MICRO-ARCHITECTURAL TEMPERATURE MODELING USING PERFORMANCE COUNTERS

A Thesis
in STS 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment
of the Requirements for the Degree

Bachelor of Science in Computer Engineering

by

Kyeong-Jae Lee

March 29, 2005

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in Science, Technology, and Society Courses.

_____
(Full Signature)

Approved _____ (Technical Advisor)
Kevin Skadron

Approved _____ (Science, Technology, and Society Advisor)
Patricia Click

Table of Contents

List of Figures

Abstract

Many thermal management techniques have been developed to effectively regulate the heat dissipated in modern processors. These techniques, however, require the ability to accurately measure the temperature of the processor. This project presents a software solution for temperature sensing that uses hardware resources known as performance counters. This methodology allows the thermal model to provide a detailed temperature profile of the processor at runtime using real workload. In particular, this project implements a software solution that models the Pentium 4 processor. The thermal model can be used in computer architecture studies or as an online temperature sensing mechanism. This report includes two case studies using the model: one that analyzes application-specific thermal stress patterns, and one that examines potential thermal security risks. Ultimately, understanding various thermal effects can help researchers develop more reliable and thermal-efficient systems.

1. INTRODUCTION

Modern computer systems need a detailed and accurate temperature sensing mechanism to effectively regulate the heat dissipated by the processor. The purpose of this project is to present a software solution for modeling the temperature of a computer processor using built-in hardware resources.

1.1. Problem Statement

Design enhancements that produce faster machines with increased resource utilization have constantly driven the development of computer processors. As a result, the power consumption of computer processors has rapidly increased in recent years as modern processors have become more complex. Increasing power and heat dissipation greatly affects the performance of the system and increases the implementation cost of cooling solutions.

Efficient temperature sensing methods are important because failure to detect overheating can produce computing errors and even cause the processor to melt down. These methods facilitate the development of advanced thermal management techniques and low-power designs, which lead to more reliable and thermal-efficient computer systems. The reliability of a system is especially important for safety-critical applications

that are used in biomedical devices or real-time control systems. Thermal-efficient systems can also lower maintenance costs and reduce energy consumption in air-conditioned rooms that store high-end server machines. In addition, if researchers can create processors that dissipate less heat, then manufacturers can reduce the functionality of some of the expensive cooling solutions such as large metal heat sinks or cooling fans. Furthermore, cooler systems will not require the bulky cooling devices such that the heat dissipated in mobile devices becomes bearable by humans. Ultimately, thermal-efficient systems can reduce the risk of user getting burned, improve system performance, and lower cooling costs.

During the early 1990s, the primary method of handling excess heat was to simply remove all the heat as efficiently as possible from the processor. Mabulikar took the approach of packaging the processor or similar circuit designs with a metal package called the MQUAD system [1]. The MQUAD package was designed to be a cost-effective system that provides excellent electrical and thermal performance. Others, such as Kamath, studied different cooling schemes using heat sinks. Kamath analyzed the heat transfer rates of two cooling methods: the first method forced air over the heat sink and the second method injected air through a fan mounted on top of the heat sink [2]. However, the use of cooling fans posed other problems. Fans required additional power consumption and provided poor reliability in smaller portable devices. Later on, heat pipes emerged as an efficient and reliable solution for portable devices such as laptop computer systems [3].

By the mid 1990s, designers began to realize that simply removing heat from the system was insufficient to solve the thermal problems. Researchers began to consider

methods that could accurately measure temperature and prevent catastrophic meltdown. In 1996, Bakker and Huijsing developed a smart temperature sensor that would switch off its power supply to achieve extremely low power consumption [4]. The PowerPC microprocessor also employed a thermal assist unit (TAU) that monitored temperature and regulated processor operations [5].

As researchers began to shift their attention from passive cooling solutions to dynamic self-monitoring solutions, thermal issues were being considered at the design stage of circuitry and architecture. Power density is proportional to the frequency and the square of the voltage. Thermal management techniques such as dynamic clock gating and dynamic voltage scaling (DVS) have aimed to reduce the operating frequency or voltage to minimize heat dissipation only when the temperature reaches a certain threshold. Lim and others at Portland State University proposed a thermal-efficient design that uses a secondary low-power processing unit when the processor gets heated up [6]. Although their design requires a 4.6 % increase in processor area, energy-performance improved by 11.4 % and other thermal management techniques such as DVS can be use in conjunction to further reduce energy consumption.

While many technical improvements are being made to control the dissipated heat, most solutions require the ability to accurately measure the temperature of the processor. Current computer systems measure temperature through thermal sensors, which are based on analog circuits. These sensors are costly to implement and can even exacerbate the thermal problem by dissipating too much power. IBM's Power5 processor, designed for high-performance servers, is known to have 24 thermal sensors. The Pentium 4 processor has only two thermal sensors. For low-cost systems, usually there are only few

sensors available on the processor, if any at all. Each sensor must be carefully placed on the processor to account for the spatial gradient of temperature. In addition to the placement problem, the sensor's response time to a temperature change can be quite slow. Hence, an accurate representation of the temperature distribution of the entire processor is difficult to obtain at runtime through thermal sensors.

1.2. Project Scope

Simulation is frequently used in architectural studies to test thermal management techniques or new designs. Simulation is a simple way to obtain temperature readings, but fails to account for system-wide hardware effects. Kevin Skadron's research lab at the University of Virginia has created a thermal simulator called HotSpot, and has shown that the performance of dynamic thermal management (DTM) techniques can substantially deteriorate if temperature sensors are inaccurate [7]. Although the HotSpot software package provides a detailed floorplan-level description of temperature, the tool requires localized power data and its use is limited to simulated architectural studies. Obtaining temperature readings from real sensors would be ideal, but the sensor circuitry is expensive and does not provide a full view of the processor as discussed in the previous section.

This project presents a software solution for temperature sensing that uses hardware information as a measure of real processor activity. Specifically, the current HotSpot thermal model has been extended to infer processor activity from performance counters. The use of real physical resources –performance counters– allows the

temperature model to present a more realistic description of the processor without using the expensive thermal sensor circuitry. And by using the HotSpot framework, the temperature model still provides a full floorplan-level detail of the temperature distribution. The model facilitates architecture studies where real workloads can be used at runtime to observe the thermal behavior of processors. In particular, this project focuses on the Intel Pentium 4 processor and its architecture.

1.3. Related Concepts in Micro-Architecture

Computer architecture is the study of the internal organization and interconnection of hardware elements in a computer system. The processor, also known as the CPU, controls all major operations and hence is the central and most active component. All programs that run on a computer consist of many instructions, which are digital bits that inform the processor about what action to take. The processor can be divided up into smaller functional units, where each block handles a single operation such as arithmetic calculations, instruction fetching, or data storage. The temperature model provides a way to analyze the thermal characteristics for each functional unit in the processor. This model relies on the ability to infer information about processor activity directly from the micro-architecture. Processors have built-in performance counters, which are used to count specific architectural events that occur during the execution of a program. For example, performance counters can be configured to gather statistics on how many integer calculations were performed, or how many times the processor accessed the data-cache unit.

1.4. Thesis Overview

Chapter 2 presents the general methodology and design of the temperature model. Chapter 3 provides an analysis of the model in terms of performance penalty. The subsequent chapters describe experiments that illustrate the potential benefits of using the model. Chapter 4 presents results from running benchmark programs and an analysis on the thermal stress patterns. Chapter 5 describes the experiments used to show the potential danger of thermal viruses. Chapter 6 concludes the paper.

2. METHODOLOGY

This chapter explains the overall design of the thermal model. The main approach is to extend the HotSpot software package to use hardware resources. Since this project focuses on the Pentium 4 processor, some information is unique to the Pentium 4 and its architecture. Subsequent chapters discuss how to use the model in various applications.

2.1. Pentium 4 Architecture

2.1.1. Overview of Architecture

The computer system used in this project is a 2.6 GHz Pentium 4 processor, 130 nm Northwood core. The typical power dissipation is 69.0 W, and the operating voltage is 1.6 V [8]. The Pentium 4 features a 20-stage pipeline and a trace cache, which eliminates the normal instruction decoding from the execution loop by storing traces of assembly instructions [9]. The Pentium 4 also has two Arithmetic and Logic Units (ALUs) that each execute in one-half the global clock cycle. The Pentium 4 supports hyper-threading technology, which allows the processor to run two threads simultaneously.

2.1.2. Processor Specifications

HotSpot has several parameters that can be modified to effectively model a specific processor. In particular, the geometric specification and the floorplan layout of the processor are required to configure the thermal model. Table 1 shows the mechanical dimensions and material characteristics for the Pentium 4 package. These settings are based on design schematics found in [10] and are used to configure the HotSpot program.

HotSpot variable	**Value**	**Description (Unit)**
t_chip	0.74	chip thickness (mm)
c_convec	131.84	convection capacitance (J/K)
r_convec	0.084	convection resistance (K/W)
s_sink	76	heat sink side (mm)
t_sink	12	heat sink thickness (mm)
s_spreader	31	heat spreader side (mm)
t_spreader	1.5	heat spreader thickness (mm)
t_interface	0.05	interface material thickness (mm)
ambient	40+273.15	ambient temperature (K) (inside box)
roughness	0.8	roughness factor of package surface (0.0~1.0)
RHO_INT	0.315	thermal resistivity of interface material (mK/W)
SPEC_HEAT_INT	3.96E+06	specific heat of interface material (J/m^3K)

Table 1. HotSpot configuration settings

Another important input to the program is the floorplan layout. Each processor has a unique floorplan layout, which partially depends on the number and type of available functional units. The floorplan of the Pentium 4 can be represented using the following functional units: L1 branch prediction unit (BPU), L2 BPU, instruction decoder, trace cache, memory order buffer (MOB), ITLB, bus control unit, DTLB, L1 cache, L2 cache, micro-coded ROM (UROM), allocation unit, rename unit, instruction queue #1, instruction queue #2, scheduler, retirement unit, floating-point (FP) execution unit, FP register file, integer execution unit, integer register file, and memory control unit. Figure

1 is an approximated floorplan layout that has been adapted from the die photo of the Northwood core [11]. The trace cache and L1 cache are divided into two units for simplicity.

