6-6-2014

A Multi-core Testbed on Desktop Computer for Research on Power/Thermal Aware Resource Management

Ashley Dierivot
adier001@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A MULTI-CORE TESTBED ON DESKTOP COMPUTER FOR RESEARCH ON POWER/THERMAL AWARE RESOURCE MANAGEMENT

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Ashley Dierivot

2014

To: Dean Amir Mirmiran
College of Engineering and Computing

This thesis, written by Ashley Dierivot, and entitled A Multi-Core Testbed on Desktop Computer for Research on Power/Thermal Aware Resource Management, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Ismail Guvenc

Ming Zhao

Gang Quan, Major Professor

Date of Defense: June 6, 2014

The thesis of Ashley Dierivot is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2014

DEDICATION

I would like to dedicate this Masters thesis to my parents, my brothers and my sister. They have been with me and have supported me since day one. Without them none of this would have been possible.

ACKNOWLEDGMENTS

First, I would like express my deepest and greatest thanks to my major professor Dr. Gang Quan for his motivation, guidance, encouragement, and patience. His dedication and sincere interests in scientific research as not only fueled my interests for research but also my interests in self improvement on my technical skills. He has been a great inspiration to me. I would also like to thank Dr. Ismail Guvenc and Dr. Ming Zhao for their patience and guidance. Their helpful tips, comments and questions, helped me address many issues that I myself would have missed. Finally I would like to thank the guys at the ARCS lab, Gustavo Chaparro, Ming Fan, Tianyi Wang, Shi Sha, Shuo Liu, Qiushi Han, and Soamar Homsi. For the short time that I was in the lab they treated me as one of their own constantly providing an endless stream of helpful tips, information and, most of all, motivation. I'll never forget the times spent in the ARCS lab.

I would also like to thank the National Science Foundation (NSF) for supporting the research described in this dissertation through grants

CNS-0917021 and CNS-1018108.

ABSTRACT OF THE DISSERTATION

A MULTI-CORE TESTBED ON DESKTOP COMPUTER FOR RESEARCH ON POWER/THERMAL AWARE RESOURCE MANAGEMENT

by

Ashley Dierivot

Florida International University, 2014

Miami, Florida

Professor Gang Quan, Major Professor

Our goal is to develop a flexible, customizable, and practical multi-core testbed based on an Intel desktop computer that can be utilized to assist the theoretical research on power/thermal aware resource management in design of computer systems. By integrating different modules, i.e. thread mapping/scheduling, processor/core frequency and voltage variation, temperature/power measurement, and run-time performance collection, into a systematic and unified framework, our testbed can bridge the gap between the theoretical study and practical implementation. The effectiveness for our system was validated using appropriately selected benchmarks. The importance of this research is that it complements the current theoretical research by validating the theoretical results in practical scenarios, which are closer to that in the real world. In addition, by studying the discrepancies of results of theoretical study and their applications in real world, the research also aids in identifying new research problems and directions.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The area of embedded systems continues to advance at an accelerating pace. Until recently, these embedded systems were developed using complex uniprocessors. However, the use of uniprocessors have diminished due to reasons including but not limited to: increase in heat power consumption/generation, diminishing instruction-Level parallelism (ILP) gains, memory bottlenecking, and the inherent complexity of designing a single core with a large number of transistors. Increasing the operating frequency and developing more complex uniprocessors is no longer an effective way to improve performance. To this end, industry usage of multicore processors or chip multiprocessors (CMP) has become much more widespread [20, 44, 67, 71].

The rapid increase in the raw performance offered by contemporary multicore architectures comes with conditions in the form of higher power dissipations and thermal implications [4, 65]. As an example, Figure 1.1 illustrates the power consumption trend of system on chip (SoC) computing systems between the years 2009 and 2024. As indicated in Figure 1.1 the primary challenge of power consumption is the exponential rise of power consumption.

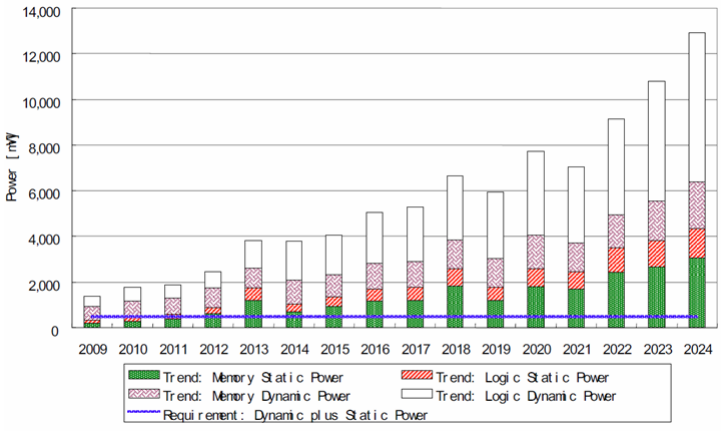

Figure 1.1: Power Consumption Trend [2]

1.1 Power/Thermal Multi-Core Design Challenges

Prior to the discussion of the power/thermal multi-core design challenges it is important to understand what Chip Multiprocessors, or CMPs, are. CMPs consist of several processor cores on a single die, each equipped with their own cache. Multiple applications can be independently run on each core of a CMP, or a single application can be split into several parallel threads and can be executed on the cores simultaneously to increase the throughput without increasing the clock rate [39]. These CMPs with their heightened number of transistors and more complex computer architectures are being designed to deliver exponentially higher peak computing performance. Figure 1.2 illustrates how more and more transistors and computing cores are integrated onto a single chip for higher performance.

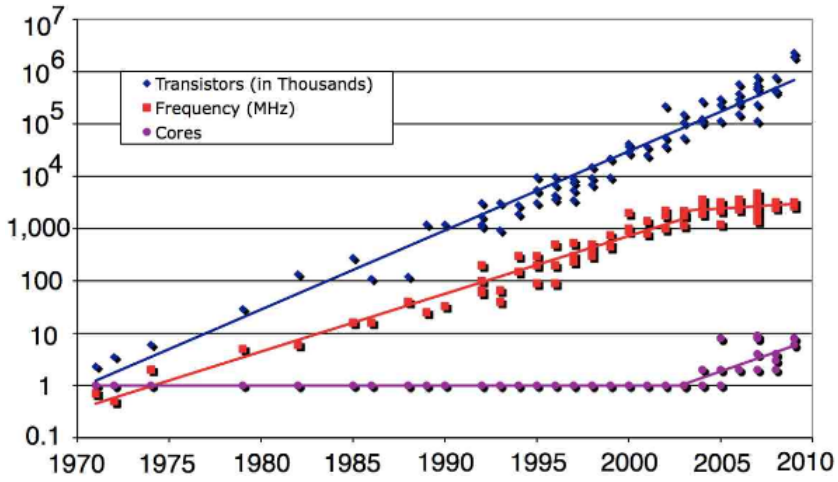

Figure 1.2: Moore's Law [55]

1.1.1 Power Challenges

Multiple platforms including portable devices and the power-rich platforms face issues due to higher power consumption. On portable or mobile devices, battery lifetime remains a primary design constraint for mobile embedded systems as developers

must carefully balance higher computing performance with efficient use of the battery source. Research [13, 54] indicates that this increased level complexity of mobile devices has caused the power consumption of batteries to rise exponentially. For the power-rich platform such as servers, super computers, etc, high power consumption is also a problem due to high cost and environmental concerns. Over the past ten years, server rated power consumptions have increased by nearly 10x. A 30,000 square feet 10MW data center can require up to 5 million dollars of cooling infrastructure [59]. From an environmental point of view the 2007 estimate of 59 billion KWhrs spent in U.S. servers and data centers translates to several million tons of coal consumption and greenhouse gas emission per year.

1.1.2 Thermal Challenges

Another adverse effect that higher power consumption causes is the increase in temperature which, in turn, causes high cooling costs, poor reliability, and performance degradation [14]. This increase in cooling costs then adversely affects the development of computing systems. In fact, it is estimated that the thermal packaging increases the total packaging cost at 1-3 dollar(s) per watt [66]. Moreover, due to this decrease in reliability and performance, it has been reported that more than 50 percent of all IC failures are related to thermal issues which are a result of power issues [56]. These reports are further illustrated by the finding that every 10 °C increase in operating temperature can cut the device by half.

1.2 Research Problem and Our Contributions

While there has been extensive work on addressing the power/thermal issues [27, 39, 65, 68, 69] most of these work has been based on idealized theoretical models and

assumptions. Theoretical research, however, has its limits. Even though some system characteristics can be parameterized into system models, practical computing system must deal with real-world scenarios and complexities which may be far beyond what theoretical models and assumptions can define. It has been a common practice to validate these theoretical research results using simulation. Simulations help us in collecting specific statistics and thereby gaining better insight on different power/thermal aware techniques. However, results obtained from a simulator are also usually built upon theoretical models and assumption themselves. These results can have drastic variations from the results obtained from an actual system due to modeling imperfections or inherent randomness associated with ambient conditions or the workload [26]. This limitation of software simulation motivates the development of a practical testbed to verify and validate the various theoretical works.

Our goal in this thesis is to develop a flexible, customizable, and practical multi-core testbed based on an Intel desktop computer that can be used to assist the theoretical research on power/thermal aware resource management in design of computer systems. Compared with the related work, we have made the following contributions in this thesis:

1. We have developed a customized, flexible, and practical hardware multi-core testing platform based on an Intel-i7 920 Bloomfield quad-core processor, running Ubuntu 12.04.1 Linux operating system with kernel version 3.5.0-49-generic as shown in Figure 1.3. By extensively using the open source software and tools, this test platform can be easily migrated to different desktop computers with different processor architectures and hardware characteristics. With the support of thread level management, researchers can customize this platform, realize a large variety of system level power/thermal aware resource management schemes (task allocation, mapping, scheduling, etc) for multi-core systems, and

study their effectiveness and efficiency. By integrating different modules, including thread mapping/scheduling, processor/core frequency and voltage variation, temperature/power measurement, and run time performance collection, into a systematic and unified framework, we expect our test bed can bridge the gap between the theoretical study and practical implementation.

2. We evaluated our testbed with real programs and three typical power/thermal scheduling policies on multi-core platforms. We investigated the power and thermal characteristics of selected benchmarks as well as how they performed and affected the system under the different DVFS policies in a practical computing system. Our experimental results verified some results established in the theoretical study. In the meantime, we also made a number of interesting findings, which can potentially become our new research problems.

