introduces the proposed thermal-aware scheduling policies and finally concluding remarks and future work are in Section 7.

2. Related Work

Operating system scheduling policies for temperature management has been studied by various researchers in the past. In this section we only discuss the closest studies. In [6], Moore et.al. investigated temperature-aware workload placement in data centers. They analyze the thermodynamic behavior specific to the data center, and propose heuristics for improvement. In [15], Ghiasi et.al. analyzed a number of temperature-aware operating system scheduling policies for asymmetric cores. The most relevant related work to our study is *[13]*. In this work, Gomaa et. al. present temperature-aware thread assignment and migration policies for a multi-core SMT scenario. The differences between the previous work and this study also highlight our main motivation and contributions:

- In this study we explore a single-thread single-core scenario, where thread migration is not a possible solution. Hence the goal is to investigate the limits of thermally effective thread assignment policies.

- Microprocessor time constants are critical in determining the feasibility of different temperature management schemes. The thermal time constant values are determined by a combination of architectural block properties such as thermal capacitance, resistances, layout area, as well as thermal characteristics of silicon and the process technology. The time constants can be quite different for different processors as a result of these factors. In [13] thermal time constant of the microprocessor is reported to be *10 msec*. Within this context, rapidly heating and cooling the processor is possible, since the time constant guarantees that a heated block will be below the thermal threshold in few milliseconds. During our experimental analysis, we found the time constant for Power4-like architecture to be much longer, in the *100 msec* range. We observed up to *200 msec* time constants for architectural blocks. Therefore, performance degradation due to cooling time of a processor block is much more prominent in this case. This fact motivates our attempts to manage the temperatures so that the critical threshold is not reached. Exceeding the threshold is bound to cause considerable degradation due to the elongated throttling periods of the dynamic thermal management. Yet another effect of our longer thermal time constant is the feasibility of slowly tuning the on-chip temperatures with each thread assignment. As *10 msec* time slices per thread constitutes a small percentage of the thermal time constant range in *100 msecs,* each thread is capable of slowly tuning the processor temperatures below a desired target threshold.

- Previous studies such as [9] and [12] reveal that thread migration can yield significant performance degradation, caused by the copying of register state between cores and the cold start of caches. According to [9] thermally-triggered activity migration causes as high as 53% performance degradation on a two-core monolithic architecture. In this study our goal is to eliminate the performance loss caused by the DTMs, minimizing and even eliminating the degradation from costly techniques such as activity migration. In order to achieve this goal we propose using the already existing task scheduling infrastructure. Our experimental analysis illustrates that our *MinTemp* policy limits the thermally-critical cycles to only *3%* with virtually no performance degradation.

- We also investigated alternative metrics to Hi-IPC/Low-IPC, INT/FP, and IPC-based resource usage of [13]. IPC-based metrics have shortcomings such as:

  - Limiting the negative effects of thermal problems to only performance loss. In reality, other factors such as fault-tolerance, reliability, packaging and cooling costs are also affected by the temperature profile.

  - Spatial distribution of hotspots on the chip is not taken into consideration. The proximity of hotspots plays an important role in exacerbating the temperatures.

To better handle these considerations we use real-time temperature readings of microprocessor blocks through thermal sensors, along with dynamic thermal profiling of threads in the scheduling run-queue. Finally, our main goal is to finely tune the processor temperatures with each scheduled thread so that excessive heating can be minimized.

3. Illustrative Case

Let us consider a scenario where a thread from benchmark *Mgrid* has been running on a Power4-like architecture and the steady-state temperatures have been reached. *Mgrid* has *367K* hotspot temperature without any dynamic thermal management on the specified architecture. It causes floating point register file (*FPU_REG*) to exceed the temperature threshold of *358K*. For current microprocessors the operating system scheduler assigns a different thread on the processor based on criteria other than temperature, usually in a form that is a variation of round-robin policy for fairness.

In this case, let us consider the presence of a monitoring facility that detects the thermal behavior of the threads in the scheduling run-queue. Furthermore, this unit collects the temperature readings of the processor blocks from on-chip sensors at the end of the time slice (*10msec* in this case). This information is passed on to the scheduler, where an informed decision can be made. Just as the *FPU_REG* is about to reach the temperature threshold, the scheduler can select the next thread from the thread queue based on the thermal profiles and the heating information from the on-chip sensors. Let us also assume that the threads extracted from SPEC2000 benchmark set are available in the scheduler queue. Since we look at the limit case where our starting point was slightly below 85^oC, the benchmark selection for the next time-slice is important and the scheduler has a number of choices:

Scenario 1: One possibility is to select threads oblivious of the temperature profile and schedule *Applu* in the next time slice. *Applu* has a steady-state hotspot temperature of *365K* for *FPU_REG*. Hence the heating problem will persist and the hotspot temperature will rise above *358K*. A hardware DTM technique such as fetch throttling, dynamic voltage scaling or clock throttling has to be activated for the core to cool down below the safety threshold.

Scenario 2: It is also possible to select a low-temperature benchmark such as *Mcf*, which has the lowest thermal profile of the traces we experimented on. In this case, core temperature decreases below the threshold and no DTM is needed.

Scenario 3: Alternatively, the scheduler can select yet another high-temperature benchmark such as *Crafty*, which reaches hotspot temperature of *363K* for *FXU_REG*. However, since the heating patterns are complementary with the processor, the *FPU_REG* cools down without exceeding the threshold and the *FXU_REG* sees an increase in temperature.

