Characterizing Thermal Behavior of Pentium-IV with Hyperthreading

A Thesis
In TCC 402

Presented to

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

In Partial Fufillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

by

Sue Kim

April 20, 2004

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 TCC Courses.

Signed _____

Approved _____ (Technical Advisor)
Technical Advisor – Dr. Kevin Skadron

Approved _____ (TCC Advisor)
TCC Advisor – Dr. Patricia C. Click

TABLE OF CONTENTS

List of Figures and Tables

Glossary

Cache: acts as a temporary storage for data, similar to memory (Silberschatz Gagne)

Central Processing Unit (CPU): the "brain" of the computer, where all the computations take place (Silberschatz Gagne)

Clock Gating: pausing the clock at intervals until the chip temperature decreases DTM mechanism used in Pentium-IV (Gunther et al.)

Cold Thread: a thread that does not utilize the CPU frequently (Smith Rohou)

Dynamic Thermal Management (DTM): controlling the temperature of the chip by clock gating, frequency scaling, and voltage scaling as the temperate of the chip increases (Gunther et al.)

Hot Thread: a thread that frequently utilizes the CPU (Smith Rohou)

Hyper-Threading: an Intel Technology, allowing each thread to have its own logical processor, therefore competing for resources and increasing performance (Intel Vol I. Intel Vol II.)

I/O: Input and Output such as mouse, keyboard, monitor (Silberschatz Gagne)

IA32_THERM_STATUS MSR: the register that holds the status of the temperature sensors on the chip (Intel Vol. II)

Kernel: core of the operating system which manages resource scheduling and thread scheduling (Silberschatz Gagne)

Linux: an operating system that is open source (Silberschatz Gagne)

Logical Processor: a virtual processor (Silberschatz Gagne)

Model Specific Registers (MSR): specific to each processor, acts as storage for data (Intel Vol II.)

MP3: compressed music format by reducing bytes (MP3)

Mpegs (Moving Picture Experts Group): digitally compressed audio and video files (MPEG)

Physical Processor: the real processor with all the components, ALU, cache, registers (Silberschatz Gagne)

Processor: another way of saying CPU (Silberschatz Gagne)

Scheduling: ordering and arranging the threads to utilize the resources in a fair manner (Silberschatz Gagne)

SPEC 2000: set of benchmarks, each benchmark has its own functionality (www.spec.org)

System Call: a function which can be called inside the kernel (Silberschatz Gagne)

Thermal Monitor: a mechanism in Pentium-4 that modulates the clock frequency when triggered by the temperature sensor (Gunther et al. Intel Vol II.)

Threads: part of a program where the flow of control is sequential (JAVA)

Trigger: the temperature sensors status changes its bit from '0' to '1' and this allows for the thermal monitor to pause the clock, which prints out 'hot' on the screen

Trigger Thread: a music file Bernard.mp3, that was used to first obtain a trigger, for the "trigger-then-run" experiment

Abstract

There is a continuous cost in cooling for processors. Therefore, there are studies in relation to packaging and Dynamic Thermal Management (DTM) Techniques. But, there is a limitation in processor design because of an increase in cooling costs. This thesis explains a new area of research, temperature-aware multithread scheduling. Four experiments were conducted: “run-then-trigger”, “trigger-then-run”, “Hyper-Threading” and DTM overhead. The results show that there is a relationship between CPU usage and thread temperatures. Also, there is interference between different temperature threads when Hyper-Threading is engaged. Temperature-aware multithread scheduling can be a way to design more complex systems, therefore creating a more productive society.

CHAPTER 1: Introduction

1.1 Purpose of Study

Our society places strong emphasis on high speed microprocessors, which significantly contribute to the performance of computers. The rate of computer performance is doubling every two years leading towards a direct increase in heat dissipation, and the increase in temperature is increasing cooling costs, limiting more complex systems from being developed. Due to the limitations in methods used for heat reduction, this thesis analyzed temperature-aware thread scheduling as one of the options in dealing with thermal management issues.

1.2 Problem Statement

In 1965, Gordon Moore proposed that the number of transistors per chip will double every eighteen months, and his observation continues to hold today. Every two years the size of the die continues to grow by 14%, and the frequency, or clock rate, is doubling. Frequency is used as a metric for computer performance, and because frequency is doubling, computer performance is also doubling. As performance doubles, power, the rate of energy consumption, also increases (Mahajan 1 - 5). Furthermore, because there is an increase in power, there is an increase in energy, and this energy gets converted into heat, and the heat is dissipated over the chip increasing chip temperature.

In order to mitigate heat dissipation, thermal management techniques both internal to the computer chip and external to the computer hardware are available. The

internal processes are known as Dynamic Thermal Management (DTM) Techniques. More specifically, researchers have found a way to manage the temperature of the computer chip through clock gating, frequency scaling, and voltage scaling. In addition, there are external devices such as packages, heat sinks, and heat pipes (Gunther et al.). The different techniques will be discussed in detail in Chapter 2, Literature Review. Both the internal and external methodologies have drawbacks. The internal methods are inexpensive, but are slow and inaccurate. On the other hand, the external devices are fast, but they are expensive. The cost of packages and heat sinks are increasing at an exponential rate. Figure 1 shows that the cost of cooling is $1.00 - $3.00 / Watt of heat Dissipation (Skadron et al. 1).

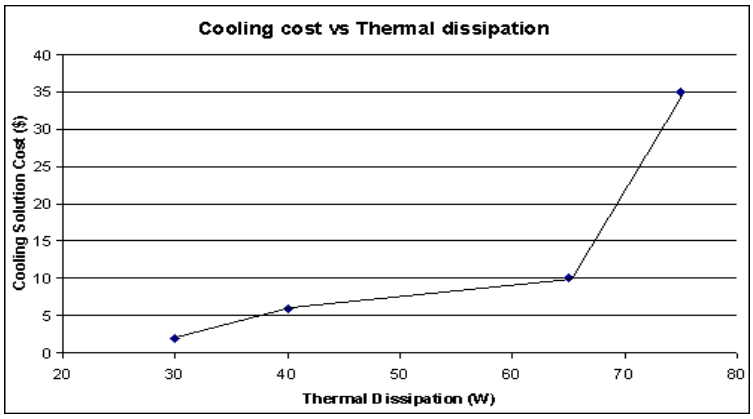

Figure 1: Cooling Cost in relation to Thermal Dissipation (Adapted from Gunther et al. 1)

1.3 Overview of Methodology

This thesis was conducted specifically on a Pentium-IV chip with a processor enabled with Hyper-Threading Technology. In order to obtain the temperature data of the chip, the status of the Thermal Model Specific Registers was observed by using privileged read instruction, RDMSR. The Thermal Model Specific Register bits were set when a change in temperature was detected and the corresponding results were analyzed.

The rest of the thesis report will be a detailed description of the experiment, an analysis of the results and the significance of the results. The second chapter will be a literature review of the past and current research of thermal management techniques. Next, the third chapter will explain the background terminology and concepts needed to understand the experiments. The fourth chapter will discuss the design procedures, and the fifth chapter will be an analysis section, which will be based upon quantitative data results. Lastly, the sixth chapter will summarize and analyze the significance of the findings and conclude the thesis report by suggesting improvements for future research.