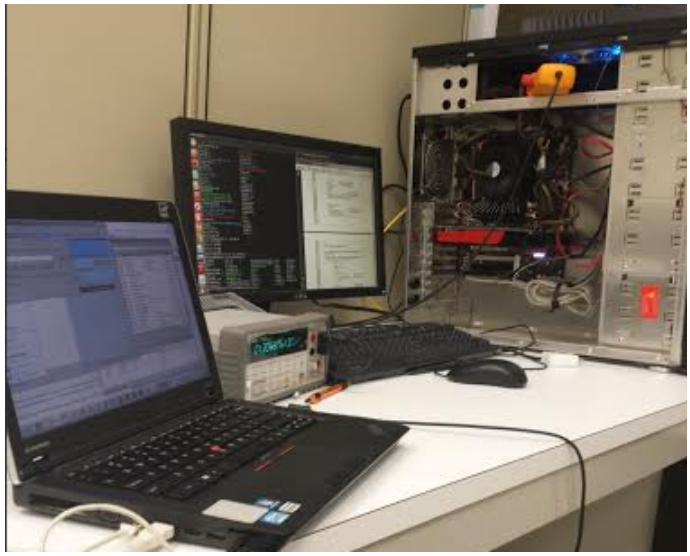

Figure 1.3: Testing Platform System

1.3 Organization of Thesis

The rest of the thesis is organized as follows: In chapter 2, we introduce the background of this work and then discuss related works in power and thermal management as well as the techniques used in their validation and testing. In chapter 3, we discuss the testing platform. With this platform we validate some existing theoretical work in chapter 4. Finally we conclude this thesis and discuss future works in chapter 5.

CHAPTER 2

BACKGROUND AND RELATED WORK

The research background and literature review is presented in this chapter. We begin by introducing the subject of power consumption. We then discuss the power and thermal relationship as well as some thermal management techniques. We then cover the various simulation and testing techniques that are currently utilized.

2.1 Power/Energy Consumption and Temperatures

As a result of continuous transistor scaling, billions of tranastors have been integrated onto a single chip [2]. While there is an immediate increase in performance, a more immediate consequence of the increase in transistor density is the increased power consumption. That has adverse effects on energy and temperature. Power, energy, and temperature have become ciritcal design objectives for system designers and engineers.

2.1.1 Power Consumption

The power consumption of an IC chip is comprised of two categories: dynamic power consumption and the static power consumption [34]. The dynamic power consumption is caused by charging and discharging the load capacitance. It is associated with the switching of the logic value of a gate and thus is essential to performing useful logic operation. The dynamic power is modeled as a function in proportion to working frequency and square of supply voltage [34]. The dynamic power is given by:

$$P_{dyn} = \alpha C V^2 f \tag{2.1}$$

where P_{dyn} is the dynamic power consumed, α is the activity factor, C is the capacitance, V is the supply voltage and f is the working frequency.

The leakage power, or static power, is due to the leakage current flowing through the transistor. It is represented by the expression:

$$P_{leak} = V I_{leak} \tag{2.2}$$

where P_{leak} is the leakage power, V is the supply voltage and I_{leak} is the leakage current through the ransistor.

As submicron technology increase, leakage power becomes more significant in the total amount of power consumption as shown in Figure 1.2. Leakage power can be approximated as a linear function of temperature and voltage [57]. That is, leakage current changes linearly with temperature and voltage. This further illustrates the notion that there is a need for incorporating leakage/temperature depenedency into design and analysis of power efficiency systems.

2.1.2 Temperature

Power consumption and heat dissipation are closey related in that high power consumption generates alot of heat which consequently raises the on-chip temperature. Subsequently, the high temperature increases the leakage power which, in turn, increases the leakage power consumption as well [2]. This characteristic degrades the temperature situtation due to the interdependency between temperature and leakge power. Moreover, the soaring chip temperature adversely impacts the performance, packaging/cooling costs, and reliablity [34]. The aforementioned issues give reason as to why temperature/thermal attributes have become a critical issue for advanced multi-core system design.

2.2 Research on Power/Thermal Aware Computing

There has been extensive theoretical research on power awareness, energy minimization, and temperature awareness, on multi-core computing systems.

2.2.1 Power/Energy Awareness

With the increased complexity of multi-core systems, comes the exponential increase in power consumption which would, in turn, increase temperature. In efforts to deal with such issues there has been extensive research done in regards to power aware computing techniques [6, 21, 28, 36, 68, 79]. Vega et al [68] proposed a technique of a SMT-sentric power-aware thread placement in CMPs where they make use of the optimum combination of core-wise SMT level and number of active cores to achieve desired power-performance efficiency. Ghasemazar et al [21] develop a robust frame work for power and thermal management of heterogenous CMPs subject to variability and uncertainty in system parameters. In detail, they model and formulate the issue of maximizing the task throughput of a heterogeneous CMP subject to a total power budget and a per-core temperature limit. Heo et al [28] proposed a method to reduce power density by moving computation of a task to a different location on the die. Ansari et al [6] proposes a scheduling algorithm which combines offline and online scheduling with DVFS to schedule fixed priority tasks on soft real-time multicore systems.

As a result of the increasing use of multi-core systems and rising performance demand, energy consumption has escalated continuously and has faced severe challenges, specifically for an energy efficient design. There has been extensive work focused on the problem of energy minimization [9, 29, 37, 41, 48, 73, 77]. For Instance, Huang et al [29] derived a closed-form energy calculation equation from which they proposed an

energy minimization scheduling method for periodic task sets. Bao et al [9] propose a temperature aware idle time distribution technique for energy optimization with dynamic voltage scaling. A temperature analysis approach is also proposed which is used inside of the optimation loop for idle time distribution and voltage selection. Yang et al. [73] proposed a procedure to determine the optimal pattern of a schedule with the minimum energy consumption at the stead state. Yao et al [48] proposed a strategy that utilized traditional DVFS for each processor after scheduling which ensured all tasks met timing requirements on synchronization. They also proposed a strategy which determined the fequency of each task before scheduling according to the total utilization of task-set and number of cores available.

2.2.2 Thermal Awareness

In regards to temperature reduction methods there has been much theoretical research done over the years [8, 19, 42, 61, 74–76].Yeo et al [75] developed a predictive dynamic thermal management technique for multicore systems. Their work proposes to maintain system temperatures below a desired level by moving and running the application from the possible overheated core to the future coolest core and by reducing the processor resources within multicore systems. Bailis et al [8] proposes the use of idle cycle injection, by way or race-to-idle schedule to implement a per-thread technique as a preventitve thermal management mechanism. Kumar et al [42] proposed a stop-n-go approach to reduce the peak temperature for tasks with data dependencies. The slack time was distributed between jobs in that the peak temperature could be minimized without make-span violations. Fisher et al [19] proposed a method to minimize peak temperature in a homogenous multicore system by utilizing global scheduling algorithms which can exploit the flexibility of multicore platforms at low temperature. Juan et al [35] proposed a systematic framework that can learn

different thermal profiles of a CMP by using an autoregressive model which can, in turn, serve as an alternative for predicting the transient temperature of a CMP.

2.3 Experimentation and Validation

In this section, we contextualize the state of the art in the validation of power/thermal aware resource management techniques, covering various validation techniques as well as related works.

2.3.1 Synthetic Simulation

These works use randomly generated test cases and simplified power/thermal models to perform their validations of their work. Gojiara et al [23] developed a technique which quickly generates very energy efficient results irrespective of the number of available voltage models. Dabiri et al [16] consider the effects of voltage on single event upsets (SEUs) to develop a method for energy optimization under SEU constraints. They also propose a convex programming formulation that can be solved efficiently and optimally. These two papers randomly generate test-cases for system functions through the use of Task Graphs For Free [18]. Lu et al [49] proposed an energy-aware fixed priority multicore scheduling technique and validated their results by way of synthetic simulation, with a simulator they developed. Other system characteristics such as different computer architectures, power/thermal characteristics can also be randomly generated. The advantage of this approach is that a large number of test cases can be generated and tested. However, the disadvantages of this approach are that the models are simplified to the extent of neglecting other factors that may, in a real situtation, have an effect on the result.

2.3.2 Architecture Level Simulation Tools

The use of architecture level simulation tools in particular is quite extensive in the research on power/thermal aware resource management. He et al [27] proposes a heat spreading model based floorplan scheme for chip multicore processor as well as a thermal aware thread mapping policy which benefits from the proposed floorplan method. This work in particular makes use of several models and tools such as sim-alpha [7], power model HotSpot [31], and a thermal model developed by Michaud et al [52]. While their work provides insight on the heat spreading behaviours during processor floorplaning and its effect on final runtime temperature, it makes use of several simulation tools as well as making some assumptions which include assuming that the heat generated in a functional unit is to be distributed evenly over the entire area of the unit. Another work is done by Salamy et al [62]. They proposed an optimal ILP solution to task scheduling of different applications on a multicore system with power and energy constraints. However, their solutions were generated using CPLEX [24] and they used SimpleScalar [7] architectural simulation to profile the used benchmarks.

This simulation methodology provides flexibility in observing the architecture details and their impacts on theoretically developed methods or techniques. However, computational costs are high and computation is time consuming. While architecture level simulation tools use much more sophisticated models, the results obtained from architecture level simulation tools do not consider the parameters that are included when dealing with an actual phyiscal system.

2.3.3 Practical Hardware Testbed

Few works have used a practical hardware bed for simulation. One in particular is by Liu et al [47]. This work presents a practical thermal aware scheduling algorithm to

optimize its throughput under a given temperature constraint. This algorithm is then validated on a practical working environment. This practical working environment is also used to validate another work by Liu et al [46]. This work uses the neighbor effect for temperature prediction on CMPs to design and develop an accurate temperature prediction method which also allows for determining better task migration destination. While obtaining real data is highly beneficial to the development of new power and thermal aware resource management techniques, a practical testbed must also deal with the issues of compatibility with specific hardware as well as the influence of other paramters such as noise and signal strength.

More closely related to our work, Hanumaiah et al [25] proposed a solution to the problem of accurately controlling the cores through the use of dynamic frequency and voltage scaling, thread migration, and active cooling. While they perform tests using architectural level simulation tools, they also implement their algorithms and solutions on a Intel quad-core Sandy Bridge processor by designing a closed loop controller. The closed-loop controller builds a system model from the power and temperature processr, then predicts the future power and temperatures of cores and corrects any mispredicitions.

Our work differs in that their approach is more of an ad-hoc model to their specific problem while our approach allows for the user to design and develop their own testing platform through the use of our provided libraries.

2.4 Summary

In this chapter we discussed the essential background of our research and introduced some related work. To start, we presented a general introduction of the basic concepts of power consumption and heat generation. Next, we discussed the some of the theoretical research addressing the issues of power consumption and heat generation

as well as power and thermal aware multicore computing techniques. Then finally we introduced the techniques used in these works to validate and verify their results.

