Soft Power Capping for Improved Performance of Computing Systems

Honors Thesis submitted by Natalie Serrino

in partial fulfillment of the Sc.B with Honors in Computer Engineering

at Brown University

April 20, 2012

Prepared under the direction of

Professor Sherief Reda, Advisor

Professor Iris Bahar, Reader

Professor Rashid Zia, Reader

Table of Contents

Abstract

The costs associated with power and cooling are increasingly becoming the largest cost of datacenter and supercomputer ownership. Power capping is a necessary decision to manage these costs and maintain system stability, and consequently dynamic power management policies are needed to optimize system performance.

Most prior work in this area has focused on methods that satisfy *hard power capping*, in which the instantaneous power may never violate the cap. We propose *soft power capping*, where it is the average power rather than the instantaneous power that must meet the budget. In some prior work, the concept of fine-grained soft power capping is used implicitly, with frequency as the control knob. We formalize the concept and extend it by using the number of active cores on the processor as the control knob.

We provide a theoretical framework for predicting the performance of soft power capped workloads and propose the concept of energy credits to control the mode of execution in soft power capping. We propose two applications of soft power capping. First, it provides an infinite set of power-performance tradeoffs, whereas hard power capping provides a finite set. Second, soft power capped workloads can outperform hard power capped workloads for the same power consumption. Finally, we propose an algorithm to dynamically size energy credits during runtime. We validate our proposed framework on a real quad-core based system and demonstrate significant improvements in performance.

Chapter 1: Introduction

1.1: Multi-core Architecture

Over the past few decades, semiconductor-manufacturing technologies have decreased the transistor feature size dramatically, from 10 μm in 1971 to 32 nm in 2012. Smaller transistors allow for faster switching time and until recently, a lower supply voltage. Because chip power consumption is proportional to the clock speed and the square of the supply voltage, processor designers have for years been able to simultaneously reduce the supply voltage and increase the clock speed of their chips to maintain the same power consumption per unit area. However, it has since become infeasible to scale down the supply voltage in transistors to the same degree, even given smaller device lengths, due to the increased impact of leakage power, noise and process variation on data integrity at low supply voltage levels. Consequently, raising the clock speed also increases the power consumption of chips today, because the supply voltages can no longer scale down accordingly.

Cooling systems have been developed to absorb the heat generated by the increased power consumption of the chips, but the heat that can be practically removed by economical cooling systems is limited. As a result, the growth in clock speeds has slowed and it is necessary for chip designers to find new strategies of increasing chip performance given these limitations on chip frequencies.

Today's state of the art processor architectures now incorporate multiple cores on a single chip to provide the increased performance that the industry is expected to provide. An example of the organization of multi-core processors is shown in Figure 1 for Intel i7-2600K 4-core processor. These chip multi-processors

(CMPs) simultaneously execute instructions in each core to increase the total throughput compared to a single-core processor. Because CMPs rely on parallelism to provide performance gains, workloads are broken into multiple threads, which can be executed on different cores. Synchronization constructs and protocols are also necessary to allow for the leveraging of the limited parallelism offered by interdependent threads.

Due to the continued shrinking of transistors, more cores can fit on the same chip area. Thus finding ways to exploit the thread level parallelism (TLP) in a given workload has become a key challenge to achieve the computing performance gains made possible by an increasing number of cores.

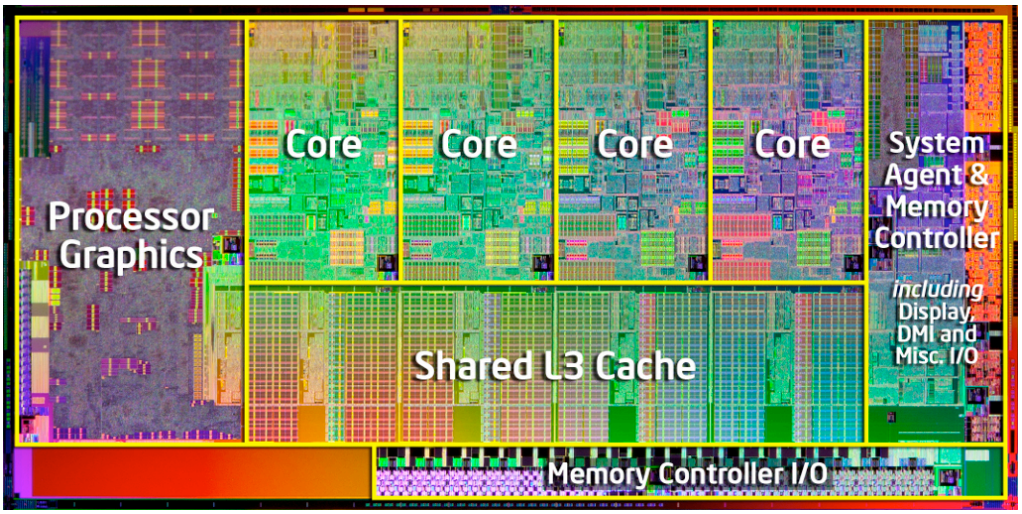

Figure 1: Die map depicting the layout of an Intel i7-2600K 4-core processor. Source: http://www9.pcmag.com/media/images/243264-intel-core-i7-2600k-die-map.jpg

1.2: Power Consumption in Datacenters and Supercomputing Facilities

The rising cost of energy relative to other expenditures is a critical issue facing modern datacenters and supercomputing facilities. Over the past decade, the cost of power and cooling for datacenters has increased 400% [6], depicted in

Figure 2. These factors constitute the largest cost of datacenter ownership, comprising up to 50% of overall spending. In 2009, datacenters consumed $4.5 billion dollars of energy, which accounted for over 1.5% of the total electricity in the US [12]. Costs associated with power and cooling are projected to continue rising. Consequently, a serious challenge for datacenters and supercomputing facilities moving forward is the minimization of costs associated with power and cooling, while still meeting the required performance goals. It is necessary to address these costs through the management of system power consumption and operating temperatures. Existing methods to control power and/or temperature, while still optimizing overall system performance, include low power sleep or nap modes and dynamic voltage and frequency scaling (DVFS).

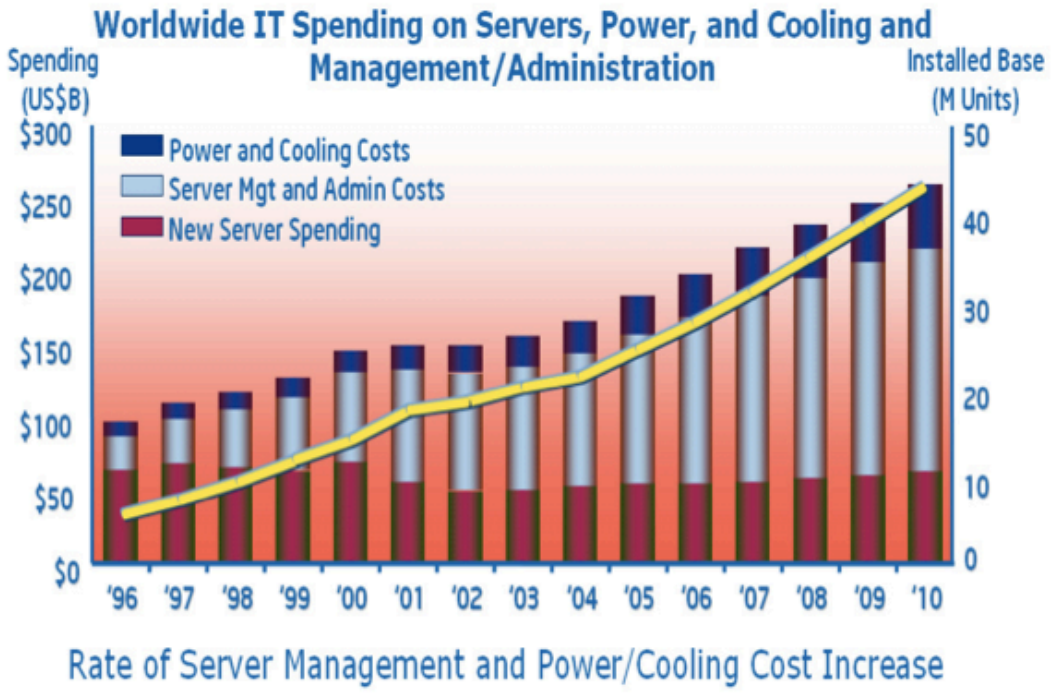

Figure 2 (from [6]) depicts world spending on different server costs.

