RC23841 (W0601-026) January 6, 2006
Computer Science

IBM Research Report

Temperature-Aware Operating System Scheduling Policies for CMP Architectures

Eren Kursun, Chen-Yong-Cher, Alper Buyuktosunoglu, Pradip Bose
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

TEMPERATURE-AWARE OPERATING SYSTEM SCHEDULING POLICIES FOR CMP ARCHITECTURES

Eren Kursun[1], Chen-Yong-Cher, Alper Buyuktosunoglu, Pradip Bose
IBM T. J. Watson Research Center

ABSTRACT- Thermal characteristics of modern microprocessors have presented numerous challenges in recent years. As the technology trends in power dissipation, feature scaling and clock frequencies continue, thermal behavior is expected to be a vital consideration in the next generation of microprocessor architecture design. There has been a wide range of dynamic thermal management (DTM) techniques proposed for reducing and managing on-chip temperatures. These schemes include: scaling the supply voltage, frequency and/or fetch rate, or even shutting down the clock signal to the processor. However, most of the proposed DTM techniques are based on the principle of "reactive throttling": i.e. some form of performance throttling is invoked, after a pre-architected temperature threshold has been exceeded. As such, there is performance degradation with each such DTM technique; and, in some cases, if the temperature threshold is set to a relatively low level to conserve package/cooling cost, the degradation can be quite severe. In this paper, we investigate the potential benefits of thermal-aware operating system scheduling for chip multiprocessing architectures. The operating system is already designed to interrupt running jobs in accordance with time slice and scheduling parameters, as well as workload characteristics (e.g. i/o and memory behavior). The intuition motivating this work is that adding thermal-awareness to the scheduling heuristics will enable us to achieve chip-level temperature reduction, without adding any extra performance overhead (unlike hardware-based DTM control mechanisms). We explored various operating system policies and their effect on temperature behavior of the processor. We developed a temperature estimation scheme for preliminary analysis of policies and eventually verified the policies with the Turandot/PowerTimer simulator on traces extracted from SPEC2000 Benchmark Set. Our results indicate that with thermal-aware O.S. scheduling, hotspot temperatures can be reduced significantly. We observed a 69.2% reduction in the number of thermally critical cycles compared to the worst case thermal scheduling and 52% reduction compared to random scheduling. On average, our* **MinTemp** *Scheduling policy yields less than 3% cycles in thermal violation. There was no appreciable change in net performance, compared to the baseline, temperature-unaware OS schedule.

1. INTRODUCTION

The continued demand for higher performance processors has led to a rapid increase in power dissipation. In fact, power density has doubled every *3* years *[Borkar1999]*. The resulting increase in on-chip temperatures have started to present significant challenges in microprocessor design. As such, on-chip thermal profiles have become a key design constraint, even as multi-core, lower frequency chips form the basis of new generation, power-efficient systems.

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

Temperature characteristics of microprocessors affect vital aspects: such as timing, reliability as well as increased packaging and cooling costs. Reliability of an electronic circuit is exponentially dependent on the junction temperature. A *10-15^oC* rise in the operating temperature results *~2X* reduction in the lifespan of the circuit *[Yeh2001]*. Due to the temperature dependence of carrier mobility, effective operating speed decreases at higher temperatures.

Leakage power has exponential dependency on temperature. In fact, there is a positive feedback loop, wherein an increase of temperature causes leakage power to rise, which causes a further increase in temperature, and so on. This phenomenon, also called "Thermal Runaway" can cause serious potential damage if not carefully controlled, through a proper design of the package and related cooling solution. Leakage power is expected to constitute as high as *50%* of the total power consumption *[ITRS2001]* in post-90nm CMOS technologies.

A wide range of packaging and cooling techniques has been proposed to alleviate thermal problems. However, cooling/packaging costs increase at a super-linear rate *(>$3* per Watt dissipated after *60^oC*, according to an older publication from Intel *[Gunther2001]*), and package impedances are known to be saturating to values that are going to be very hard to decrease cost-effectively in the future. Due to the increases in transistor count and clock frequencies, 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, not to handle the absolute worst power/thermal behavior, but a safe threshold below it. For instance, Intel's Pentium 4 is known to use packaging that is designed to handle *80%* of the worst-case power dissipation. Beyond the *80%* threshold, the very "hot" (but infrequent) workloads are handled with global clock throttling *[Gunther2001]*.

1.1. Dynamic Thermal Management Techniques

As mentioned earlier, DTM response mechanisms that are currently in use in one or more commercial microprocessors are as described below in additional detail:

Global Clock Throttling: As the temperature exceeds the thermal threshold the clock signal to the bulk of the microprocessor is shut down. However, such global throttling causes very significant performance degradation.

Fetch Throttling: Some Intel CPUs have reportedly integrated a linear throttle mechanism, stopping the CPU instruction fetch mechanism for short periods of time and allowing it to cool. A prior-generation PowerPC processor also used such fetch-throttling to conserve power, when needed *[Sanchez1997]*. (Other related techniques such as: "Decode throttling" varies the number of instructions that can be decoded per cycle *[Brooks2001]*)

Dynamic Voltage and Frequency Scaling (DVFS): This provides power savings that follows the cubic rule (with regard to supply voltage). In other words, it can result in 3% reduction in active power, for only about 1% loss in performance (resulting from a 1% reduction in voltage and frequency). Thus, it is generally much more efficient than plain dynamic frequency scaling or clock throttling, which tends to follow a linear rule (i.e., 1% reduction in power for 1% loss in performance). However, due to the resynchronization of clocks, DVFS schemes may have to stall *10-50μsec* each time the voltage is

adjusted, so the performance loss can be much more pronounced than that dictated by an ideal DVFS mechanism.

In summary, all of the hardware-based DTM response mechanisms currently in use, generally imply a significant loss of performance at the chip-level, in order to meet affordable peak temperature limits. As such, there is a need to investigate the use of alternate hardware-software hybrid schemes towards the goal of improved thermal efficiency: i.e. achieving significant temperature reduction without noticeable performance loss. In the section below, we further motivate the general concepts behind our research.

2. MOTIVATION

It is important to note that the aforementioned DTM techniques are designed to work on the principle of "reactive throttling" and as such can cause performance degradation of various degrees. Chip multiprocessing (CMP) architectures provide unique challenges and opportunities in thermal management. Independently managed DTM techniques for each individual core are quite difficult to justify in a multi-core or CMP scenario, since a global view of power and temperature would be missing in such a scheme. A globally-coordinated DTM controller, that is able to swap jobs executing on multiple cores, in response to localized temperature overruns, has more promise. Such "activity migration" strategies *[Kursun2004]* have been proposed in the past. However, implementing such a scheme entirely in hardware is likely to introduce significant additional hardware complexity and area overhead, since process context swaps have to be managed entirely in hardware. Instead, if one can leverage the existing context swap infrastructure supported by the system software (e.g. the operating system and hypervisor layers) and implement a temperature-aware task scheduling heuristic, it may be possible to achieve the goal of hot spot mitigation, without additional on-chip hardware complexity. This is the basic motivation of the work reported here.

2.1. Illustrative Case

Figure 1 illustrates a CMP scenario with 4 cores and a number of threads, where the operating system is responsible for scheduling the next thread to each core every time the current time slice is exhausted. Note that in operating systems such as Linux the time slice is configurable, however in AIX this is fixed to *10ms*ec. Conceptually, we assume the presence of a monitoring facility called the GMU (global power-performance-temperature monitoring unit), that collects thermal profile information for the threads (tasks) in the queue. This information is passed on to the OS-level. As each benchmark is divided and executed in 10msec operating system time slices (usually in a round robin fashion) this profiling information can be acquired at run time at the first time slice of the scheduled task. During the next time slice, the operating system 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 schedule has been exceeded.