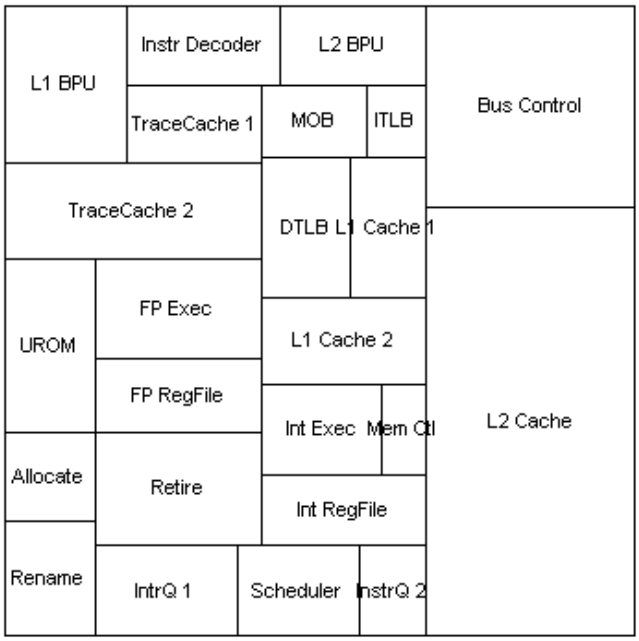

Figure 1. Floorplan layout of Pentium 4. Adapted from [11]

2.2. HotSpot Extension

2.2.1. Performance Counters

The Pentium 4 includes an extensive set of performance monitoring features, with 45 configurable events and 18 physical performance counters [12, 13]. The performance counters are used to count specific micro-architectural events for debugging and performance measurements. Each counter is associated with one counter configuration control register (CCCR), which determines the specific counting scheme. The event selection control registers (ESCRs) determine which event is to be counted. However,

the thermal model is a user application and cannot directly modify these counters. A special device driver must be written and installed on the computer in order to access these counters. A simplified device driver, adapted from the Abyss device driver [14], is used in this project. The thermal model will then indirectly use the device driver to read the performance counter values.

2.2.2. Power Modeling

The existing HotSpot framework models the processor as a network of thermal resistors and conductors per functional unit, with power dissipation in each unit treated as a current source in the RC network. The thermal model needs to estimate power dissipation from performance counters. Isci and Martonosi have already shown that power can be accurately modeled from performance counters [15]. Their power model uses the following equation:

$$Power = \left(MaxPower \times ArchitecturalScaling \times AccessRate\right) + NonGatedClockPower$$

Several micro-architectural events, which are measured through performance counters, are combined to closely approximate the number of accesses to each functional unit. This project uses similar metrics found in [15] and extends HotSpot to interface with the Pentium 4 performance counters. Appendix A lists the configuration settings for the performance counters. For the Pentium 4, not all performance metrics can be measured simultaneously using the 18 performance counters. Four sets of counter rotations are required to sample all necessary architectural events. Thus, the performance counters are periodically sampled but a different set of architectural events is measured each time. Appendix B lists the complete metrics and parameters for the power model.

2.2.3. Runtime Requirements

One goal of this project is to be able to use the thermal model at runtime to dynamically calculate temperature values. The main temperature computation algorithm must be optimized to satisfy these runtime requirements. For example, if the temperature values are updated every 10 milliseconds, the actual sampling and calculation performed by the program must be less than 10 milliseconds. The model must be programmed such that the computation and the counter sampling are performed concurrently. HotSpot currently uses a fourth order Runge-Kutta numerical solution to calculate temperature, and this solution proved to be inadequate for runtime measurements. The Runge-Kutta-Fehlberg (RKF) method, which uses an adaptive step size to minimize the calculation time, replaced the original method. Despite the complexity, the RKF method is more efficient and can be easily integrated into the existing HotSpot framework.

3. ANALYSIS OF THE MODEL

Measuring the performance and efficiency is important for any software solution. This information is useful to understand how the software solution affects the computer system. This chapter explains some of the implementation details of the thermal model and evaluates its performance and thermal overhead.

3.1. Implementation of the Model

The source code for the thermal model was compiled using the gcc-3.2.2 compiler with optional flags of "-O3 -march=pentium4 -mfpmath=sse -mmmx -msse -msse2". The additional compiler flags are used to optimize the source code for faster performance. When executed, the thermal model periodically prints out a list of temperature values for each functional unit. In this project, the default sampling interval is used for all experiments: 5 milliseconds for each counter-rotation, and 20 milliseconds to update temperature values. Although the model updates temperature values infrequently, the program continually monitors access to performance counters and updates power values. Given the new power values, the program performs a large set of calculations based on the RKF method to obtain the temperature values. Ideally, the thermal model should add

very little overhead to the system, but monitoring the performance counters and calculating the temperature can require a substantial amount of processor resources. The subsequent sections describe two metrics that are used to find the inherent overhead of the program.

3.2. Performance Overhead

This project uses the CPU2000 benchmarks to estimate the performance overhead. CPU2000 is a benchmark suite designed by the Standard Performance Evaluation Corporation (SPEC) to measure performance of computer processors [16]. CPU2000 contains two types of benchmark applications: one that mainly supports integer operations, and one that supports floating-point operations. All SPEC benchmarks are compiled using the base tuning option. For the purposes of this study, the execution time of the benchmark programs is used as the measure of performance. On the first trial, the time for each benchmark to complete its task is recorded. Then, the benchmark and the thermal model are executed simultaneously. The benchmarks take longer to execute in the second trial since the thermal model is being processed concurrently. This procedure is repeated three times. The overhead is measured as the percentage difference in the average execution time of each benchmark between the two experiments. Hence, a longer time difference means a higher overhead.

Table 2 lists the average execution times of each benchmark. A total of 20 benchmark programs have been selected from the CPU2000 benchmark suite. Note how running the model simultaneously increases the execution time of each benchmark.

Figure 2 shows the performance overhead results that have been calculated from data in Table 2.

Integer Benchmarks	Without model (seconds)	With model (seconds)	Floating-point Benchmarks	Without model (seconds)	With model (seconds)
gzip	181.82	238.13	wupwise	167.64	257.49
vpr	261.87	314.93	swim	435.61	462.62
gcc	112.94	146.02	mgrid	280.34	426.34
mcf	284.28	304.23	applu	323.65	439.48
crafty	123.23	169.48	mesa	210.15	307.83
parse	250.35	317.36	art	880.05	910.47
gap	119.46	156.72	equake	141.62	160.95
vortex	300.52	297.85	ammp	502.59	606.88
bzip2	237.05	288.57	sixtrack	8.30	11.43
twolf	541.59	555.47	apsi	586.31	695.55

Table 2: Average execution time of SPEC benchmarks

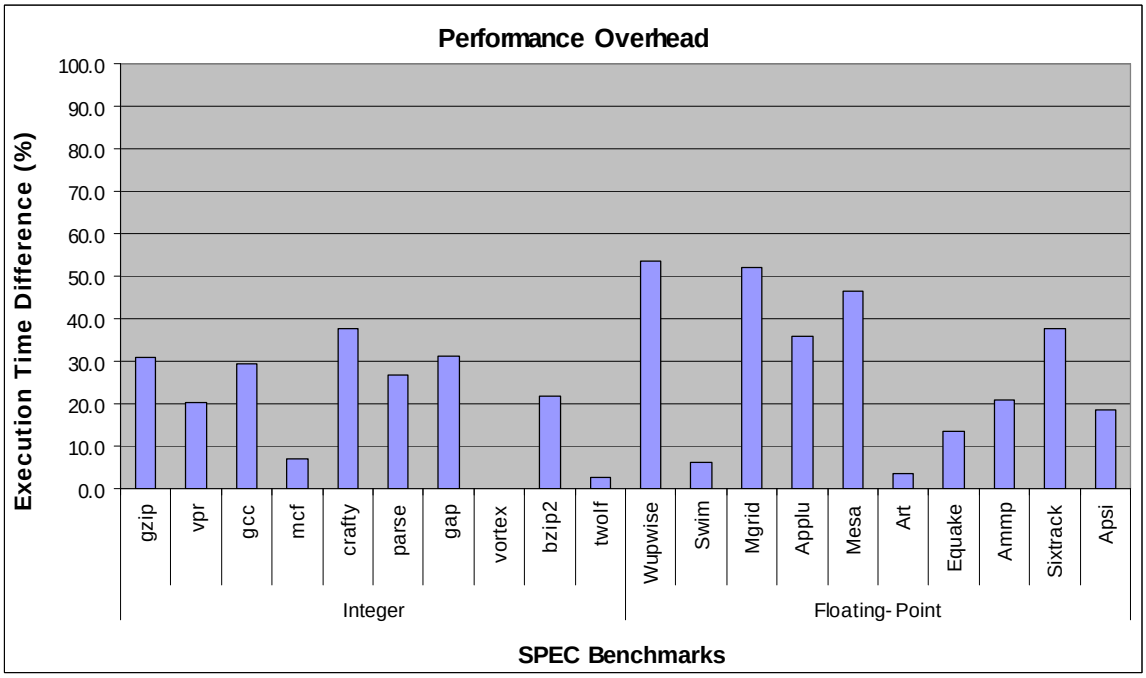

Figure 2. Performance overhead of SPEC benchmarks

The results indicate that the performance overhead can vary across applications, and the variation is larger across floating-point benchmarks than integer benchmarks.

The average overhead is 20.6 % for integer applications, and most of them are in the range of 20 ~ 30 %. In contrast, half of the floating-point benchmarks have an overhead that is near or below 20 %, while three benchmarks –wupwise, mgrid, and mesa– have a very high overhead above 45 %. The thermal model's main computation algorithm requires several iterations of the RKF algorithm. Benchmarks with iterative numerical methods that use lots of floating-point operations are more likely to compete for computing resources, and hence these benchmarks have greater overhead. Thus, the thermal model would impede performance were it to be used in conjunction with high-precision scientific applications.

3.3. Thermal Overhead

The thermal model inevitably uses computing resources, and hence the estimated temperature values reflect the amount of heat added by the model itself. Obtaining the thermal overhead, however, is not an easy task since the temperature of each functional unit cannot be estimated without using the thermal model. Thus, the amount of heat generated by the model is approximated using the following procedures.

First, steady-state temperature values for each functional unit are measured and recorded. Steady-state conditions mean that the thermal model is the only software running on the computer system. The second experiment concurrently executes two versions of the thermal model. One version is the original program. The other program is a modified version where the counter values are statically assigned instead of obtaining real values via the device driver. Hence, only one version prints out the real temperature

values. The other program uses the same algorithm to computer temperature but is using false data. One reason for using this experimental setup is because only one program can access the performance counters at any given time. Furthermore, the majority of the processor activity is in the main temperature calculation algorithm and not in the performance monitoring functionality. Hence, the thermal overhead can be approximated as the temperature difference between the second experiment and the steady-state condition. Table 3 lists the results of these experiments.