An additional challenge stemming from the need to manage power consumption and temperature in datacenters is the necessity to do so in the face of changing system constraints and workload characteristics. The datacenter cannot control the resources made available to it by the power grid. In fact, the power provided by the grid may change with its overall load and access to additional energy reserves. Thus a datacenter may be faced with a fluctuating power supply during the execution of its workloads, so it needs to adapt to these changing power constraints during runtime without sacrificing performance unnecessarily. New strategies to adjust the behavior of a system to meet dynamically changing constraints on power and temperature have been explored, such as thread packing [3] and online workload phase detection [8]. The techniques mentioned earlier (DVFS, sleep and nap states) also have the capacity to be applied during execution time to meet such constraints.

1.3: Power Capping and Dynamic Voltage and Frequency Scaling

Upper bounds on power consumption are necessary in servers due to fundamental limits of a given system's capacity as well as the need to maintain stable operating temperatures. Finding the optimal system configuration for a given power budget is necessary to maximize performance and minimize energy consumption.

Power caps are subject to change, and dynamic voltage and frequency scaling (DVFS) is a common technique of meeting different power budgets on the same system. DVFS varies the supply voltage and clock speed of the system to produce

different power-performance tradeoffs. A higher clock frequency requires a correspondingly higher supply voltage to maintain data integrity given a fixed transistor length. Therefore increasing the frequency produces a generally linear increase in runtime and a cubic increase in power, due to power's quadratic dependence with voltage and linear dependence with frequency.

A system utilizing DVFS selects the highest frequency that allows for power consumption at or below the power budget. DVFS settings implemented on a per-core basis or exclusively chip-wide. Because DVFS overheads are only on the order of microseconds [1], it is an efficient control knob that can also be employed dynamically for power budgets that change during runtime. Figure 3 shows the power and performance of the PARSEC 2.0 benchmark *blackscholes,* running at different DVFS settings on our 4-core system. The specifics of our system are detailed in Chapter 3.

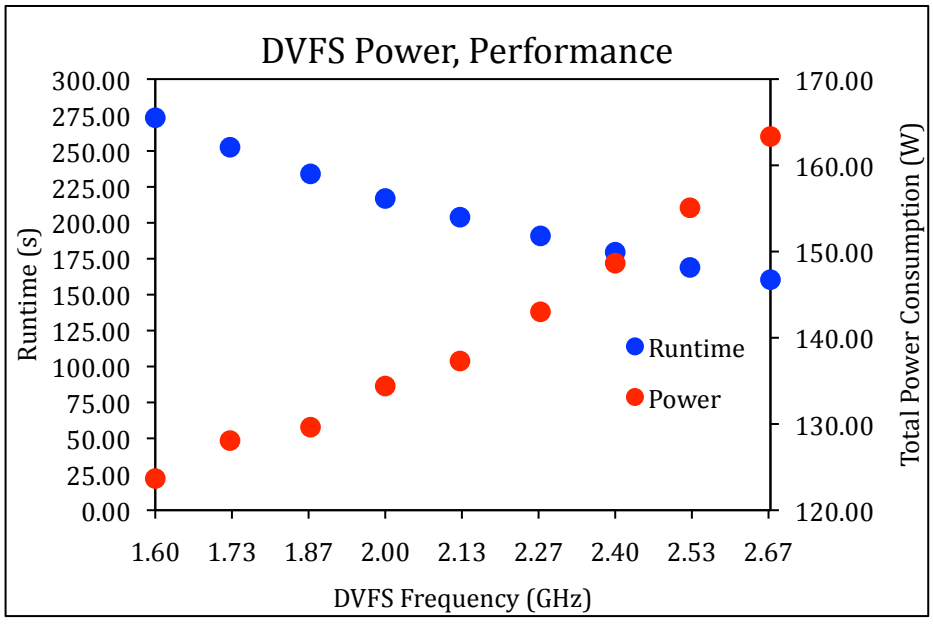

Figure 3: Power and runtime of the workload blackscholes by DVFS Frequency.

1.4: Multi-threaded applications

To exploit the performance gains made possible by multi-core architectures, individual applications are broken into multiple threads that can be executed simultaneously on different cores. Exploiting the thread level parallelism (TLP) of the workload is necessary to maximize throughput and minimize runtime. For the sequential pieces of the workload, multiple synchronization constructs such as spin locks or transactions are used.

Because servers implementing new power management techniques run a variety of applications, the selection of a benchmark suite to evaluate new power management techniques is important. The techniques being tested need to provide benefits for a diverse set of workloads in order to be applicable to real systems. This work utilizes the PARSEC 2.0 benchmark suite in all of our experiments [5]. The PARSEC 2.0 benchmark suite targets "emerging applications", which they define as applications that are difficult but possible to execute well, and for which there exists strong demand. These thirteen multithreaded benchmarks represent a diverse set of applications, such as data compression, 3D rendering, and video encoding, for different domains such as finance, computer vision, and data mining.

Each PARSEC 2.0 benchmark can be broken into a variable number of threads. Parameters such as the rate of data sharing and the size of the working set of the benchmarks are varied to evaluate the performance of processors under different types of workloads. The proportions of floating point operations, reads, and writes to the total number of instructions in the workload also vary, as do the frequency of synchronization primitives, such as locks, barriers, and condition

variables. For example, an 8-threaded version of *canneal*, a simulated annealing algorithm, uses 0.45 billion floating-point operations out of 7 billion total instructions, and 34 locks. In contrast, an 8-threaded version of *bodytrack*, a computer vision algorithm used to determine the pose of a human body, uses 6.08 billion floating point operations out of 14.04 billion total instructions, and 28,538 locks [5]. The diversity and relevance of the PARSEC 2.0 benchmark suite, as well as its ability to be broken into a variable number of threads, make it a good metric for the performance of our power management strategies for multi-core systems.

1.5: Research Overview

In this work we investigate a technique which seeks to improve the performance of a server for a given power budget. Most of the previous work in this area has relied on what we will refer to as *hard power capping*, in which the selected settings must always produce power consumption at or under the power budget. This technique is contrasted with our idea of *soft power capping*, where it is the average power of the system that is capped rather than the instantaneous power consumption. A soft power capped system switches between two modes of operation during runtime, one consuming less power than the budget and one consuming more power than the budget. These modes are balanced to achieve the average power consumption specified by the cap.