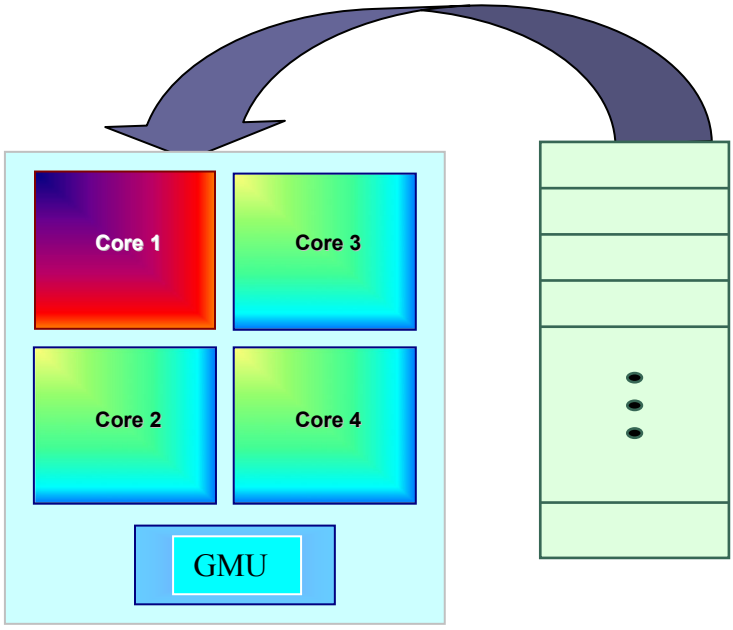

Figure1. 4-core CMP with Thermal-aware O.S. Scheduling

As Core 1 in the 4-core CMP heats up to the temperature threshold of 358K (*85^{o}C)*, the operating system can select the next thread from the thread queue based on the temperature profiles of the threads in queue. Let us assume that a thread from benchmark *Mgrid* was running on Core 1. [*Mgrid* has *367K* maximum temperature and causes *FPU_REG* unit (floating point register file) to exceed the threshold].

Scenario 1: Operating System (O.S) has a variety of threads to select from in the queue. One possibility is to select oblivious of the temperature and schedule *Applu* on the same core which as a steady state temperature of *365K* for *FPU_REG*. This in turn will increase the core temperature even above *358K* and a hardware DTM technique such as fetch throttling (or clock throttling, etc.) has to be activated for Core 1 to cool down below a safe threshold.

Scenario 2: O.S can select a low-temperature benchmark such as *Mcf*, which has the lowest thermal profile of the traces we experimented on. In this case, Core 1 temperature will decrease and no DTM will be needed.

Scenario 3: Alternatively, the O.S can select yet another high-temperature benchmark such as *Crafty*, which reaches *363K* for *FXU_REG*. However, since the heating patterns are complementary, the *FPU_REG* file can cool down without degrading the performance with a hardware DTM technique.

NOTE: Figure 2 displays the difference between the maximum and average temperature for threads extracted from SPEC 2000 benchmark set. Notice that the maximum deviation between maximum and average temperatures is less than *2.5-3^{o}C*. This in fact 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. For our experimental analysis we used static thermal profiles.

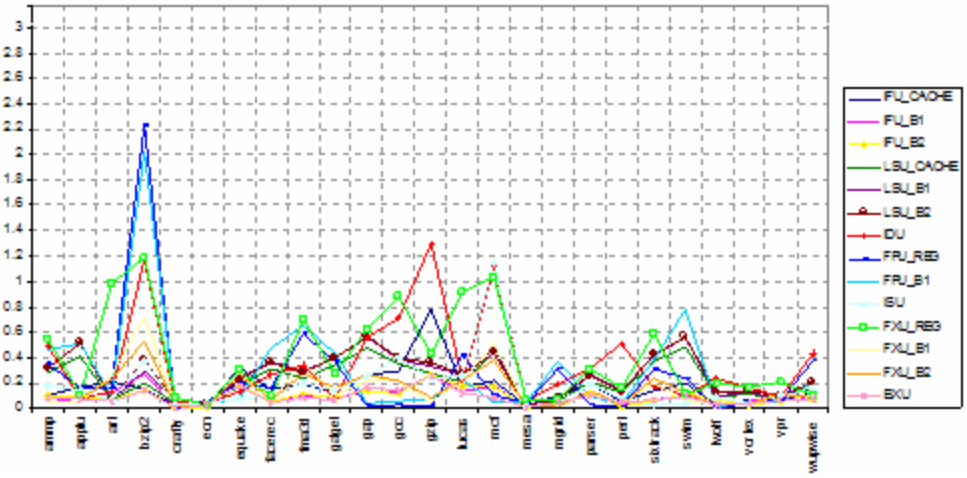

Figure 2. Difference between maximum and average temperatures of architectural blocks for threads extracted from SPEC2000

3. PRELIMINARIES

3. 1. Temperature Modeling Basics

Temperature modeling for microprocessors is based on the duality of the electrical and thermal phenomena in Table 1.

THERMAL	**ELECTRICAL**
Heat Flow (W)	Current Flow (A)
Temperature Difference (K)	Voltage Difference (Volts)
Thermal Capacitance (C_T)	Electrical Capacitance (C_E)
Thermal Resistance (R_T)	Electrical Resistance (R_E)
Thermal Time Constant: $\tau = R_T.C_T$ (sec)	Electrical Time Constant: $\tau = R_E.C_E$ (sec)

Table 1. Analogy between Thermal and Electrical parameters

Temperature behaves according to the exponential RC curves represented by the time constant. In electrical phenomena, RC time constant is defined, as the time required to charge a capacitor to *63.2%* of the final value; or equivalently, the time required to discharge the capacitor to *36.8%* of the initial value. Thermal transient behavior of an architectural block can be similarly estimated with the corresponding thermal time constant. The time constant for an architectural block (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). It is independent of the power dissipation. A thermal model (derived from University of Virginia's HotSpot *[Skadron2003]* built on top of the energy models within the Turandot/PowerTimer simulator *[Moudgill1999, Brooks2003]* was used to estimate the temperature characteristics in our experimental analysis.

Figure 3 displays the exponential decay curve for benchmark *Mcf*, when the active power dissipation is reduced to *0* Watts, as the benchmark stops executing.

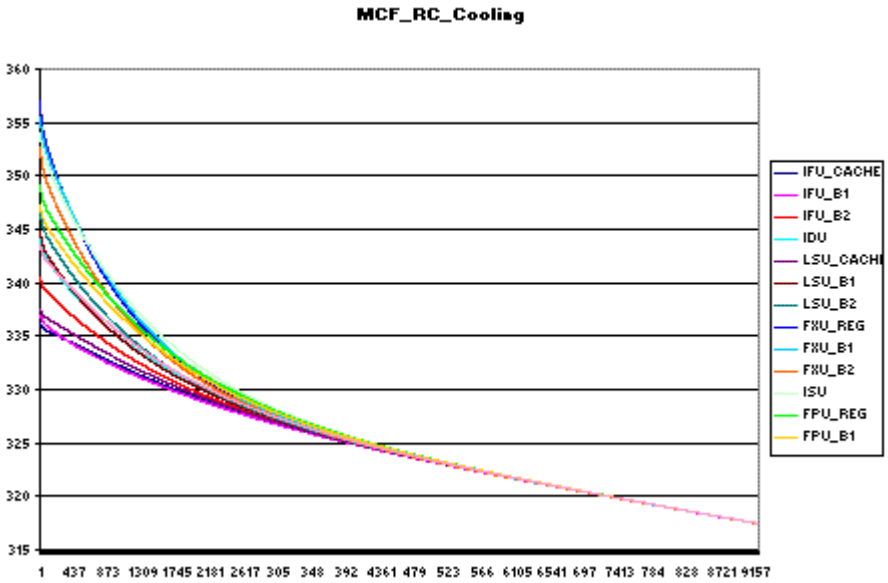

Figure 3. Exponential Temperature decay curve for Mcf (x axis represents temperature samples (each sample is taken at *100K* cycles hence RC time constants are in *100msec* range)

Figure 4 shows the time constant values for *5W*, *50W* and *500W* constant power dissipation for each block extracted using the Turandot/PowerTimer simulator. As the figure illustrates 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 *80msec* to *200msec* for various architectural blocks. These numbers are then used in temperature estimation models for each block in a standalone Temperature and Power Analysis Tool *(TPAT),* as described in section 4.