CHAPTER 3

POWER/THERMAL AWARE COMPUTING MULTICORE TESTBED

Our testing platform utilizes select tools and APIs to meet our design objectives and needs. In this chapter we will discuss the system as well as its components. We begin with discussing the overall system overview. Then we cover each individual component of the testing platform, discussing its functionality while also covering our implementation of the component.

3.1 System Overview

We have constructed a flexible, customizable, and practical power/thermal aware resource management test platform that is capable of testing different resource management schemes and measure their power/thermal/computing performance. The flexibility of this testing platform allows it to migrate to different architectures and computers, as well as testing different benchmarks. Its customizable attribute is credited due to its several different benchmarks and different program allocation and scheduling schemes. Its practicality allows it to run real programs that measure true power consumption, temperature, execution times, and other performance parameters.

The system is shown in Figure 3.1

As shown by Figure 3.1 there are three sections to the testing platform. The inputs section denotes the information that the user will provide to set up the experiments. The system configuration uses information from the inputs to configure the testing platform for the experiments that will run. The run-time information/output section refers to the output of the experiments which involves the architectural parameters, power consumption, temperature, and performance.

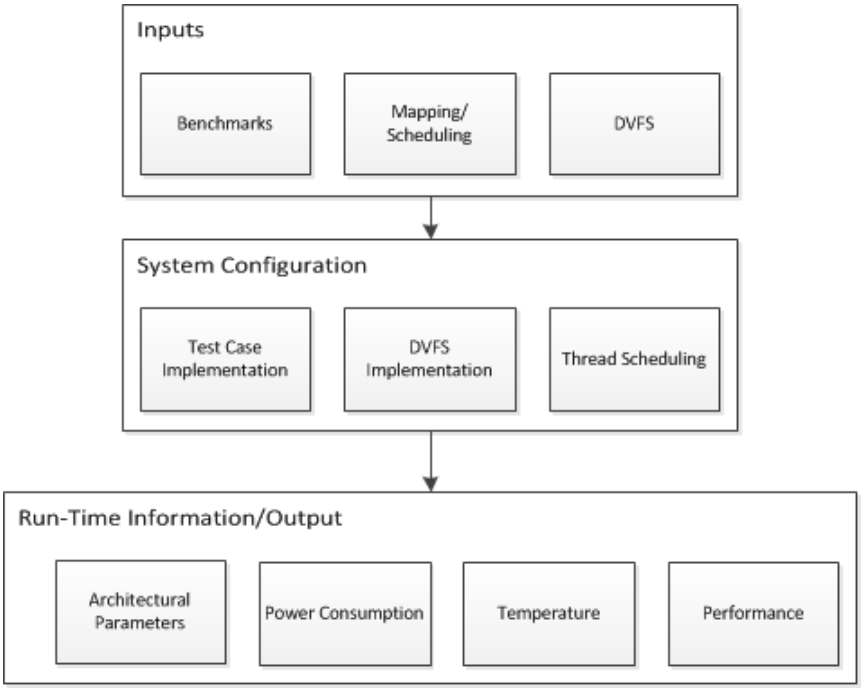

Figure 3.1: TestBed: System overview

3.2 Inputs

This section discusses the information input by the users to validate their theoretical results with the experimentations. This includes the information for the system workload, task mapping/scheduling methods, and other settings such as the DVFS schedule.

3.2.1 System Workload

The benchmarks chosen to be used as workloads vary in computation time. These benchmarks include:

- **Classic Matrix Multiplication**: Standard 1024 x 1024 matrix multiplication algorithm

- **Swapped Matrix Multiplication**: Same input parameters as the classic matrix multiplication
- **Catalan Numbers** : In combinatorial mathematics, the Catalan numbers form a sequence of natural numbers that occur in various counting problems.
- **Fibonacci**: Fibonacci number detector by way of anonymous recursion
- **Ackermann Function** : Well-defined total function which is computable but not primitive recursive.
- **Pythagorean Triples**: Computes a Pythagorean triple
- **Bubble Sort**: Bubble Sort algorithm of 1.0e5 elements
- **Insertion Sort**: Insertion sort algorithm of 1.5e5 elements

3.2.2 Task Mapping and Schdeduling

This section of the inputs denote which of the programs are allocating to which core. That is, this core handles the allocations of task to cores. Moreover, the scheduling policy is also denoted here. To map threads to cores we make use of CPU affinity through the use of the function call pthread_setaffinity_np. Details of its implementation are discussed in later sections as well as details regarding our thread scheduling.

3.2.3 Dynamic Voltage Frequency Scaling

Another facet of the input section is the selection of the Dynamic Voltage Scaling schedule. The schedule will determine how to vary the processor's working frequency throughout the experiment. The process of dynamically altering the supply voltage and operating frequency is commonly referred to as dynamic voltage frequency scaling(DVFS), and the corresponding schedule is called DVFS schedule [78]. DVFS can

decrease the supply voltage and the operating frequency of a chip when the detected temperature is higher than the predefined thermal threshold and after a period of interval when the temperature returns back to safe region then increases the voltage and frequency step by step.

DVFS can be divided into two categories. The first is the hardware approach presented by [38], [51] and [40] which propose monitoring hardware to predict execution patterns and manipulate the system's DVFS configuration. Hardware methods cannot be changed based on design and policy variations, which result in its lack of use among many chip manufacturers. The second category is the management of power, controlled by compiler and user space runtime systems discussed in [72], [63], and [60].

3.3 System Configurations

This section takes the attributes acquired in the user input section to configure the testbed system. Parameters in this section include: programming and mapping on different cores, thread level scheduling, priority setting, and DVFS implementation. Each of the component characteristics as well as implementation are covered in detail.

3.3.1 Test Case Implementation

Techniques to map threads to cores have been presented in numerous research works [15, 17, 53]. For our research we have modified the method presented by Cruz et al [53] to use the function call pthread_setaffinity_np instead of the sched_setaffinity system call which is present in the Linux kernel to dynamically map the threads. This particular method makes use of a system call named CPU affinity on the computing system. By way of CPU affinity, tasks can be bound to one or more cores. The initial

purposes for the use of CPU affinity was for optimizing the cache performance. To prevent a data synchronization problem which would subsequently lead to an increase in cache miss rate and reduction in system performance, the operating system tries to keep task data on only one core's cache at a time for as long as possible for the multicore computing system. Moreover, there is an increase in context switching overhead in the case of continuous migration of tasks from one core to another. Hence, CPU affinity is only used to bind our tasks to cores. Our method to bind specific cores to tasks is presented in the code snippet in Figure 3.2 below:

```
/**
* Binds core to specified thread
*/
int bind_to_core ( pthread_t thread, int core_id)
{
    int num_cores = sysconf(_SC_NPROCESSORS_ONLN); //Number of cores machine has
    int j;
    int ret;
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(core_id, &cpuset);
    thread = pthread_self();
    //Testing function.
    printf("Cores returned by pthread_getaffinity_np() contained:\n");
    for (j = 0; j &lt; CPU_SETSIZE; j++)
    if (CPU_ISSET(j, &cpuset))
    printf(" CPU %d\n", j);
    //Bind to core here
    ret = pthread_setaffinity_np(thread,sizeof(cpu_set_t), &cpuset);
    return core_id; //returns the core number thread was binded two.
}
```

Figure 3.2: Method to bind thread to core

From the code listing above it is apparent that the primary parallel programming paradigm used is POSIX Threads (Pthreads). Pthreads are an interface with a set of C language procedures and extensions used for creating and managing threads [5]. Pthreads has a very low-level of abstraction which translates to difficult implementation from a developer standpoint. With Pthreads the parallel application developer has more responsibilities like work load partitioning, worker management, communication and snychronization, and task mapping [3]. Primary alternatives to using Pthreads include, OpenMP and MPI.

OpenMP or Open Message passing is an application program interface, which defines a set of program directives, runtime library routines and environment variables that are used explicitly to express direct multi-threaded, shared memory parallelism. OpenMP stands at high-level abstraction which makes development of parallel application much easier. The developers only need to specify the directives in order to parallelize the applications. As a result of its ease of use, OpenMP is not as widely used as Pthreads as the flexibility of OpenMP is less compared to Pthreads [5, 58]. MPI or Message Passing Interface, on the other hand, is a message passing library standard designed to function on a wide variety of parallel computers [50]. MPI works on both SMP and distributed memory systems. MPI uses objects called communicators and groups to define which collection of processes may communicate with each other. While much more difficult in regards to implementation and use, we have chosen Pthreads due to its extensive flexibility and its granularity.

The usage of Pthreads is not without limitations. Due to the fact that we are binding threads to cores, the simple act of binding entire programs to threads is not possible. The method of using the system() command was explored. On Linux, it spawns /bin/sh (forking and executing a full shell process), which parses the command, or benchmark in this case, and spawns the second program. Due to the unpredictability of disk access and Linux processes scheduling, timing system() calls has a very high inherent variability. To this end, we have implemented our benchmarks as methods and have coded them as such in our library file for the benchmarks. While this limitation impacts our ability to use more commercial benchmarks, it does give us more control in regards to the behavior and parameters of the benchmark. This attribute also allows us to simply code our own computing intensive tasks.

3.3.2 Thread Scheduling

Linux by default, uses the Native POSIX Thread Library, or nptl, implementation which considers a thread as a light-weight process, so the scheduler schedules threads with other processes. Using the Pthreads scheduling feature, you can designate how threads share available processing power. It can be decided that all threads should have equal acess to all available CPUs, or some threads can be given some preferential treatment [11]. There are two thread-specific attributes to consider when creating a schedule for pthreads: The scheduling priority and scheduling policy. The scheduling priority determines which thread get preferential treatment and access to available CPUs at any given time. The scheduling policy is a way of expressing how threads of the same priority run and how they share available CPUs, [11]. To make use of more popular scheduling policies such as Liu and Layland Rate Monotonic Scheduling(RMS) [45] or Earliest Deadline First(EDF) [43], we must modify and patch the kernel. The realtime policies available to us is SCHED_FIFO and SCHED_RR which translates to first-in-first-out and round robin, respectively.

While our testing platform does implement some form of thread scheduling, limitations do apply. First of all, our implementation is referred to as userspace scheduling. The limitation exists in the earlier notion that the Linux considers a thread as a lightweight process. That is, while we can perform scheduling, in the userspace, there is no real indication that the threads are actually executing at their scheduled time since the lightweight processes are then scheduled by the kernel where we have no control. In order for this to happen, we must change a system file sched.c to perform kernel level scheduling and fully implement our scheduling policy but that is outside the scope of this work.