An important point to note is that in none of these scenarios will there be a very significant temperature change: Neither *Crafty* reaches *363K* at the end of the next cycle, nor *Mcf* presents a dramatic decrease in temperatures. As the thermal time constants are in the order of 100 msecs, the temperature change for each benchmark will be incremental. This is caused by the fact that the thermal time constants of the processor are significantly longer than the scheduling time slices. Thread assignment takes place at pre-determined time slices in the scheduling infrastructure. Hence temperature-awareness provides the opportunity to finely tune the on-chip heating with each thread scheduled. As the operating system has more information about the characteristics of each thread such as whether it is real-time or not, deadlines, priorities, processor/memory behavior…etc, the thermal optimization at operating system level is capable of making more informed decisions than pure throttling at hardware DTM techniques. We discuss this in more detail while presenting our temperature-aware scheduling policies in the following sections.

4. Preliminaries

Temperature modeling for microprocessor architectures is based on the duality of the electrical and thermal phenomena. Thermal transient behavior of an architectural block can be estimated with the corresponding thermal time constant.

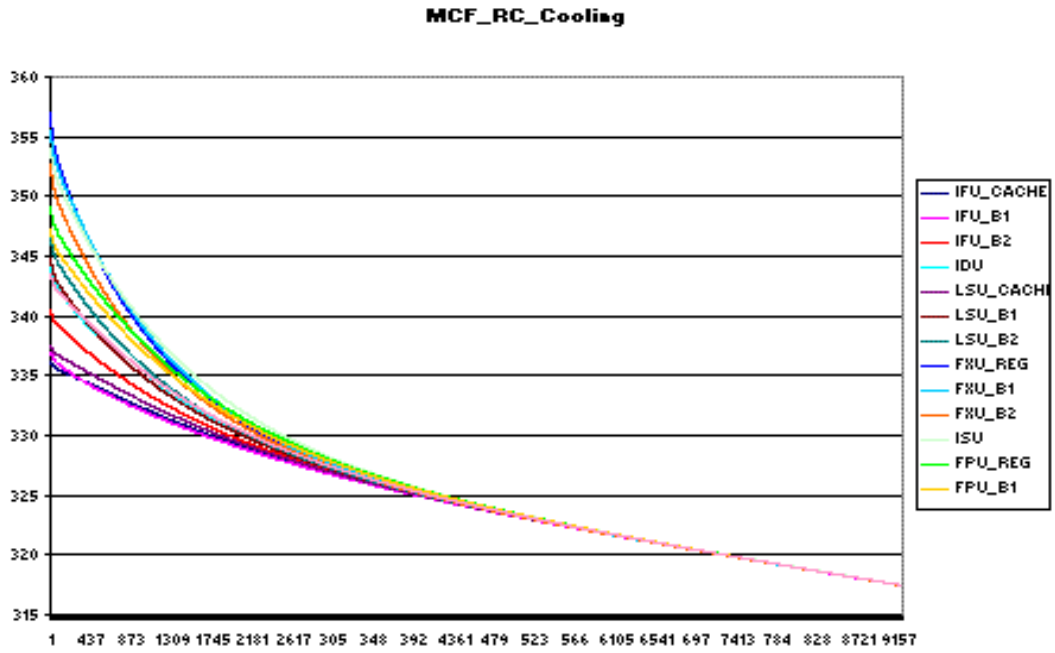

Figure 1. Temperature decay curve for *Mcf*

The time constant for an architectural block or floorplanned unit is dependent on the physical properties such as the thermal capacitance and resistance; which are dictated by the physical dimensions of the block and the intrinsic thermal properties of the silicon die material and specific process technology, yet the time constant independent of the power dissipation.

Figure 1 displays the exponential decay curve for benchmark *Mcf*, when the active power dissipation is reduced to *0* Watts, as the benchmark stops executing. The x axis represents temperature samples. Each sample is taken at *100K* cycles hence the figure illustrates that RC time constants are in *100 msec* range. Thermal time constant is independent of the power dissipation of the architectural block and program behavior. We have observed thermal time constants in the range of *80 msec* to *200 msec* for various architectural blocks.

4.1. SPEC2000 Temperature Profile

Initial thermal analysis of the traces from SPEC2000 benchmark set was performed using Turandot power performance and thermal simulator. We used *400* million and *1* billion instruction traces. Figure 2 illustrates the maximum temperatures architectural blocks over the *400* million traces. SPEC2000 is a thermally challenging benchmark set with *12* out of *25* benchmarks have blocks with hotspot temperatures above the *360K*. Keeping the temperatures below *358K* threshold without a dynamic temperature management scheme is a challenging task over such a pool of threads.

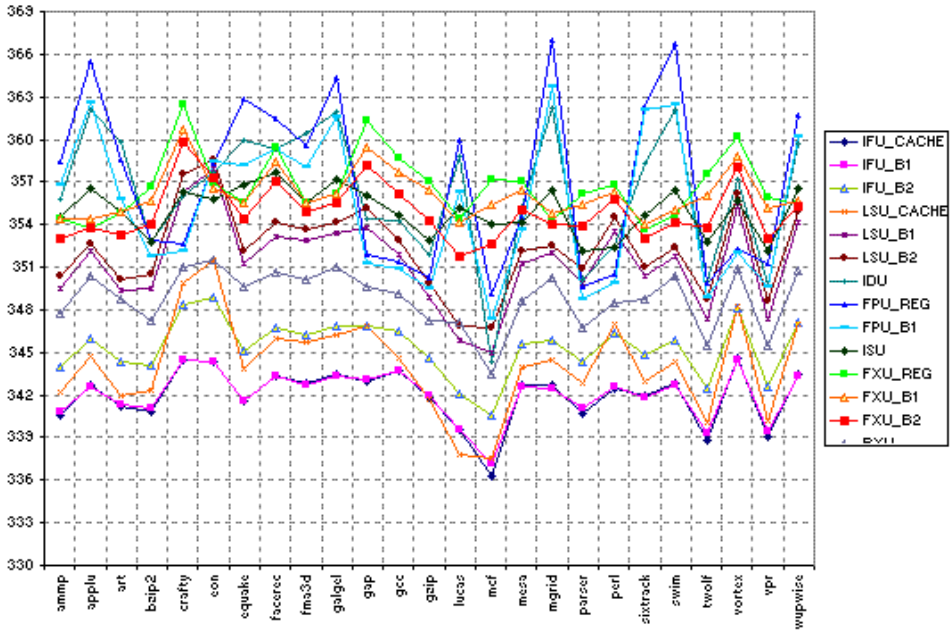

Figure 2. Thermal profile of the traces from SPEC2000 benchmark set.