The two control knobs that we consider in this work are DVFS and the number of active cores on which the workload operates. Higher clock speeds as well as higher number of active cores (cores turned ON rather than in a sleep state)

increase the power consumption of the system. As a result, the higher power setting employs a higher frequency and/or a higher number of active cores than does the lower power setting. The authors of [2] and [10] have shown the effectiveness of soft power capping by varying frequency, so we focus in this work on improving performance through soft power capping by switching of the number of active cores.

We first develop a theoretical framework for the power consumption, performance, and energy consumption of a system employing soft power capping as a function of the fraction of instructions in the workload executed in the higher power setting. We implement soft power capping in a real system and compare theoretical and experimental soft power capping data to show the soundness of the theoretical model.

We show that soft power capping has two applications. First, it provides a finer grained set of power-performance tradeoffs than does hard power capping without the need to increase the number of settings of the system. Second, employing soft power capping outperforms certain existing settings in a system for the same power budget. We propose an implementation of soft power capping based on *energy credits*, which are accumulated when the system uses less power than the budget and spent when the system uses more power than the budget.

The subsequent chapters are organized in the following manner. Chapter 2 reviews related work. Chapter 3 describes our theoretical framework and motivations for soft power capping. Chapter 4 discusses the power-runtime Pareto frontier and our first set of experimental results for soft power capping. Chapter 5 improves upon the implementation of soft power capping discussed in Chapter 4 by

using energy credits, evaluates the performance of this improved version of soft power capping, and proposes a simple algorithm to select the size of the energy credit during runtime. Chapter 6 concludes the work.

1.6: Experimental Setup

All of the experiments in this work were performed on an Intel quad-core Core i7 940 45 nm processor. Figure 4 shows the motherboard of our experimental setup. The operating frequency of the processor is set using the Linux *cpufreq* library, with nine equally spaced settings between 1.60 GHz and 2.67 GHz. The operating system is the 2.6.10.8 Linux kernel OS. We used an Agilent A34401 digital multimeter to measure the power consumption of the server. We used *pfmon*, a performance-monitoring tool for Linux, to sample the power data from the multimeter every 100 milliseconds. The power data from the system served as the primary input to our control algorithm.

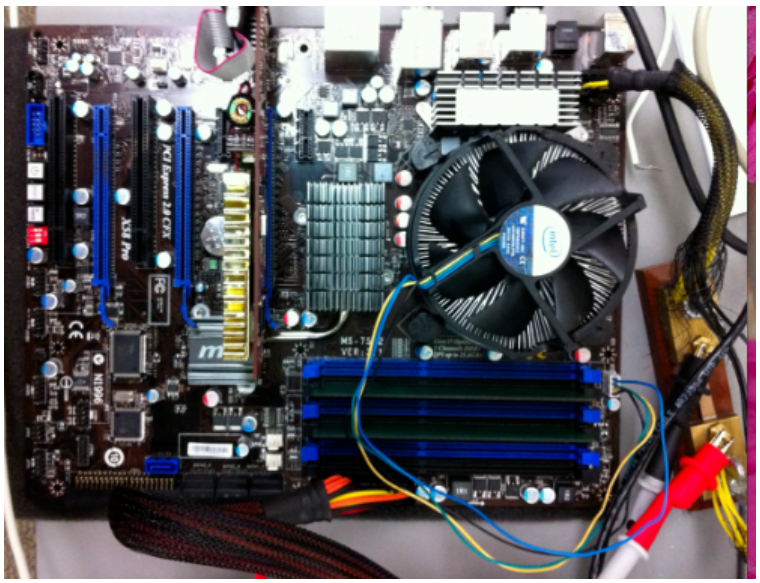

Figure 4: Motherboard used for experimental data, containing an Intel i7 processor.

The control algorithms were implemented through a MATLAB interface with the pfmon utility. This algorithm was responsible for taking the power measurements provided by the pfmon tool and adjusting the number of active cores according to the policy. Dynamically switching the number of active cores was accomplished through the migration of threads. We used the packing strategy proposed by [3] so that the workloads could be launched with a set number of threads. These threads are then packed on to a variable number of cores.

We utilized the PARSEC 2.0 benchmark suite [5] to evaluate the performance of our soft power capping management policy. The benchmarks in the PARSEC 2.0 suite have the advantage of being broken into a variable number of threads. We used four-thread versions of the benchmarks for our experiments and this parameter was set using the command line option provided by the suite.

Chapter 2: Previous Work

The importance of effective power provisioning in the context of real datacenters is explored in [11]. Fan *et al.* found that power capping can be effective to allow additional machines to be hosted and also as a safety mechanism to stay within the operating capacity of the datacenter. They evaluate the ability of DVFS to reduce peak power consumption on a large-scale system, and conclude that it is an effective technique. They also highlight the importance of power-efficiency during non-peak power consuming phases of datacenter operation.

In [12], Harchol-Balter *et al*. determine that the optimal power allocation strategy for a datacenter depends on multiple factors, such as the maximum and minimum server frequencies, the power-to-frequency relationship in the processors, the arrival rate of jobs and the configuration of the server itself. For example, they show that in some cases it might be optimal to run fewer servers at the highest operating frequency, and in other cases it would be more advantageous to distribute that same amount of power among a greater number of machines running at a lower frequency. They highlight the importance of finding the optimal power allocation for a given datacenter, which can typically improve performance by a factor of 1.4, but can reach as high as a factor of 5.

There has been significant progress in developing dynamic power management policies for multi-core architectures to achieve the ultimate goal of maximizing performance under a variable power cap [1], [2], [3], [4]. Our work intends to expand on these works by proposing soft power capping as an additional

control knob, in particular by varying the number of active cores in the processor to improve the overall performance of the server.

Isci *et al*. simulated four distinct global power management policies on multi-core systems and analyzed the performance of each under a variable power budget [1]. The control knob used for each of the policies is per-core DVFS with three settings, whereas our work utilizes chip-wide DVFS with nine settings. The policy that performed the best while meeting the power budget, MaxBIPS, performs an exhaustive search of all 3^N combinations of the cores' DVFS settings, where N is the number of cores in the system. MaxBIPS determines the optimal setting for each core such that the overall performance is maximized and the power budget is met. One drawback of this approach is that the number of combinations to be analyzed increases exponentially with the number of cores in the system. As a result, while it may be feasible to examine 81 (3^4) possibilities as it does in the four-core system analyzed by their paper, it becomes impractical to do so as the number of settings and/or the number of cores increases past a certain threshold.

Wang *et al*. also use per-core DVFS as the manipulated control knob for their power management policy, and improve upon the work done by Martonosi *et al*. [2]. They guarantee that the power and temperature of the system remains at or below the desired level by basing their power management policy on multiple-input multiple-output (MIMO) control theory. This approach is more scalable than the one proposed by Martonosi *et al*. because it does not rely on an exhaustive search of all possible configurations or the assumption that performance and power for different DVFS settings can be accurately estimated during runtime. Their control algorithm

uses the online power, performance, and temperature data as inputs to produce the ideal DVFS levels for each core, which are then adopted by the system. These ideal DVFS levels are then realized through the modulation of existing DVFS settings. For example, they state that "to approximate 2.89 GHz … the modulator would output the sequence, 2.67, 3, 3, 2.67, 3, 3, etc, on a smaller timescale" [2]. Interestingly, this approach to DVFS is a form of fine-grained soft power capping because it relies on a low power mode and a high power mode to approximate a setting whose performance and power consumption lies between the two modes. An important distinction to note between our work and theirs is that Isci *et al*. run multiple copies of a single-threaded workload on each core, whereas we use multi-threaded applications.