3.3.3 DVFS Implementation

For our implementation of DVFS we make use of the latter approaches by way of the CPUfreq module illustrated by Wang et al [69] which will be discussed later. We develop numerous methods and policies. Among them we can change indivdual core speeds, oscillate the clock speed between two frequencies, and step up or step down the clock speed to another frequency step. We also include functions to facilitate the restoration of chip speed to its lowest level or setting the chip speed to its highest level which normally find its use for those developing their own testing platform using our libraries.

CPUFreq

Implementing DVFS we must make use of Intel's SpeedStep technology to change the desired DVFS level. Enhanced Intel SpeedStep Technology was introduced in the Pentium M processor. The technology enables them management of processor power consumption via performance state transitions. These states are defined as discrete operating points associated with different voltages and frequencies [32]. In Linux, this can be acheieved through the installation and utilization of the CPUfreq, also known as CPU speed scaling, package which allows for the scaling of the processor speed.

CPUfreq comes with various rules which determine how and when the cpu frequency is changed based on system load. These rules are defined by a CPUfreq governor. These governors are:

- **performance** forces CPU to use highest possible clock frequency.
- **powersave** forces CPU to use lowest possible clock frequency.
- **ondemand** is dynamic governor that allows CPU to achieve maximum frequency when the system load is high and minimum frequency when the system is idle.

- **conservative** is similar to ondemand, this is a dynamic governor that allows for the adjustment of clock frequency based on CPU usage but does so in a far less aggressive manner than the ondemand governor
- **userspace** allows userspace programs(essentially any process running as root) to set the frequency

For our implementation, we make use of the method presented by Wang et al. [69] which proposes using the userspace governor which allows us to manually change the frequency. This is done by modifying and updating the system file located at /sys/devices/system/cpu/cpu[X]/cpufreq/scaling_setspeed where X is the index of the core in the processor. We have also extended his proposed extension by implementing the CPUfreq frequency modulator as a function within a library called setCoreSpeed(). The desired core and frequency step were passed as parameters. The method used C function calls to modify the aforementioned system file. We also implemented other methods that facilitate in changing the operating frequency for a number of cores at a time. The Intel i7 processor supports 10 working frequencies ranging from 1.60 GHz to 2.67 GHz as shown in the following table. The available frequency levels are shown in Table 3.1

Available Frequencies in GHz	
1	1.600
2	1.733
3	1.867
4	2.000
5	2.133
6	2.267
7	2.400
8	2.533
9	2.667
10	2.668

Table 3.1: List of available frequencies

3.4 Run-Time Information Collection and Output

This section returns values after running the benchmarks. The values return are in the form of power consumption, temperature, execution times and performance results .

Hardware Locality

An Open MPI sub-project, Hardware locality or hwloc, provides command line tools as well as a C API to obtain the hierarchical map of key computing elements, such as: NUMA memory nodes, shared caches, processor sockets, processor cores, processing units (hardware threads), and even IO devices. It also obtains various attributes such as cache and memory information [12]. Hwloc is a result of the evolution and merger of the libtopology project and the Open MPI project: Portable Linux Processor Affinity (PLPA) due to their functional and ideological overlap. These attributes allow us to determine the communication pattern as well as the locality of the CPU cores that will be used for the experiments. The role that hwloc plays in our testbed is to provide us with a physical topology of the architecture that is being tested. This allows us to analyze the effects of a cores temperature on neighboring cores.

3.4.1 Power Consumption

Power and energy reading for SandyBridge architecture Intel processors involve the use of onboard power counters through the use of Running Average Power Limit or RAPL. It returns the chip level power consumption without the need for external peripherals. The aforementioned power counters are not available to all Intel processors and for the sake of portability for the testbed to read the CPU power consumption, it was necessary to follow the method illustrated in [33, 69] which used an Agilent

34410A digital multimeter (DMM) along with a Fluke i410 current probe to measure the voltage running through the 12V power lines that powers the processor. The main power lines for the CPU operate at 12V, and are then fed to a voltage regulator module, which converts this voltage to the actual processor operating voltage which provides control on voltage variations [33]. The probe is clamped to the 12V lines and produce a voltage signal proportional to the current running through the lines with a coefficient of 1mv/A. The voltage sample obtained is then converted to processor power dissipation with the power relation :

$$P = VI = 12 \cdot (VoltageSample[V]) \cdot 1000 \tag{3.1}$$

where P is the power, V is the supply voltage and I is the current. The current I is obtained from the measured voltage signal, $VoltageSample[V]$ that is proportional to the current. The resolution of the DMM for the power reads was set at a fast 5 digit read at 200 Hz. This resulted in high speed sampling of the voltage.

Unlike what is presented in the above methods, we made use of an external machine to collect the readings and a software tool, Agilent BenchVue, to log our results. Agilent BenchVue, designed by engineers at Agilent Technologies, is a free software suite that provides a wide array of capabilities that vary based on the functionality of the instrument types and models connected to the PC running the BenchVue software. Supported functionality for the utilization of digital multimeters include:

- Measurment Configuration: Function and range selection, integration time, input impedance, auto zero, null state
- Visualization and Annotation: Chart, with full annotation, zoom in/out, change trace color, display sample count and markers, tabulated results.
- Data Logger: Basic data log with strip chart, start control(IMM/time/trig), sample interval, stop control (IMM/time/samples)

- Exported Information: Screen shots by clipboard or file, data by MATLAB, Microsoft Excel, Microsoft Word, and CSV

Figure 3.3 shows the Agilent Benchvue software interface. This particular interface is unique to digital multimeters and shows the trace of an experiment that was run.

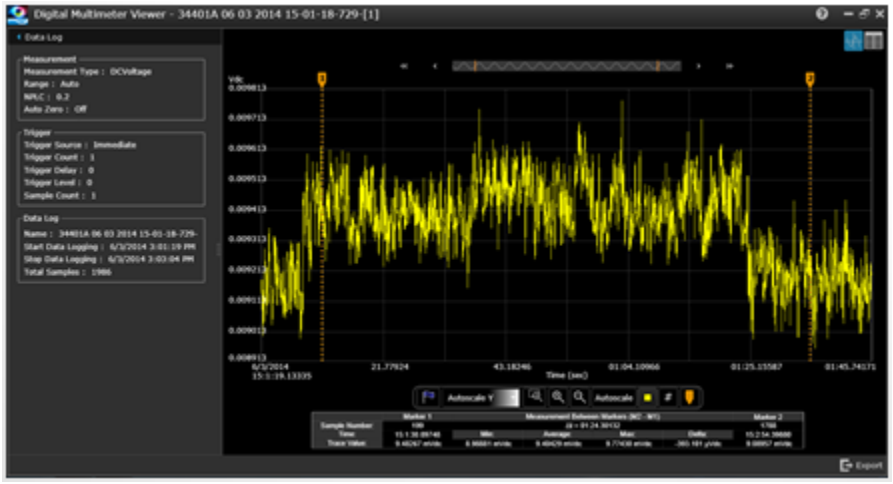

Figure 3.3: Agilent Benchvue Digital Multimeter Viewer Interface

In addition to the capabilities on the PC, Agilent BenchVue also comes as a mobile app, which can be installed on an Android or Applie IOS device to control BenchVue bench applications running on the Windows computer through WiFi, VPN, 3G or 4G.

A communication structure of the Agilent BenchVue software, running computer, instrument and IO is shown in Figure 3.4

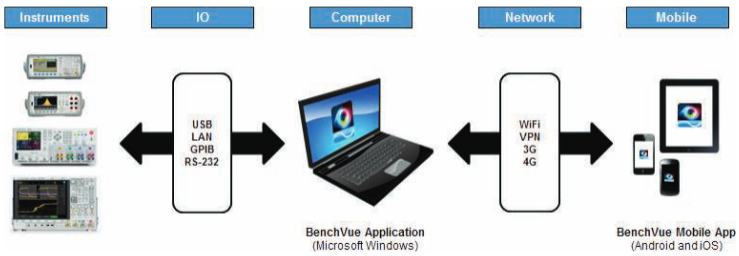

Figure 3.4: Communication Protocol of Agilent BenchVue

Based on the communication structure above, an outline of the system is presented in the following Figure 3.5.

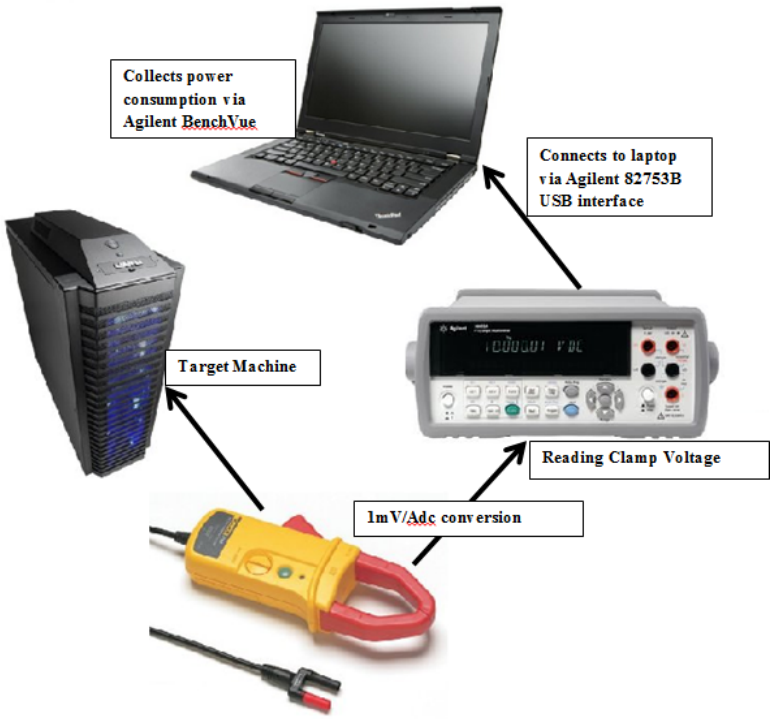

Figure 3.5: Overview of power trace collection system

3.4.2 Temperature Reading

On die digital thermal sensor can be read using an MSR (no I/O interface). MSRs or Model Specific Register are used to provide access to features that are generally tied to implmentation dependant aspects of a particular processor. In Intel Core Duo processors, each core has a unique digital sensor whose temperature is accessible using an MSR. The digital thermal sensor is the preferred method for reading the die temperature because (a) it is located closer to the hottest portions of the die, and (b) it enables software to accurately track the die temperature and the potential activation of thermal throttling [32]. On modern machines temperature is read using

these sensors by way of third party applications. One popular method in particular is through the use of the tool, lm-sensors, which is a free open source software tool that provides monitoring information of temperatures, voltages, humidity and fan speeds. It can also detect chassis intrusions [1]. While an effective tool, it does not provide methods to use its functions programmatically.