The average temperature of the traces is *351.2K*. On average, the temperature profile of SPEC2000 on a Power4-like architecture is closest to *Gcc*, which has average temperature of *351K* and similarities hold in terms of heating for individual blocks as well. In general *IDU* (Instruction decode unit), *FPU_REG* (Floating Point Register File), *FXU_REG* (Fixed Point Register File), *FXU_B1* (Fixed Point Unit) are the most thermally challenging blocks. According to our experimental results, SPEC2000 has limited thermal variation within phases of individual benchmarks. The maximum deviation between maximum and average temperatures was less than 5^oC on a Power4-like architecture over the experimented traces.

5. Methodology

For our experimental analysis we used PowerPC simulator, Turandot *[10]* which utilizes HotSpot *[4]* models for temperature analysis of the Power4-like microprocessor architecture. Each microarchitectural block is represented by a node in the analogous RC network, where the nodes are connected to each other with corresponding thermal resistance and capacitances. Similarly the power dissipation information for each block is generated by Turandot's built-in energy models [11]. The leakage power dissipation is taken into consideration along with the positive feedback loop between leakage power and temperature. Then, the corresponding differential equations are solved by 4^{th} order Runge-Kutta method *[4]* during the HotSpot simulations. We assume process technology of *180 nm, 1.1 GHz* clock frequency, and *1 V* supply voltage for our experimental analysis. We used an ambient temperature of *45* ^oC, initial temperature *60* ^oC. We also used a quite conservative thermal threshold value of *85* ^oC for our thermal management techniques based on ITRS data *[8]* and packaging and cooling characteristics.

As mentioned earlier we used traces generated from SPEC2000 benchmark set *400 million* instructions. After the initial power and temperature analysis of the traces a simulated scheduler keeps track of the temperatures and initiates Turandot simulations for each time slice. Thread scheduling is based on round-robin policy for fairness, accompanied with additional temperature-aware policies. Each thread is given *10 msec* time slices before being interrupted. It is then replaced by the next thread dynamically selected by the policy based on the on-chip temperatures at the end of that particular time slice. The threads continue from the point of context switch at the next-time slice they are scheduled.

6. Temperature-Aware Scheduling Policies

We have implemented and investigated a number of task scheduling policies. We assume that our baseline architecture is equipped with thermal sensors where the block temperatures can be read at the end of each scheduling time slice. This information is passed on to the scheduler so that next thread selection can be based on the heating patterns observed in the current scheduling window (or time slice). We used the default time slice length of linux, *10msec,* for our experimental analysis.

As the execution of threads is in round-robin fashion this profiling information can be acquired dynamically at the first time slice of the scheduled task. During the next time slice, the scheduler has the estimated temperature behavior of the task and can schedule intelligently based on this information. Also, in some cases, a task can lose its time slice prematurely, if a pre-specified

temperature threshold has been exceeded. According to our experimental results on traces SPEC2000, we observed limited thermal variation within phases of individual benchmarks. For SPEC2000 traces the maximum deviation between maximum and average temperatures was less than 5^oC. This shows that the thermal profile of a thread is a good enough indicator of the thermal behavior of the thread next time it is assigned to a core. The results presented in the experimental analysis section are for initial profiles of the traces, yet the dynamic profiling results are consistent.

It is worth noting that the temperature threshold is a function of silicon thermal behavior as well as the packaging and cooling characteristics of the processor. We used a conservative thermal threshold value of 85^oC and a safety threshold value of 82^oC during our experiments on a Power4-like architecture. The rest of the temperature values and benchmark characteristics in this study are also for the same infrastructure. The temperature characteristics of the threads and architectural blocks are expected to vary for other processors, yet the scheduling results should still hold.

In this section, we discuss a number of scheduling policies and compare their effectiveness in maintaining on-chip temperatures below the pre-set temperature threshold. For the sake of comparison, we assume that all policies start with steady-state temperatures of *Mgrid*. The first benchmark selection at the start point is also set to be *Ammp* for similar reasons. Furthermore, for clarity of the display, the figures present results for the temperature behavior of the corresponding policy on a limited number of SPEC2000 runs. Notice that this also limits the policies in the number and variation of available threads; in a less restricted pool of threads the differences between policies become more prominent. We have investigated alternative cases starting with steady-state temperatures of different benchmarks, different scheduling time slices, and longer simulation durations. The results are consistent with the set we present in this section.

6.1. Random Policy

This policy is closest to a scheduling policy that is oblivious of the temperature profile of the threads as well as the heating patterns of the processor. Thread assignment is based on fairness and implemented through random number generation, each thread is given equal number of time slices. Figure 3 illustrates the temperatures for individual blocks in Kelvins, where the x-axis represents the temperature samples at every *100K* cycles. (the values can be translated to time by multiplying with *0.1msec*. Note that the x-axis marks might be appear to be slightly different for individual policies, yet this is only due to the automatic display settings of the graphics software) As mentioned earlier we assume steady-state block temperatures of *Mgrid*, and display single run of SPEC benchmarks for clarity. It is interesting to note that random benchmark selection performs fairly well, the critical block temperatures for *FPU_REG*, *FPU_B1* and *IDU* are eventually reduced. However, the temperatures are still above the threshold for over *96%* of the time.