CHAPTER 2: Literature Review

Research has been done in the past and is continuing in terms of thermal management of microprocessors. The researchers are trying to find the most efficient way to minimize heat dissipation on the chips by adding external devices to the chips and manipulating the internal signals of the chips. The development of three main external cooling techniques; packaging, heat sinks and heat pipes started in the early 90's and are still in development today.

2.1 Packaging and Heat Sinks

In 1993, researchers started studying a packaging technology MQUAD. This packaging technology had the capability to stand a power dissipation of up to 14 watts (Mahulikar 176). Heat sinks were a major component of surface mounting packages. In 1994 researchers began to focus on ways to enhance heat sinks. One study was conducted on pin-fin heat sinks, which were commonly used in desktop computers at that time. An axial flow fan directed air over the pin-fin heat sink, injecting air onto the sink (Kamath 83). Researchers continued to study ways of improving heat sinks, but by 1998 researchers were reaching a limit in terms of heat sinks in relation to package size. Therefore, they moved in a new direction in order to develop heat pipes.

2.2 Heat Pipes

During 1998, many thorough studies of heat pipe designs took place. Heat pipes are cooling devices, which maneuver heat away from obstructions and neighboring boards (Toth 449). One group of researchers decided to study the relationship between heat sinks and heat pipes. These researchers wanted to optimize the use of both the heat sinks and heat pipes to minimize power dissipation. Therefore, the researchers decided upon best ways to utilize these components. One solution was to connect the CPU and aluminum metal plate using a heat pipe, which would be placed underneath the keyboard. But, they concluded that the best thermal solution would be to connect the heat pipes to a heat dissipating plate and an external heat sink (Rujano 11).

2.3 Dynamic Thermal Management Techniques

The researchers of thermal cooling devices decided that there are limitations such as costs and size to the further advancement of these hardware devices. Therefore, they decided to utilize the internal structures of the chip, which came to be known as Dynamic Thermal Management techniques. As the operating temperature of the computer reached close to a defined threshold temperature the voltage of the chip would decrease or the frequency of the clock would be modulated.

In 2000, a study was done which manipulated the clock frequency, because frequency is a metric for performance and an increase in performance increases heat dissipation. The computer's clock stalls for a couple of cycles until the operating temperature of the computer decreases; this technique came to be known as clock gating

(Q X Pedram 479). The Pentium-4 uses clock gating as the primary DTM technique (Gunther et al.). Studies in dynamic thermal management techniques continued (Brooks Martonosi) and in 2002, one study was conducted which manipulated the voltage of the computer chip and this technique came to be known as voltage scaling. The chip was designed so that when it reached close to the threshold temperature, the "overall" voltage was decreased through the manipulation of the electrical circuitry of the chip (Chunhong Sarrafzedeh). Both techniques are much cheaper than the hardware device cooling techniques mentioned above such as packaging, heat sinks and heat pipes. But, the dynamic thermal management techniques are slow. There is a delay in the chip design because clock gating and voltage scaling are not instantaneous to the chip's increase in temperature.

2.4 Scheduling

In addition, research has been conducted in terms of scheduling of threads in the Central Processing Unit. Threads are within programs and the flow of control is sequential (Java). The studies of scheduling have ranged from single threading to simultaneous multithreading. The researchers have tried to study if performance can be increased by manipulating the threads in a single scheduling thread processor (Eggers et al. 39; Goncalves Navaux 327).

The literature review shows that there have been studies and research in both thermal management techniques and scheduling. However, very few studies have linked the two areas of research. The performance of a computer is related to scheduling of

threads, but is also related to the heat dissipated on the chip. My thesis project will create a bridge between the cooling techniques and scheduling.

CHAPTER 3: Background

This chapter will address background information such as how the thermal monitor is triggered, how to access the thermal register, how threads can have different temperatures, and Hyper-Threading Technology. An explanation of these terms will allow for a better understanding of the experiments described in chapter four.

3.1 Thermal Monitor

The Pentium-IV is designed with a mechanism known as a Thermal Monitor 1(TM1). This controls the temperature of the chip and reduces the temperature by modulating the frequency of the clock. The thermal monitor is triggered by the status of the temperature sensor. The sensor is factory calibrated to trigger the monitor when the temperature increases above the recommended design temperature; then the TM1 will stay engaged for 1 ms or until the temperature falls below the designed threshold. The status of the sensors can be read from Model Specific Registers (Gunther et al., 6-7).

3.2 Model Specific Registers

The Pentium IV chip has built-in Model Specific Registers (MSRs) . These registers are specific to the processor model type and act as storage for particular values and data. For example, in my thesis I utilized the IA32_THERM_STATUS MSR. Figure 2 shows

the arrangement of the bits for the IA32_THERM_STATUS MSR (Intel Software Architecture Manual).

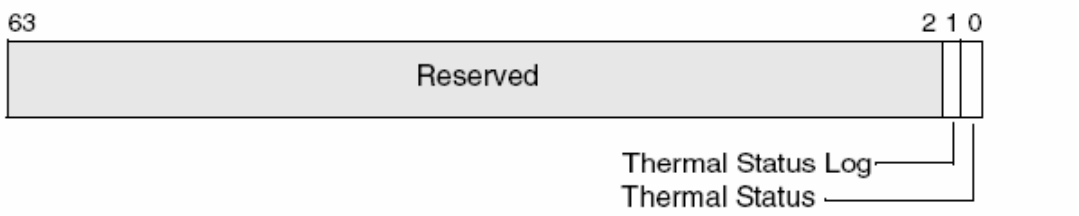

Figure 2: IA32_THERM_STATUS Model Specific Register (Adapted from Intel)

The register stores the status of the temperature sensor. The register is 64-bits; I read the 0^{th} and the 1^{st} bit to determine the temperature of the chip. The 0^{th} bit by default is set to '0', but when the bit gets set to '1' this infers that the chip temperature is hot enough to trip the thermal monitor. Also, the 1^{st} bit is set to '0' by default, but the bit is set to 1 when the thermal sensor has been tripped since the last reset or power up of the machine. Software is written to reset the 1^{st} bit to '0'(Intel Software Architecture Manual).

The RDMSR instruction was used to access the bits. This instruction reads the 64-bits of the IA32_THERM_STATUS MSR and places the most significant 32-bits into the high register and the least significant 32-bits into the low register. I created a system call, temp_stat because RDMSR can only be used in the Linux kernel and a system call is a way to use instructions inside the kernel. The system call read the IA32_THERM_STATUS using RDMSR, I parsed the low register and then I checked to see if either bit 0 or bit 1 was set to '1' (Intel Software Architecture Manual). The implementation of the system call and the user level code can be found in Appendix A.