To read the temperatures, we make use of the method presented in Wang et al [69] which utilizes the coretemp driver in Linux to read the temperature values reported by the thermal sensors in each core through the system file:

/sys/devices/platform/coretemp.[X]/temp1_input, where X is the index of the core used.

The coretemp driver permits reading the digital temperature sensors(DTS) embedded inside Intel CPUs. Each core on these processors has a DTS that reports temperature data relative to TJMax which is the safe maximum operating core temperature for the CPU. Unlike traditional analog thermal devices, the output of the DTS is a temperature relative to the maximum supported operating temperature of the processor [32]. Figure 3.6 presents the register that deals with reading the digital sensor.

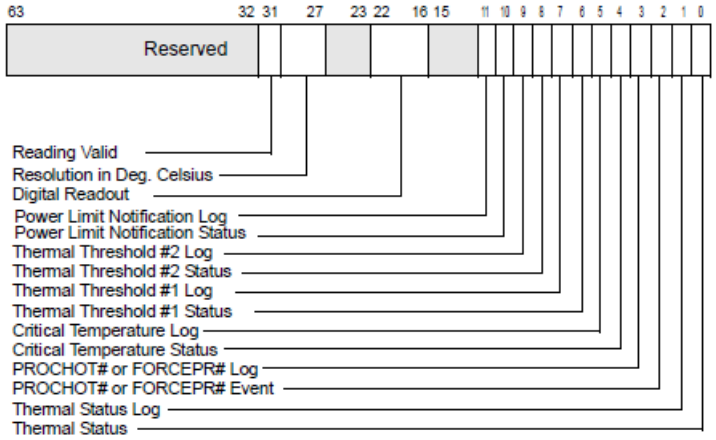

Figure 3.6: IA32 Thermal Status Register

It can read on a per-core and per-package basis but based on our system configuration we will only utilize the per-core functionality as per-package sensor is new. As of late, it is present only in the SandyBridge platform and beyond. Our implementation of the temperature reader is presented in Figure 3.7

```
/**
* Gets the temperature of the core
*/
int getTemp(int core)
{
    int cpu;
    //System Specific
    if(core == 0)
    {
        cpu = 2;
    }
    if(core == 1)
    {
        cpu = 3;
    }
    if(core == 2)
    {
        cpu = 4;
    }
    if(core == 3)
    {
        cpu = 5;
    }
    char ch;
    float temp;
    FILE *fp;
    char address[50];
    //Printing the label
    sprintf (address, "/sys/devices/platform/coretemp.0/temp%d_label", cpu);
    fp = fopen(address, "r");
    while( (ch = fgetc(fp)) !=EOF && ch != '\n')
    putchar(ch);
    //Printing, and returning, the value
    sprintf (address, "/sys/devices/platform/coretemp.0/temp%d_input", cpu);
    fp = fopen(address, "r");
    fscanf(fp, "%f",&temp);
    temp = temp/1000;
    printf(" %2.3f\n",temp );
    return temp; //returns the temperature of the core.
}
```

Figure 3.7: Method for collecting temperature

The system specific information is based on the filenames provided by the coretemp driver which corresponds to the core. That is, the file names are in the format temp[X]_label where X is refers to the core. However, in the system this file may have

the number 2 but actually correspond to core 0. Our implementation provides a way to pass the desired core as a parameter for collecting temperature while the correct file is accessed. As the code shows, the number 0 would be passed however the file temp2_label would be accessed that would thus return the temperature of core 0.

Temperature is measured in Celsius and measurement resolution is 1 °C. Valid temperatures are from 0 to the maximum temperature of the core due to the fact that the actual value of the temperature register is in fact a delta from the maximum temperature. The maximum temperature is dependent upon the model of the CPU.

3.5 Performance

There are numerous tools available for obtaining the performance for a section of code or a program. These include but are not limited to perf profiler, PAPI, perfsuite, Valgrind, likwid, and more. The performance for the benchmarks were acquired through the use of the Performance API(PAPI) [70]. This specific tool provides the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. PAPI enables software engineers to see, in near real time, the relation between software performance and processor events. As opposed to the other listed tools, we have chosen PAPI due to its extensive API that allows us to gather selected performance parameters that gives insight as to how a particular piece of code and/or method is functioning.

In our utilization of the PAPI tool, the API code is added into each of our benchmark methods. That is, every benchmark has a copy of the code that does the measurements. Due to the limitations of our system, we are limited to the use of 4 counters per experiment. For that reason, each benchmark has two versions: one version that gets information such as instruction counts and cache misses and another

version that gets information such as total cycles, branch mispredicts, etc. PAPI obtains information as follows:

- Start Counters
- Do Work
- Read Counters
- Stop Counters

The workload of the benchmark is utilized in the "Do Work" section of the list which begins after the counters have begun. Upon completion of the benchmarks, the counters are read and reported to the user. The counters then end following the aforementioned procedures. While limited in information that can be attained per experiment, slight modification of the performance gathering procedures in the benchmark libraries gives way for the measurement of other performance parameters such as loads, stores, floating point operations, etc.

CHAPTER 4

EXPERIMENTS AND RESULTS

This section is the culmination of the previous sections as it integrates them as well as performing benchmark testing and power/thermal and performance readings. This is done to demonstrate the functionality of this testing platform as well as provide an analysis on the results. Each experiment returns the power consumption, operating temperature and performance parameters of its respective run.

4.1 Preliminary Work

This section illustrates the work done prior to the experimentation of our testing platform. That is, in this section we determine our processor architecture, examine any potential overhead when retrieving the temperatures as well as examining the idle temperatures of the system.

Architecture Data Collection

Using the hwloc tool, Figure 4.1 illustrates the topology of target machine used in our system. Based on the figure, it shows that there are 4 physical cores and 4 logical cores by way of hyper-threading.

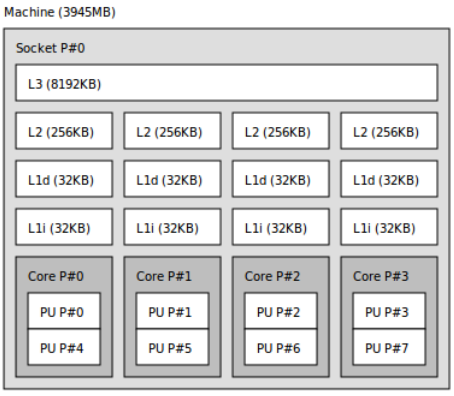

Figure 4.1: Topology of System using hwloc tool

Temperature Retrieval Overhead

As a result of the fact that the temperature reading function would be running on the actual machine that is being tested, it necessary to determine the amount of time it takes to read the temperature system file. Presented in Figure 4.2 are the times for the retrieval of the temperature values

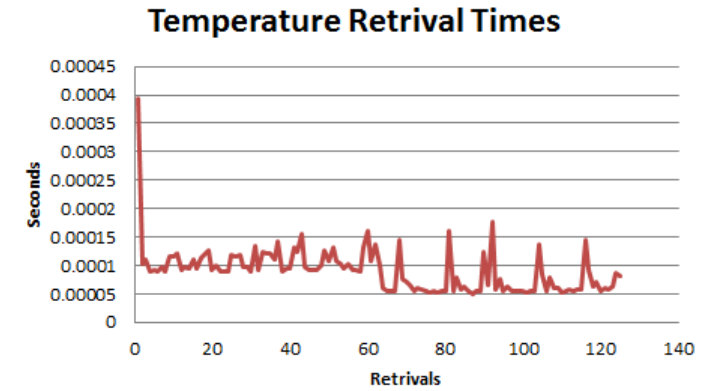

Figure 4.2: Temperature retrival time from system file

The information provided to us by the above figure indicate that the load on the processor as a result of returning the temperature. Results indicate that due to the relatively low times required to capture the temperature on the temperature sensor, we can utilize our current method of reading temperatures. In the case where the times are too high we must either utilize another method or make use of an external peripheral.

Idle Temperatures

Presented in Figure 4.3 is the running temperature of the core while the system is idle.

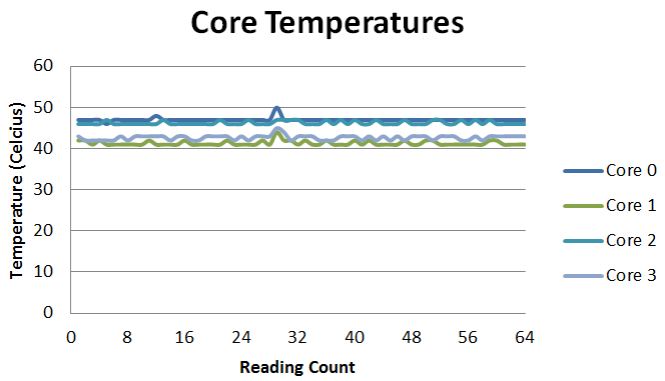

Figure 4.3: Running temperature of each core

The expression to determine the average temperature is described by:

$$T_{core} = \frac{\sum_0^n T_i}{n} \tag{4.1}$$

where T_{core} is the average temperature for all the cores, T_i is the temperature at core i. The n represents the number of cores used in the calculation. It can be seen in Figure 4.3 that after doing 64 consecutive readings with 2 second sampling time the average temperature for all the cores was: $T_{core} = 44.3359\,^{\circ}\mathrm{C}$. The sampling time can be scaled to meet the needs or requirements of any desired scheduling/mapping algorithm.

4.2 Benchmark Performance

For this section we will test appropriately selected benchmarks and obtain output data accordingly. As stated in chapter 3, the benchmarks available for the testing platform are: Classic Matrix Multiplication, Swapped Matrix Multiplication, Catalan Numbers, Fibonacci Numbers, Pythagorean Triples, Insertion Sort and Bubble Sort. Using the perf profiler the average runtimes for the benchmarks are presented in Table 4.1 below.

Runtimes	
Classic Matrix	9.30
Swapped Matrix	6.34
Catalan	4.18
Fibonacci	13.25
Ackermann	25.31
Pythagorean Triples	48.45
Insertion Sort	46.29
Bubble Sort	79.87

Table 4.1: Average run times of benchmarks

PAPI Hardware info	
Model	Intel
CPU Revision	5.0000
Max Speed	2.668
Min Speed	1.600
Hardware threads per core	2
Sockets	4
NUMA Nodes	1
CPUs per node	1
Total CPUs	8
Hardware Counters	7
Multiplex Counters	64

Table 4.2: CPU Information acquired by PAPI

Details of our hardware were aquired through the methods of Weaver et al [70]. The Table 4.2 presents detailed CPU information not found with hardware locality.