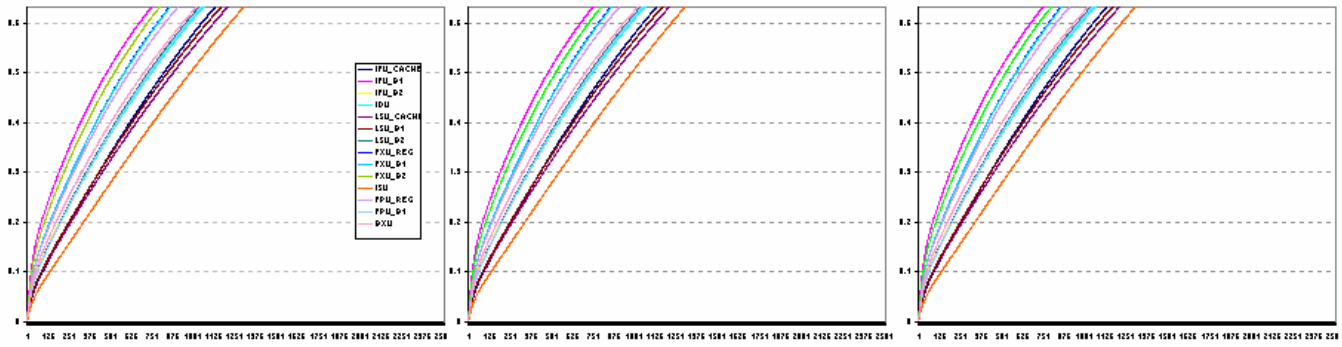

Figure 4. Time constant values for *5W*, *50W* and *500W* power dissipation for different architectural blocks. (*x-axis* represents temperature samples and *y-axis* is the normalized temperature values where the upper division is *0.632* of the final value i.e. R.C. time constant)

3.2. Turandot Thermal Simulation

Our baseline PowerPC simulator, Turandot *[Moudgill1999]* utilizes HotSpot *[Skadron2003]* models for temperature analysis of 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 power density for each block is generated by Turandot's built-in energy models [*Brooks2003]* and corresponds to the current sources in the

analogous electrical circuit. Then, the corresponding differential equations are solved by 4th order Runge-Kutta method *[Skadron2003]*.

4. TEMPERATURE ESTIMATION USING TPAT

We model the temperature of a block with approximate RC curves. Our thermal sampling interval is *10msec* and we are trying to estimate the potential temperature for each possible thread we can schedule on a core.

We have investigated various parameters for temperature estimation. Our experimental analysis shows that the correlation between *Power Density* and the next scheduled time slice temperature is *0.25,* which is consistent with previous experimental results by Skadron et al. *[Skadron2003]*. The correlation value with *ΔPower* (Change in Power Dissipation) is *0.32* and finally correlation of temperature with current power is *0.41*. In summary, a simpler metric that is based on solely power dissipation in the previous cycle seems to be a better choice:

$$T_i = \begin{cases} if\ P_{wi} << P_{i-1} & T_{i-1}e^{-\Delta t/RC} \\ if\ P_i \sim P_{i-1} & T_{i-1} + \Delta T_{i-1} \\ if\ P_{wi} >> P_{i-1} & T_{i-1}(1-e^{-\Delta t/RC}) \end{cases}$$

If the power dissipation at scheduled time slice *i* is significantly larger than the previous slice, the temperature change is estimated with RC saturation curve based on the RC time constant for the block. Power dissipation differences larger than *20%* gives *<1C* temperature estimation error. In the case that power dissipation is less than *20%* different compared to the previous time slice (OS schedule), temperature is estimated based on the temperature change in the previous OS scheduled slice. Similarly the drop in power dissipation is estimated using the exponential decay curve.

In general, temperature change acts as a low pass filter on the power dissipation of the architectural block. The smaller finer-grain fluctuations do not affect the overall temperature of the block. To account for this effect we use a weighted average of the power dissipation of the previous *3* thermal sampling intervals. Each thermal sampling interval is *100K*. The correlation of the temperature values estimated by *TPAT* and *Turandot* simulations are displayed in Figure 5. SPEC2000 benchmark set is used for the simulations.

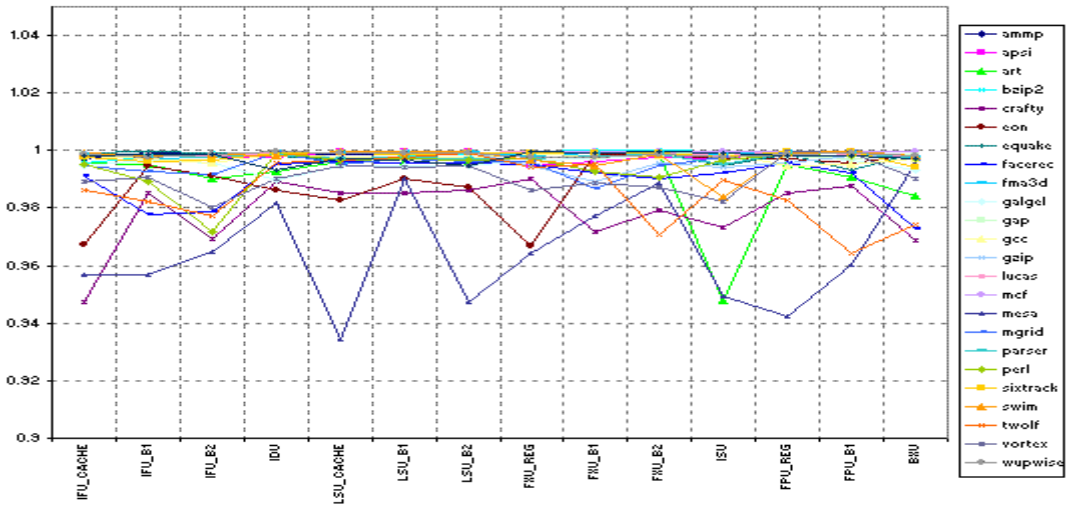

Figure 5. Correlation between the estimated temperatures by *TPAT* and *Turandot* simulations for architectural blocks over SPEC

5. METHODOLOGY

We assume process technology *180nm,* clock frequency *1.1GHz*, supply voltage *1V* with Power4-like architecture for our experimental analysis. Ambient temperature: 45^oC, initial temperature: 60^oC, thermal threshold for DTM (85^oC) based on ITRS data *[ITRS2001]*.

We used traces generated from SPEC2000 Benchmark set (400 million instructions were used). After the initial power and temperature analysis of the traces, static Thermal and Power Analysis Tool (TPAT) analyzes the trace data and generates the estimated temperatures. The initial analysis of the thermal O.S. scheduling policies is done with TPAT; this step is followed by the actual validation with the Turandot simulations.

6. SPEC 2000 TEMPERATURE PROFILE

Initial thermal analysis of the traces from SPEC2000 Benchmark set was done with Turandot thermal simulator. The average and maximum temperatures are displayed in Figure 6. Among the *25* benchmarks, *10* of them have blocks with temperatures above the *360K*, to keep the temperatures below *358K* temperature threshold is a challenging task in such a pool of threads.

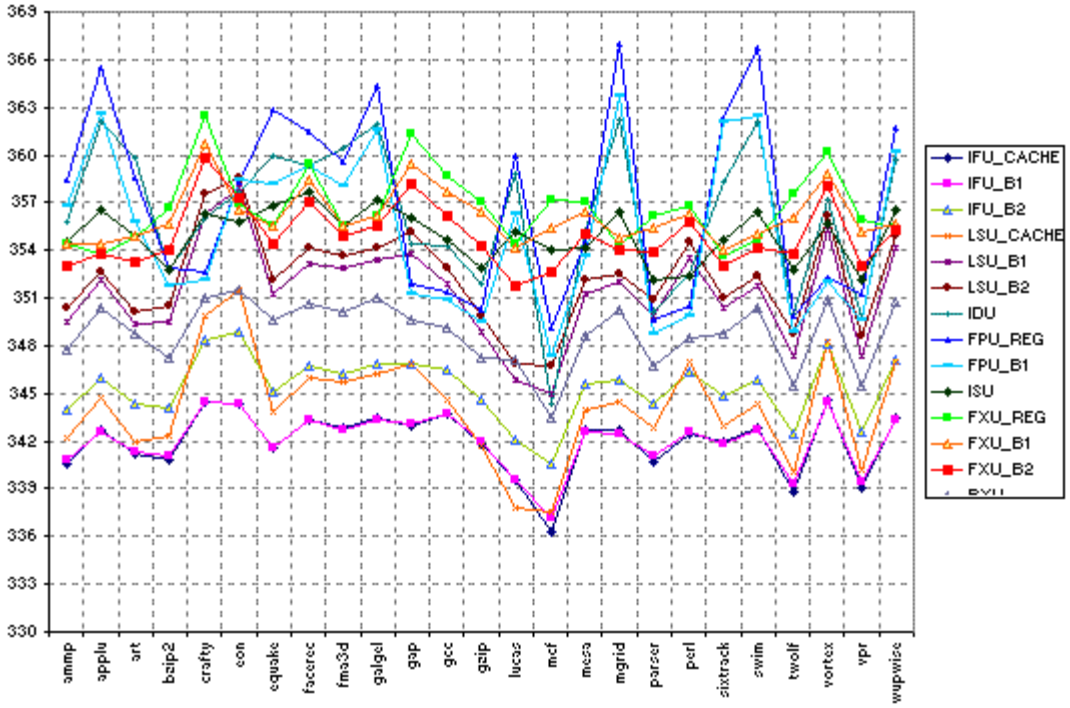

Figure 6. Thermal profile of the traces from SPEC2000 benchmark set.