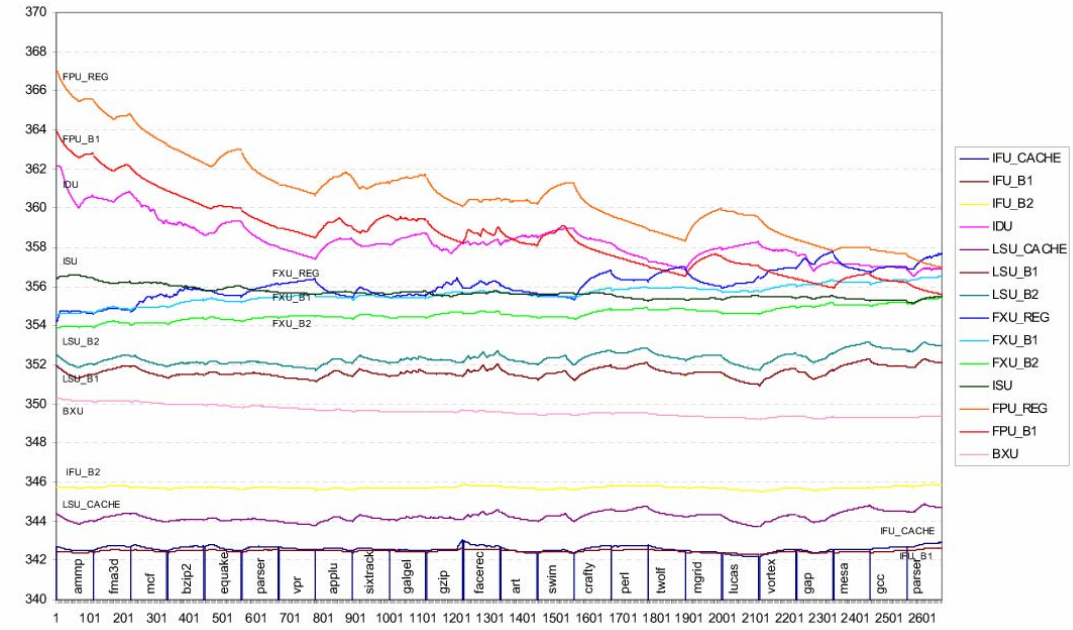

Figure 3. *Random* policy for*10msec* time slice

6.2. AvgTemp Policy

The simplest temperature-aware scheduling policy involves representing each thread with a single temperature value. In this case we represented threads with average temperature of the processor for the initial profiling. The average values are calculated as a

weighted average, where the block areas are taken into consideration. As SPEC2000 traces have limited thermal variation, the *AvgTemp* policy is expected to reduce the temperatures effectively. *AvgTemp* schedules the threads based on the average core temperature. In the case that maximum temperature threshold is exceeded a thread that has minimum average temperature is scheduled on the core. However Figure 4 shows that, although for most benchmarks average temperature profile is a good indicator, there are others such as *Lucas* and *Swim* with higher thermal variation and cause increase in the temperature of the thermally critical blocks gradually.

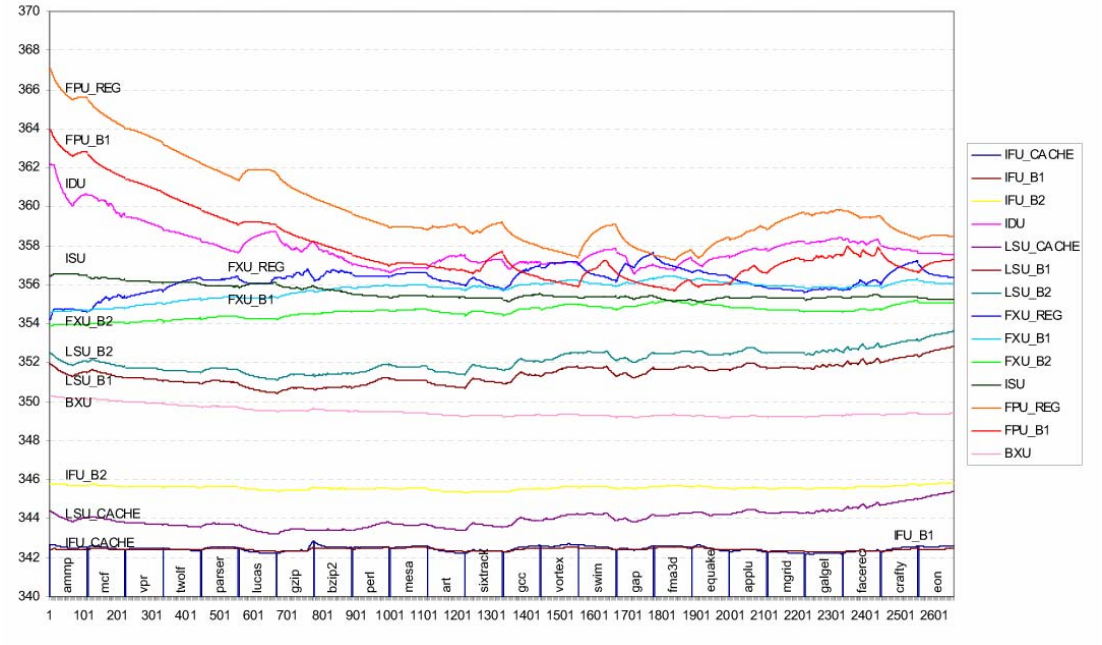

Figure 4. *AvgTemp* Policy for *10msec* time slices

6.3. MaxTemp Policy

MaxTemp policy aims to extract maximum performance from the processor; it schedules the most aggressive threads consecutively. The thermal behavior of SPEC with *MaxTemp* is displayed in Figure 5. We assume that *Mgrid* has been running for several time slices and its steady-state temperatures have been reached for the initial temperatures. The high-performance high-temperature threads keep the temperatures above the threshold temperature *98%* of the execution time. For longer execution periods the threshold is exceeded *42%* of the execution time.