Gandhi *et al*. explore a different form of soft power capping (although they also do not use the term itself) in their work, IdleCap, which achieves a given power cap over a 1 second granularity timescale [10]. IdleCap exploits non-linearity with respect to frequency in the set of possible power-performance settings of a server. In other words, they improve upon the marginally decreasing gains in power as frequency decreases by employing soft power capping. The low power mode utilized by IdleCap is, as the name suggests, a completely idle state, and the high power mode is a high-frequency state (3 GHz for their work). Alternating between these two modes achieves a higher average frequency for the same power budget compared to the existing frequency settings. As a result, they achieve better performance for the same power budget by using soft power capping compared to the static configurations available on the system.

Shi *et al*. explore the idea of instantaneous thermal cap violations [7]. They find that by applying what they refer to as a soft thermal constraint, in which a given temperature cap can be violated for a short period of time, as opposed to hard thermal constraint that may never be violated, leads to better performance. This idea of soft thermal constraints differs from our idea of soft power capping because they do not guarantee that the average temperature will be at or below the constraint, but rather that it will be satisfied for the majority of the workload's execution. The authors found improvements of 13% in a single core processor when they allowed the temperature constraint to be violated by 10°C for 100 seconds.

Cochran *et al*. propose a technique that allows for a graceful transition in the number of active cores in the system [3]. This technique, thread packing, is to break the workload into multiple threads and then place those threads onto a variable number of cores. The cores that are not assigned threads are automatically switched into a low-power sleep state. For example, four threads may be allocated to four cores, such that each core has one thread, and four threads can also be allocated to two cores, such that each of the two active cores has two assigned threads and the other two cores are in the sleep state. This technique allows them to introduce the number of active cores in addition to chip-wide DVFS as a control knob for their power management policy, because threads can be migrated between cores during runtime. In contrast, thread reduction, in which the total number of workload threads is changed, cannot be performed once the workload has started. Our work relies on their technique of thread packing in order to change the number of active cores in the system during runtime

Chapter 3: Models and Methods

We envision soft power capping by executing workloads in two phases: one that has lower power consumption than the desired average power cap, and one that has higher power consumption than the cap. We define *mode* as the combination of clock speed and number of active processor cores. We use two different modes in our execution of soft power capped workloads. The mode that has the high power consumption should have a higher throughput than the low power mode in order to gain performance benefits from soft power capping. In order to better understand the advantages offered by soft power capping, we derive the relationships of average power, runtime, and total energy consumption to the fraction of instructions in the workload executed in the high power mode.

If a workload spends a total of t_L seconds in the low power mode consuming an average of P_L watts, and t_H seconds in the high power mode consuming P_H Watts, the average power consumption is as follows:

$$P_{AVG} = \frac{P_L \cdot t_L + P_H \cdot t_H}{t_L + t_H}$$

We can express this in terms of the fraction of instructions executed in the high power mode (henceforth referred to as α) by substituting

$$t_L = \frac{(1-\alpha)\cdot n}{T_L} \quad \text{and} \quad t_H = \frac{\alpha \cdot n}{T_H}$$

where n refers to the total number of instructions in the workload and T_L and T_H refer to the throughputs in the low and high power modes, respectively. We then obtain the average power of the workload by the relation:

$$P_{AVG} = \frac{P_L \cdot \frac{(1-\alpha)\cdot n}{T_L} + P_H \cdot \frac{\alpha \cdot n}{T_H}}{\frac{(1-\alpha)\cdot n}{T_L} + \frac{\alpha \cdot n}{T_H}}$$

This reduces to the following equation for the average power of the workload:

(1) $$P_{AVG} = \frac{P_L \cdot \frac{(1-\alpha)}{T_L} + P_H \cdot \frac{\alpha}{T_H}}{\frac{(1-\alpha)}{T_L} + \frac{\alpha}{T_H}}$$

The runtime of the workload can be constructed from the sum of the runtimes of each phase:

$$t_{total} = t_L + t_H$$

Substituting t_L and t_H *as* functions of α results in the following relation describing the total runtime of the workload:

(2) $$t_{total} = \frac{(1-\alpha)\cdot n}{T_L} + \frac{\alpha \cdot n}{T_H}$$

Finally, we derive the total energy consumption of the workload, where E_L and E_H refer to the energy consumption of the workload in the low and high power phases, respectively:

$$E_{total} = E_L + E_H$$

We know that the energy consumption is the product of power and runtime, so we express E_{total} as the following function of α:

(3) $$E_{total} = P_L \cdot \frac{(1-\alpha)\cdot n}{T_L} + P_H \cdot \frac{\alpha \cdot n}{T_H}$$

One caveat to these relations is that *n*, the total number of instructions in the workload, varies with the mode in which the workload is executed. Figure 5 shows the total number of instructions in the workload *blackscholes*, with respect to both frequency and number of active cores. We observe a small variation in the total number of instructions retired when the number of active cores is changed. This difference (6% in this case) derives from the fact that a thread that is sharing a core with other threads will take longer to relinquish a lock, because it is periodically swapped out for other threads. Other threads attempting to acquire the lock will therefore waste more instructions spinning on the lock. As a result, the total number of instructions executed is smallest when each thread is allocated its own core, because the relinquishing of locks is not inhibited. We do not observe significant variation in the total number of instructions retired as a function of frequency.

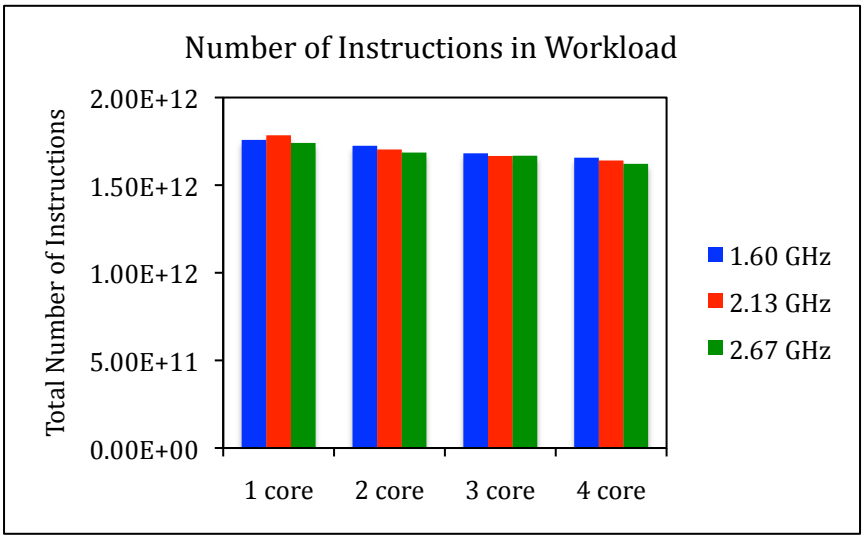

Figure 5: The number of instructions in the benchmark blackscholes, by mode.

We account for these differences in our model by estimating n as a function of α, where n_L and n_H are the number of instructions executed in the workload run in the low power mode and high power mode, respectively:

(4) $$n = (1 - \alpha) \cdot n_1 + \alpha \cdot n_2$$

In our equations (1-4) there is no term accounting for any overhead resulting from switching modes. The choice to exclude overhead in our theoretical framework will be evaluated in Chapter 4, in which the predictions made by our theoretical framework are compared to experimental results.