The average temperatures of the traces are shown in Figure 7. (Average temperature is *351.2K*) On average the temperature profile of SPEC is closest to *Gcc*, which has average temperature of *351.1K* and similarities in terms of heating for individual blocks as well.

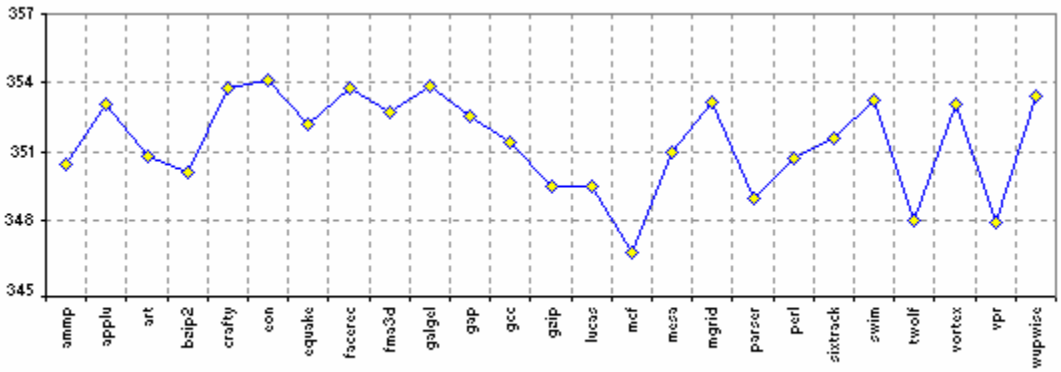

Figure 7. Average temperature of the traces from SPEC2000

Figure 8 illustrates the average temperatures of the architectural blocks over SPEC trace runs. 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 thermally challenging blocks.

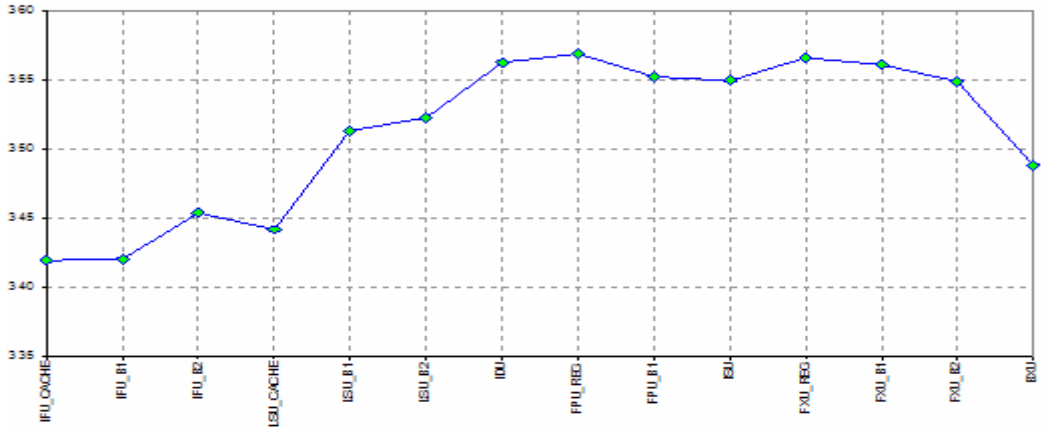

Figure 8. Average temperatures of architectural blocks over the experimental traces

7. THERMAL-AWARE OPERATING SYSTEM POLICIES

We have implemented and investigated a number of operating system task scheduling policies. We assume that our baseline architecture is equipped with thermal sensors (monitored by the GMU) and at the end of the O.S. time slice the temperatures are read. This information is passed on to the operating system so that the next thread selection can be based on the heating patterns observed in the current scheduling window (or time slice).

8. EXPERIMENTAL RESULTS

8.1. Random Policy

In this experiment, we used random numbers for next benchmark selection. This policy is closest to an operating system policy that is oblivious to temperature information. It is interesting to note that with random benchmark selection temperatures are eventually reduced. For the given execution time frame for over *96%* of the time, the temperatures are above the thermal threshold. The x-axis represents the temperature samples at every *100K* cycles (the values can be translated to time by multiplying with **0.1msec*) and y-axis represent the individual block temperatures in Kelvins.

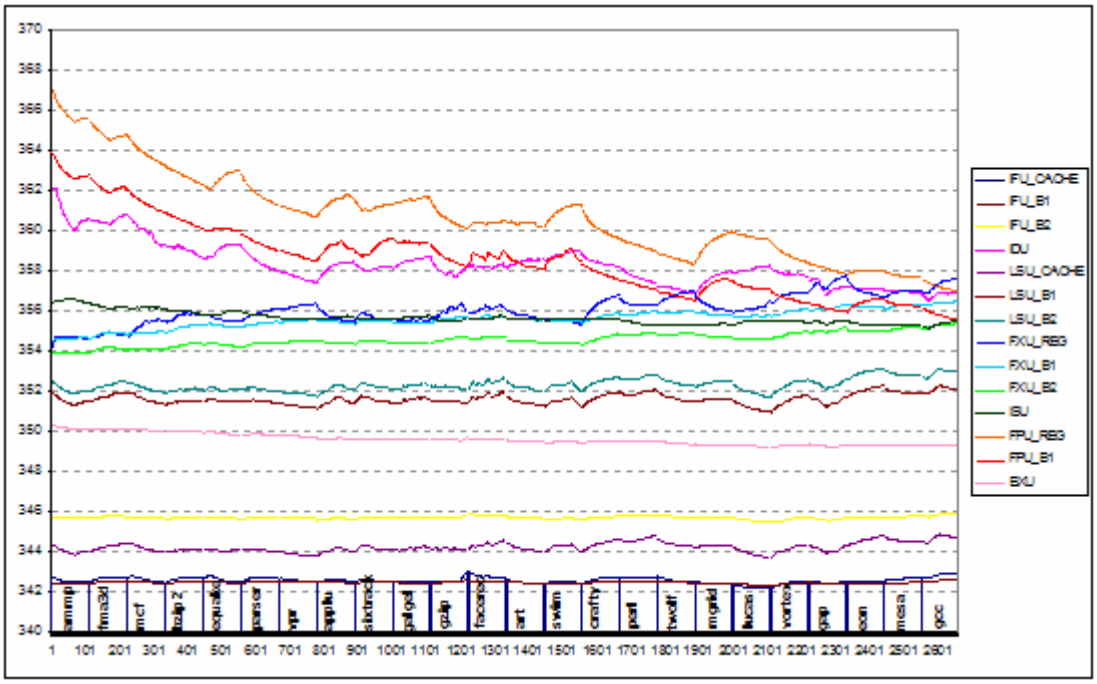

Figure 9. *Random* policy for*10msec* time slice

8.2. AvgTemp Policy

The simplest temperature-aware scheduling policy involves representing each thread with a single temperature value. We used average temperature of all the blocks for each benchmark for the AvgTemp policy. It is important to note that SPEC2000 benchmark thermal profile gives us maximum *2.2K* degrees deviation between average and maximum core temperatures. Hence, average temperature based scheduling policy manages to reduce the temperatures. However, Figure 10 shows that although some benchmarks with relatively low average temperature such as *Lucas* cause increase in the temperature of the thermally critical blocks.