3.3 Hot and Cold Threads

Threads are parts of a program which flow in a sequential manner (Java). Each thread can be characterized as hot or cold, a state that is determined by the thread's utilization of the CPU; the more the thread utilizes the CPU, the hotter the thread (Rohou Smith 4). Also, performance is another metric used in determining whether the thread is hot or cold. As the performance increases, the thread becomes hotter, the heat dissipation increases, and the overall temperature of the chip increases.

3.4 Hyper-Threading

The threads run on a Central Processing Unit (CPU), the brain of the computer, and are scheduled so that each thread will be able to utilize the resources of the CPU without conflicts. There are two main types of scheduling for threads: single threading and multithreading. Single threading is when only one "thread" is allowed to utilize the resources, and multithreading is when multiple threads are running concurrently, but the threads are scheduled so that there will be no conflicts between resources such as cache which is used for temporary storage and arithmetic logic unit (ALU) used to perform arithmetic operations (Silberschatz Gagne 96). Intel uses the word Hyper-Threading instead of multithreading. Hyper-Threading allows for each thread to have its own logical processor rather its own physical processor. The logical processor is a virtual processor and therefore, because there is only one physical processor, the threads are competing for resources (Intel Manual).

CHAPTER 4: Experiment

This chapter describes the design decisions, what types of experiments were conducted, and how they were conducted. The basics of the experiments consisted of invoking the system call, running threads while placing a dryer near the intake vent and then viewing the log file to count the number of triggers.

4.1 Design Decisions

The first step was to get the temperature sensor to trigger the thermal monitor. In order to trigger, an mp3 thread was selected because it is known that mp3s are hot temperature threads. I tried to run the mp3 for hours and still the chip was not hot enough to trigger the thermal monitor because of the over engineering in the packages and heat sinks for cooling. Therefore, I decided to add an external heat source, a hair dryer because it is easy to hold, maneuver and flexible to change the settings. After applying the dryer to the mp3 for about five minutes, there were several triggers from the thermal monitor.

Next, I needed to consistently and repeatedly trigger the thermal monitor. I tried mp3 music files of different duration, short duration (approx. 1min 30sec), medium duration (approx. 3min 30sec) and long duration (approx. 9min 30sec). From the different triggering results such as the number of times the thermal monitor triggered, the time it took for the first trigger and the duration of the mp3, it was found that the medium

duration mp3 was consistently and repeatedly triggering and also the duration was not too long. Therefore, I decided to use bernard.mp3, a medium file, as the trigger thread.

Following the success of triggering the thermal monitor, I had to decide how long to run each experiment. I varied the time and observed that on average it took about five minutes for the first trigger to occur with the dryer. Therefore, I decided that fifteen minutes would be enough time for the thermal monitor to trigger multiple times. Also, I realized that turning the dryer on for fifteen minutes straight would bring too much external heat, which would result in inaccurate chip temperatures. So, I decided to leave the dryer on in two minute intervals with a one minute gap in between for the duration of fifteen minutes.

The preliminary experiments helped to establish a consistent and repeatable number of triggers. Therefore, my design decisions were to use the music file bernard.mp3 as the trigger thread, run each experiment for fifteen minutes and to use a dryer. Next, I had to decide what types of threads would be used in the experiments to allow for a variety of temperature ranges of the chip.

4.2 Description of Hot and Cold threads

In order to get a range of thread temperatures and a range of the usage of the CPU, a total number of eight threads were used. The eight threads are date, find, gcc, grep, ls, make, mplayer and mult. Below is a list with a description of each thread.

date: writes the current date and time to standard output (linux manual)

find: recursively descends the directory hierarchy for each path

seeking files that match a Boolean expression (linux manual)

- 2 types of finds

  - find in a directory for file not present
  - find in all directory hierarchy for file not present

gcc: GNU project C and C++ compiler, does preprocessing, compilation, assembly and linking (linux manual)

grep: searches text files for a pattern and prints all lines that contain that pattern (linux manual)

ls: for each file that is a directory, lists the contents of the directory (linux manual)

make: maintain, update, and regenerate related programs and files, multiple gccs (linux manual)

mplayer: movie player for Linux (http://www.mplayerhq.hu/homepage/design6/news.html)

- used mplayer to play mpegs and mp3s

  - 3 different mp3, music files (bernard, willa, peace)
  - 1 mpeg, movie file (football.mpeg)

mult: multiplying two matrices

- Different Sizes (10*10, 100*100, 1k*1k, 8k*8k)

Appendix B has the command lines that were used to call each type of thread.

4.3 Design and Procedures

Four different types of experiments were conducted. The first two experiments and the last experiment described in this section used only one thread, while the third experiment involved two threads, therefore utilizing Hyper-Threading.

4.3.1 Single Thread: " Run-then-Trigger"

Listed below are the step by step procedures that were taken to conduct the "run-then-trigger" experiment.

Step 1: Acquired a Pentium-IV desktop with Hyper-Threading enabled

Step 2: Acquired Linux kernel with smp

Step 3: Acquired the seven threads

Step 4: Acquired a dryer

Step 5: Implemented temp_stat system call, in Appendix A

Step 6: Recompiled Linux kernel

Step 7: Took the computer case off

Step 8: Placed the dryer close to the middle of the intake vent

Step 9: Set the dryer setting to medium

Step 10: Invoked the system call

Step 11: Simultaneously, turned on dryer and ran a thread
(command line to run threads can be found in the Appendix B)

Step 12: Started timing and stopped after 15 minutes

Step 13: After 20 seconds, changed dryer settings to hot

Step14: Turned dryer on for two minutes and off for one minute in intervals

Step 15: Stopped the thread and the system call

Step 16: Looked in /var/log/messages and count number of hot

I repeated these steps for each of the threads.

4.3.2 Single Thread: " Trigger-then-Run"

Listed below are the steps that were taken to conduct the "trigger-then-run" experiment. Steps one to ten for this experiment are the same as in section 4.3.1. But, there are differences starting from step 11.

Step 11: Simultaneously, turned on dryer and ran bernard.mp3 (medium duration mp3)

Step 12: After 20 seconds, changed dryer settings to hot

Step 13: Turned the dryer on for two minutes and off for one minute in intervals

Step 14: Typed tail /var/log/messages and constantly checked and saw for the word 'hot' to print out on the screen

Step 14: As soon as hot printed out, stopped bernard.mp3 and ran another thread

Step 15: Continued to run this new thread for fifteen minutes

Step 16: Followed step 15 and 16 from section 4.3.1

I repeated these steps for each of the threads.

4.3.3 "Hyper-Threading" Trigger

The single thread experiments helped with deciding the combinations of threads for the "Hyper-Threading" experiment. The "Hyper-Threading" experiment is similar to the experiment described in section 4.3.1. All the steps are the same except for step 11.

Step 11: Simultaneously, turned on dryer and ran two threads
(command line to run threads can be found in the Appendix B)

To utilize Hyper-Threading, I ran two threads simultaneously instead of running only one thread. I repeated steps one thru eleven for different combinations of threads.