Units	**Temperature (^{o}C)**		**Thermal Overhead (^{o}C)**
	Steady-State	**Two Versions**	
BusCtl	42.79	43.60	0.81
L2_Cache	43.20	44.02	0.82
L2_BPU	46.19	49.28	3.09
InstrDecoder	44.72	46.35	1.63
L1_BPU	45.02	46.81	1.79
ITLB	43.85	45.28	1.43
MOB	44.05	45.60	1.55
TrCache_Top	46.39	47.98	1.59
TrCache_Bot	46.37	47.91	1.54
DTLB	45.44	47.48	2.04
L1_Cache_Top	45.14	47.07	1.93
L1_Cache_Bot	46.38	48.60	2.22
IntExe	50.65	53.82	3.17
MemCtl	50.06	51.89	1.83
IntReg	52.35	55.72	3.37
FpExe	43.84	45.22	1.38
FpReg	44.63	46.14	1.51
UROM	43.33	44.25	0.92
Alloc	50.54	51.75	1.21
Rename	50.83	52.01	1.18
Retire	49.24	50.68	1.44
InstrQ1	51.61	53.11	1.50
Sched	51.95	53.37	1.42
InstrQ2	51.12	52.87	1.75

Table 3. Thermal overhead

For most units, the results show that the temperature increases by roughly 0.9 ~ 2.1 ^{o}C. The integer register file, integer execution unit, and the L2 BPU have the largest thermal overhead. This pattern indicates the compute-intensive nature of the thermal model, and reaffirms the fact that the majority of the processor activity lies in the RKF algorithm.

4. THERMAL STRESS PATTERNS

The thermal model can be used to characterize thermal behavior of applications or study temperature-aware design techniques. This chapter includes a case study on thermal stress patterns of benchmark programs. Understanding the temperature variations of certain programs can help designers to efficiently allocate resources to alleviate thermal design concerns.

4.1. Spatial Variations

This section examines several SPEC benchmarks in detail to illustrate the thermal characteristics of different types of applications. While the information is applicable to most of the benchmarks, only a few are selected and presented in this report because their high overhead makes the thermal stress patterns easily noticeable.

For each benchmark, a temperature trace is created by recording the output values of the thermal model. Initially, the thermal model runs on the system by itself. The benchmark starts executing after roughly two minutes.

4.1.1. Integer Applications

Figure 3 and Figure 4 show the transient temperature trace of the processor for the gzip and gcc benchmarks respectively. Note that the time axis of each figure is scaled to fit the execution time of each benchmark. See Table 2 in section 3.2. for details on the execution time of SPEC benchmarks. In each figure, the sharp rise in temperature indicates the point when the benchmark starts running. The data before that point are the steady-state conditions.

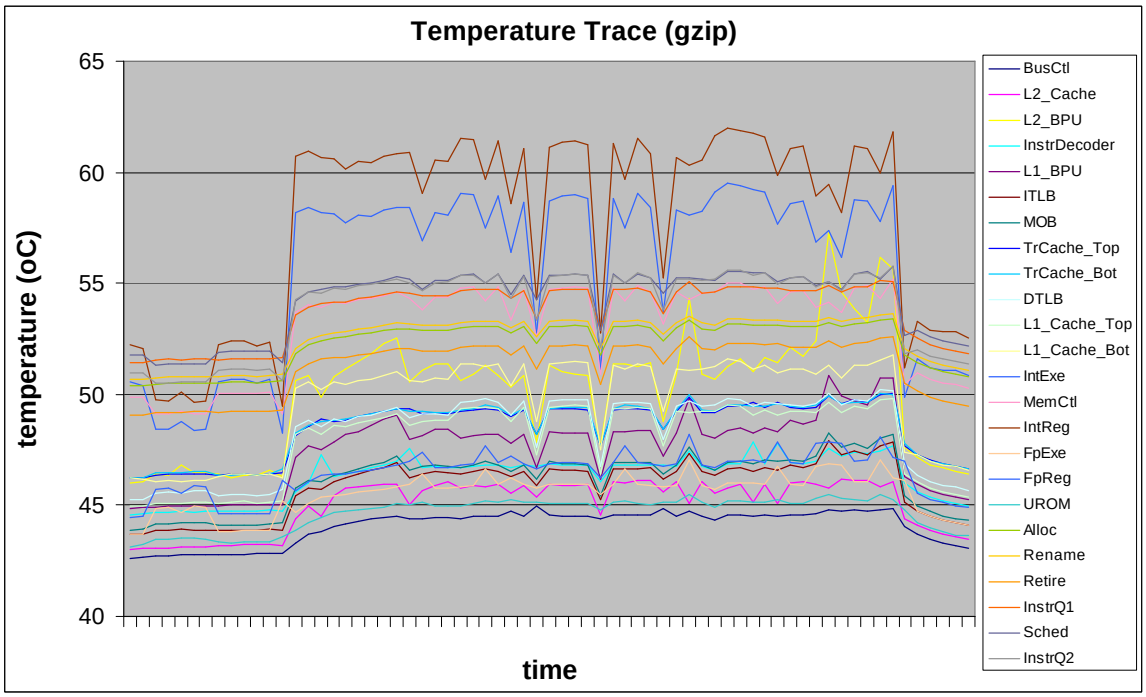

Figure 3. Temperature trace of gzip benchmark

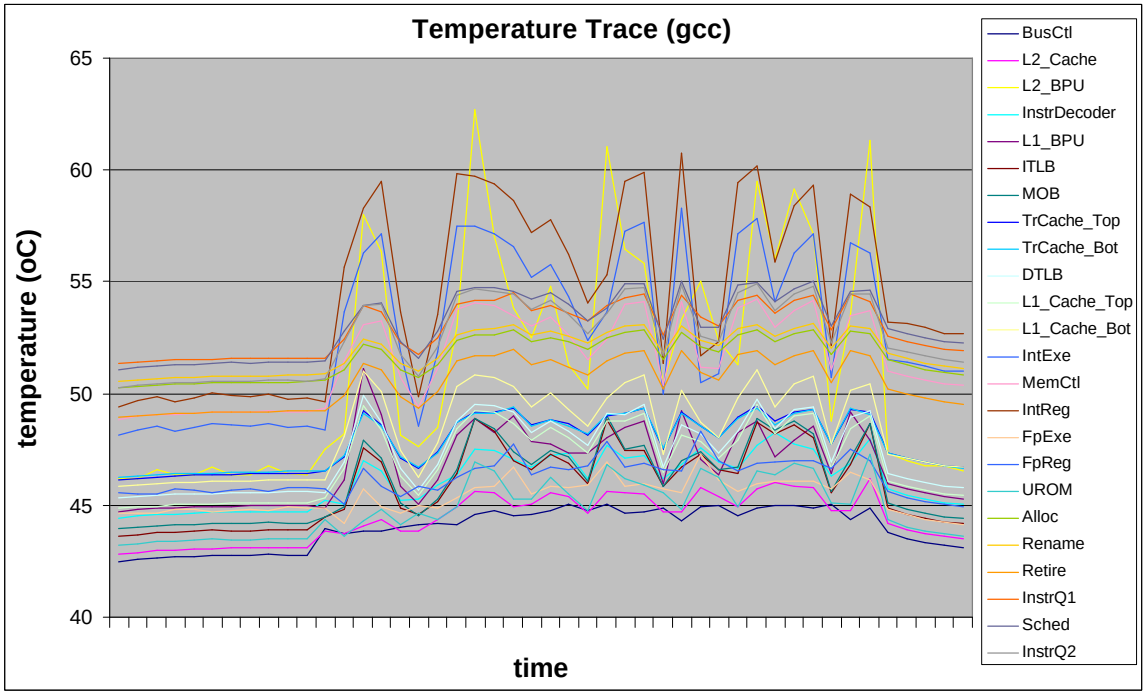

Figure 4. Temperature trace of gcc benchmark

For integer benchmarks, the integer units –IntReg and IntExe– are typically the hottest units on the chip. Even considering that the thermal model itself adds heat mostly to the integer units, the overall temperature gradient indicates that certain units are likely to heat up more than other units. In comparison to the steady-state condition, the amount of increase in temperature for each functional block varies from 2 to 10 ^{o}C.

4.1.2. Floating-Point Applications

Figure 5 and Figure 6 show the temperature trace of the wupwise and mesa benchmarks respectively.