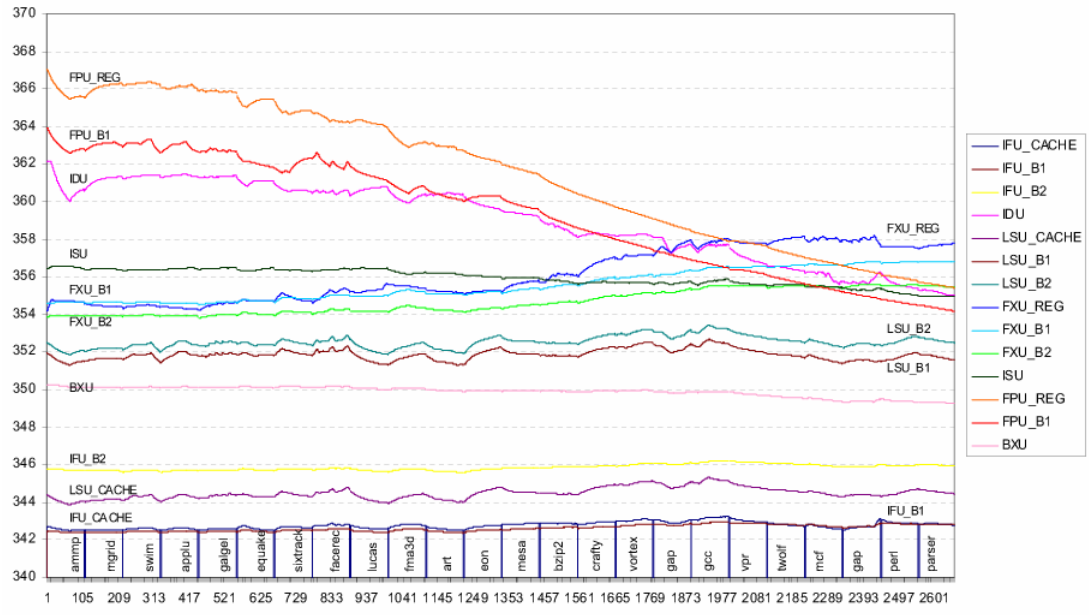

Figure 5. *MaxTemp* Policy with *10msec* operating system time slices

6.4. MinTemp Policy

MinTemp policies goal is to maintain the on-chip temperatures below the thermal threshold. It has two operation modes:

- For the case that hotspot temperatures are above the safety thermal threshold, the goal is to manage the temperature of the thermally critical blocks on the chip and avoid the performance degradation due to the dynamic thermal management schemes. In this case *MinTemp* policy selects the thread that has the minimum temperature for the current cycle's hottest block. If there are more than one hotspots with equal temperature (or threads with same profile) the tie is broken randomly or with other scheduling criteria such as fairness and deadlines.
- Otherwise if the temperatures are below the safety thermal threshold, then the schedulers goal is to balance the threads in keeping the temperatures within the boundaries, yet execute the thermally challenging threads on the currently cool chip as much as possible. Assuming that the packaging is designed so that a percentage of challenging benchmarks are handled with dynamic thermal management, we give priority in servicing these challenging traces during this cooler operation mode.