4.3.4 DTM Overhead

Studies have been done, as mentioned in the literature review, on different dynamic thermal management techniques. These techniques range from voltage scaling to clock gating. But, all these techniques have the limitation of slowing down the processor because it takes time for the DTMs to function on the chip. In addition, the processor slows down because DTM reduces voltage or temporarily pauses the clock. As a result, computer performance worsens. Thus, this Hyper-Threading experiment is a method to validate DTM interference with processor performance.

The "DTM Overhead" experiment involves the coldest and hottest threads, ls and mult (10 * 10). The difference between this experiment and the "Run-Then-Trigger" experiment was that the total number of iterations for the thread was printed out onto the screen instead of the word hot. Also, in this experiment each thread ran with and without the use of a dryer.

CHAPTER 5: Discussion of Results and Analysis

This chapter presents the data collected from the four experiments described in the previous chapter. Also, this chapter analyzes the implications of the data and acts as a source of validation for further research. The tables are organized to show the thread name, the number of times the thermal monitor was triggered and the temperature of each thread. The temperature was defined to be cold if the number of triggers were less than five, medium hot if the number of triggers were five or more, and hot if the number of triggers were fifteen or more. This identification of temperatures was determined by looking at the range of the number of triggers and choosing a system that would create an even distribution of thread temperatures.

5.1 Single Thread Analysis

Table 1 shows the results for the experiment that consisted of running the thread and then triggering the thermal monitor. The table shows that the majority of the thread temperatures are cold. Also, the range of the number of triggers was as small as zero to as large as twenty-two. The mp3s at least triggered on average four times. Furthermore, the four mult threads (10*10, 100*100, 1k*1k and 8k*8K) varied in terms of the number of triggers and temperature ranges but, triggered more than once. Of the four mult threads, mult(10*10) had the greatest number of triggers and it was also the smallest matrix. Conversely, mult(8k*8k) was the largest matrix and of the four this thread triggered the fewest number of times.

Programs	# Triggers	Temperature
bernard.mp3	3	cold
Date	0	cold
find2(search all .)	8,0	cold
find1 (no file.c)	0	cold
Gcc	0	cold
Grep	0	cold
Ls	0	cold
Make	1	cold
Mpeg	0	cold
mult(10*10)	22	hot
mult(100*100)	11	medium hot
mult(1k*1k)	16	hot
mult(8k*8k)	4	cold
peace.mp3	4	cold
willa.mp3	4	cold

Table 1: Single Thread: "Run then Trigger" Results

The results were not surprising; they were as expected. The threads which use the CPU more frequently were hotter than the threads which do not use the CPU as much. As a hot thread uses the CPU frequently, the temperature of the chip will increase and will trigger the thermal monitor. For example, threads like grep and gcc use the CPU infrequently because most of the time is spent in file I/O.

On the other hand, the mp3 triggered at least four times because of the relationship between CPU usage and temperature of a thread. Mp3 music files are known to be hot threads because mp3s need to be decoded and that process utilizes the CPU, therefore there was at least on average four triggers compared to the other cold threads with no triggers.

Also, the mult thread is a hot thread because of its use of the cache. The cache is part of the CPU, a temporary storage similar to memory, and the mult threads utilize the cache to store data. Mult(10*10) had the greatest number of triggers because the matrix

was small enough to fit into the cache. Therefore, all the data could be stored in the cache and there was no need to access memory. On the other hand, mult(8k*8k) was not triggering as much because the size of the matrix was too large for all the data to fit into the cache. Therefore, this thread did not utilize the cache as frequently as mult(10*10) instead this thread had to access memory.

Table 2 below shows the results for the experiment that consisted of triggering the thermal monitor and then running the thread. The table shows that there was a great amount of variation in terms of the number of triggers and the temperature of the threads. The table shows that there was a concentration of hot threads in the mult and mp3s.

Programs	# Triggers	Temperature
bernard.mp3	13	Medium hot
date	1	Cold
find2(search all .)	2	Cold
find1 (no file.c)	2	Cold
gcc	7	Medium hot
grep	0	Cold
ls	0	Cold
make	15,1	hot, cold
mpeg	0	Cold
mult(10*10)	31	Hot
mult(100*100)	28	Hot
mult(1k*1k)	23	Hot
mult(8k*8k)	2	Cold
peace.mp3	16	Hot
willa.mp3	21	Hot

Table 2: Single Thread: "Trigger then Run" Results

Once again there was a relationship between CPU usage and thread temperatures (i.e. grep cold and mult hot). The difference between table 1 and table 2 is the change in the experimental procedure. Table 1 is based upon the results when the thread ran first and then the thermal monitor was triggered, but table 2 is based upon triggering the

thermal monitor and then running the thread. Therefore, table 2 shows more of a variation in triggers and temperatures. Also, the overall number of triggers was more than the results from table 1. This is due to the fact that the thermal monitor was first triggered, which implies that the chip has already reached a high temperature. Also, another new thread starts running with the high chip temperature and therefore increases the triggers.

More importantly, the thread which ran after the thermal monitor triggered was either cold, medium hot or hot. Before, running the "trigger-then-run" experiment I predicted that if the new thread running after the chip temperature was high was cold or medium then there would be a decrease in the number of triggers. Furthermore, I predicted that if a new hot thread ran then that would increase the number of triggers.

Comparing table 1 to table 2, it is evident that all the triggers increased except for grep, ls and mpeg. There was a significant increase of triggers for gcc, make, mp3 and mult threads. In table 1 gcc had no triggers, but in table 2 gcc had seven triggers. The triggers increased because the thermal monitor was already triggered, and by adding a cold thread there was still an influence in the thermal monitor because of the delay in clock gating. Also, with the hot mp3 and mult threads there is a significant increase in the number of triggers. Both experiments validated that threads can be identified as hot or cold depending on the usage of the CPU.

5.2 Hyper-Threading Analysis

The results of the first two experiments helped to decide what combinations of cold and hot threads would be used in conducting the Hyper-Threading experiment. Table 3 shows that there was a range in the number of triggers and in temperatures. Also, by looking at all the hot temperature combinations it can be seen that the combinations involved the mult(10*10) or the mp3 thread. In addition, the last row in the table shows a combination of mult(10*10) and peace.mp3, and this row has the greatest number of triggers. On the other hand, there were some relationships in including cold threads into the combinations. For example, with the combination of bernard.mp3 and gcc, the number of triggers was zero even though bernard.mp3 is a hot thread.