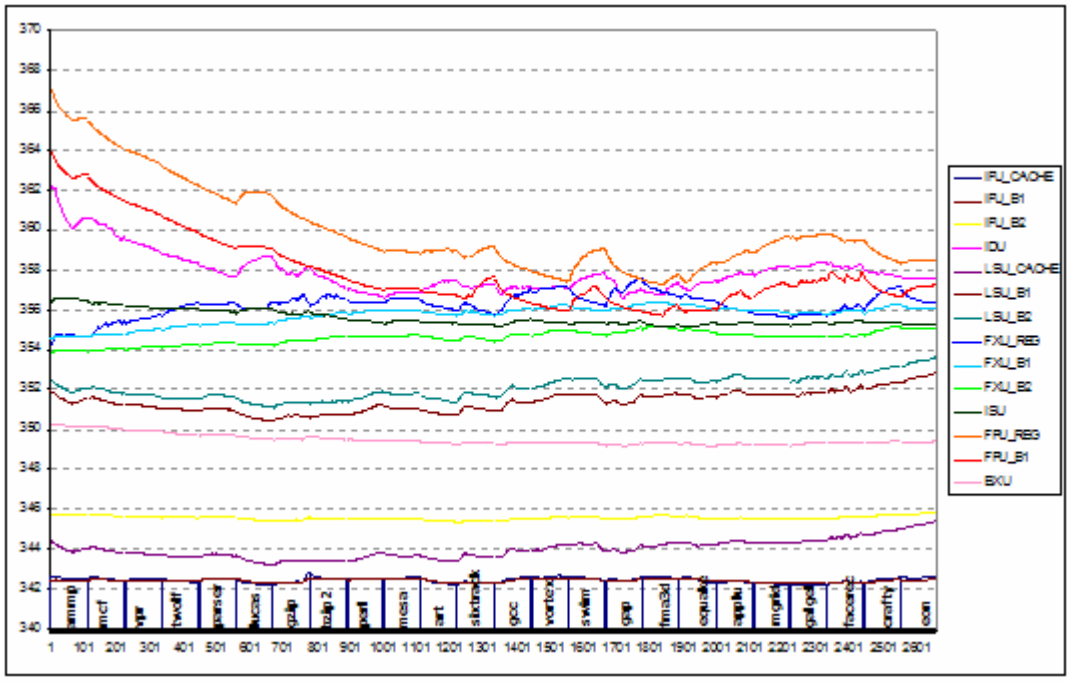

Figure 10. *AvgTemp* Policy for *10msec* time slices

8.3. MaxTemp Policy

In this set of experiments, the next thread selection is based on the hottest block on the core. *MaxTemp* policy schedules the thread with highest temperature profile for the current hottest block. The thermal behavior of SPEC with *MaxTemp* is displayed in Figure 11. We assume that *Mgrid* has been running for several O.S. time slices and its steady state temperatures has been reached for the initial temperatures. Temperatures are above the thermal threshold *96%* of the execution time. Similarly Figure 12 shows *MaxTemp* with initial temperatures for *Gcc*. Thermal threshold is exceeded *42%* of the execution time.

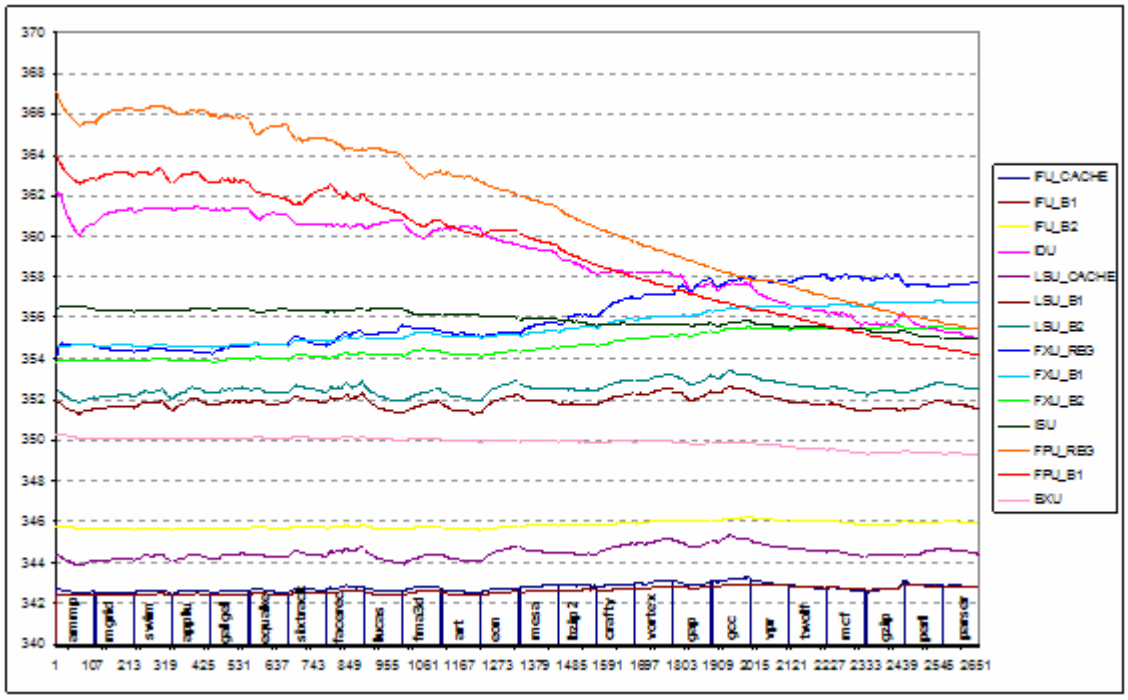

Figure 11. *MaxTemp* Policy with *10msec* operating system time slices

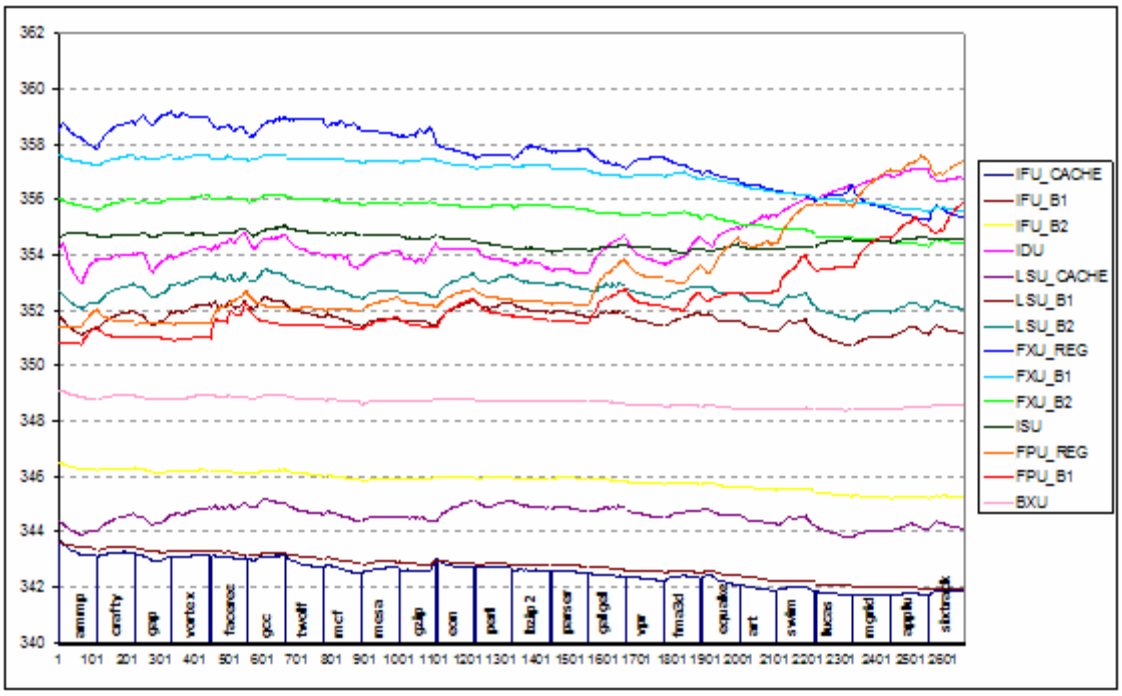

Figure 12. *MaxTemp* Policy for *10msec* operating system time slice (starting with steady state temperatures of *Gcc*)

8.4. MinTemp Policy

In this case, the next benchmark selection is based on the temperature of the hottest block on the chip. *MinTemp* policy selects the thread that has the minimum temperature for the current hottest block. Furthermore, assuming that there is almost always number of thermally challenging benchmarks in the queue, this policy schedules these threads if the hottest block temperature is below the safety threshold of *356K*.