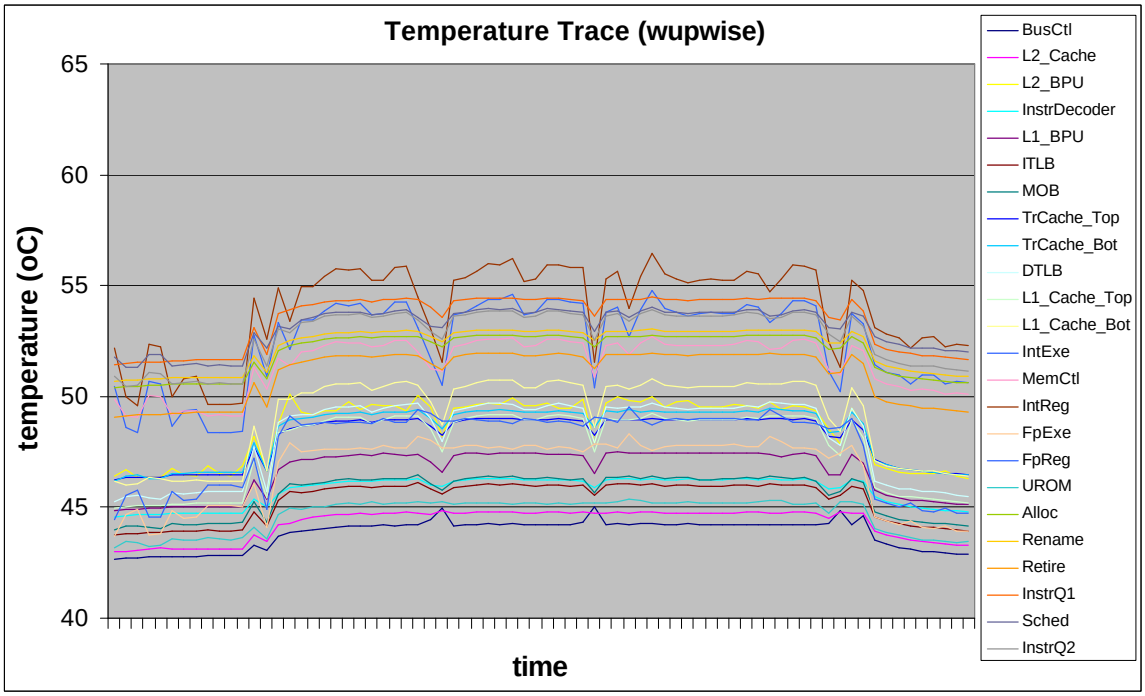

Figure 5. Temperature trace of wupwise benchmark

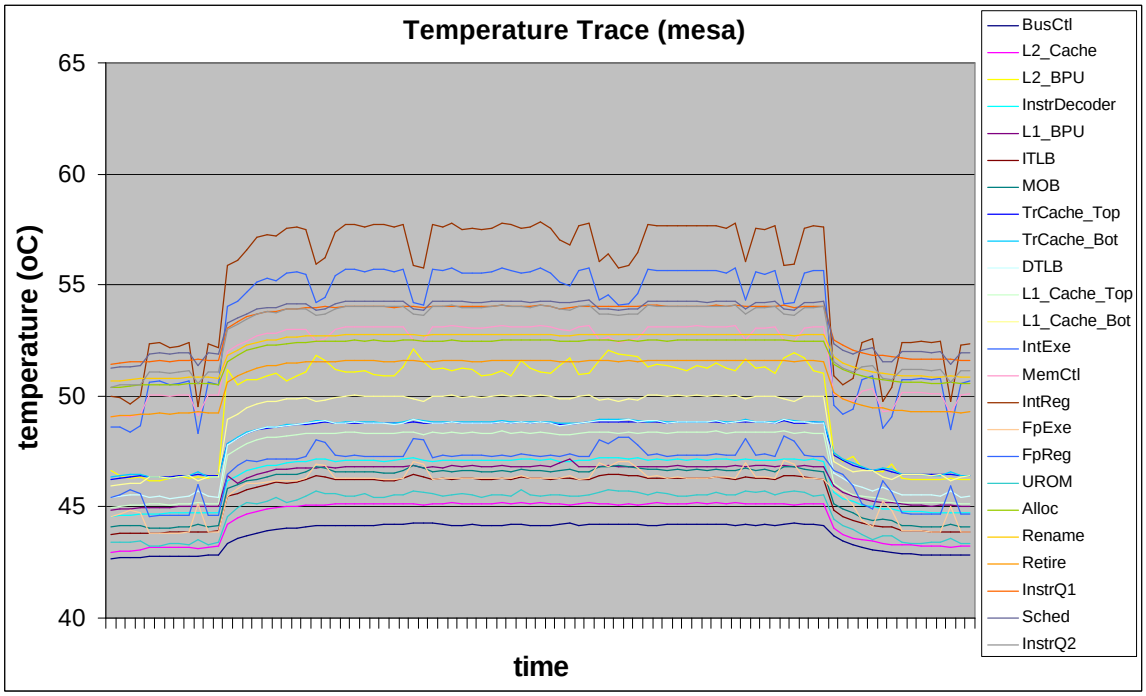

Figure 6. Temperature trace of mesa benchmark

In contrast to integer benchmarks, both figures show a relatively uniform temperature increase over all functional blocks. In addition, the amount of increase in temperature ranges from 1.5 to 5 ^{o}C, which is considerably smaller than that of integer benchmarks. For the wupwise and mesa benchmarks, the floating-point units heat up by the largest amount, but not enough to significantly change the thermal gradient of the processor. The overall thermal distribution does not differ much from the steady-state condition other than the fact that the average temperature is higher.

4.1.3. Application-Specific Thermal Behavior

The temperature traces presented in the previous two sections clearly show a general pattern of how integer benchmarks differ from floating-point benchmarks. While integer benchmarks tend to heat up the processor more than floating-point benchmarks, the relative temperature change across each functional unit is more uniform for floating-point benchmarks. Table 4 shows the average increase in temperature when the SPEC benchmarks are running. This table only includes the four benchmarks –gzip, gcc, wupwise, and mesa– that were used in the previous sections. Note that the floating-point benchmarks have a smaller range of values and a smaller standard deviation than the integer benchmarks. These numerical results support the analysis.

Units	Integer		Floating-Point	
	gzip	**gcc**	**wupwise**	**mesa**
BusCtl	1.69	1.79	1.41	1.35
L2_Cache	2.51	1.87	1.48	1.89
L2_BPU	5.39	8.07	3.19	5.08
InstrDecoder	2.16	2.16	1.42	2.36
L1_BPU	3.43	2.77	2.21	1.77
ITLB	2.74	3.13	2.01	2.40
MOB	2.83	3.09	2.11	2.56
TrCache_Top	2.85	2.21	2.39	2.36
TrCache_Bot	2.90	2.20	2.76	2.43
DTLB	3.90	2.95	3.81	3.32
L1_Cache_Top	3.89	2.94	3.62	3.16
L1_Cache_Bot	4.54	3.12	3.84	3.53
IntExe	7.32	4.18	2.85	4.61
MemCtl	4.26	2.67	2.10	2.86
IntReg	7.95	4.43	2.66	4.86
FpExe	2.15	1.94	3.78	2.55
FpReg	2.33	2.05	4.16	2.79
UROM	1.72	2.26	1.79	2.19
Alloc	2.36	1.69	2.04	1.91
Rename	2.32	1.66	1.99	1.87
Retire	2.79	1.97	2.47	2.30
InstrQ1	2.92	2.04	2.59	2.37
Sched	3.13	2.02	1.71	2.17
InstrQ2	3.88	2.50	2.36	2.77
Average	3.41	2.74	2.53	2.73
Standard Deviation	1.58	1.35	0.82	0.96

Table 4. Average temperature increase of SPEC benchmarks

Another way to study thermal behavior is by visualizing the thermal map of the processor. Consider the gzip benchmark and the wupwise benchmark. Figure 7 shows the thermal gradient of the processor at a particular instance when these benchmarks are executing. The uniform temperature increase for the wupwise benchmark creates a thermal map similar to that in steady-state. The non-uniform temperature change for the gzip benchmark creates an easily noticeable hot spot near the integer units.

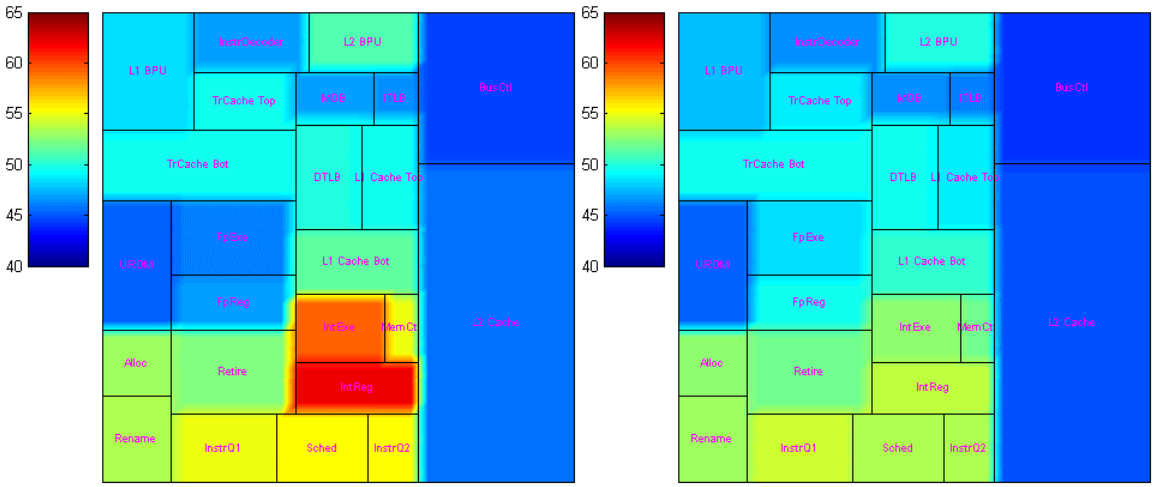

Figure 7. Thermal plot for integer and floating-point benchmark.

(a) gzip benchmark (b) wupwise benchmark

Understanding these spatial variations can assist manufacturers to customize thermal design packages for processors. For example, if a Pentium 4 processor used in a server machine mainly supports numerical computations, it may be feasible to use inexpensive packaging materials. Since integer benchmarks tend to create larger temperature gradients around the integer units, differentiating the heat spreader material around the known hot spots would be a possible solution to effectively remove heat and minimize packaging costs for processors running integer applications. Customized packaging can help manufacturers to efficiently allocate resources where it is needed the most and minimize cooling costs.

4.2. Temporal Variations

The previous section analyzed thermal stress patterns across the functional units. Understanding these spatial differences is very important but not sufficient to manage temperature. Typically, DTM techniques operate in small time intervals. Thus, in addition to the global picture of the temperature distribution, it is also important to understand how temperature changes as time progresses.

Although most applications exhibit predictable thermal behavior over time, some applications may abruptly shift the temperature gradient in unexpected ways. Consider the gcc benchmark. Figure 8 is identical to Figure 4, except that it only shows three units: the L2 BPU, integer register file, and scheduler unit. Figure 9 also lists the temperature trace for the same three units while running the vortex integer benchmark.

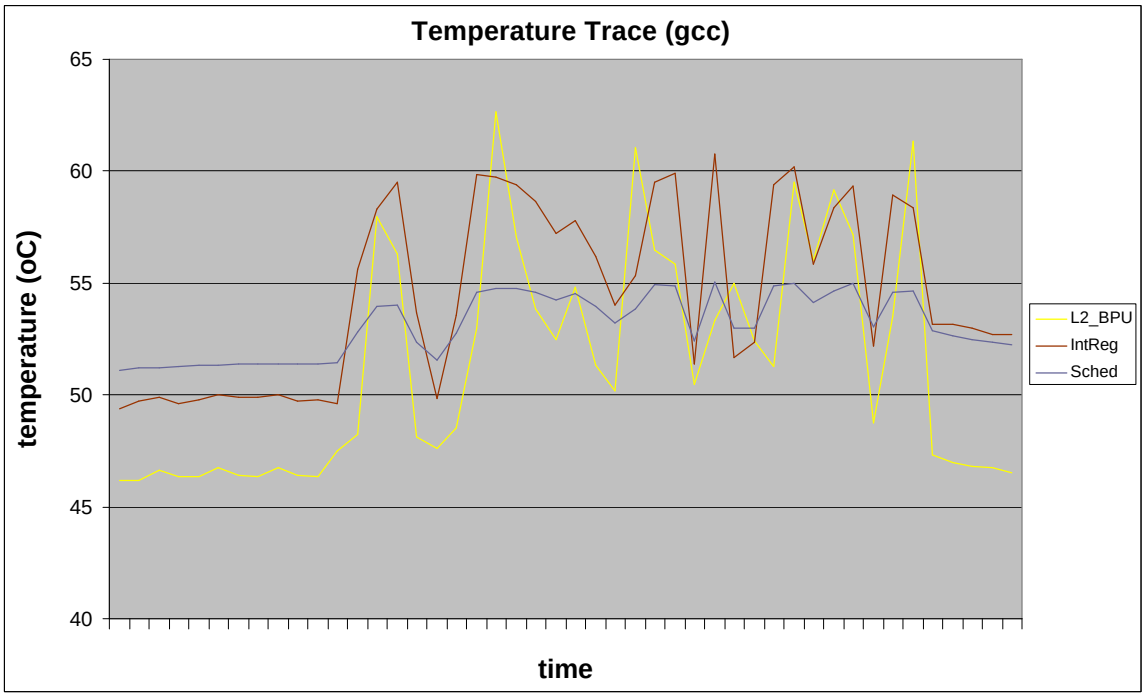

Figure 8. Temperature trace of gcc benchmark. Only three units are shown.