Programs	# Triggers	Temperature
bernard.mp3 & gcc	0	cold
bernard.mp3 & make	7	medium hot
date & find2(search all .)	11	medium hot
find1(no file.c) & ls	3	cold
find2(search all .) & gcc	0	cold
find2(search all .) & make	6	medium hot
gcc & make	5	medium hot
gcc & willa.mp3	1	cold
grep & ls	1	cold
grep & make	13	medium hot
grep & peace.mp3	29	hot
ls & mpeg	1	cold
ls & mult(10*10)	42	hot
make & mpeg	6	medium hot
make & mult(10*10)	18	hot
make & mult(100*100)	1	cold
make & mult(1k*1k)	0	cold
make & mult(8k*8k)	7	medium hot
make & willa.mp3	16	hot
mpeg & mult(10*10)	34	hot
mpeg & peace.mp3	16	hot
mult(10*10) & mult(10*10)	59	hot
mult(10*10) & mult(1k*1k)	25	hot
mult(10*10) & peace.mp3	63	hot

Table 3: "Hyper-Threading" Trigger Results

Also, the results showed that there is an interference of different temperature threads. For example, when a hot thread ran together with another hot thread the overall temperature was only moderately hot. But, when two threads of contrasting temperature ran together, the results showed that the overall temperature became mostly colder. Appendix D includes a color coded representation of table 3 to easily identify the relationship between different combination of threads. I predicted that a hot thread in combination with a cold thread would create a hot thread because the hot thread will influence the cold thread to become hotter, but instead the combination produced a cold overall temperature. This could be due to Hyper-Threading. The two threads are competing for the resources because there is only one physical processor in Hyper-Threading. For example, a hot thread needs the CPU and a cold thread also needs to use the CPU; both threads are competing for the CPU. Also, the threads are competing for other resources such as fetch bandwidth, ALU and cache. A hot thread is not constantly using the CPU, and therefore the hot thread has time to cool down.

5.3 DTM Overhead

Table 4 shows the result of the experiment that consisted of running threads with and without the dryer to analyze DTM overhead. The table shows that the number of iterations was greater for the threads that ran without a dryer compared to the threads with a dryer. The difference in number of iterations with mult thread is 2,485,931. For the ls thread the difference is 98,294. Appendix E shows the statistical data results which validate the statistical mean difference using T-test analysis.

Thread	Trial 1: #Iter.	Trial 2: #Iter.	Trial 3: #Iter.
ls(w/out)	663722	600073	613592
ls(with)	565428	519289	575267
mult(w/out)	81979732	81222313	87629140
mult(with)	79493801	77182501	79333417

Table 4: DTM Overhead Results

The results once again validated that DTM does affect overall performance. The dryer was a way to trigger the thermal monitor and therefore engage a DTM mechanism. The results show that the threads which ran with the dryer had fewer number of iterations compared to the threads which ran without the dryer, for both the hottest thread, mult and coldest thread, ls. The fewer number of iterations for the thread with a dryer validated that DTM is slowing down the processor and reducing overall performance. The dryer is increasing heat, which starts the Pentium-IV DTM technique, where the clock is stopped for a few microseconds (Intel).

CHAPTER 6: Conclusion

6.1 Summary

The results of the experiments show the significant relationship between different combinations of cold and hot threads and the temperature variation of the chip. This thesis explores a new area of research in temperature-aware multithreaded scheduling. Studies have been done on both hot and cold threads, but only in terms of single threading. To achieve better control of chip temperature, researchers can analyze the implications of multithreading, which will lead to further research in various ways of scheduling different temperature threads. This will allow for an internal software modification rather than having to deal with the limitations of cooling costs.

6.2 Societal Implications

The thesis project will have an impact on three main groups: the system designers, the thermal management researchers, and consumers.

6.2.1 Influence on designers

System designers are reaching limitations in designing new systems because the cost of cooling is increasing rapidly. If scheduling different temperature threads can help mitigate temperature increase, therefore decreasing cooling costs, the system designers will have more freedom in designing new systems with high performance.

6.2.2 Influence on researchers

The thesis project will motivate researchers who are interested in dynamic thermal management techniques. They will recognize that the present internal and external cooling techniques are reaching limitations and there is a need for new areas of research. Furthermore, researchers who have not experimented with their ideas of thermal techniques will be motivated, encouraged, and challenged to explore temperature - aware multithread scheduling.

6.2.3 Consumers

Cost of cooling continues to increase because of expensive cooling techniques. If there is a way to reduce temperature by utilizing Hyper-Threading and not affect computer performance, manufacturers can produce a computer with higher performance at a given price point. The price reduction will attract the consumers and will increase revenue. The revenue could possibly benefit shareholders and stakeholders. In addition, the increased revenue can go to the Research and Development teams and they can develop new technology for society.

6.2.4 Ripple Effect

The further research in temperature-aware multithreaded scheduling will allow complex systems to be developed. Corporations will be able to run more intensive

simulations and computations, which will create an IT revolution and improve the technology in society.

6.3 Further Extensions

The experiments described in chapter four can be modified and improved. The experiment can be modified to alter variables such as the different types of hardware which can be stressed on the chip. For example, implement a multiplication thread that actually stresses both the floating point and integer registers and then analyze the increase and the differences in temperature. The modified experiments can use complicated benchmarks, such as SPEC 2000 to create more experimental data variation. Also, have more than two threads running together at the same time. Furthermore, instead of counting the number of times of trigger, researchers could actually measure the chip temperature under different trigger conditions. The experiment can utilize the results in chapter five by creating a temperature-aware thread-scheduling algorithm. The algorithm could identify the thread temperature, which determines the combination of threads. Lastly, the next step would be to trying to see if it is possible to trigger the thermal monitor without using an external heat source because it took a long time to get the thermal monitor to trigger.

Bibliography

Brain, M. How MP3 Files Work. HowStuffWorks. <http://entertainment.howstuffworks.com/mp31.htm>

Brooks, D., and Martonosi, M. Dynamic thermal management for high-performance Microprocessors. High-Performance Computer Architecture, 19-24 Jan. 2001.

Chunhong C., Sarrafzadeh, and M. Chen. Simultaneous voltage scaling and gate sizing for low-power design. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions. 2002.

Eggers, S.J., Emer, J.S., Leby, H.M., Lo, J.L., Stamm, R.L., and Tullsen, D.M. "Simultaneous multithreading: a platform for next-generation processors." Micro, IEEE. 17 (1997): 12 – 19.

Goncalves, R., and Navaux, P. Improving SMT performance scheduling processes Parallel, Distributed and Network-based Processing Proceedings. Spain, 2002.

Guenin, B.M., and Mahulikar, D. Methodology for the thermal characterization of the MQUAD microelectronic package. Semiconductor Thermal Measurement and Management Symposium. New Haven, CT, 1993.

Gunther, S.H., Binns, F., Carmean, D.M., and Hall, J.C. "Managing the Impact of Increasing Microprocessor Power Consumption". Intel Technology Journal Q1 (2001): 1-8.

"Hyper-Threading Technology". Intel Corporation. 28 March. 2004 <http://www.intel.com/technology/hyperthread/index.htm?iid=sr+hyper&>.

Intel Corporation.The IA-32 Intel Architecture Software Developer's Manual: Basic Architecture. Vol. I. 2004.

Intel Corporation.The IA-32 Intel Architecture Software Developer's Manual: System Programming Guide. Vol. II. 2004.

Kamath, V. Air injection and convection cooling of multi-chip modules: a computational study . I-THERM IV. 'Concurrent Engineering and Thermal Phenomena'., InterSociety Conference. Washington, D.C., 1994.

Mahajan, R. "Thermal Management of CPUs : A Perspective on Trends, Needs and Opportunities". THERMINIC Workshop, 1-4 Oct 2002.