In order to predict the average power, energy consumption, total runtime, and workload size of a soft power capped workload, we rely on the performance data for prior workloads that remained in a given mode for their entire execution. In other words, we calculate P_L, T_L, n_L and P_H, T_H, n_H, and from the performance of a workload that was run exclusively in the low power mode or high power mode. We can improve the accuracy of our predictions by taking the data from the portions of the workload that will correspond to its soft power capping phase. For example, we can use the first 40% of the power and throughput data from a workload in the 2 core mode of execution if the soft power capped version of the workload will spend the first 40% of its instructions in that same mode.

For $\alpha = 0.6$, assuming a single transition from the low power mode to the high power mode, Figure 6 shows the parameters P_L and P_H for the thirteen PARSEC 2.0 benchmarks, Figure 7 shows T_L and T_H, and Figure 8 shows n_L and n_L. In these figures, the low power mode for all benchmarks uses 2 active cores at 1.60 GHz, and the high power mode for the benchmarks uses 4 active cores at 1.60 GHz.

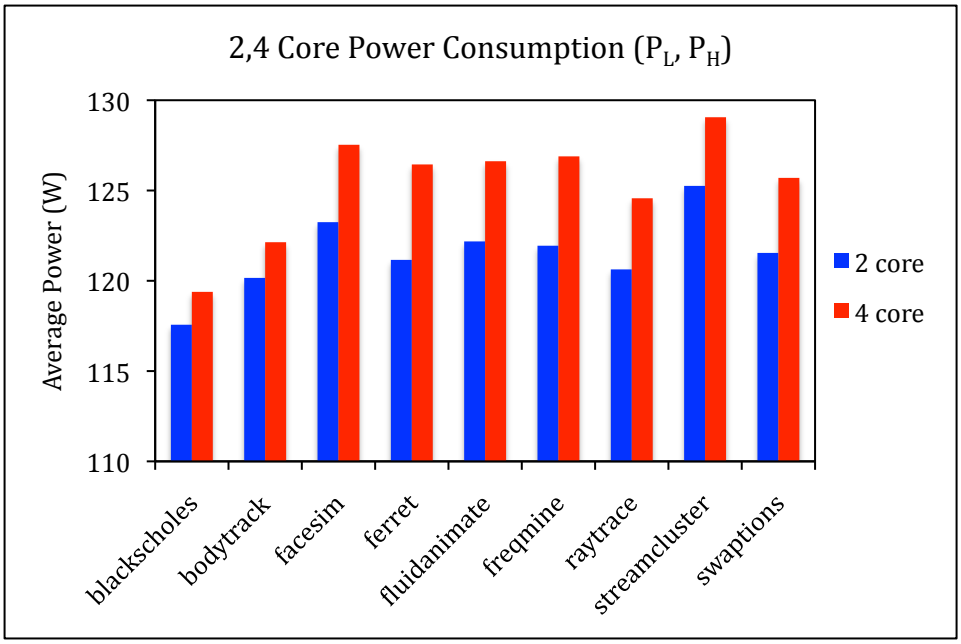

Figure 6: Input values for the power parameters P_L and P_H to equations (1) and (3), taken from experimental data the executing on 2 cores at 1.60 GHz, and on 4 cores at 1.60 GHz. In this example, α is 0.6, so the power values for the 2 core and 4 core mode are taken from the first 40% and last 60% of each workload, respectively.

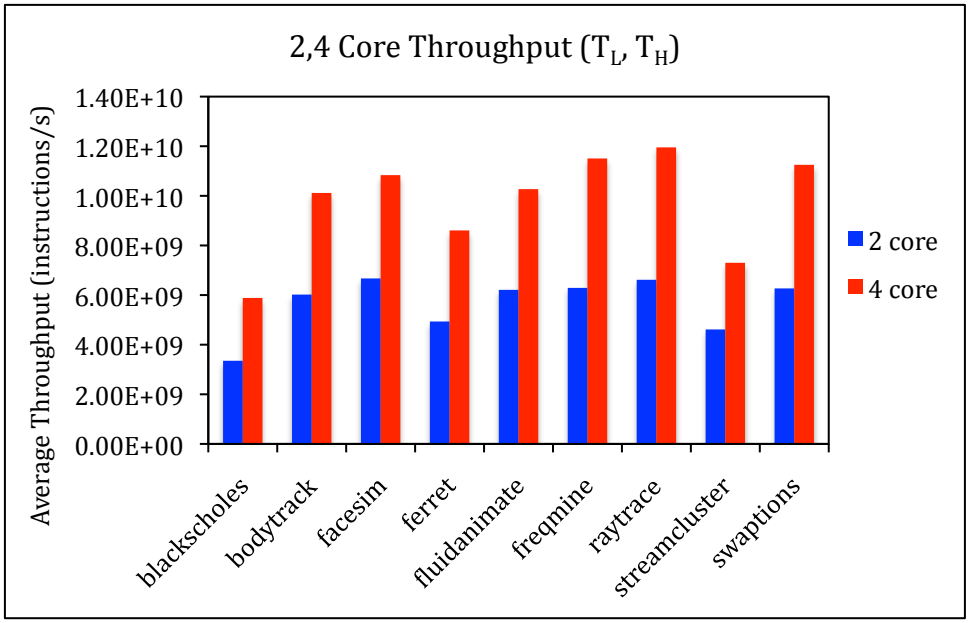

Figure 7: Input values for the throughput parameters T_L and T_H to equations (2) and (3), taken from experimental data the workloads executing on 2 cores at 1.60 GHz and on 4 cores at 1.60 GHz. In this example, α is 0.6, so the power values for the 2 core and 4 core mode are taken from the first 40% and last 60% of each workload, respectively.

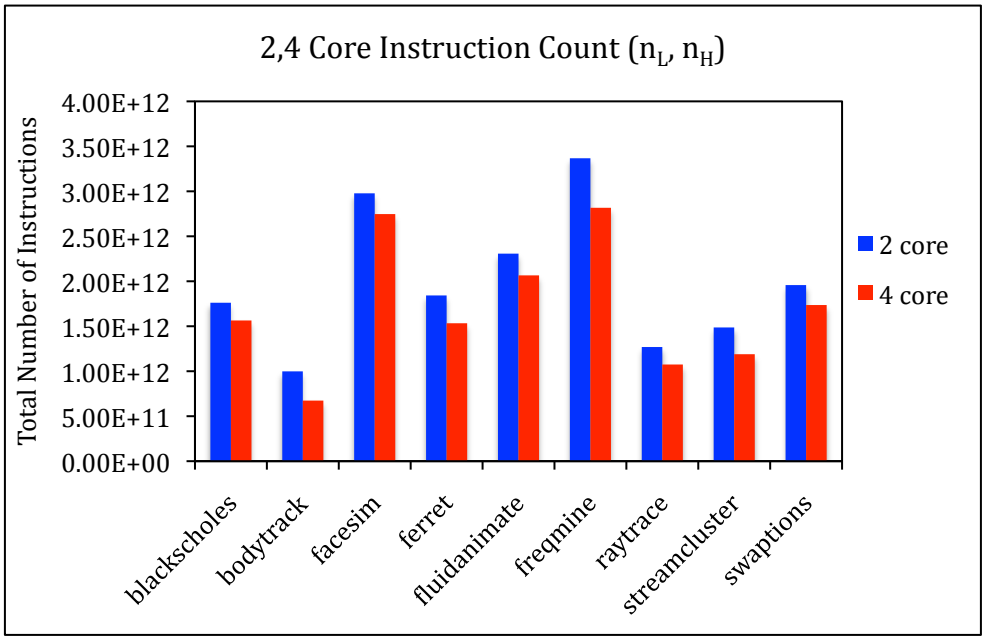

Figure 8: Input values for the workload size parameters n_L and n_H to equations (4), taken from experimental data the workloads executing on 2 cores at 1.60 GHz and on 4 cores at 1.60 GHz. In this example, α is 0.6, so the power values for the 2 core and 4 core mode are taken from the first 40% and last 60% of each workload, respectively.