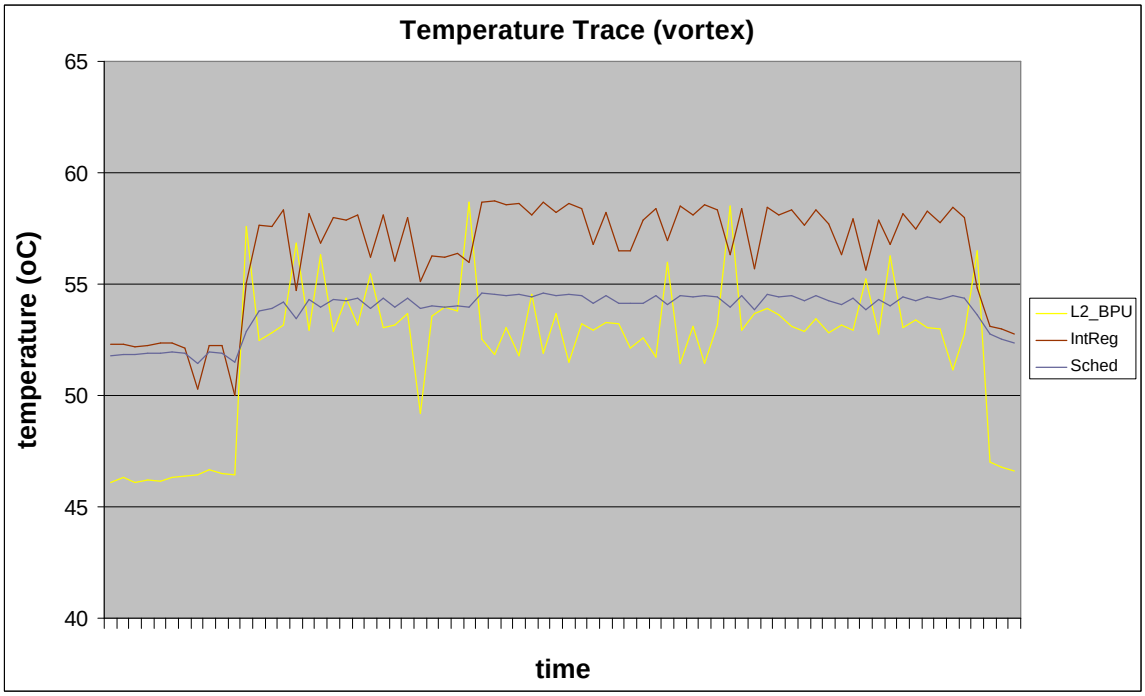

Figure 9. Temperature trace of vortex benchmark. Only three units are shown.

In Figure 8 and Figure 9, the integer register file is typically the hottest unit on the processor. However, as the temperatures sporadically change, the L2 BPU or the scheduler often becomes the hottest unit. That is, the hot spot is not fixed but moves around the processor during the execution of the program. Also note that the L2 BPU is located far away from the integer units, which are placed in the lower half of the processor (see Figure 1).

The results indicate that thermal stress patterns are not fixed to certain regions on the chip and can dramatically change as time passes. To effectively operate any thermal management technique, researchers must be able to characterize the movement of hot spots and quantify the thermal gradient bounds. Understanding these temporal variations can ultimately help chip designers to intelligently place thermal sensors on the processor.

Gunther et al. have explained the importance of examining thermal maps to find the optimal location of sensors on the Pentium 4 [17]. The thermal model is a useful tool to effectively locate hot spots on the processor.

5. THERMAL SECURITY RISKS

The main purpose of this chapter is to show how the thermal model can be used in computer architecture studies in addition to its use as a temperature sensing mechanism. This chapter includes a case study of potential thermal security attacks on microprocessors that only have a small number of thermal sensors.

5.1. Thermal Monitors of the Pentium 4

The thermal model produces a detailed temperature profile of the processor. For most applications, the hottest spot on the Pentium 4 processor tends to be near the integer execution unit or its neighboring units, such as the rename or instruction queue units. The coolest unit is typically the bus control unit. The Pentium 4 has two thermal sensors, which seem to be placed in optimal locations to account for temperature variations across the chip. While the exact locations of the sensors are unknown, it can be inferred that one is near the upper corner of the bus control unit, and the other is near the integer execution unit and acts as a catastrophic shutdown detector [12, 17]. Each thermal sensor triggers a thermal monitor when the temperature reaches a certain threshold. The thermal monitor will then regulate the processor's clock frequency, and hence reduce activity to

cool down the chip. However, consider a program that is designed to overheat certain regions that are far away from the thermal sensors. This poses a potential security risk if localized heating can occur such that the thermal sensors fail to detect noticeable rise in temperature. The experiments in the following section target the branch prediction units.

5.2. Overheating the Branch Prediction Units

The Pentium 4 has two branch prediction units –L1 BPU and L2 BPU– that are located relatively far away from the thermal sensors. Two benchmark programs are devised to increase activity in the BPUs. The first experiment uses a test program that contains approximately 90 if-statements within a large for-loop. Appendix C lists the source code for this program. The results show that the temperature of the L2 BPU increases by roughly 12 ^{o}C at the most. Figure 10-(a) contains a thermal map of the Pentium 4 at this particular instance. While the L2 BPU does heat up more than usual, the integer execution unit is still the hottest unit on the chip. The general pattern indicated in Figure10-(a) is consistent with most applications; the hot spots are located in the lower region of the chip near the integer units and the instruction queue units.

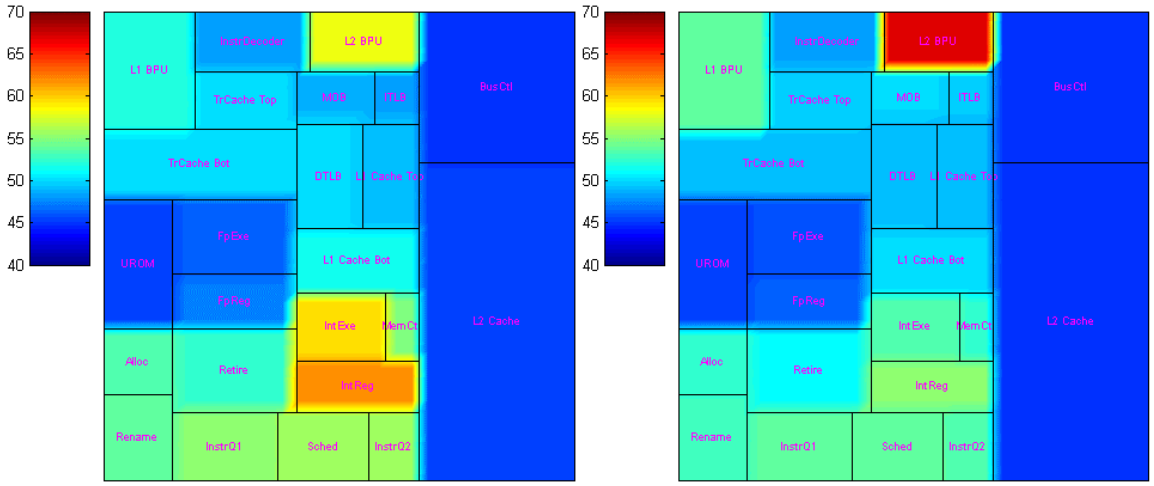

Figure 10. Thermal plot of BPU test programs. (a) experiment #1 (b) experiment #2

The second experiment slightly varies the program used in the first experiment. Within the compiled assembly code, several assembly instructions are removed, thus creating a series of conditional branch instructions inside the main loop. Appendix D includes the source code for this test program. The temperature of the L2 BPU increases by 21 ^{o}C during the program's peak execution, resulting in a temperature slightly above 67 ^{o}C. Most other units only showed an average of 3 ~ 5 ^{o}C increase in temperature. Figure 10-(b) shows the thermal map for this experiment. Note that the lower region of the chip is noticeably cooler than that in Figure 10-(a). The L2 BPU is clearly the hottest unit on the chip, where the next hottest unit –IntReg– is only 55 ^{o}C. Thus, experiment #2 demonstrates that it is possible to stress certain units while minimizing activity in other units. Further optimizations of the program may allow one to achieve higher temperature gradients. If most functional units are below normal operating temperatures, the thermal monitor may not operate or may respond too late when the chip has already been permanently damaged.

5.3. Benchmark Analysis

The results in the previous section show early work indicating the possibility of thermal security attacks on the Pentium 4 processor through *thermal viruses*. Thermal viruses are programs that may cause significant overheating to occur such that the processor is physically damaged. The test program used in experiment #2 is of special interest because of the way it significantly increases activity in the L2 BPU. Figure 11 shows how the access rates of the BPUs change across time. The sharp rise in the access rate indicates the point where the benchmark program started to run. The L2 BPU only has an 8 % access rate at steady-state, which means that it is consuming 8 % of its maximum power. When the benchmark is running, the L2 BPU is consuming more than 40 % of its maximum power.