Mahulikar, D., Pasqualoni, A., Crane, J., and Braden, J. Development of a cost effective high performance metal QFP packaging system. Electronic Components and Technology Conference. New Haven, CT, 1993.

MPEG Pointers and Resources. May 2000. <http://www.mpeg.org>.

Rohou E., and Smith, M.D. Dynamically Managing Processor Temperature and Power. 2nd Workshop of Feedback Directed Optimization, 1999.

Rujano, J.R., Cardenas, R., Rahman, M.M., and Moreno, W.A. Development of a thermal management solution for a ruggedized Pentium based notebook computer. Thermal and Thermomechanical Phenomena in Electronic Systems Conference. Seattle, WA, 1998.

Silberschatz, G., and Greg G. Operating System Concepts. New York: John Wiley & Sons, 2003.

Skadron, K., Stan,M.R., Huang, W., Velusamy, S., Sankaranarayanan K., and Tarjan D. Temperature Aware Microarchitecture. Proceedings of the 30th International Symposium on Computer Architecture. San Diego, CA, 2003.

The One Page Linux Manual. A summary of useful Linux commands. May 1999. < http://homepage.powerup.com.au/~squadron/linuxmanual.pdf>.

Toth, J., DeHoff, R., and Grubb, K. Heat pipes: the silent way to manage desktop thermal Problems. Thermal and Thermomechanical Phenomena in Electronic Systems Conference. Seattle, WA, 1998.

What is a thread. The JAVA Tutorial. <http://java.sun.com/docs/books/tutorial/essential/threads/definition.html>.

Wu, Q., Wu, X., and Pedram, M. Clock-gating and its application to low power design of sequential circuits. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions, 2000.

Xie, H., Aghazadeh, M., and 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. Electronic Components and Technology Conference. Chandler, AZ, 1995.

Appendix A : System Call Implementation

Modifications of System Call Table

Entry.S

```
.long SYMBOL_NAME(sys_ni_syscall)
        .long SYMBOL_NAME(sys_ni_syscall)      /* 255 sys_epoll_ctl */
        .long SYMBOL_NAME(sys_ni_syscall)      /* sys_epoll_wait */
        .long SYMBOL_NAME(sys_ni_syscall)      /* sys_remap_file_pages */
        .long SYMBOL_NAME(sys_set_tid_address)
        .long SYMBOL_NAME(sys_temp_stat)   /*  added code to test system call, 259*/
```

Unistd.h

```
#define __NR_exit_group             252
#define __NR_lookup_dcookie         253
#define __NR_set_tid_address        258
#define __NR_temp_stat          259     /* added this new syscall */

/* user-visible error numbers are in the range -1 - -124: see <asm-i386/errno.h> */

#define __syscall_return(type, res) \
```

User Code

```
#include <linux/unistd.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

#define __NR_temp_stat 259
_syscall0(int, temp_stat);


int main() {

  while(1){

    temp_stat();
    sleep(3);
  }
  return 0;

  }
```

System Call Implementation

```
#include <linux/linkage.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/unistd.h>
#include <linux/kernel.h>
#include <asm/temp_stat.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/config.h>
#include <linux/smp.h>
#include <asm/processor.h>
#include <asm/system.h>
#include <asm/msr.h>
//#include <linux/resource.h>


#define rdmsr(msr,val1,val2)__asm__ __volatile__("rdmsr" : "=a" (val1), "=d" (val2):
"c" (msr))
```

```
#define wrmsr(msr,val1,val2)__asm__ __volatile__("wrmsr" : /* no outputs */ : "c"
(msr), "a" (val1), "d" (val2))
asmlinkage int sys_temp_stat(void)
{

        unsigned int low, high;


        rdmsr( MSR_IA32_THERM_STATUS, low, high);


        // isolate LSB, if a 1, then processor hot

        if (low & 1 ){

                printk ("Hot: %x\n", low);

        }

        else if (low & 2) {


          printk( "Sticky: %x\n", low);

          low &= ~2;

          printk( " Reset: %x\n", low );


        }

        else {

                printk ("Cold: %x/n", low);
        }

  return 0;
}
```

Makefile for temp_stat

```
#
# Makefile for the linux ipc.
#
# Note! Dependencies are done automagically by 'make dep', which also
# removes any old dependencies. DON'T put your own dependencies here
# unless it's something special (ie not a .c file).
#
# Note 2! The CFLAGS definition is now in the main makefile...

O_TARGET := ipc.o

obj-y   := util.o temp_stat.o

obj-$(CONFIG_SYSVIPC) += msg.o sem.o shm.o

include $(TOPDIR)/Rules.make
```

Appendix B : Shell Scripts and Command Lines

Date.sh

```
#!/bin/sh
while [ 1 ]
do
date
done
```

command line: ./date.sh

Find.sh

```
#!/bin/sh
while [ 1 ]
do
find /home/ssk4s/linux-2.4.20-24.9/include/config/scsi/generic/ncr5380 -name module.c -
print
done
```

command line: ./find.sh

Find2.sh

```
#!/bin/sh
while [ 1 ]
do
find . -name module.c -print
done
```

command line: ./find2.sh

gcc.sh

```
#!/bin/sh
while [ 1 ]
do
  gcc –c test.c
  echo "Looping through gcc"
  done
```

command line: ./gcc.sh

grep.sh

```
#!/bin/sh
while [ 1 ]
do
grep 'NCR5381' /home/ssk4s/linux-2.4.20-
24.9/include/config/scsi/generic/ncr5380/module.h
done
```

command line: ./grep.sh

ls.sh

```
#!/bin/sh
while [ 1 ]
do
  let "m+=1"
  echo "$m"
  ls
done
```

command line: ./ls.sh

make.sh

```
#!/bin/sh
while [ 1 ]
do
  Make
  echo "Looping through gcc"
done
```

command line: ./make.sh

mplayer

command line: mplayer bernard.mp3

command line: mplayer football.mpeg

mult(10* 10)

```
#define N 10
int A[N][N];
int B[N][N];
int C[N][N];

int main(void) {

  while(1){
  int i,j,k,m;
  m++;
  printf(" %i\n", m);
  for(k=0; k<N; k++) {
    for(j=0; j<N; j++) {
      for(i=0; i<N; i++) {
           C[i][k] += A[i][j] * B[j][k];
       }
    }
  }
  }
  return 0;
}
```

command line: ./mult10.c

mult (100*100)

```
#define N 100
```

command line: ./mult100.c

mult(1K*1K)

```
#define N 1000
```

command line: ./mult1k.c

mult(8k*8K)

#define N 8846
command line: ./mult8k.c

Appendix C : Log Files

Mar 15 16:08:35 slarnaq kernel: Cold: 0/nCold:
Mar 15 16:09:44 slarnaq kernel: 0/nCold: 0/nSticky: 2
Mar 15 16:09:47 slarnaq kernel: Sticky: 2
Mar 15 16:09:50 slarnaq kernel: Sticky: 2
Mar 15 16:09:53 slarnaq kernel: Hot: 3
Mar 15 16:09:56 slarnaq kernel: Sticky: 2
Mar 15 16:10:17 slarnaq last message repeated 7 times
Mar 15 16:10:20 slarnaq kernel: Hot: 3
Mar 15 16:10:23 slarnaq kernel: Hot: 3
Mar 15 16:10:26 slarnaq kernel: Sticky: 2