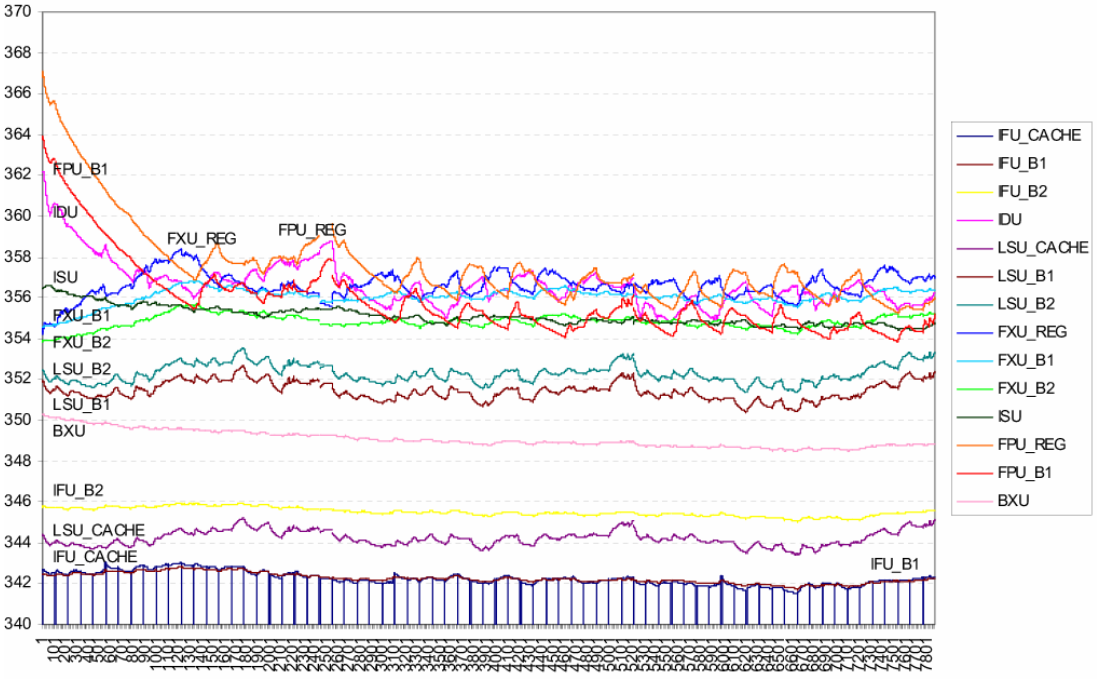

Figure 6. *MinTemp* Policy with 10msec time slices

Figure 6 demonstrates the execution of *MinTemp* policy in consecutive round-robin scheme. Notice that this figure has longer execution duration just to illustrate the long-term performance of *MinTemp*. Although the cooling times appear shorter than other traces, they are consistently in *100 msec* range. It is worth noting that, after *130msec* for the heated blocks, which is around the RC time constant necessary for the initial cooling, *MinTemp* is effective at keeping the maximum temperature below thermal threshold. The zigzag patterns in the figure illustrate that *MinTemp* interleaves the thermally challenging threads whenever the on-chip temperatures are below safety threshold. Figure 7 shows the thermal behavior of *MinTemp* policy with the initial temperatures for *Gcc,* which has a moderate thermal profile. Similar observations are valid for this case and when consecutive round-robin runs were executed: the maximum on-chip temperatures are below thermal threshold more than *99%* of the execution time.

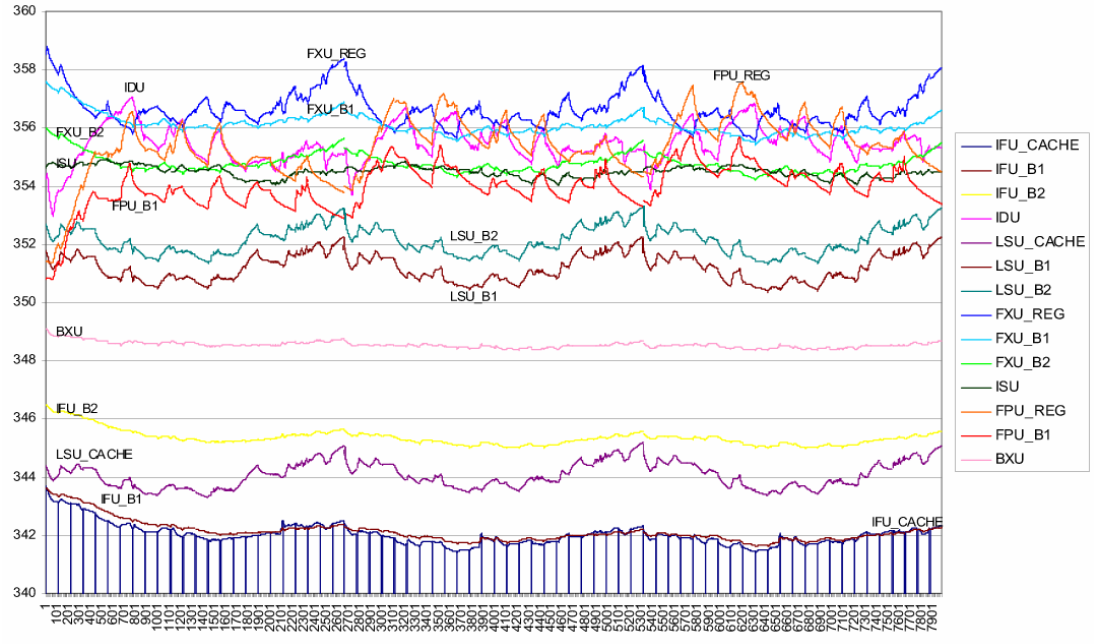

Figure 7 *MinTemp* Policy, Round-Robin, *10msec* operating system time slice (*Gcc* steady state temperatures)

6.5. Fetch Throttling

Current microprocessor architectures have reportedly integrated a linear throttle mechanism, stopping the CPU instruction fetch stage for short periods of time in order to allow the processor to cool. A prior-generation PowerPC processor also used such fetch-throttling to conserve power when needed *[2]*. Other related techniques such as: Decode throttling varies the number of instructions that can be decoded per cycle *[3]*. We compared the thermal behavior of temperature-aware operating system scheduling policies with fetch throttling.

For the sake of comparison we used the same thread execution sequence that was used by *MinTemp* and throttle fetch above the temperature threshold value of *358K*. Notice that this is quite optimistic, since this sequence was carefully selected by *MinTemp* policy for best thermal behavior, which is not the case for a thermally-oblivious policy coupled with fetch-throttling. We selected a coarse-grained aggressive fetch throttling policy in order to effectively reduce the temperatures below the thermal threshold. Figure 8 illustrates the effect of fetch throttling after *Mgrid's* steady state temperature has been reached. The initial benchmark sequence until *Lucas* is not executed due to the coarse grained fetch throttling. After the sequence completes we continue with the initial benchmark sequence that was throttled (*Ammp, Mgrid, Swim, Applu, Galgel, Equake, Sixtrack* and *Facerec*). For this specific scenario, the IPC is *0.808* compared to *1.02* for the analogous case in *MinTemp* policy. The corresponding *20%* IPC degradation is due to the halt during the thermal time constant as opposed to continuing the execution with complementary benchmarks in the case of *MinTemp*.

Notice that this degradation is for the minimum temperature execution stream that was previously selected by *MinTemp*. When the experiment is repeated with *MaxTemp* execution stream accompanied with fetch throttling the performance degradation is significantly higher. For the rest of the experimental analysis we used existing scheduling criteria such as fairness for fetch throttling, not *MinTemp* execution stream.Although *20%* degradation is the upper bound as this particular figure indicates is a single run of SPEC, we have observed around *15%* performance degradation for longer runs of fetch-throttling. The performance degradation depends on the baseline scheduling policy as well as severity of thermal problems. In general for thermally challenging benchmark sets the degradation can be quite severe.