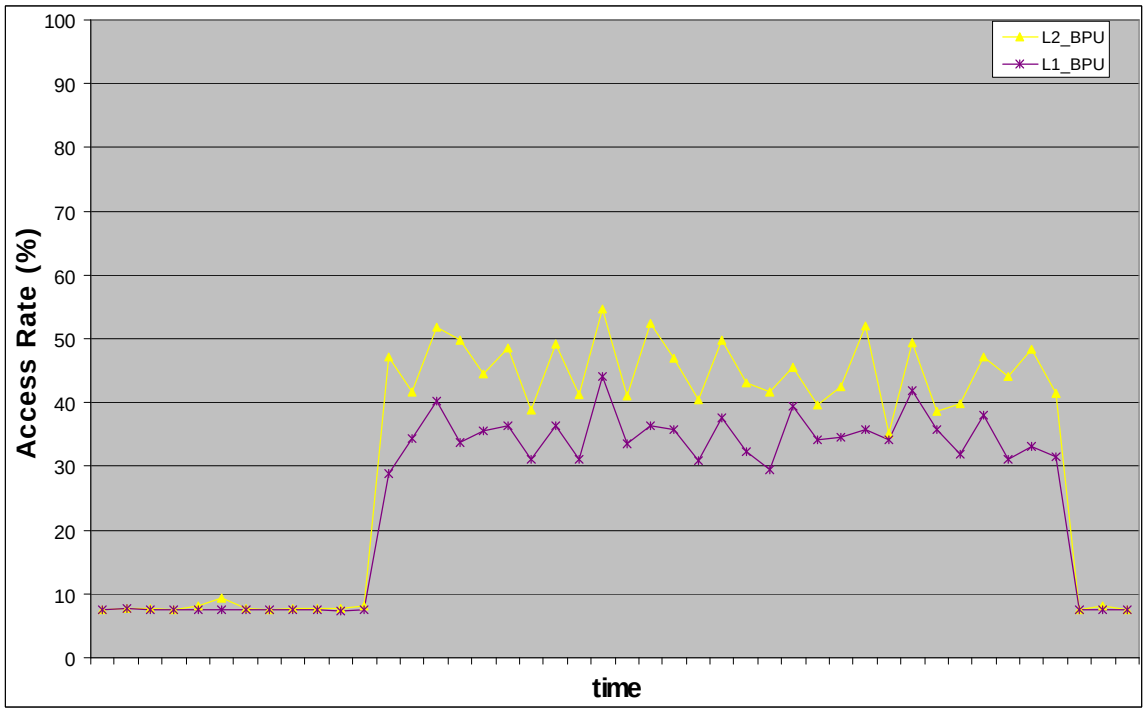

Figure 11. Access rate of BPUs (experiment #2)

Recall from section 2.2. that the thermal model estimates the power dissipated by each functional unit based on the following equation [15]:

$$Power = (MaxPower \times ArchitecturalScaling \times AccessRate) + NonGatedClockPower$$

Thus, the high increase in the access rates is manifested in higher power dissipation for the BPUs. As the results indicate, the temperature of the L2 BPU rises rapidly because of its large power density. Figure 12 shows the average power consumption of the processor for experiment #2. The power consumption of the L2 BPU increases by a factor of seven when the benchmark program is being executed. Under steady-state, the L1 BPU and L2 BPU only consume a combined total of 8 % of the total power. The percentage rises to 30 % when the benchmark is running.

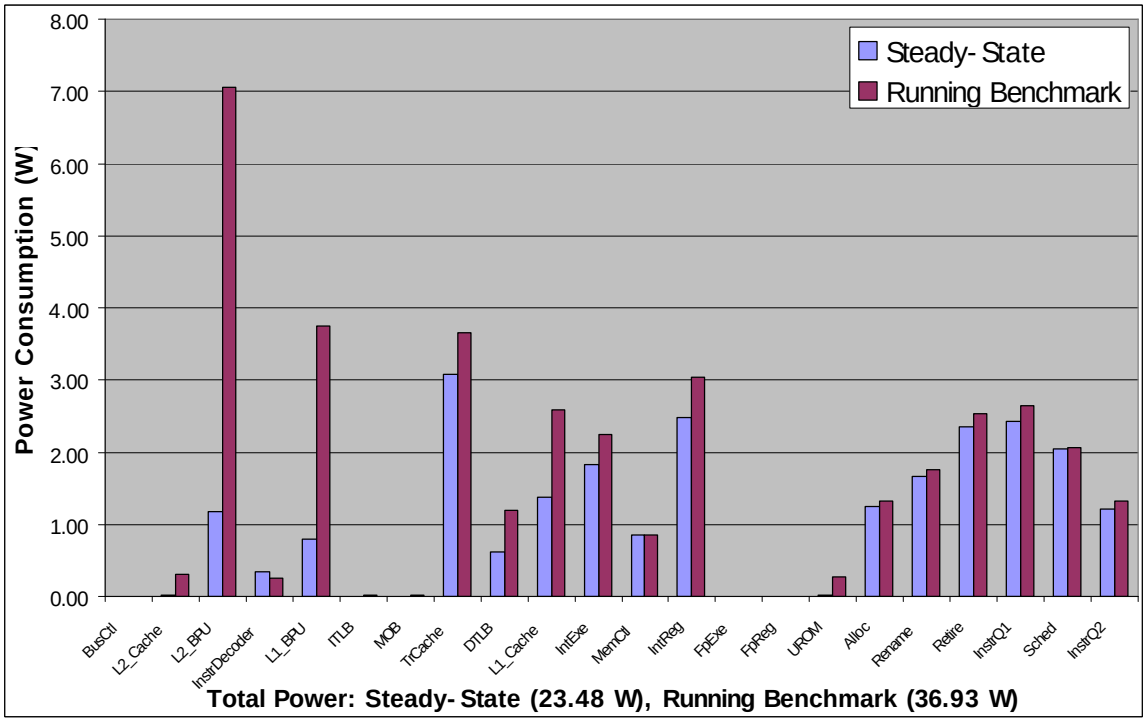

Figure 12. Average power dissipation (experiment #2)

5.4. Future Considerations

The localized heating effect discussed in section 5.2. suggests that thermal security attacks might be possible. However, the temperature value of the L2 BPU is not high enough to cause permanent damage to the chip for either experiment. The benchmark programs need to be further developed to realistically pose a big threat. The most convincing evidence could be gathered by refining the existing benchmarks to produce more thermal stress in highly concentrated areas and then observe the chip fail. Even if the chip does not fail, such experiments may help researchers understand the safety margin that is designed into the chip, and how that affects the temperature gradient as we vary the voltage or clock frequency of the system. Nonetheless, researchers can study localized heating effects by using the thermal model. These studies will also enable chip designers to build more safety mechanisms and place thermal sensors more intelligently on the chip.

6. CONCLUSIONS

6.1. Summary of Results

The goal of this project was to implement and use a runtime thermal model of a Pentium 4 processor. The implementation of the model was successful, and the resulting software solution provides detailed temperature information at the floorplan level and uses the hardware performance counters as a measure of real processor activity. The current implementation of the model adds about 1 to 4 ^{o}C of thermal overhead and can slow down the execution time of SPEC benchmarks by up to 54 %.

The software can be used for both architecture studies and online temperature sensing. The model was used in two case studies. The first study involved characterizing thermal stress patterns of various SPEC benchmarks. The results indicated that integer benchmarks tend to show more variation than floating-point benchmarks in terms of temperature changes across functional units. The thermal traces of the gcc and vortex benchmarks also showed how hot spots on the chip are not fixed but can move around during the execution of a program. The second study used the thermal model to study localized heating effects caused by thermal viruses. Two benchmark programs were

created to artificially induce high thermal stress around the L2 BPU. One benchmark program showed that the temperature of the L2 BPU increased by 20 ^{o}C while all other units only showed an average of 3 ~ 5 ^{o}C increase in temperature.

6.2. Runtime Temperature Sensing

This project shows that runtime temperature sensing is possible via software in the absence of fine-grained hardware sensors. Using hardware performance counters in the model provides several advantages over other thermal sensing methods. First, unlike purely software-based simulations that cannot account for system-wide hardware activity, the thermal model represents a more realistic view of the processor and can thus be used under real workload. Second, performance counters are present in most of today's processors, so the thermal model would be using existing hardware resources. Lastly, the fact that performance counters serve as the basis for estimating temperature indicates that computer systems do not have to rely solely on hardware sensors to employ DTM techniques. Ultimately, the thermal model could replace some of the hardware temperature sensors. Eliminating all or a few of the hardware sensors could add more flexibility to some of the rigid design requirements of processors.

Nonetheless, a great deal of future work is necessary for the thermal model to fully develop as a low-cost and reliable temperature sensing mechanism. The current implementation is insufficient for applications that require low runtime costs and high precision. The high overhead suggests that the software solution needs to be further optimized. A faster and simpler algorithm would have to replace the existing RKF

numerical solution, which is computationally too expensive. The model also needs to be further validated for accuracy. Modeling the processor in software inevitably introduces errors when certain parameters are modeled incorrectly. Few hardware parameters are even unknown, and thus force the thermal model to use rough estimates. Future research may include ways to improve the model, which can then be used as a cost-effective alternative to thermal sensors for runtime DTM techniques.

6.3. Future Architecture Studies

In addition to its use as a temperature sensing mechanism, the thermal model is also a valuable tool for computer architecture research. Chapter 4 and 5 provide a few applications of the thermal model. Understanding application-specific thermal stress patterns allows hardware designers to customize thermal-efficient processor packages, floorplan layouts, and sensor placement methods to fit a specific purpose. Hardware designers can also understand the safety margins that must be built into the processor by examining thermal security attacks. Future research areas such as these facilitate the development of more reliable and thermal-efficient computer systems. As the system's performance and stability improves, productivity will also increase and power consumption can be greatly reduced.

While the potential benefits of the thermal model are clear, the information provided by the thermal model must be handled with caution in regard to thermal security attacks. The concept of a thermal security attack is at an early stage and future work is necessary to realistically examine the risks involved. Nonetheless, information provided

by the thermal model has the potential to assist malicious programmers in creating thermal viruses. Thermal viruses would be very dangerous since they would go undetected by normal anti-virus software and would cause permanent damage to the chip. If thermal viruses prove to be possible, researchers can use the current information to provide extra safety measures in the chip. Another solution is to limit access to information regarding the thermal model or the processor. In some cases, the details of the processor or its architecture are already proprietary information. Educating the public about computing ethics is another way to alleviate concerns about thermal security attacks.

The main methodology introduced in this report can be applied to any processor. Each processor has a unique architecture, and thus would require a different thermal model. Extending the thermal model to other processors is another promising area for future research and can inspire new thermal-aware applications. One possibility is to model the IBM's Power5 processor, which has 24 thermal sensors. A new sensor-fusion algorithm can be studied using the thermal model and the 24 hardware sensors. If performance counters are used as a proxy for temperature, an optimal solution could use the sensors to calibrate the new temperature values of the thermal model only when necessary. This algorithm would effectively reduce the computation overhead of the software while using existing hardware sensors. This hybrid hardware-software temperature sensing solution may prove to be more efficient than a hardware solution of many sensors. Ultimately, the overall methodology presented in this report should stimulate many new applications that can alleviate thermal concerns in processors.

Works Cited

[1] Mabulikar, D., Pasqualoni, A., Crane, J., Braden. J.S., "Development of a Cost Effective High Performance Metal QFP Packaging System" *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 16, issue 8, pp. 902-908, Dec. 1993.

[2] Kamath, V., "Air Injection and Convection Cooling of Multi-chip Modules: A Computational Study," *Intersociety Conference on Thermal Phenomena in Electronic Systems*, Washington D.C., May 1994.

[3] Xie, H., Aghazadeh, M., Toth, J., "The Use of Heat Pipes in the Cooling of Portables with High Power Packages-A Case Study with the Pentium Processor-based Notebooks and Sub-notebooks," *Proceedings of 45th Electronic Components and Technology Conference*, Las Vegas, NV, May 1995.