Mar 15 16:52:11 slarnaq kernel: Cold: 0/nSticky: 2
Mar 15 16:52:14 slarnaq kernel: Sticky: 2
Mar 15 16:52:17 slarnaq kernel: Hot: 3
Mar 15 16:52:20 slarnaq kernel: Sticky: 2
Mar 15 16:52:23 slarnaq kernel: Sticky: 2
Mar 15 16:52:26 slarnaq kernel: Hot: 3
Mar 15 16:52:29 slarnaq kernel: Sticky: 2
Mar 15 16:52:35 slarnaq last message repeated 2 times
Mar 15 16:52:38 slarnaq kernel: Hot: 3
Mar 15 16:52:44 slarnaq last message repeated 2 times
Mar 15 16:52:47 slarnaq kernel: Sticky: 2
Mar 15 16:53:05 slarnaq last message repeated 6 times

Mar 15 16:53:08 slarnaq kernel: Hot: 3
Mar 15 16:53:26 slarnaq last message repeated 6 times
Mar 15 16:53:29 slarnaq kernel: Sticky: 2
Mar 15 16:53:32 slarnaq kernel: Hot: 3
Mar 15 16:53:35 slarnaq kernel: Hot: 3
Mar 15 16:53:38 slarnaq kernel: Sticky: 2
Mar 15 16:53:41 slarnaq kernel: Sticky: 2

Mar 15 16:08:35 slarnaq kernel: Cold: 0/nCold:
Mar 15 16:09:44 slarnaq kernel: 0/nCold: 0/nSticky: 2
Mar 15 16:09:47 slarnaq kernel: Sticky: 2
Mar 15 16:09:50 slarnaq kernel: Sticky: 2
Mar 15 16:09:53 slarnaq kernel: Hot: 3
Mar 15 16:09:56 slarnaq kernel: Sticky: 2
Mar 15 16:10:17 slarnaq last message repeated 7 times
Mar 15 16:10:20 slarnaq kernel: Hot: 3
Mar 15 16:10:23 slarnaq kernel: Hot: 3
Mar 15 16:10:26 slarnaq kernel: Sticky: 2

Mar 30 12:57:35 slarnaq kernel: Cold: 0/nCold:

Mar 30 12:57:38 slarnaq kernel: 0/nSticky: 2
Mar 30 12:57:38 slarnaq kernel: Reset: 0
Mar 30 12:57:41 slarnaq kernel: Sticky: 2
Mar 30 12:57:41 slarnaq kernel: Reset: 0
Mar 30 12:57:44 slarnaq kernel: Sticky: 2
Mar 30 12:57:44 slarnaq kernel: Reset: 0
Mar 30 12:57:47 slarnaq kernel: Sticky: 2
Mar 30 12:57:47 slarnaq kernel: Reset: 0
Mar 30 12:57:50 slarnaq kernel: Sticky: 2
Mar 30 12:57:50 slarnaq kernel: Reset: 0
Mar 30 12:57:53 slarnaq kernel: Sticky: 2
Mar 30 12:57:53 slarnaq kernel: Reset: 0
Mar 30 12:57:56 slarnaq kernel: Hot: 3
Mar 30 12:57:59 slarnaq kernel: Sticky: 2
Mar 30 12:57:59 slarnaq kernel: Reset: 0
Mar 30 13:04:57 slarnaq kernel: Sticky: 2
Mar 30 13:04:57 slarnaq kernel: Reset: 0
Mar 30 13:05:00 slarnaq kernel: Hot: 3
Mar 30 13:05:03 slarnaq kernel: Sticky: 2
Mar 30 13:05:03 slarnaq kernel: Reset: 0
Mar 30 13:05:06 slarnaq kernel: Sticky: 2
Mar 30 13:05:06 slarnaq kernel: Reset: 0
Mar 30 13:05:09 slarnaq kernel: Sticky: 2
Mar 30 13:05:09 slarnaq kernel: Reset: 0
Mar 30 13:05:12 slarnaq kernel: Sticky: 2
Mar 30 13:05:12 slarnaq kernel: Reset: 0
Mar 30 13:05:15 slarnaq kernel: Sticky: 2
Mar 30 13:05:15 slarnaq kernel: Reset: 0
Mar 30 13:05:18 slarnaq kernel: Sticky: 2
Mar 30 13:05:18 slarnaq kernel: Reset: 0
Mar 30 13:05:21 slarnaq kernel: Sticky: 2
Mar 30 13:05:21 slarnaq kernel: Reset: 0
Mar 30 13:05:24 slarnaq kernel: Hot: 3
Mar 30 13:05:27 slarnaq kernel: Sticky: 2
Mar 30 13:05:27 slarnaq kernel: Reset: 0
Mar 30 13:05:30 slarnaq kernel: Sticky: 2
Mar 30 13:05:30 slarnaq kernel: Reset: 0
Mar 30 13:05:33 slarnaq kernel: Sticky: 2
Mar 30 13:05:33 slarnaq kernel: Reset: 0
Mar 30 13:05:36 slarnaq kernel: Hot: 3
Mar 30 13:05:39 slarnaq kernel: Sticky: 2
Mar 30 13:05:39 slarnaq kernel: Reset: 0
Mar 30 13:05:42 slarnaq kernel: Hot: 3
Mar 30 13:05:45 slarnaq kernel: Sticky: 2
Mar 30 13:05:45 slarnaq kernel: Reset: 0
Mar 30 13:05:48 slarnaq kernel: Sticky: 2

Mar 30 13:05:48 slarnaq kernel: Reset: 0

Mar 30 12:58:14 slarnaq kernel: Reset: 0
Mar 30 12:58:17 slarnaq kernel: Sticky: 2
Mar 30 12:58:17 slarnaq kernel: Reset: 0
Mar 30 12:58:20 slarnaq kernel: Hot: 3
Mar 30 12:58:23 slarnaq kernel: Hot: 3
Mar 30 12:58:26 slarnaq kernel: Sticky: 2
Mar 30 12:58:26 slarnaq kernel: Reset: 0
Mar 30 12:58:29 slarnaq kernel: Sticky: 2
Mar 30 12:58:29 slarnaq kernel: Reset: 0
Mar 30 12:58:32 slarnaq kernel: Hot: 3
Mar 30 12:58:35 slarnaq kernel: Sticky: 2
Mar 30 12:58:35 slarnaq kernel: Reset: 0
Mar 30 12:58:38 slarnaq kernel: Sticky: 2
Mar 30 12:58:38 slarnaq kernel: Reset: 0

Appendix D :Color Coded Hyper-Threading Results

Appendix D is a color coded scheme of Table 1 – Table 4 shown in chapter four. The charts are organized to emphasize the "Hyper-Threading" experiment. Exp. 1 refers to the first "Run-then-Trigger" experiment, Exp. 2 refers to the second "Trigger-then-Run experiment and Exp. 3 is the third "Hyper-Threading" experiment. All the individual threads were used in all three experiments. But, for each experiment the same thread temperature varied. For example, for make thread: Exp. 1 identified the thread to be cold, Exp. 2 identified the thread to be medium and Exp. 3 identified the thread to be hot.