4.3 DVFS Experimental Results

The experiments were performed on a system consisting of an Intel i7 920 Bloomfield processor. The physical topology of the processor is illustrated in Figure 4.1. We implement and experiment on three selected DVFS scheduling policies: *a*) Constant speed; *b*) Two level frequency; and *c*) Oscillating frequency. While there are multiple benchmarks available for use in the current version of this testing platform, we utilize the Ackermann function as our workload. As an external tool, the running voltage

was collected before and after the benchmarks executed to illustrate the disparity in the voltage between when it is running and when execution has completed. The data presented represents an entire run of the testing platform to illustrate the changes in power consumption. The performance metrics we are obtaining for these experiments is the instruction count as well as the L1/L2/L3 cache misses. For the listed experiments, we utilize two cores. Due to current system limitations, it is currently not possible to tell which core corresponds to which performance readings.

4.3.1 Constant Speed Experiments

In these experiments, the operating frequency is kept at a constant level while the benchmark executes. The following figures illustrate the power and temperature profiles of the constant speed DVFS tests utilizing 2 cores.

Constant Speed Step 5		
Instruction Count	48670802836	48670787535
L1 Cache Miss	1330713267	1330744355
L2 Cache Miss	143611047	143206959
L3 Cache Miss	1368930271	1368918376

Table 4.3: Performance Values for Constant Speed Experiment 1

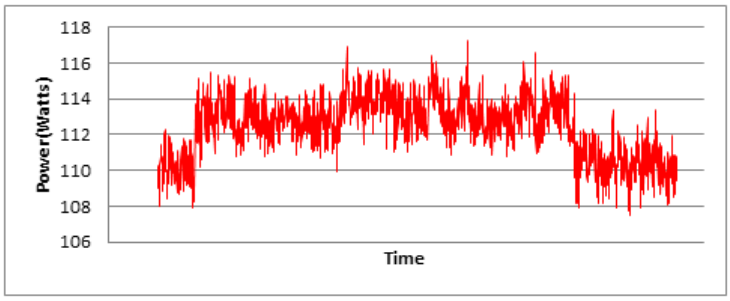

Figure 4.4: Power Trace for Constant Speed Experiment 1

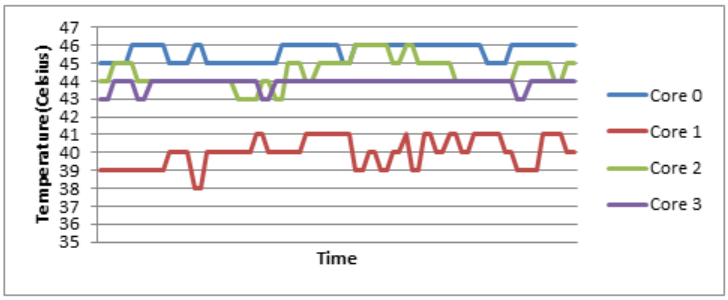

Figure 4.5: Temperature Trace for Constant Speed Experiment 1

Table 4.3 refers to the performance parameters of the first experiment where the benchmark is excuted at a constant speed step 5, or 2.16 GHz. There is a noticeable difference in cache misses among the cores. Figure 4.4 illustrates the power trace for the experiment. The time in which the benchmark is executed is shown by the region of the graph that is slightly more elevated than the rest of the graph. The temperature trace, illustrated by Figure 4.5 shows the that core 0 still exhibits a rise in temperature. The active cores show their activity in the temperature trace. Core 1 has a generally lower temperature than the other cores, which will be a trend among the experiments that will become much more apparent. Moreover, Core 3 shows a more constant temperature throughout the experiment where it only dips in temperature at some instances. These temperature drops are, however, not exactly evident in the power trace.

Constant Speed Step 10		
Instruction Count	48670802739	48670787471
L1 Cache Miss	1328556598	1328406671
L2 Cache Miss	142538023	143295344
L3 Cache Miss	1367513547	1367511978

Table 4.4: Performance Values for Constant Speed Experiment 2

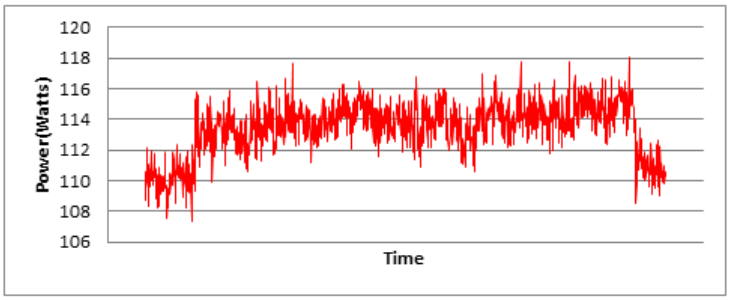

Figure 4.6: Power Trace for Constant Speed Experiment 2

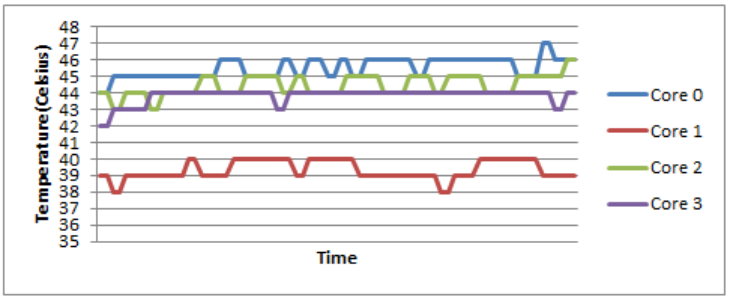

Figure 4.7: Temperature Trace for Constant Speed Experiment 2

In this experiment, shown by Table 4.4, the instruction count is an output parameter that stays in a specific range as evident by the previous Table 4.3. This specific range is one that is seen throughout the experiments considering that we are using the same benchmark, the Ackermann function, as our workload for the experiments. As opposed to the previous experiment, this experiment has slightly higher power consumption than the previous experiment shown by Figure 4.6. The temperature shown in Table 4.7 however, illustrates a more familiar pattern that is more similar to the previous speed step 5 experiment.

4.3.2 Two Level Experiments

In these experiments, while the benchmark is executing, the operating frequency will step up to another frequency, or step, after a given amount of time. For the

experiments, we have chosen 15 seconds.

Two Level Neighboring Speeds		
Instruction Count	48670802836	48670787752
L1 Cache Miss	1331130993	1330949609
L2 Cache Miss	141702154	141801795
L3 Cache Miss	1368544571	1368404495

Table 4.5: Performance Values for Two Level Experiment 1

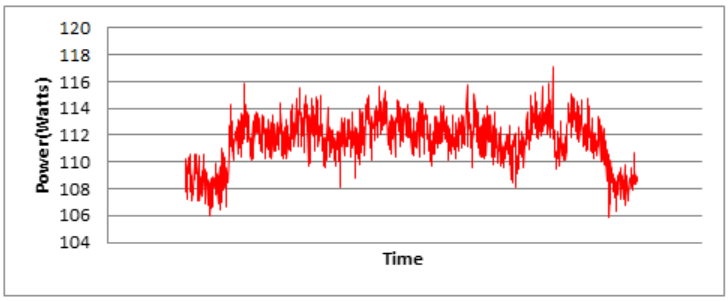

Figure 4.8: Power Trace for Two Level Speed Experiment 1

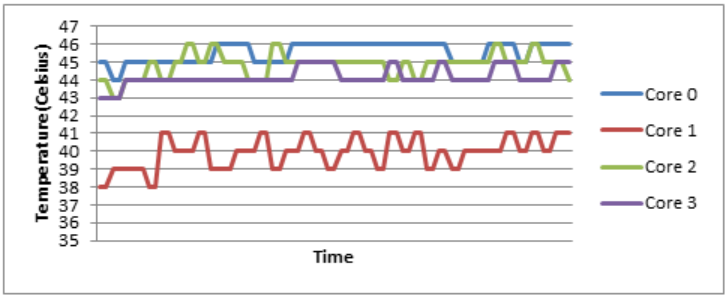

Figure 4.9: Temperature Trace for Two Level Speed Experiment 1

In this experiment, the operating frequency is set at step 5, or 2.16 GHz, then after the aforementioned 15 seconds steps up to step 6, or 2.26 GHz. This is also known as neighboring speeds. Considering that operating frequency has little effect on power, there is no real indication as to when the operating frequency changes as shown in Figure 4.8. While the other cores exhibit expected behaviors, the most unique characteristic of this experimental run is the temperature of core 1, shown in

Figure 4.9 as it displays a unique oscillating characteristic. This could be a result of switching activities outside the scope of this experiment.

Two Level Min-Max Speeds		
Instruction Count	48670802786	48670787483
L1 Cache Miss	1329282011	1329106371
L2 Cache Miss	143515069	143095169
L3 Cache Miss	1368382457	1368148803

Table 4.6: Performance Values for Two Level Experiment 2

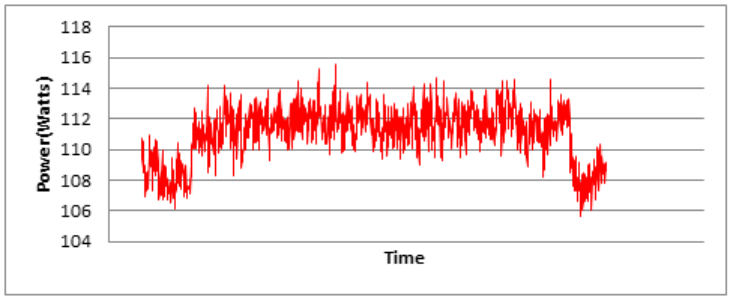

Figure 4.10: Power Trace for Two Level Speed Experiment 2

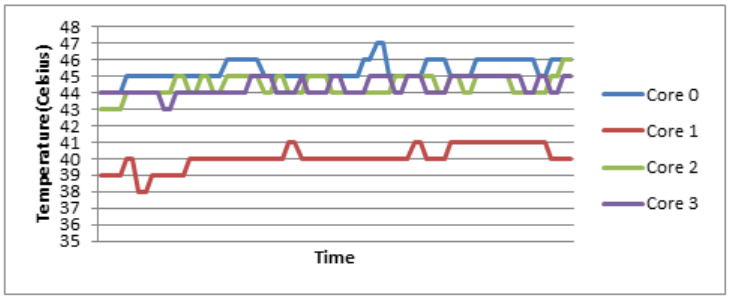

Figure 4.11: Temperature Trace for Two Level Speed Experiment 2

In this experiment, as opposed to the previous who changed levels from neighboring speeds, this experiment changes levels between the minimum speed and maximum speed. That is, the operating frequency steps from 1.67 GHz to 2.67 GHz. There is little disparity between execution and non-execution, illustrated by the power trace in Figure 4.10. The temperature also displays similar characteristics as the previous one

where the core one temperature was oscillating. Also while slight, the temperature of core 2 exceeds 45 degrees on more occasions than the previous, shown in Figure 4.11.