[4] Bakker, A., Huijsing, J., "Micropower CMOS Temperature Sensor with Digital Output," *IEEE Journal of Solid-State Circuits*, vol. 31, issue 7, pp. 933-937, Jul. 1996.

[5] Sanchez, H., "Thermal Management System for High Performance PowerPC™ Microprocessors," *Proceedings of IEEE Compcon 97*, San Jose, CA, Feb. 1997.

[6] Lim, C., Daasch, W., Cai, G., "Thermal-aware Superscalar Microprocessor," *Proceedings of International Symposium on Quality Electronic Design*, Mar. 2002.

[7] Skadron, K., Stan, M. R., Huang, W., Velusamy, S., Sankaranarayanan, K., and Tarjan, D., "Temperature-Aware Microarchitecture," *Proceedings of 30th ACM/IEEE International Symposium on Computer Architecture*, San Diego, CA, Jun. 2003.

[8] Intel Corporation, "Intel Pentium 4 Processor in the 423-pin Package / Intel 850 Chipset Platform Design Guide," order no. 29824505, Intel Corporation, Santa Clara,

CA, 2004, Available at HTTP: http://developer.intel.com/design/chipsets/designex/298245.htm.

[9] Hinton, G., Sager, D., Upton, D., Boggs, D., Carmean, D., Kyker, A., and Roussel, P., "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal*, Q1 2001.

[10] Intel Corporation, Intel Pentium 4 technical documents, Available at HTTP: http://www.intel.com/design/Pentium4/documentation.htm.

[11] Chip Architect, Intel Pentium 4 Northwood die photo, Available at HTTP: http://www.chip-architect.com/news/2003_04_20_Looking_at_Intels_Prescott_part2.html.

[12] Intel Corporation, "IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide," order no. 253668, Intel Corporation, Santa Clara, CA, 2004, Available at HTTP: http://developer.intel.com/design/pentium4/manuals/index_new.htm#sdm_vol3.

[13] Sprunt, B., "Pentium 4 Performance Monitoring Features" *IEEE Micros*, vol. 22, no. 4, pp. 72-82, Jul.-Aug. 2002.

[14] Sprunt, B., Brink and Abyss Pentium 4 Performance Counter Tools For Linux, Feb. 2002, Available at HTTP: http://www.eg.bucknell.edu/bsprunt/emon/brink_abyss.

[15] Isci, C., and Martonosi, M., "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," *Proceedings of 36th ACM/IEEE International Symposium on Microarchitecture,* San Diego, CA, Dec. 2003.

[16] Standard Performance Evaluation Corporation, SPEC CPU2000 Benchmark, Available at HTTP: http://www.spec.org/cpu2000/.

[17] Gunther, S., Binns, F., Carmean, D.M., and Hall, J.C., “Managing the Impact of Increasing Microprocessor Power Consumption,” *Intel Technology Journal*, Q1 2001.

Bibliography

Bakker, A., Huijsing, J., "Micropower CMOS Temperature Sensor with Digital Output," *IEEE Journal of Solid-State Circuits*, vol. 31, issue 7, pp. 933-937, Jul. 1996.

Bellosa, F., Weissel, A., Waitz, M., and Kellner, S., "Event-driven Energy Accounting for Dynamic Thermal Management," *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, New Orleans, LA, Sep. 2003.

Brooks, D., and Martonosi, M., "Dynamic Thermal Management for High-Performance Microprocessors," *Seventh International Symposium on High-Performance Computer Architecture*, Monterrey, Mexico, Jan. 2001.

Chip Architect, Intel Pentium 4 Northwood die photo, Available at HTTP: http://www.chip-architect.com/news/2003_04_20_Looking_at_Intels_Prescott_part2.html

Fleischmann, M., "Crusoe Power Management: Cutting x86 Operating Power through LongRun," *Embedded Processor Forum*, Jun. 2000.

Gunther, S., Binns, F., Carmean, D.M., and Hall, J.C., "Managing the Impact of Increasing Microprocessor Power Consumption," *Intel Technology Journal*, Q1 2001.

Hinton, G., Sager, D., Upton, D., Boggs, D., Carmean, D., Kyker, A., and Roussel, P., "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal*, Q1 2001.

Intel Corporation, Intel Pentium 4 technical documents, Available at HTTP: http://www.intel.com/design/Pentium4/documentation.htm

Intel Corporation, "IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide," order no. 253668, Intel Corporation, Santa Clara, CA, 2004, Available at HTTP: http://developer.intel.com/design/pentium4/manuals/index_new.htm#sdm_vol3.

Intel Corporation, "Intel Pentium 4 Processor in the 423-pin Package / Intel 850 Chipset Platform Design Guide," order no. 29824505, Intel Corporation, Santa Clara, CA, 2004, Available at HTTP: http://developer.intel.com/design/chipsets/designex/298245.htm

Isci, C., and Martonosi, M., "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," *Proceedings of 36th ACM/IEEE International Symposium on Microarchitecture*, San Diego, CA, Dec. 2003.

Kamath, V., "Air Injection and Convection Cooling of Multi-chip Modules: A

Computational Study," *Intersociety Conference on Thermal Phenomena in Electronic Systems*, Washington D.C., May 1994.

Korn, W., Teller, P., Castillo, G., "Just How Accurate are Performance Counters," *IEEE International Conference on Performance, Computing, and Communications*, Phoenix, AZ, Apr. 2001.

Lim, C., Daasch, W., Cai, G., "Thermal-aware Superscalar Microprocessor," *Proceedings of International Symposium on Quality Electronic Design*, Mar. 2002.

Lee, K.-J., and Skadron, K., "Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors," *Proceedings of the Workshop on High-Performance, Power-Aware Computing*, Denver, CO, Apr. 2005.

Mabulikar, D., Pasqualoni, A., Crane, J., Braden. J.S., "Development of a Cost Effective High Performance Metal QFP Packaging System" *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 16, issue 8, pp. 902-908, Dec. 1993.

Mahajan, R., "Thermal management of CPUs: A perspective on trends, needs and opportunities," Keynote presentation at the *8th International Workshop on THERMal INvestigations of ICs and Systems*, Madrid, Spain, Oct. 2002.

Russell, J.T., Jacome, M.F., "Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors," *Proceedings of International Conference on Computer Design*, Austin, TX, Oct. 1998.

Sanchez, H., "Thermal Management System for High Performance PowerPC™ Microprocessors," *Proceedings of IEEE Compcon 97*, San Jose, CA, Feb. 1997.

Seng, J.S., Tullsen, D.M., "The Effect of Compiler Optimizations on Pentium 4 Power Consumption," *Proceedings of 7th Workshop on Interaction Between Compilers and Computer Architectures*, Anaheim, CA, Feb. 2003.

Skadron, K., Stan, M. R., Huang, W., Velusamy, S., Sankaranarayanan, K., and Tarjan, D., "Temperature-Aware Microarchitecture," *Proceedings of 30th ACM/IEEE International Symposium on Computer Architecture*, San Diego, CA, Jun. 2003.

Sprunt, B., Brink and Abyss Pentium 4 Performance Counter Tools For Linux, Feb. 2002, Available at HTTP: http://www.eg.bucknell.edu/bsprunt/emon/brink_abyss

Sprunt, B., "Pentium 4 Performance Monitoring Features" *IEEE Micros*, vol. 22, no. 4, pp. 72-82, Jul.-Aug. 2002.

Standard Performance Evaluation Corporation, SPEC CPU2000 Benchmark, Available at

HTTP: http://www.spec.org/cpu2000/.

Xie, H., Aghazadeh, M., Toth, J., “The Use of Heat Pipes in the Cooling of Portables with High Power Packages-A Case Study with the Pentium Processor-based Notebooks and Sub-notebooks,” *Proceedings of 45th Electronic Components and Technology Conference*, Las Vegas, NV, May 1995

Appendix A: Performance Counter Configuration

Performance Metric	Counter Rotation	Performance Counter	ECSR		CCCR	
		Number	Address	Value	Address	Value
BSQ_cache_ref	1234	0x0	Bsu0	0x180a0e0f	Bpu0	0x0003f000
uop_q_writes (0x07)	1234	0x4	Ms0	0x12000e0f	Ms0	0x00031000
TC_deliver_mode	1234	0x5	Tc0	0x0201740f	Ms1	0x00033000
uop_q_writes (0x04)	1234	0x6	Ms1	0x1200080f	Ms2	0x00031000
LD_port_replay	1234	0x9	Saat0	0x0800040f	Flame1	0x00035000
ST_port_replay	1234	0xB	Saat1	0x0a00040f	Flame3	0x00035000
uops_retired	1234	0xD	Cru0	0x0200060f	Iq1	0x00039000
front_end_event	1234	0xE	Cru3	0x1000060f	Iq2	0x0003b000
uop_type	1234	0xF	Rat1	0x04000c0f	Iq3	0x00035000
IOQ_allocation	12	0x1	Fsb0	0x07dfc20f	Bpu1	0x0107d000
FSB_data_activity	34	0x1	Fsb0	0x2e007e0f	Bpu1	0x0107d000
BPU_fetch_req	12	0x2	Bpu1	0x0600020f	Bpu2	0x00031000
MOB_ld_replay	34	0x2	Mob1	0x0600740f	Bpu2	0x00035000
ITLB_reference (0x07)	12	0x3	Itlb1	0x30000e0f	Bpu3	0x00037000
ITLB_reference (0x01)	34	0x3	Itlb1	0x3000020f	Bpu3	0x00037000
branch_retired	12	0xC	Cru2	0x0c001e0f	Iq0	0x0003b000
machine_clear	34	0xC	Cru2	0x0400020f	Iq0	0x0003b000
packed_SP_uop	1	0x8	Firm0	0x1100000f	Flame0	0x00033000
scalar_SP_uop	2	0x8	Firm0	0x1500000f	Flame0	0x00033000
64bit_MMX_uop	3	0x8	Firm0	0x0500000f	Flame0	0x00033000
x87_FP_uop	4	0x8	Firm0	0x0900000f	Flame0	0x00033000
packed_DP_uop	1	0xA	Firm1	0x1900000f	Flame2	0x00033000
scalar_DP_uop	2	0xA	Firm1	0x1d00000f	Flame2	0x00033000
128bit_MMX_uop	3	0xA	Firm1	0x3500000f	Flame2	0x00033000
x87_SIMD_moves_uop	4	0xA	Firm1	0x5c00300f	Flame2	0x00033000