	Exp. 1	Exp. 2	Exp. 3
grep	COLD	COLD	HOT
peace	COLD	HOT	HOT
ls	COLD	COLD	HOT
mult(10*10)	HOT	HOT	HOT
make	COLD	MEDIUM	HOT
mult(10*10)	HOT	HOT	HOT
make	COLD	MEDIUM	HOT
willa.mp3	COLD	HOT	HOT
mpeg	COLD	COLD	HOT
mult(10*10)	HOT	HOT	HOT
mpeg	COLD	COLD	HOT
peace.mp3	COLD	HOT	HOT
mult(10*10)	HOT	HOT	HOT
mult(10*10)	HOT	HOT	HOT
mult(10*10)	HOT	HOT	HOT
mult(1k*1k)	HOT	HOT	HOT
mult(10*10)	HOT	HOT	HOT
peace.mp3	COLD	HOT	HOT

	Exp. 1	Exp. 2	Exp. 3
bernard.mp3	COLD	HOT	COLD
gcc	COLD	MEDIUM	COLD
date	COLD	COLD	COLD
gcc	COLD	MEDIUM	COLD
find1	COLD	COLD	COLD
ls	COLD	COLD	COLD
find2	COLD	COLD	COLD
gcc	COLD	MEDIUM	COLD
gcc	COLD	MEDIUM	COLD
willa.mp3	COLD	HOT	COLD
grep	COLD	COLD	COLD
ls	COLD	COLD	COLD
ls	COLD	COLD	COLD
mpeg	COLD	COLD	COLD
make	COLD	MEDIUM	COLD
mult(100*100)	MEDIUM	HOT	COLD
make	COLD	MEDIUM	COLD
mult(1k*1k)	HOT	HOT	COLD

	Exp. 1	Exp. 2	Exp. 3
bernard.mp3	COLD	HOT	MEDIUM
make	COLD	MEDIUM	MEDIUM
date	COLD	COLD	MEDIUM
find2	COLD	COLD	MEDIUM
find2	COLD	COLD	MEDIUM
make	COLD	MEDIUM	MEDIUM
gcc	COLD	MEDIUM	MEDIUM
make	COLD	MEDIUM	MEDIUM
grep	COLD	COLD	MEDIUM
make	COLD	MEDIUM	MEDIUM
make	COLD	MEDIUM	MEDIUM
mpeg	COLD	COLD	MEDIUM
make	COLD	MEDIUM	MEDIUM
mult(8k*8k)	COLD	COLD	MEDIUM

Appendix E : DTM Experiment, T-Test Results

Appendix E shows the T-test results for the different variables. The T-tests involve different combination of the ls and mult threads with and without the use of a dryer. Overall, the two-tail probability shows that P(T<=t) two-tail > 0.05 which means that the two variables statistically analyzed do not have a different distribution. But, when ls and mult threads with and without dryer were statistically analyzed there was a different distribution because P(T<=t) two tail < 0.05.

ls without and with dryer

t-Test: Paired Two Sample for Means

	ls (w/out)	*ls (with)*
Mean	625795.6667	553328
Variance	1124495910	893191621
Observations	3	3
Pearson Correlation	0.532206121	
Hypothesized Mean Differ.	0	
df	2	
t Stat	4.070315304	
P(T<=t) one-tail	0.027695884	
t Critical one-tail	2.91998731	
P(T<=t) two-tail	0.055391767	
t Critical two-tail	4.302655725	

t-Test: Two-Sample Assuming Unequal Variances

	ls (w/out)	*ls (with)*
Mean	625795.7	553328
Variance	1.12E+09	893191621
Observations	3	3
Hypothesized Mean Differ.	0	
df	4	
t Stat	2.794332	
P(T<=t) one-tail	0.024547	
t Critical one-tail	2.131846	
P(T<=t) two-tail	0.049095	
t Critical two-tail	2.776451	

mult without and with dryer

t-Test: Paired Two Sample for Means

	mult(w/out)	*mult(with)*
Mean	83610395	78669906.33
Variance	1.22562E+13	1.66571E+12
Observations	3	3
Pearson Correlation	0.539473852	
Hypothesized Mean Differ.	0	
df	2	
t Stat	2.844999597	
P(T<=t) one-tail	0.052266	
t Critical one-tail	2.91998731	
P(T<=t) two-tail	0.104532	
t Critical two-tail	4.302655725	

t-Test: Two-Sample Assuming Unequal Variances

	mult(w/out)	*mult(with)*
Mean	83610395	78669906.33
Variance	1.23E+13	1.66571E+12
Observations	3	3
Hypothesized Mean Differ.	0	
df	3	
t Stat	2.293411	
P(T<=t) one-tail	0.052813	
t Critical one-tail	2.353363	
P(T<=t) two-tail	0.105625	
t Critical two-tail	3.182449	

ls and mult without dryer

t-Test: Paired Two Sample for Means

	ls(w/out)	*mult(w/out)*
Mean	625795.6667	83610395
Variance	1124495910	1.23E+13
Observations	3	3
Pearson Correlation	-0.21065599	
Hypothesized Mean Differ.	0	
Df	2	
t Stat	-40.9719490	
P(T<=t) one-tail	0.000297584	
t Critical one-tail	2.91998731	
P(T<=t) two-tail	0.000595167	
t Critical two-tail	4.302655725	

t-Test: Two-Sample Assuming Unequal Variances

	ls(w/out)	*mult(w/out)*
Mean	625795.6667	83610395
Variance	1124495910	1.22562E+13
Observations	3	3
Hypothesized Mean Differ.	0	
Df	2	
t Stat	-41.0545309	
P(T<=t) one-tail	0.000296389	
t Critical one-tail	2.91998731	
P(T<=t) two-tail	0.000592777	
t Critical two-tail	4.302655725	

ls and mult with dryer

t-Test: Paired Two Sample for Means

	ls(with)	*mult(with)*
Mean	553328	78669906
Variance	893191621	1.67E+12
Observations	3	3
Pearson Correlation	0.974225626	
Hypothesized Mean Differ.	0	
df	2	
t Stat	-107.252504	
P(T<=t) one-tail	4.34609E-05	
t Critical one-tail	2.91998731	
P(T<=t) two-tail	8.69218E-05	
t Critical two-tail	4.302655725	

t-Test: Two-Sample Assuming Unequal Variances

	ls(with)	*mult(with)*
Mean	553328	78669906.33
Variance	893191621	1.66571E+12
Observations	3	3
Hypothesized Mean Differ.	0	
df	2	
t Stat	-104.806329	
P(T<=t) one-tail	4.5513E-05	
t Critical one-tail	2.91998731	
P(T<=t) two-tail	9.1026E-05	
t Critical two-tail	4.302655725	