4.3.3 Oscillating Speed Experiments

In these experiments, while the benchmark is executing, the operating frequency will oscillate between two selected frequencies, or steps. The duration of each step was predetermined at 5 seconds each step. That is, every 5 seconds the frequency changes.

Oscillating: Neighboring Speeds		
Instruction Count	48670802760	48670787502
L1 Cache Miss	1329211980	1328790568
L2 Cache Miss	142646183	142790757
L Cache Miss	1368021261	1367634857

Table 4.7: Performance Values for Oscillating Experiment 1

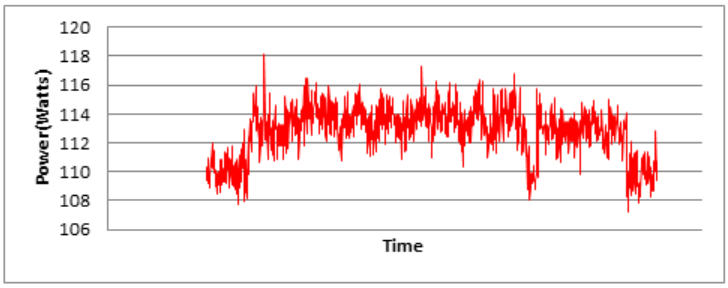

Figure 4.12: Power Trace for Oscillating Speed Experiment1

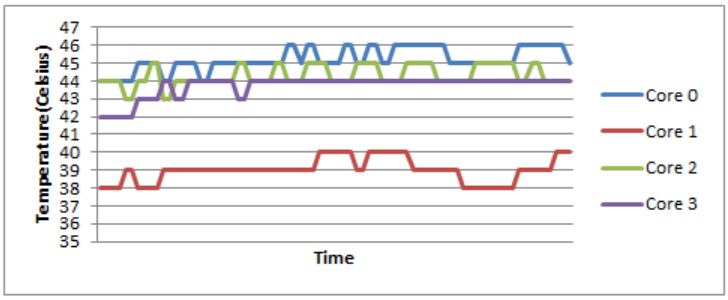

Figure 4.13: Temperature Trace for Oscillating Speed Experiment 1

In the experiment above, the frequency was oscillated between the neighboring steps 5 and 6, or 2.16 GHz and 2.26 GHz, respectively. The power trace, in Figure 4.12, is slightly more pronounced as there are periodic drops in the power. The temperature however remained, similar to the other experiments. However, core 3 exhibited a constant speed for a majority of the experimental run as shown in Figure 4.13. The performance parameters as shown in Table 4.7 remain in similar ranges. Though considering that the cache misses are not the same gives rise to the notion that multicore operations are more unpredictable.

Oscillating Min-Max Speeds		
Instruction Count	48670800415	48670785142
L1 Cache Miss	1332962098	1333059656
L2 Cache Miss	142817511	142400212
L3 Cache Miss	1367426904	1367428881

Table 4.8: Performance Values for Oscillating Experiment 2

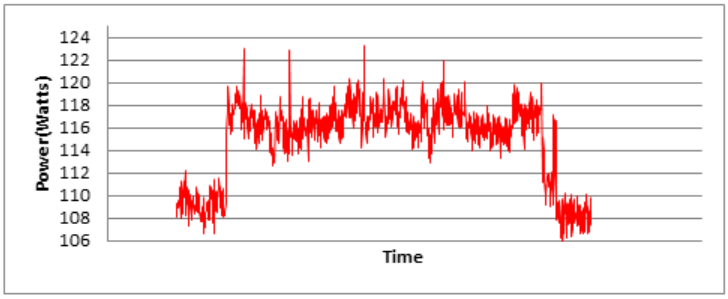

Figure 4.14: Power Trace for Oscillating Speed Experiment 2

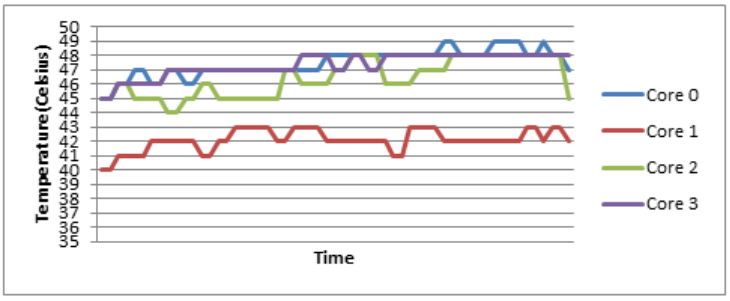

Figure 4.15: Temperature Trace for Oscillating Speed Experiment 2

Finally, in the last experiment presented, the frequency was oscillated between the minimum speed of 1.67 GHz and the maximum speed of 2.67 GHz. This experiment produced the highest temperatures as shown in Figure 4.15. The entire core experienced some form of increased heat as even core one exceed its standard operating temperatures of 39-40 and into the 42 degree and above range. The power produced, in Figure 4.14, in this experiment are also at its highest as a bulk of the operation is done between 116 and 118 Watts while also exceeding 120 Watts at certain instances. There are noticeable spikes in the power, indicating when the operating frequency changes. Moreover, the disparity between when the worload is undergoing computation versus when it is not is more pronounced.

4.4 Summary

In this chapter we performed experiments of three different DVFS policies: *a*) Constant speed; *b*) Two level frequency; and *c*) Oscillating frequency. For each DVFS policy we ran 2 experiments. For the constant speed policy we executed the experiments at speed step 5 and 10, or operating frequencies 2.13 GHz and 2.67 GHz. In regards to the two level experiments, we examine the behaviors of stepping up the operating frequency from the slowest speed to fastest speed as well as stepping up the operating frequency from speed step 5 to speed step 6, or in other words, neighboring speeds. The oscillating speeds experiments require the same operating frequencies as the two level experiments. However, they oscillate between said frequency levels rather than stepping up to them just once. General observations show that despite there being a minimum amount of work done on core 0. There is still a rise in temperature during the experiments.

Moreover, core 1 exhibits the lowest temperature, several degrees lower than others, than the rest of the cores. As evident in equation (2.2), the operating frequency

does not have much of an effect on the power compared to the capacitance or supply voltage. This attribute gives reason as to why many of the power figures look similar. By way of the performance parameters, it is seen that a multicore platform is more unpredictable than a single core platform due to the different results of the cache parameters given by the different runs. Lastly, the experiments also show that modifying the operating frequency has little effect on the instruction count, cache misses, as well as other performance parameters.

CHAPTER 5

CONCLUSION

In this chapter, we summarize our research presented in this thesis and discuss possible future work of this research.

5.1 Concluding Remarks

The rapidly growing power consumption and heat generation not only significantly increase the packaging and cooling costs, but also severely degrade the performance and reliability of computing systems. It has been well recognized that power and thermal issues have posed enormous challenges in computer system design and posed a serious threat to slow down the continuous evolution of computer technology. For the past a few decades, there has been extensive theoretical research efforts that deals with power and thermal issues in computer system design. However, most of them are based on idealized assumptions and/or simplified theoretical models. While these assumptions and models help to greatly simplify complex problems and make them theoretically manageable, engineers and practitioners have to deal with complicated factors in the real world that may not be captured by these assumptions and models.

In this thesis, we focus on the development of a practical multi-core hardware testing platform. When compared to other methodologies to validate theoretical results of power and thermal aware resource management, our system differs in that, unlike other works, it does not rely on simplified models or idealized assumptions. This essentially results in a system which can obtain real experimental results directly from an actual computing system. This attributes to the practicality of our testing platform. With the aim for flexibility and portability, we developed a hardware testing platform based on an Intel-17 920 "Bloomfield" processor, running the Ubuntu Linux

operating system with kernel version 3.5.0.43. Due to the libraries we created, we also allow researchers to develop their own testing tools. Furthermore, we investigated the various phenomena involving the effect of different DVFS policies on the temperature and power consumption of the CPU which are illustrated by Haung et al [30].

5.2 Future Work

In this thesis, we have done extensive research work on obtaining accurate thermal and power characteristics as well as testing various frequency scaling policies. In the future, we would like to utilize a method that allows us to use commercially available benchmarks such as SPEC2000, Splash-2, PARSEC, and NAS NPB [10, 22, 64]. This would require utilizing a different control and mapping structure than pthreads. Moreover, when we aimed for portability and flexibility in this testing platform, we have not patched the kernel. However, if we wish to implement more sophisticated scheduling structures as well as real-time scheduling algorithms, then it will be imperative to patch the kernel. Another avenue in which I would like to explore is the prospect of allowing the user to throttle onboard fan speeds as the only method of cooling cores on our system is a combination of the sleep function call and reducing the CPU operating frequency to its lowest level.

Moreover, Due to the increased usage of graphical processing units(GPUs) it would be very logical to extend our research to allow for the practical validation of various theoretical works centered around the power and thermal aware resource management of both NVidia and AMD GPUs.

BIBLIOGRAPHY

[1] *Lm-sensors linux hardware monitoring*, 2006 (accessed April 25, 2014).

[2] *System Drivers*. INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS, 2009.

[3] D. Buttlar . Nichols and J. P. Farrell. *Pthreads Programming*. O'Reilly and Associates, Inc, 1996.

[4] A.H. Ajami, K. Banerjee, and M. Pedram. Modeling and analysis of nonuniform substrate temperature effects on global ulsi interconnects. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(6):849–861, June 2005.

[5] E. Ajkunic, H. Fatkic, E. Omerovic, K. Talic, and N. Nosovic. A comparison of five parallel programming models for c++. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1780–1784, May 2012.

[6] K.H. Ansari, P. Chitra, and P. Sonaiyakarthick. Power-aware scheduling of fixed priority tasks in soft real-time multicore systems. In *Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on*, pages 496–502, March 2013.

[7] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb 2002.

[8] P. Bailis, V.J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer. Dimetrodon: Processor-level preventive thermal management via idle cycle injection. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 89–94, June 2011.

[9] Min Bao, A. Andrei, P. Eles, and Zebo Peng. Temperature-aware idle time distribution for leakage energy optimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(7):1187–1200, July 2012.

[10] C. Bienia, S. Kumar, and K. Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pages 47–56, Sept 2008.

[11] Dick Buttlar Bradford Nichols and Jacqueline Proulx Farrell. *Pthreads Programming*. O'Reilly and Associates, Inc., 1996.