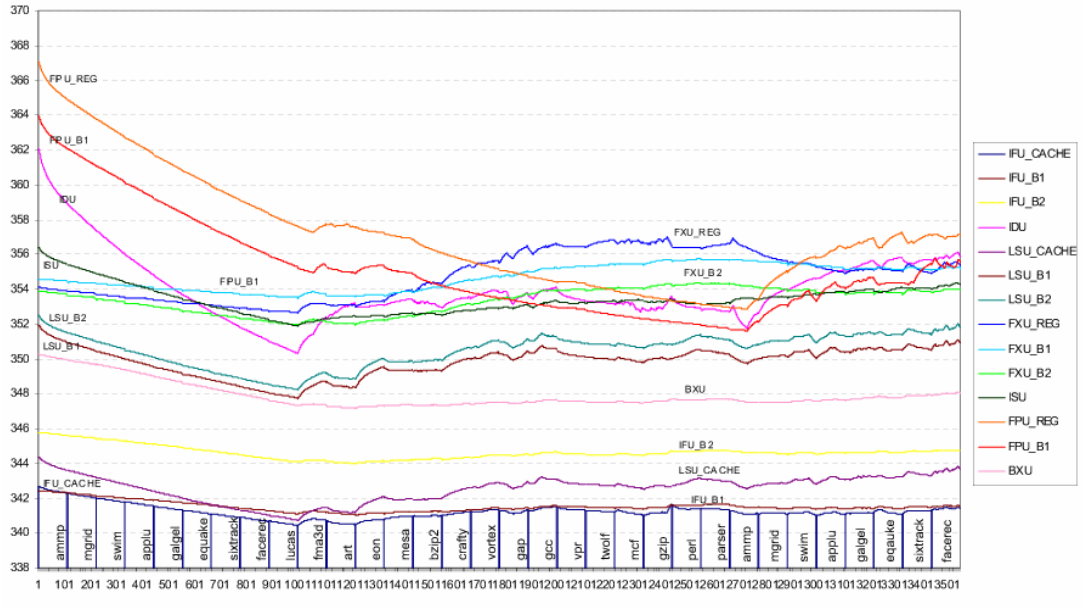

Figure 8. Fetch throttling above *358K*

6.6. Effects of Varying Scheduling Time Slices

In some operating systems such as linux, operating system time slice length can be adjusted in the orders of baseline time-slice. We experimented on various scheduling time slices from *10 - 50 msec*. The conclusions for the thermal-aware scheduling policies still hold for the different time slices. The only difference we observed was that the heating and cooling behavior is stronger for the longer scheduling slices, as the thermally critical benchmarks have longer time to heat the processor blocks.

7. Conclusions and Future Work

Today's microprocessor architectures present unique thermal challenges. We investigated the effectiveness of temperature-aware thread scheduling as a thermal management technique. Since system thread scheduling is already incorporated in the current microprocessors execution, thermal-awareness can be incorporated with virtually no performance degradation. Most of the architectures in the market today employ packaging systems designed for less than worst-case temperature, with the assumption that threads with absolute worst-case thermal behavior are of limited percentage in the distribution and can be dealt dynamically with hardware-level reactive dynamic thermal management techniques.

We investigated the processor heating/cooling behavior and showed that the RC time constant that determines the heating/cooling speed is in the range of *100 msec* for the baseline Power4-like architecture. The range of time constants enables effectively utilizing the thread scheduling infrastructure to manage the on-chip heating. We experimented on traces generated from SPEC2000 benchmark set where *12* of the *25* benchmarks have maximum block temperatures above *360K*, which is higher than our conservative temperature threshold of *358K*. Having *40%* of the benchmarks in the thermally critical range is quite aggressive, compared the current reported packaging assumptions with only *20%* benchmarks are above the threshold values.

Even with a thermally challenging benchmark set, our *MinTemp* policy is effective in reducing the temperatures from a high initial temperature such as *Mgrid* steady-state temperature. *MinTemp* manages to keep the temperatures lower than the thermal threshold *99%* of the execution time for cases with average initial temperature values (such as *Gcc*) over longer periods of execution. The reason *MinTemp* is effective in reducing the temperature is the diversity in the location of hotspots and the effectiveness of the policy in interleaving the challenging threads. Furthermore, *MinTemp* adapts to the processor heating behavior through its distinct operation modes: For thermally critical cycles it aims to reduce the maximum block temperature through scheduling threads with lower temperature profiles for the corresponding hotspot block. For the non-critical cycles, the goal is to execute the thermally critical traces with higher priority, while the temperatures are below the safety threshold.

Our experimental analysis indicates that scheduling selection can result in temperature differences around $5^{o}C$ on average, between different benchmarks for the hotspots. Hardware based techniques such as fetch throttling is also quite effective in reducing the maximum temperatures below the thermal threshold. However, the corresponding performance degradation can be up to *20%* for a single run of SPEC2000 even over the execution sequence identical to *MinTemp* policy. For longer durations the degradation can still be significant depending on the severity of heating problems. *Random* thread selection and *AvgTemp* policies are comparable

effectiveness to each other, which indicates the importance of localized heating information for individual processor blocks and threads. As the localized heating is not taken into consideration for *AvgTemp*, the thermal behavior is not improved drammatically over a thermally-oblivious policy such as *Random*.

Figure 9 displays the percentage of cycles above threshold for the investigated policies. It is important to notice that for conservative thresholds such as *364K* none of the policies provide sufficient thermal alleviation. Also, notice that fetch throttling and *MinTemp* are the most effective in temperature reduction for slightly higher thermal thresholds. Fetch throttling has significant performance degradation for lower thresholds. *MinTemp* displays the percentage of cycles above the thresholds for steady-state behavior of the policy and shows that the temperatures are effectively reduced for thermal thresholds of *358K* and above. For lower temperature thresholds, the number of cycles above threshold increases dramatically, which implies that other hardware based DTM techniques have to be activated for lower thresholds. This fact is also consistent with having a hierarchical activation of various DTM techniques for effective temperature management.