It should be noted that these equations apply to any form of soft power capping utilizing two modes of operation. Frequency and the number of active cores are both control knobs that can be manipulated by soft power capping, and equations (1-4) are extendable to any two desired configurations given prior data on power consumption, throughput, and number of instructions in each configuration. However, as mentioned in the earlier chapters, our work is concerned with soft power capping by changing the number of active cores. As a result that is the context in which we utilize and validate the model.

Figure 9 shows the theoretical power consumption of our experimental system using soft power capping while running the PARSEC benchmark *ferret* at 1.60 GHz and switching between 2 active cores and 4 active cores. The power

consumption varies with α, the fraction of the workload (by number of instructions) spent in the 4 core state. Figure 10 shows the theoretical runtime of the system with respect to α. Figure 11 models the theoretical energy consumption, and Figure 12 depicts the predicted total number of instructions in the soft power capped workload. We choose to switch the workload between 2 and 4 cores for two reasons, the first being symmetry. In both modes, each active core has the same number of threads (2 and 1 per core, respectively), whereas for 3 cores there is an inherent imbalance because one core must have an extra thread. Second, we wanted to compare soft power capping to an existing configuration for the same power budget. In this case, we can compare switching between 2 and 4 cores to a static mode of 3 active cores, because the power-performance tradeoff of 3 active cores for a given workload lies in between those of 2 and 4 active cores.

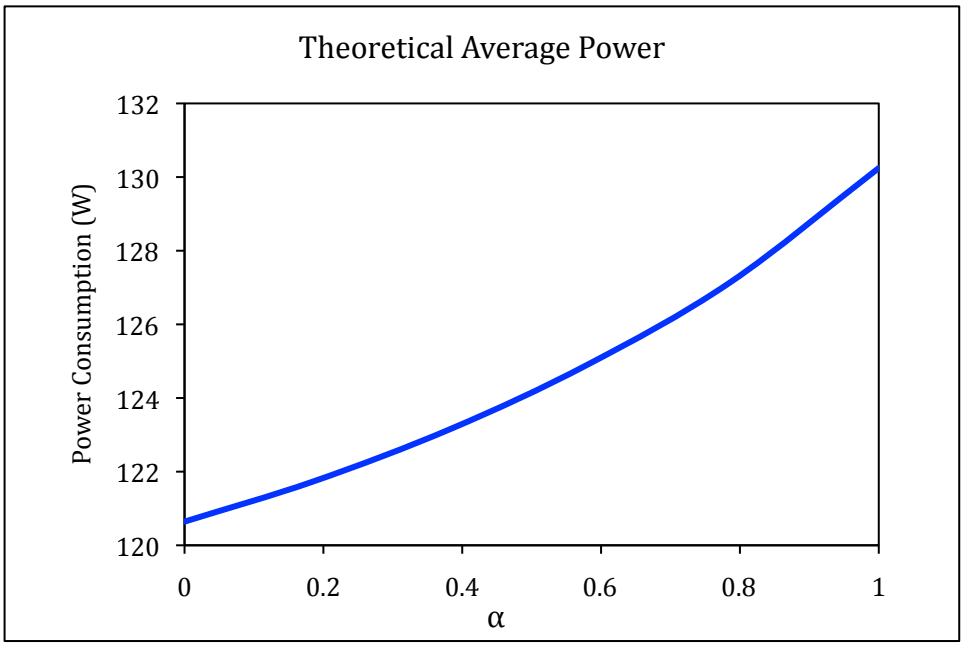

Figure 9: Predicted average power for the workload ferret as a function of α, the fraction of instructions executed in the high power mode

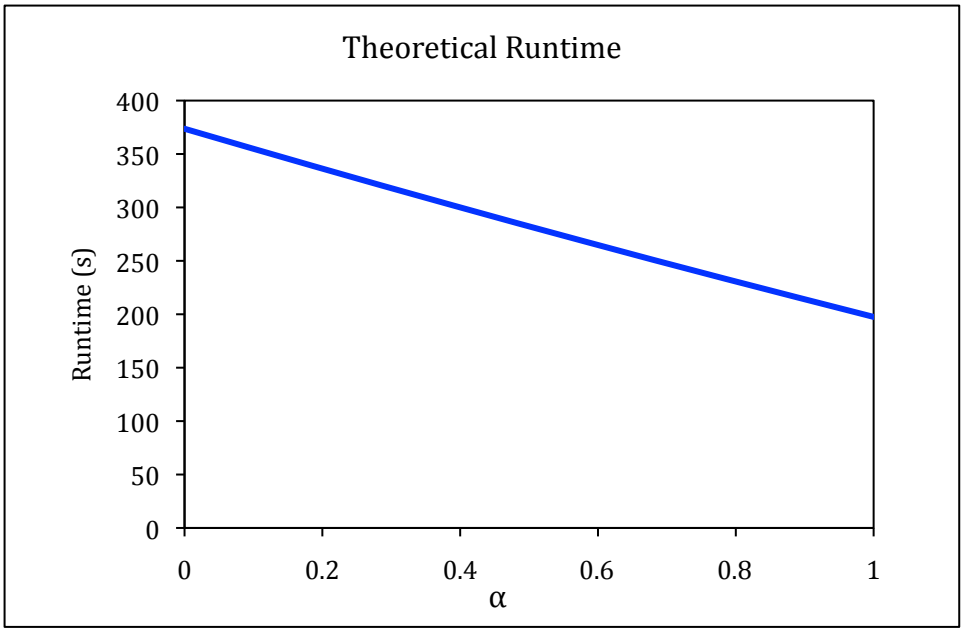

Figure 10: Predicted for the workload ferret as a function of α, the fraction of instructions executed in the high power mode

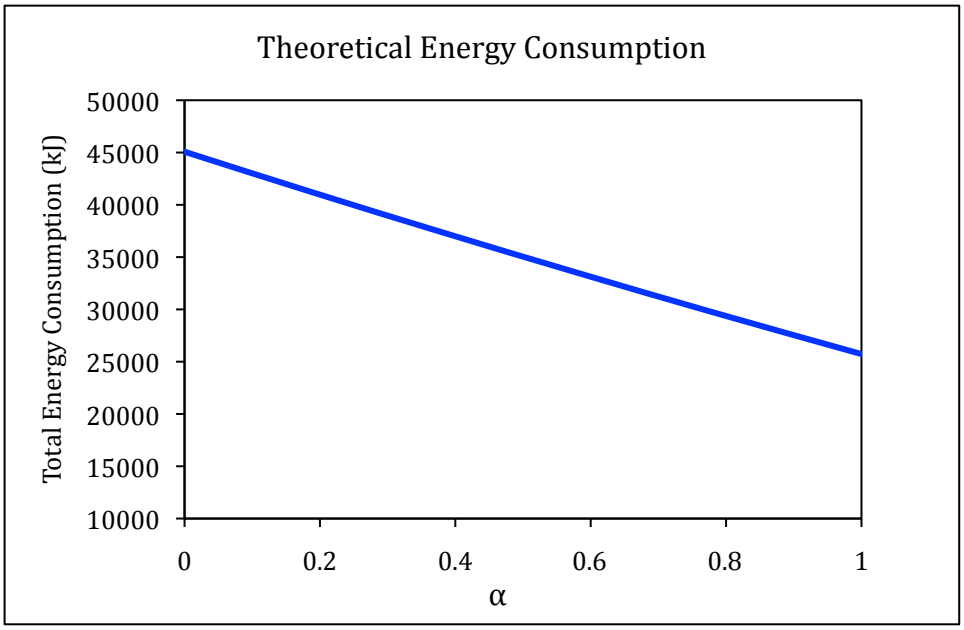

Figure 11: Predicted energy consumption for the workload ferret as a function of α, the fraction of instructions executed in the high power mode