Appendix B: Power Model and Performance Metrics

$$
\begin{aligned}
Power[BusCtl] &= 8.6 \times \left(\frac{ioq\_alloc}{\Delta Cycles} + BUS\_RATIO \times \frac{fsb\_data\_act}{\Delta Cycles}\right) \\
Power[L1\_Cache] &= 5.5 \times (L1\_access) \\
Power[L2\_Cache] &= 45.0 \times \left(\frac{bsq\_cache\_ref}{\Delta Cycles}\right) \\
Power[L1\_BPU] &= 10.5 \times \left(\frac{branch\_retired}{\Delta Cycles}\right) \\
Power[L2BPU] &= 15.5 \times \left(8 \times \frac{itlb\_ref\_01}{\Delta Cycles} + \frac{branch\_retired}{\Delta Cycles}\right) \\
Power[ITLB] &= 0.9 \times \left(\frac{itlb\_ref\_07}{\Delta Cycles} + \frac{bpu\_fetch\_req}{\Delta Cycles}\right) \\
Power[MOB] &= 1.6 \times \left(\frac{mob\_ld\_replay}{\Delta Cycles}\right)
\end{aligned}
$$

$$
\begin{aligned}
Power[MemCtl] &= 0.85 \times \left(1 - \frac{machine\_clear}{\Delta Cycles}\right) \\
Power[DTLB] &= 2.5 \times \left(L1\_access + \frac{mob\_ld\_replay}{\Delta Cycles}\right) \\
Power[Int\_Exe] &= \frac{3.4}{2} \times (Int\_exe) \\
Power[FP\_Exe] &= \frac{4.5}{2} \times (FP\_exe) \\
Power[Int\_Reg] &= 2.3 \times (Int\_exe) \\
Power[FP\_Reg] &= 2.3 \times (FP\_exe) \\
Power[Instr\_Decoder] &= 2.4 \times \left(\frac{tc\_deliver}{\Delta Cycles}\right) \\
Power[Tr\_Cache] &= \frac{4.0}{2.6} \times \left(\frac{uop\_q\_all}{\Delta Cycles}\right) + 2.0 \\
Power[UROM] &= 3.3 \times \left(\frac{uop\_q\_wr\_04}{\Delta Cycles}\right) \\
Power[Alloc] &= \frac{0.6}{3} \times \left(\frac{uop\_q\_all}{\Delta Cycles}\right) + 1.1 \\
Power[Rename] &= \frac{0.7}{3} \times \left(\frac{uop\_q\_all}{\Delta Cycles}\right) + 1.5 \\
Power[Instr\_Q1] &= 0.6 \times \left(\frac{uop\_q\_all}{\Delta Cycles}\right) + 2.0 \\
Power[Instr\_Q2] &= 0.3 \times \left(\frac{uop\_q\_all}{\Delta Cycles}\right) + 1.0 \\
Power[Sched] &= \frac{0.4}{6} \times \left(\frac{uop\_q\_all}{\Delta Cycles}\right) + 2.0 \\
Power[Retire] &= \frac{1.5}{3} \times \left(\frac{uops\_retired}{\Delta Cycles}\right) + 2.0
\end{aligned}
$$

Appendix C: Source Code for BPU Experiment #1

```
int main()
{
   int a=0;
   int i;
   unsigned long long int liter = 2000000000000000;

   for (i=0; i < liter; ++i) {
      a = 13*a;

      if (a < 893) a = 1223;
      if (a > 7891) a = 986;
      if (a > 3191) a = 5432;
      if (a > 1190) a = 7843;
      if (a < 439) a = 54;
      if (a < 59) a = 610;
      if (a < 49) a = 150;
      if (a < 429) a = 530;
      if (a < 859) a = 161;
      if (a > 1389) a = 1229;
      if (a < 419) a = 511;
      if (a < 333) a = 861;
      if (a > 5881) a = 1001;
      if (a < 459) a = 1060;
      if (a > 5105) a = 1111;
      if (a < 783) a = 537;
      if (a < 69) a = 70;
      if (a < 39) a = 40;
      if (a < 29) a = 30;
      if (a < 19) a = 320;
      if (a < 9) a = 10;
      if (a < 0) a = 0;
      if (a < 299) a = 300;
      if (a < 89) a = 390;
      if (a < 79) a = 480;
      if (a < 279) a = 580;
      if (a < 289) a = 1290;
      if (a < 673) a = 199;
      if (a < 99) a = 100;
      if (a < 269) a = 270;
      if (a < 259) a = 860;
      if (a < 229) a = 31;
      if (a > 5861) a = 84;
      if (a > 3821) a = 61;
      if (a < 323) a = 511;
      if (a < 203) a = 19;
      if (a < 200) a = 200;
      if (a < 489) a = 5490;
      if (a < 449) a = 5020;
      if (a < 219) a = 229;
      if (a < 209) a = 2000;
      if (a > 6439) a = 91;
```

```
        if (a > 5929) a = 169;
        if (a < 499) a = 510;
        if (a > 5339) a = 5644;
        if (a < 479) a = 38;
        if (a < 469) a = 970;
        if (a > 5003) a = 348;
        if (a < 899) a = 3;
        if (a < 889) a = 593;
        if (a < 213) a = 4001;
        if (a > 2091) a = 214;
        if (a < 409) a = 1230;
        if (a < 400) a = 9;
        if (a < 879) a = 3800;
        if (a < 849) a = 2048;
        if (a > 4851) a = 289;
        if (a > 4841) a = 8543;
        if (a < 869) a = 94;
        if (a > 4831) a = 41;
        if (a < 839) a = 541;
        if (a < 819) a = 1103;
        if (a > 639) a = 24;
        if (a > 529) a = 23;
        if (a < 249) a = 52;
        if (a < 239) a = 240;
        if (a > 519) a = 3122;
        if (a < 809) a = 1;
        if (a > 9699) a = 323;
        if (a > 9389) a = 209;
        if (a > 9179) a = 8008;
        if (a < 800) a = 465;
        if (a > 1179) a = 88;
        if (a > 1069) a = 72;
        if (a > 859) a = 261;
        if (a > 749) a = 3125;
        if (a > 8059) a = 136;
        if (a > 7049) a = 97;
        if (a < 563) a = 12345;
        if (a > 1699) a = 33;
        if (a > 509) a = 27;
        if (a > 503) a = 1021;
        if (a < 453) a = 981;
        if (a < 829) a = 53;
        if (a < 343) a = 241;
        if (a > 5871) a = 436;
        if (a > 9069) a = 762;
        if (a < 103) a = 6781;

    }
    return 0;
}
```

Appendix D: Source Code for BPU Experiment #2

```
      .file "bpu_simple.c"
      .text
.globl main
      .type main,@function
main:
      pushl %ebp
      movl  %esp, %ebp
      subl  $24, %esp
      andl  $-16, %esp
      movl  $0, %eax
      subl  %eax, %esp
      movl  $0, -4(%ebp)
      movl  $1233977344, -16(%ebp)
      movl  $465661, -12(%ebp)
      movl  $0, -8(%ebp)
.L2:
      movl  -8(%ebp), %eax
      movl  %eax, -24(%ebp)
      cltd
      movl  %edx, -20(%ebp)
      movl  -20(%ebp), %eax
      cmpl  -12(%ebp), %eax
      jb    .L5
      movl  -20(%ebp), %edx
      cmpl  -12(%ebp), %edx
      ja    .L3
      movl  -24(%ebp), %eax
      cmpl  -16(%ebp), %eax
      jb    .L5
      jmp   .L3
.L5:
      movl  -4(%ebp), %eax
      movl  %eax, %edx
      sall  $1, %edx
      addl  %eax, %edx
      sall  $2, %edx
      leal  (%eax,%edx), %eax
      movl  %eax, -4(%ebp)
      cmpl  $892, -4(%ebp)
      jg    .L7
;     movl  $1223, -4(%ebp)
.L7:  jle   .L8
.L8:  jle   .L9
.L9:  jle   .L10
.L10: jg    .L11
.L11: jg    .L12
.L12: jg    .L13
.L13: jg    .L14
.L14: jg    .L15
.L15: jle   .L16
.L16: jg    .L17
```

```
.L17: jg    .L18
.L18: jle   .L19
.L19: jg    .L20
.L20: jle   .L21
.L21: jg    .L22
.L22: jg    .L23
.L23: jg    .L24
.L24: jg    .L25
.L25: jg    .L26
.L26: jg    .L27
.L27: jns   .L28
.L28: jg    .L29
.L29: jg    .L30
.L30: jg    .L31
.L31: jg    .L32
.L32: jg    .L33
.L33: jg    .L34
.L34: jg    .L35
.L35: jg    .L36
.L36: jg    .L37
.L37: jg    .L38
.L38: jle   .L39
.L39: jle   .L40
.L40: jg    .L41
.L41: jg    .L42
.L42: jg    .L43
.L43: jg    .L44
.L44: jg    .L45
.L45: jg    .L46
.L46: jg    .L47
.L47: jle   .L48
.L48: jle   .L49
.L49: jg    .L50
.L50:
      cmpl  $5339, -4(%ebp)
      jle   .L51
      movl  $5644, -4(%ebp)
.L51: jg    .L52
.L52: jg    .L53
.L53: jle   .L54
.L54: jg    .L55
.L55: jg    .L56
.L56: jg    .L57
.L57: jle   .L58
.L58: jg    .L59
.L59: jg    .L60
.L60: jg    .L61
.L61: jg    .L62
.L62: jle   .L63
.L63: jle   .L64
.L64: jg    .L65
.L65: jle   .L66
.L66: jg    .L67
.L67: jg    .L68
.L68: jle   .L69
.L69: jle   .L70
.L70: jg    .L71
```

```
.L71: jg    .L72
.L72: jle   .L73
.L73: jg    .L74
.L74: jle   .L75
.L75: jle   .L76
.L76: jle   .L77
.L77: jg    .L78
.L78: jle   .L79
.L79: jle   .L80
.L80: jle   .L81
.L81: jle   .L82
.L82: jle   .L83
.L83: jle   .L84
.L84: jg    .L85
.L85: jle   .L86
.L86: jle   .L87
.L87: jle   .L88
.L88: jg    .L89
.L89: jg    .L90
.L90: jg    .L91
.L91: jle   .L92
.L92: jle   .L93
.L93: jg    .L94
.L94: jg    .L95
.L95: jle   .L96
.L96: jle   .L97
.L97: jle   .L98
.L98: jg    .L99
.L99: jg    .L100
.L100:      jg    .L101
.L101:      jle   .L102
.L102:      jle   .L103
.L103:      jg    .L4

.L4:
      leal  -8(%ebp), %eax
      incl  (%eax)
      jmp   .L2
.L3:
      movl  $0, %eax
      leave
      ret
.Lfe1:
      .size main,.Lfe1-main
      .ident      "GCC: (GNU) 3.2.2 20030222 (Red Hat Linux 3.2.2-5)"
```