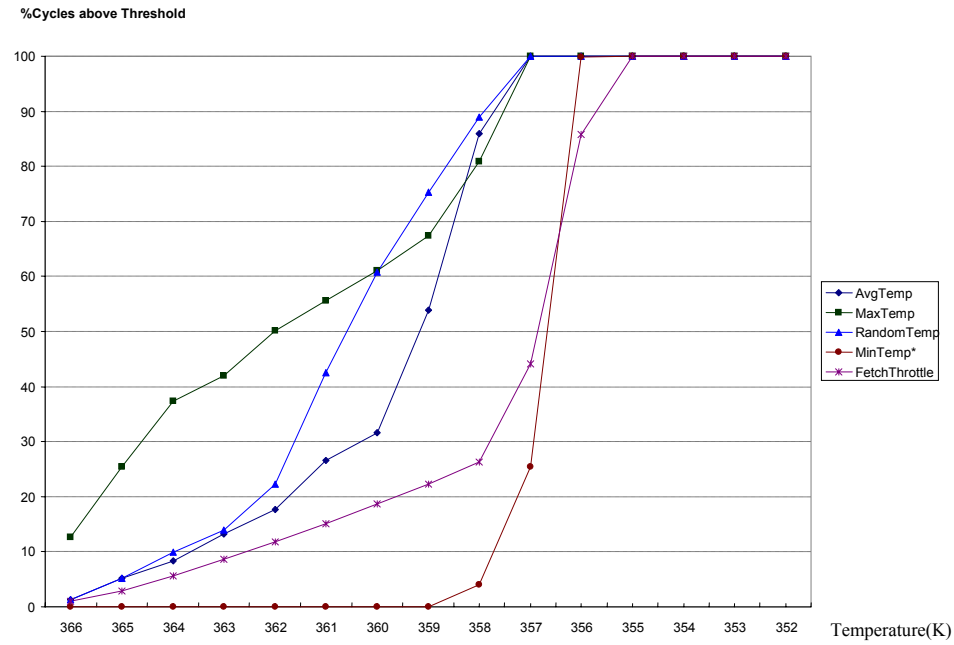

Figure 9. Percentage of cycles above the thermal threshold for different thermal management policies and threshold values

Our preliminary results in this study, indicate that thermal-aware operating system task scheduling can be effectively used to manage the on-chip temperatures. Our current and future work include implementation of these policies in the operating system kernel on a multi-core processor. For CMP architectures utilizing system and software thermal management techniques along with hardware DTMs enhances the performance of thermal management. The increased number and diversity of cores and threads enable CMP architectures to be good candidates for effective task scheduling policies.

References

[1] S. Borkar, "Design Challenges of Technology Scaling", *IEEE MICRO*, pp. 23-29, August 1999.

[2] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, J.Alvarez, "Thermal Management System for High-Performance PowerPC Microprocessors", *Proceedings of IEEE International Computer Conference*, pp.325, 1997.

[3] D. Brooks, M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors", *International Symposium on High-Performance Computer Architectures*, pp. 171-182. Jan 2001.

[4]K.Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, "Temperature-Aware Microarchitecture", *Proceedings of 30th International Symposium on Computer Architecture*, pp.2-13, June 2003.

[5] S. Gunther, F. Binns, D.M. Carmean, J.C. Hall, "Managing the Impact of Increasing Microprocessor Power Consumption", *Intel Technology Journal*, 2001.

[6] J. Moore, J. Chase, P. Ranganathan, R. Sharma, "Making Scheduling 'Cool': Temperature-Aware Workload Placement in Data Centers", *USENIX Annual Technical Conference*, April 2005.

[7] L.T. Yeh, R. Chy, "Thermal Management of Microelectronic Equipment", *American Society of Mechanical Engineers*, *(ISBN 0791801683),* 2001.

[8] SIA. *International Road Map for Semiconductors*, 2001.

[9] E. Kursun, G. Reinman, S. Sair, A. Shayesteh, T. Sherwood, "Low-Overhead Core Swapping for Thermal Management", *Power-Aware Computer Systems Workshop MICRO*, Dec 2004.
[10] M. Moudgill, J-D. Wellman, and J. Moreno, "Environment for PowerPC Microarchitecture Exploration," *IEEE Micro***,** Vol. 19, No. 3, pp. 15–25 (May/June 1999) (see also *http://www.research.ibm.com/MET/*.)
[11] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New Methodology for Early-Stage, Microarchitecture-Level Power Performance Analysis of Microprocessors," *IBM Journal of Research and Development,* vol. 47, no. 5/6, 2003.
[12] S. Heo, K. Barr, K. Asanovic, "Reducing Power Density through Activity Migration", *International Symposium on Low Power Electronics and Design*, pp.217-222, 2003.
[13] M. Gomaa, M.D. Powell and T.N. Vijaykumar,"Heat-and-Run: Leveraging SMT and CMP to Manage Power Density through Operating System", *Proceedings of 11th International Conference on Architectural Support for Programming Languages and Operating Systems,* pp.260-270, 2004.
[14] A. Shayesteh, E. Kursun, T. Sherwood, S. Sair, G. Reinman, "Reducing the Latency and Area Cost of Core Swapping Through Helper Engines", *International Conference on Computer Design,* pp.17-23, 2005
[15]S. Ghiasi, D. Grunwald, "Thermal Management with Asymmetric Dual-Core Designs", *University of Colorado Technical Report CU-CS-965-03,* 2004.
[16] J. Srinivasan and S. V. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications", *In Proceedings of International Conference on Supercomputing*, June 2003.
[17] A. Weissel and F. Bellosa, "Dynamic Thermal Management for Distributed Systems", *In Proceedings of the First Workshop on Temperature-Aware Computer Systems*, June 2004