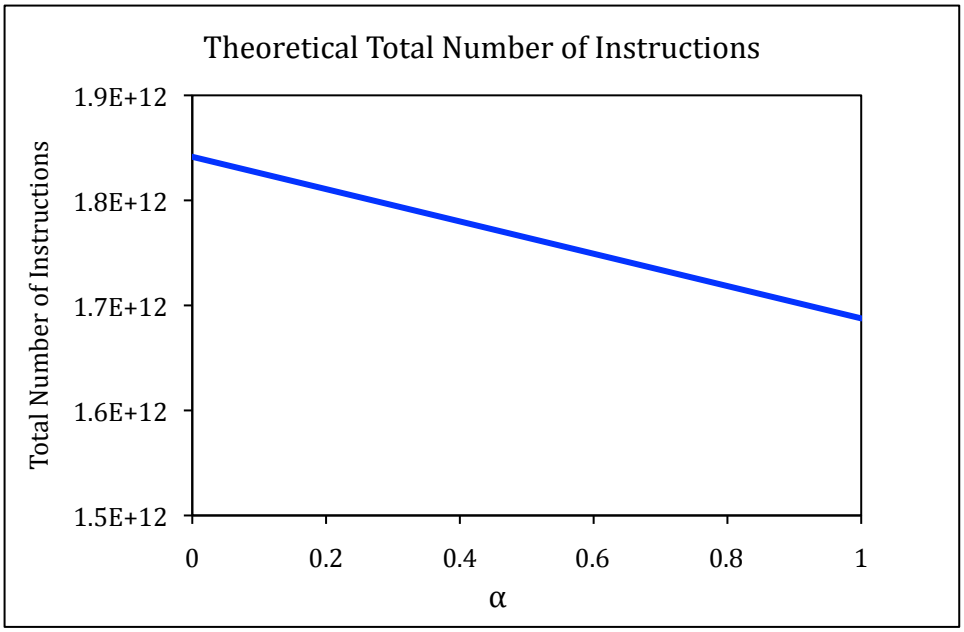

Figure 12: Predicted energy consumption for the workload ferret as a function of α, the fraction of instructions executed in the high power mode

The theoretical benefits of soft power capping are highlighted in Figure 13. Figure 13 shows the power-performance tradeoffs of a system running at 1.60 GHz on 2 cores, 3 cores, 4 cores, and the theoretical power-performance tradeoffs of switching between 2 and 4 cores with α (the fraction of time spent running in the high power mode, 4 cores) varying from 0 to 1. We can observe that there are a finer grained set of power-performance tradeoffs made possible by soft power capping: instead of being limited to discrete settings, there are an infinite number of settings made possible by varying α.

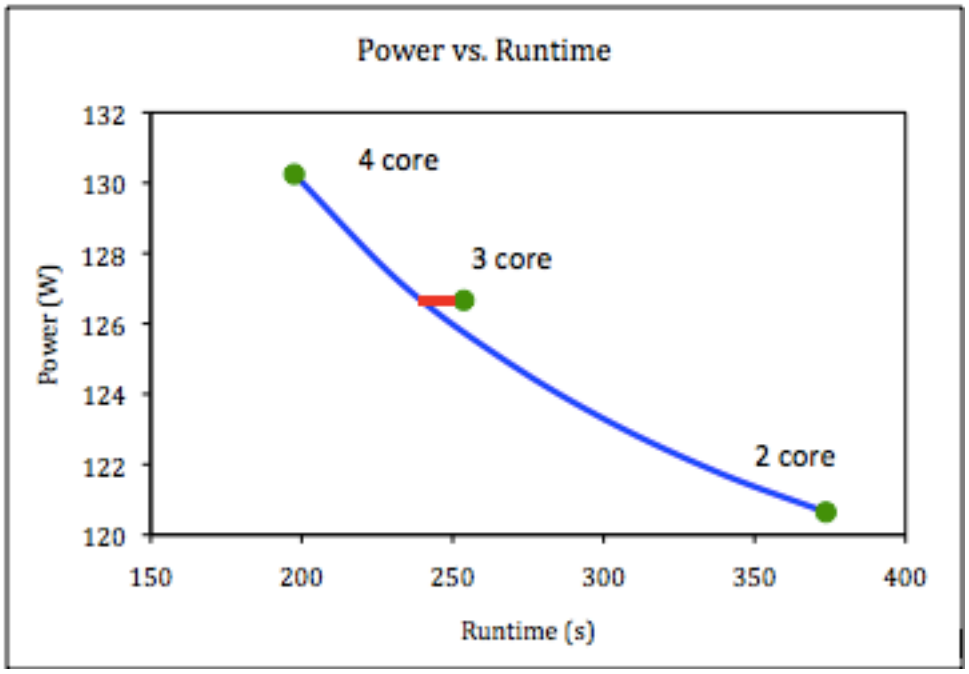

Figure 13: Predicted power vs. predicted runtime of ferret, for α (the fraction of instructions executed in the high power mode) from 0 to 1. The experimental runtime and power consumption of ferret in three static modes (2, 3, and 4 active cores, all at 1.60 GHz) are also shown. The improvement in runtime compared to the 3 core case for the same power budget is marked with the red line.

We can also see that the theoretical performance of the soft power capped configuration, with α set such that the power consumption is equal to the power consumption of 3 cores, is faster and thus has a better performance for the same power budget (the improvement is marked with the red line). As a result, we propose two main applications of soft power capping.

1. Soft power capping allows the server to choose from a continuous, infinite set of power-performance tradeoffs as opposed to the finite, discrete set offered by hard power capping.
2. Soft power capped systems can outperform hard power capped systems for the same power budget consumed by an existing mode.

Both applications improve server performance. The first does so by allowing the server to run exactly at the cap, instead of selecting the best performing setting that falls under the cap. The second application improves performance by achieving a shorter runtime for the same average power consumption of an existing static setting. The authors of [2] have shown the performance advantages provided by the first application of soft power capping by switching the frequency. We build upon their work by focusing on the implementation of soft power capping through switching the number of active cores in the system, and extend the benefits of soft power capping to the second application.