[12] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. hwloc: A generic framework for managing hardware affinities in hpc applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 180–186, Feb 2010.

[13] Slo-Li Chu, Shiue-Ru Chen, and Sheng-Fu Weng. Design a low-power scheduling mechanism for a multicore android system. In *Parallel Architectures, Algorithms and Programming (PAAP), 2012 Fifth International Symposium on*, pages 25–30, Dec 2012.

[14] A.K. Coskun, T.S. Rosing, and K. Whisnant. Temperature aware task scheduling in mpsocs. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, April 2007.

[15] E.H.M. Cruz, M. Diener, and P.O.A. Navaux. Using the translation lookaside buffer to map threads in parallel applications based on shared memory. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 532–543, 2012.

[16] F. Dabiri, N. Amini, M. Rofouei, and M. Sarrafzadeh. Reliability-aware optimization for dvs-enabled real-time embedded systems. In *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*, pages 780–783, March 2008.

[17] Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi. Application-to-core mapping policies to reduce memory system interference in multi-core systems. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 107–118, 2013.

[18] R.P. Dick, D.L. Rhodes, and W. Wolf. Tgff: task graphs for free. In *Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on*, pages 97–101, Mar 1998.

[19] N. Fisher, Jian-Jia Chen, Shengquan Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 131–140, 2009.

[20] David Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, May 2005.

[21] M. Ghasemazar, H. Goudarzi, and M. Pedram. Robust optimization of a chip multiprocessor's performance under power and thermal constraints. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 108–114, Sept 2012.

[22] R. Giladi and N. Ahitav. Spec as a performance evaluation measure. *Computer*, 28(8):33–42, Aug 1995.

[23] B. Gorjiara, N. Bagherzadeh, and P. Chou. An efficient voltage scaling algorithm for complex socs with few number of voltage modes. In *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, pages 381–386, Aug 2004.

[24] S. Grothklags and A. Streit. On the comparison of cplex-computed job schedules with the self-tuning dynp job scheduler. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 250–, April 2004.

[25] V. Hanumaiah and S. Vrudhula. Temperature-aware dvfs for hard real-time applications on multicore processors. *Computers, IEEE Transactions on*, 61(10):1484–1494, Oct 2012.

[26] V. Hanumaiah and S. Vrudhula. Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling. *Computers, IEEE Transactions on*, 63(2):349–360, Feb 2014.

[27] Liqiang He and Cha Narisu. Heat spreading aware floorplan for chip multicore processor. 2:231–239, 2009.

[28] Seongmoo Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. In *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on*, pages 217–222, Aug 2003.

[29] Huang Huang and Gang Quan. Leakage aware energy minimization for real-time systems under the maximum temperature constraint. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.

[30] Huang Huang, Gang Quan, J. Fan, and Meikang Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 363–368, June 2011.

[31] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(5):501–513, May 2006.

[32] Intel. Intel 64 and ia-32 architecturessoftware developers manual, Feb 2014.

[33] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: methodology and empirical data. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 93–104, Dec 2003.

[34] A.Chandrakasan J. Rabaey and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2003.

[35] Da-Cheng Juan, Huapeng Zhou, D. Marculescu, and Xin Li. A learning-based autoregressive model for fast transient thermal analysis of chip-multiprocessors. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 597–602, Jan 2012.

[36] N. Kamiura, A. Saitoh, T. Isokawa, and N. Matsui. A two-pronged approach of power-aware voltage scheduling for real-time task graphs in multi-processor systems. In *Multiple-Valued Logic, 2009. ISMVL '09. 39th International Symposium on*, pages 151–156, May 2009.

[37] G. Karmakar and A. Kabra. Energy-aware thermal comfort-band maintenance scheduling under peak power constraint. In *Intelligent Computational Systems (RAICS), 2013 IEEE Recent Advances in*, pages 122–127, Dec 2013.

[38] Wonyoung Kim, M.S. Gupta, Gu-Yeon Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134, Feb 2008.

[39] T. Kolpe, A. Zhai, and S.S. Sapatnekar. Enabling improved power management in multicore processors through clustered dvfs. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, 2011.

[40] Masaaki Kondo, Hiroshi Sasaki, and Hiroshi Nakamura. Improving fairness, throughput and energy-efficiency on a chip multiprocessor through dvfs. *SIGARCH Comput. Archit. News*, 35(1):31–38, March 2007.

[41] Fanxin Kong, Wang Yi, and Qingxu Deng. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, 2011.

[42] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 123–128, Jan 2011.

[43] Wenming Li, Krishna Kavi, and Robert Akl. A non-preemptive scheduling algorithm for soft real-time systems. *Comput. Electr. Eng.*, 33(1):12–29, January 2007.

[44] Yingmin Li, K. Skadron, D. Brooks, and Zhigang Hu. Performance, energy, and thermal considerations for smt and cmp architectures. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 71–82, Feb 2005.

[45] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[46] Guanglei Liu, Ming Fan, and Gang Quan. Neighbor-aware dynamic thermal management for multi-core platform. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 187–192, March 2012.

[47] Guanglei Liu, Gang Quan, and Meikang Qiu. Throughput maximization for intel desktop platform under the maximum temperature constraint. In *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*, pages 9–15, Aug 2011.

[48] Junyang Lu and Yao Guo. Energy-aware fixed-priority multi-core scheduling for real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 277–281, Aug 2011.

[49] Junyang Lu and Yao Guo. Energy-aware fixed-priority multi-core scheduling for real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 277–281, Aug 2011.

[50] A. Schulz W. E. NagelTools M. S. M uller, M. M. Resch. *Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing.* 2009.

[51] K. Malkowski, P. Raghavan, M. Kandemir, and M.J. Irwin. Phase-aware adaptive hardware selection for power-efficient scientific computations. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 403–406, Aug 2007.

[52] Pierre Michaud, Yiannakis Sazeides, Andr Seznec, Theofanis Constantinou, and Damien Fetis. An analytical model of temperature in microprocessors. Technical report, 2005.

[53] E.H. Molina da Cruz, M.A. Zanata Alves, A. Carissimi, P.O.A. Navaux, C.P. Ribeiro, and J. Mehaut. Using memory access traces to map threads and data on hierarchical multi-core platforms. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 551–558, 2011.

[54] M. Nikitovic, T. De Schampheleire, and M. Brorsson. A study on periodic shut-down for adaptive cmps in handheld devices. In *Computer Systems Architecture Conference, 2008. ACSAC 2008. 13th Asia-Pacific*, pages 1–7, Aug 2008.

[55] Kunle Olukotun and Lance Hammond. The future of microprocessors. *Queue*, 3(7):26–29, September 2005.

[56] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501, Aug 2006.

[57] Gang Quan and V. Chaturvedi. Feasibility analysis for temperature-constraint hard real-time periodic tasks. *Industrial Informatics, IEEE Transactions on*, 6(3):329–339, Aug 2010.

[58] L. Dagum D. Kohr D. Maydan J. McDonald R. Chandra, R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2000.

[59] R. Raghavendra, P. Ranganathan, V. Talwar, Xiaoyun Zhu, and Zhikui Wang. Motivating co-ordination of power management solutions in data centers. In *Cluster Computing, 2007 IEEE International Conference on*, pages 473–473, Sept 2007.

[60] Arun Rangasamy, Rahul Nagpal, and Y.N. Srikant. Compiler-directed frequency and voltage scaling for a multiple clock domain microarchitecture. In *Proceedings of the 5th Conference on Computing Frontiers*, CF '08, pages 209–218, New York, NY, USA, 2008. ACM.

[61] S. Saha, Ying Lu, and J.S. Deogun. Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, pages 41–50, Aug 2012.

[62] H. Salamy, S. Aslan, and D. Methukumalli. Task scheduling on multicores under energy and power constraints. In *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*, pages 1–4, 2013.

[63] Hiroshi Sasaki, Yoshimichi Ikeda, Masaaki Kondo, and Hiroshi Nakamura. An intra-task dvfs technique based on statistical analysis of hardware events. In *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, pages 123–130, New York, NY, USA, 2007. ACM.

[64] Sangmin Seo, Gangwon Jo, and Jaejin Lee. Performance characterization of the nas parallel benchmarks in opencl. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 137–148, Nov 2011.

[65] H.F. Sheikh and I. Ahmad. Dynamic task graph scheduling on multicore processors for performance, energy, and temperature optimization. In *Green Computing Conference (IGCC), 2013 International*, pages 1–6, June 2013.

[66] K. Skadron, M.R. Stan, Wei Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware computer systems: Opportunities and challenges. *Micro, IEEE*, 23(6):52–61, Nov 2003.

[67] Pengliu Tan. Task scheduling of real-time systems on multi-core architectures. In *Electronic Commerce and Security, 2009. ISECS '09. Second International Symposium on*, volume 2, pages 190–193, May 2009.

[68] A. Vega, A. Buyuktosunoglu, and P. Bose. Smt-centric power-aware thread placement in chip multiprocessors. In *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pages 167–176, 2013.

[69] Xiaorui Wang, Kai Ma, and Yefu Wang. Adaptive power control with online model estimation for chip multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 22(10):1681–1696, Oct 2011.

[70] V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *Parallel Processing*

Workshops (ICPPW), 2012 41st International Conference on, pages 262–268, Sept 2012.

[71] W. Wolf, A.A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(10):1701–1713, Oct 2008.

[72] Q. Wu, V.J. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D.W. Clark. A dynamic compilation framework for controlling microprocessor energy and performance. In *Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on*, pages 12 pp.–282, Nov 2005.

[73] Chuan-Yue Yang, Jian-Jia Chen, L. Thiele, and Tei-Wei Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 9–14, March 2010.

[74] Inchoon Yeo and Eun Jung Kim. Temperature-aware scheduler based on thermal behavior grouping in multicore systems. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 946–951, 2009.

[75] Inchoon Yeo, Chih Chun Liu, and Eun Jung Kim. Predictive dynamic thermal management for multicore systems. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 734–739, June 2008.

[76] Buyoung Yun, K.G. Shin, and Shige Wang. Thermal-aware scheduling of critical applications using job migration and power-gating on multi-core chips. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 1083–1090, Nov 2011.

[77] Gang Zeng, T. Yokoyama, H. Tomiyama, and H. Takada. Practical energy-aware scheduling for real-time multiprocessor systems. In *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE International Conference on*, pages 383–392, Aug 2009.

[78] Bo Zhai, D. Blaauw, D Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 868–873, July 2004.

[79] Lin Zhou and Lei Yu. Frequency-utilization based power-aware schedule policy for real-time multi-core system. In *Green Computing and Communications*

(GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, pages 200–207, Aug 2013.