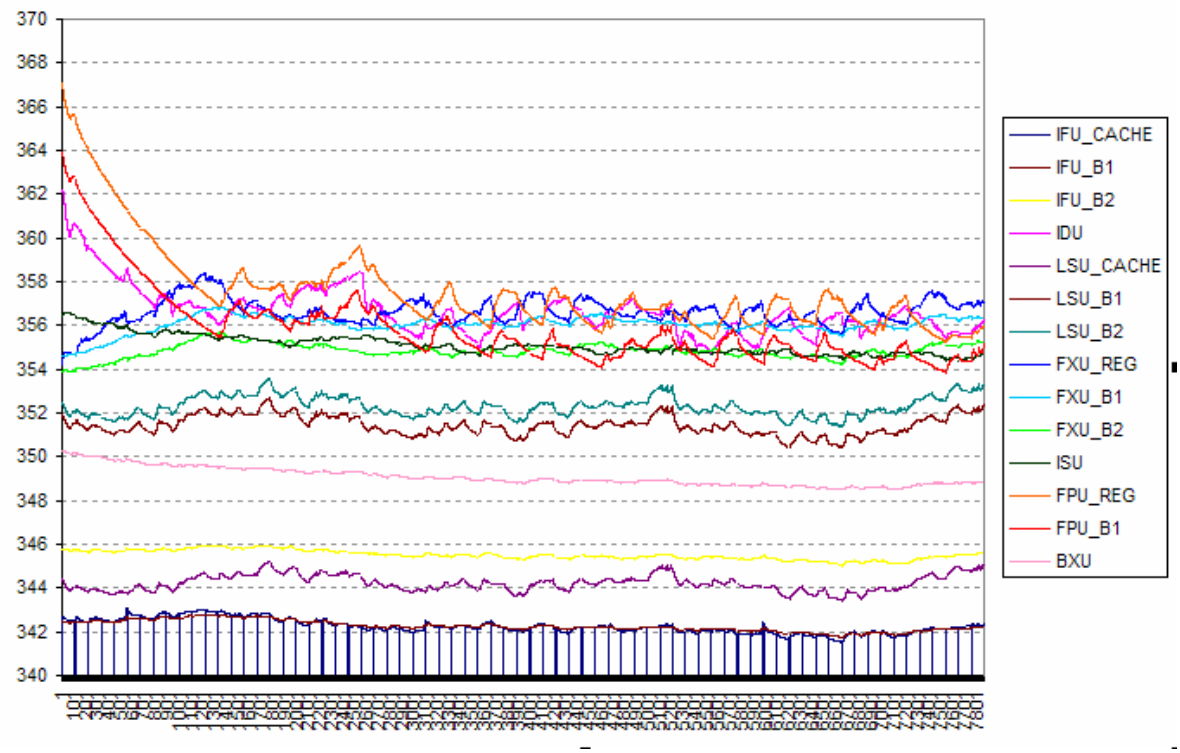

Figure 13. *MinTemp* Policy 3 Round-Robin runs with 10msec time slices

Figure 13 shows the execution of *MinTemp* policy in consecutive round robin scheme. It is important to note that after *130msec* (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 illustrates that *MinTemp* interleaves the thermally challenging threads whenever the on-chip temperatures are below safety threshold.

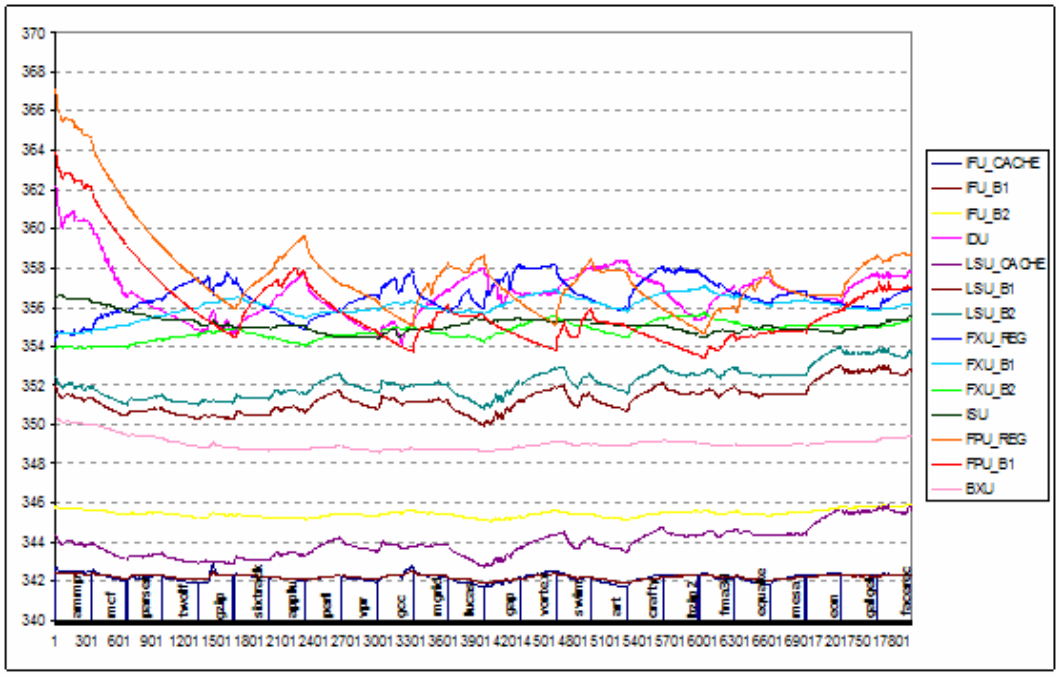

Figure 14. *MinTemp* Policy for *30msec* operating system time slice

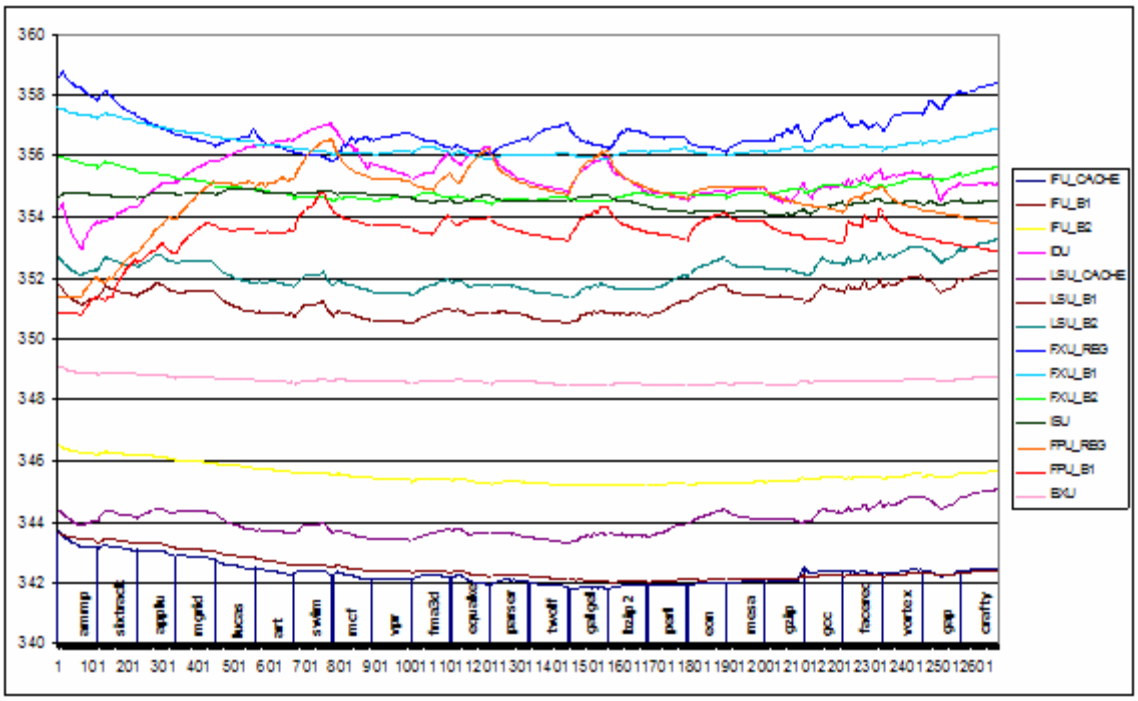

Figure 15. *MinTemp* Policy for *10msec* operating system time slice (*Gcc* starting temperature)

Figure 15 shows the thermal behavior of *MinTemp* policy with the initial temperatures for *Gcc*. Similar observations are valid for this case and consecutive round-robin runs displayed in Figure 16, where maximum on-chip temperatures are below thermal threshold more than *98%* of the execution time.

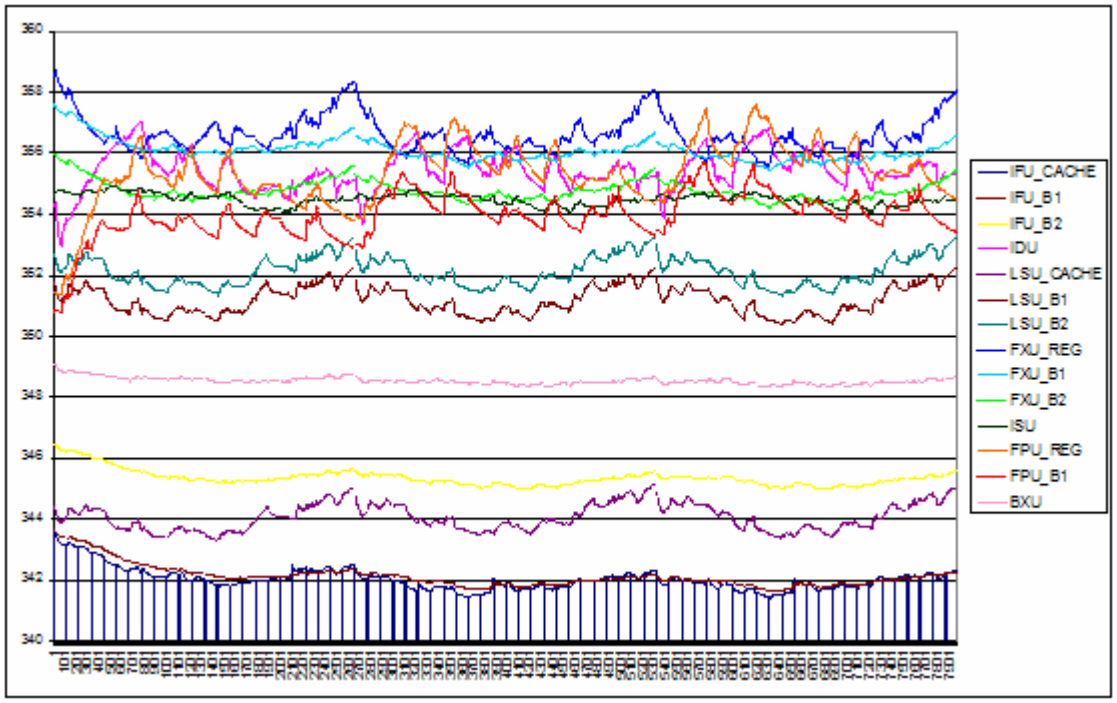

Figure16 *MinTemp* Policy, Round-Robin, *10msec* operating system time slice

8.5. Fetch Throttling

We compared the temperature-aware operating system scheduling policies with fetch throttling. We throttle fetch above the temperature threshold value of *358K*. We selected a coarse-grained aggressive fetch throttling policy in order to affectively reduce the temperatures below the thermal threshold. Figure 17 illustrates the effect of fetch throttling after *Mgrid's* steady state temperature has been reached. It is worth noting that the initial benchmark sequence until *Lucas* has not executed due to 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*). The IPC for this scenario 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 cooling time as opposed to executing complementary benchmarks in *MinTemp*.

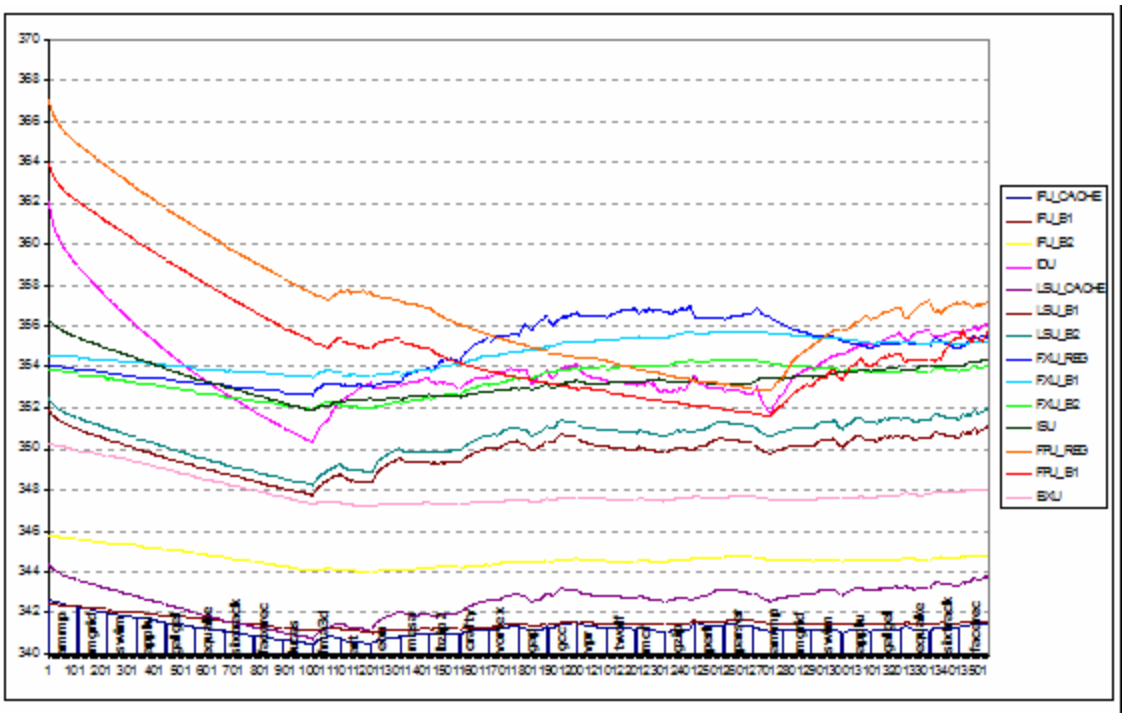

Figure 17. Fetch throttling above *358K*

8.6. Effects of Varying O.S. Time Slices

We investigated the affects of varying the length of operating system time slices. In some operating systems such as Linux, O.S. time slice length can be adjusted, in others such as AIX it is fixed (AIX case it is 10msec). Figures 18-20 display the affects of increasing the length of time slice for MaxTemp policy.

It is important to note that in all the experimented time slice lengths (from *10msec* to *50msec*) temperature threshold of *358K* was exceeded over *90%* of the execution time. As the time slice lengths increase to more than *30msec* the heating behavior in *FXU_REG* is more prominent and reach *360K*.

(The figures also illustrate the RC time constant concept. Although the transition from floating point sequence of benchmarks to fixed point sequence happens around time sample *2600*. It takes blocks *FPU_REG* and *FPU_B1 100msec* (*~1000* temperature samples) to cool down to *355K*.)

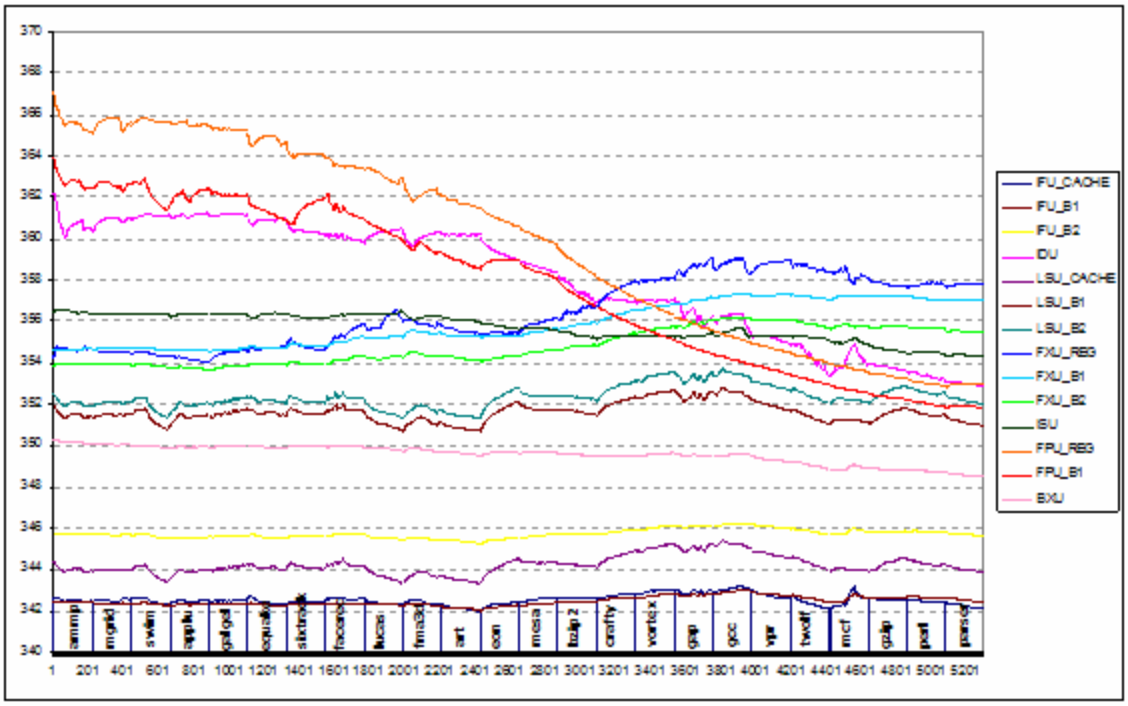

Figure 18. *MaxTemp* Policy for *20msec* operating system time slice

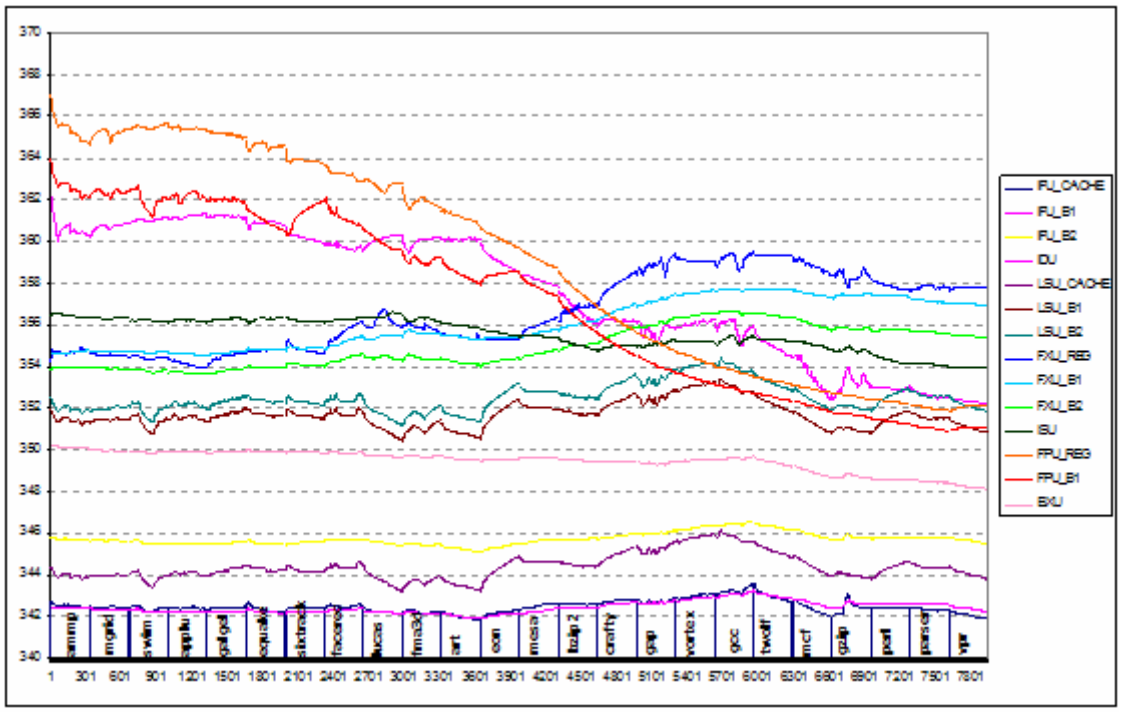

Figure 19. *MaxTemp* Policy for *30msec* operating system time slice

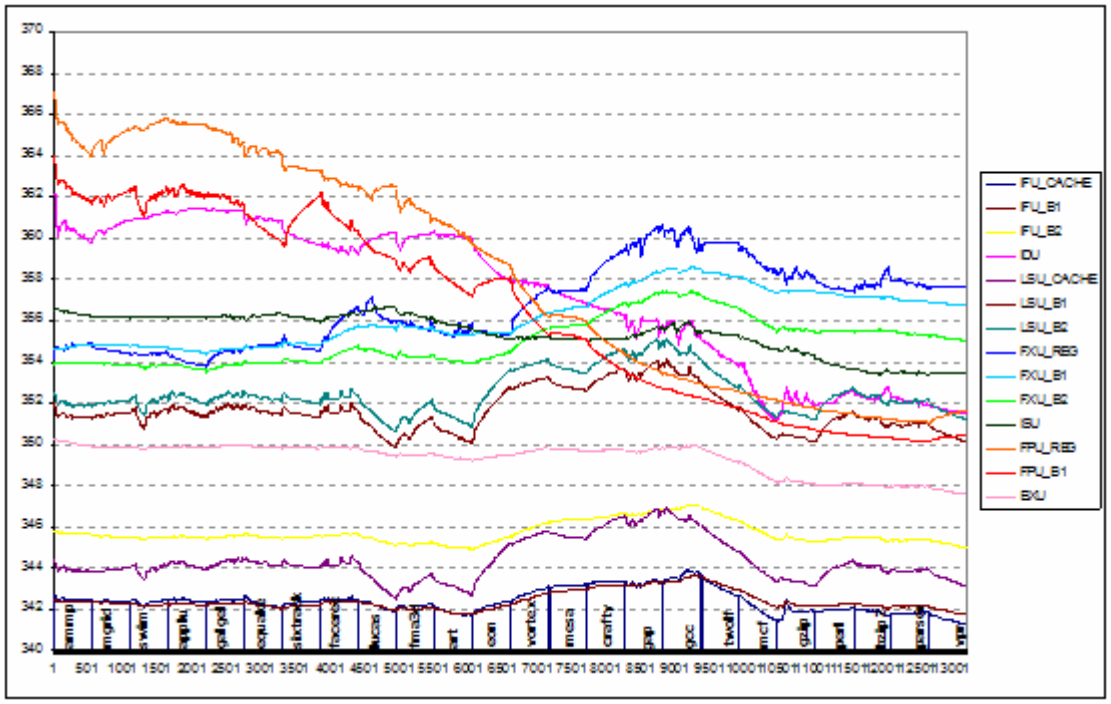

Figure 20. *MaxTemp* Policy for *50msec* operating system time slice

6. CONCLUSIONS AND FUTURE WORK

Today's microprocessor architectures present unique thermal challenges. We investigated the effectiveness of temperature-aware operating system scheduling as a software-hardware DTM technique. Since operating system thread scheduling is already incorporated in the CMP microarchitectures, thermal-aware O.S. scheduling can be performed with virtually no performance degradation.

Most architectures employ packaging systems designed for less than worst-case temperature, with the assumption that threads with absolute worst-case thermal behavior are of limited number in the distribution and can be dealt dynamically with hardware DTM ("reactive throttling") techniques.

We used traces generated from SPEC2000 benchmark set where *10* of the *25* benchmarks have maximum block temperatures above *360K*. We assumed a thermal threshold value of *358K*. (Having *40%* of the benchmarks thermally critical: quite aggressive, compared the Intel's packaging assumption that only *20%* benchmarks are above the packaging /threshold values).

Even with a thermally challenging benchmark, our *MinTemp* policy is effective in reducing the temperature from a high initial temperature (such as *Mgrid* steady state temperature) and keeping the temperatures lower than the thermal threshold *98%* of the execution time for cases with average initial temperature values (such as *Gcc*). The reason *MinTemp* is effective in reducing the temperature even with *40%* of the benchmarks having maximum block temperatures above thermal threshold is the diversity of hotspot blocks and the policy to interleave the hot benchmarks.

Our experimental analysis indicates that scheduling selection can result in temperature differences around $5^{o}C$ on average between different benchmarks for hottest blocks. Fetch throttling is also effective

at reducing the maximum temperatures below the thermal threshold. However, the corresponding performance degradation can be up to *20%. Random* thread selection and *AvgTemp* policies are comparable effectiveness.

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 100msec for the baseline POWER4-like architecture we looked at. With a time constant in the order of *100msec*, we looked at thermal-aware O.S. scheduling policies that intelligently interleave threads in the execution queue. We also studied a temperature estimation technique and employed it as a static thermal analysis tool (TPAT) for initial probing of various policies.

REFERENCES:

[Borkar1999] S.Borkar, "Design Challenges of Technology Scaling", *IEEE MICRO*, p23-29, August 1999.

[Sanchez1997] "Thermal management system for high-performance PowerPC microprocessors", H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, J.Alvarez, *Proc. COMPCON*, page 325, 1997.

[Brooks2001] D.Brooks, M.Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors", *International Symposium on High-Performance Computer Architectures*, p. 171-182. Jan 2001.

[Skadron2003] K.Skadron, M.Stan, W.Huang, S.Velusamy, K.Sankaranarayanan, D.Tarjan, "Temperature-Aware Microarchitecture", *Proceedings of 30th International Symposium on Computer Architecture* (ISCA 30) p.2-13, June 2003.

[Gunther2001] S.Gunther, F.Binns, D.M. Carmean, J.C. Hall, "Managing the Impact of Increasing Microprocessor Power Consumption", *Intel Technology Journal*, 2001.

[Moore2005] J.Moore, J.Chase, P.Ranganathan, R.Sharma, "Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers", *USENIX Annual Technical Conference*, Anaheim CA, April 2005.

[Yeh2001] L.T. Yeh, R.Chy, "Thermal Management of Microelectronic Equipment", American Society of Mechanical Engineers, (ISBN: 0791801683), 2001.

[ITRS2001] SIA. International Road Map for Semiconductors, 2001.

[Kursun2004] E.Kursun, G.Reinman, S.Sair, A.Shayesteh, T.Sherwood, "Low-Overhead Core Swapping for Thermal Management", Power-Aware Computer Systems (PACS) Workshop, in conjunction with MICRO Symposium, Dec 2004, Portland.

[Moudgill1999] 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/*.

[Brooks2003] 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 J. R&D,* vol. 47, no. 5/6, 2003.