Chapter 4: The Power-Runtime Pareto Frontier and Initial Experimental Results

4.1: The Power-Runtime Pareto Frontier

Each system mode has its power consumption and performance. For example, Figure 14 plots the power consumption and performance for all possible DVFS frequencies and number of core modes. The mode that achieves the shortest runtime for a given power budget is therefore optimal when runtime minimization under a power constraint is the goal. The power-runtime Pareto frontier denotes the set of modes which produce the shortest runtime for a given power range. In other words, for each point in the power-runtime frontier, there does not exist a point which achieves both lower power consumption and lower runtime. These modes which correspond to the points that lie on the frontier *dominate* the other modes because they represent the set of optimal choices for runtime minimization as a function of power. Figure 14 shows the power-runtime Pareto frontier for *ferret* with all possible modes in our 4-core, 9-frequency system. The other workloads in the PARSEC 2.0 benchmark suite have similar properties.

When energy minimization is the desired outcome, selecting configurations which lie on the power-*energy* Pareto frontier is the optimal choice. For each of those configurations, there does not exist another configuration which achieves both lower power and lower energy consumption. Figure 15 shows the power-energy Pareto frontier for *ferret* with all 36 (= 9 frequencies x 4 cores) existing configurations of our system.

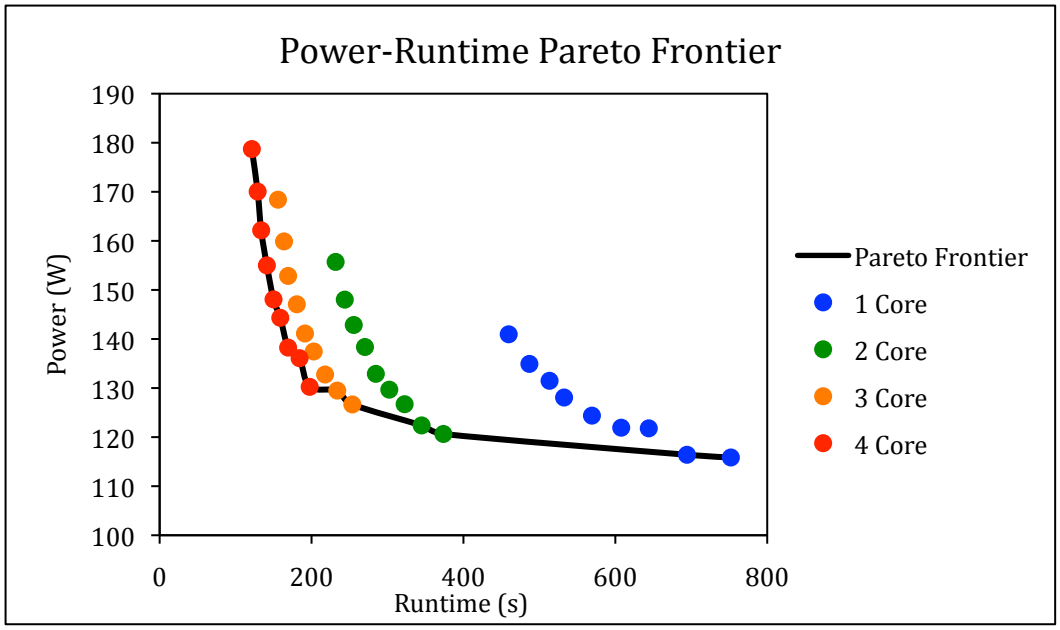

Figure 14: For each possible mode of execution on our system, the experimental power and runtime for the benchmark ferret are plotted. The workload can be run on 1, 2, 3, or 4 active cores at 9 different frequencies (ranging from 1.60 GHz to 2.67 GHz). The power-runtime Pareto frontier, which consists of all configurations that have the shortest runtime for a given power budget, is also marked.

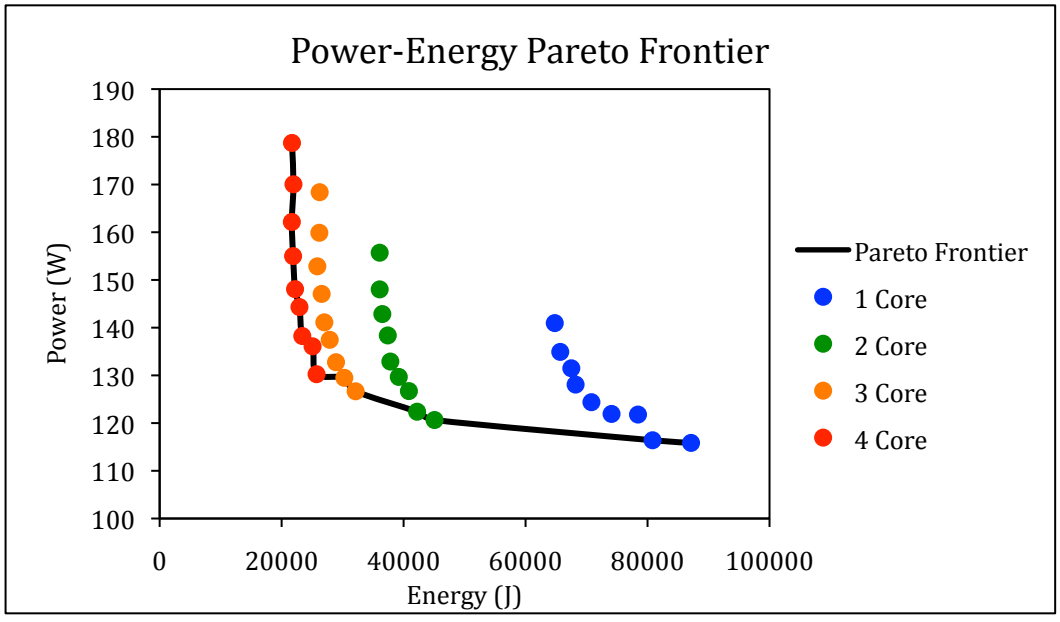

Figure 15: For each possible mode of execution on our system, the experimental power and total energy consumption for the benchmark ferret are plotted. The power-energy Pareto frontier, which consists of all configurations that have the smallest total energy consumption for a given power budget, is also marked.

We observe that for both the power-runtime and power-energy Pareto frontiers, it is always advantageous to run at the highest number of active cores allowed by the power constraint, selecting for the frequency that fits the cap, rather than running at a lower number of active cores and a higher frequency.

Our first application of soft power capping (finer grained power-performance tradeoffs) seeks to increase the number of points along the power runtime and/or power-energy Pareto frontiers. This allows the user to make a better use of the full power budget allocated to the system, leading to better performance. The second application of soft power capping (shorter runtime than an existing static configuration for the same power budget) seeks to provide new points which dominate existing portions of the power-runtime and power-energy frontiers. Therefore soft power capping has the potential to increase the granularity of the set of optimal system configurations in terms of runtime and/or energy, as well as achieve lower runtime and/or energy of existing points on the frontier.

4.2: Initial Experimental Results for Soft Power Capping

We first implemented soft power capping by setting a timer in our control algorithm which switched the system mode from the low power mode to the high power mode after a certain time had elapsed during execution. There are a few benefits to this approach. This setup implements soft power capping through a single transition between the low power and high power mode, so it is a good base case with which we can compare the experimental and theoretical data. It is also very simple. There are also significant disadvantages. The user has to have oracle

knowledge about the specific workload in order to set the switching point to achieve the desired α (the fraction of instructions executed in the high power mode) and average power consumption. Additionally, it is a very coarse grained approach and therefore the periods of violation of the power budget are long in duration. Figure 16 shows the power consumption for the workload ferret, executing a switch from 2 to 4 cores at 1.60 GHz for $\alpha = 0.6$, for our timer based implementation of soft power capping.

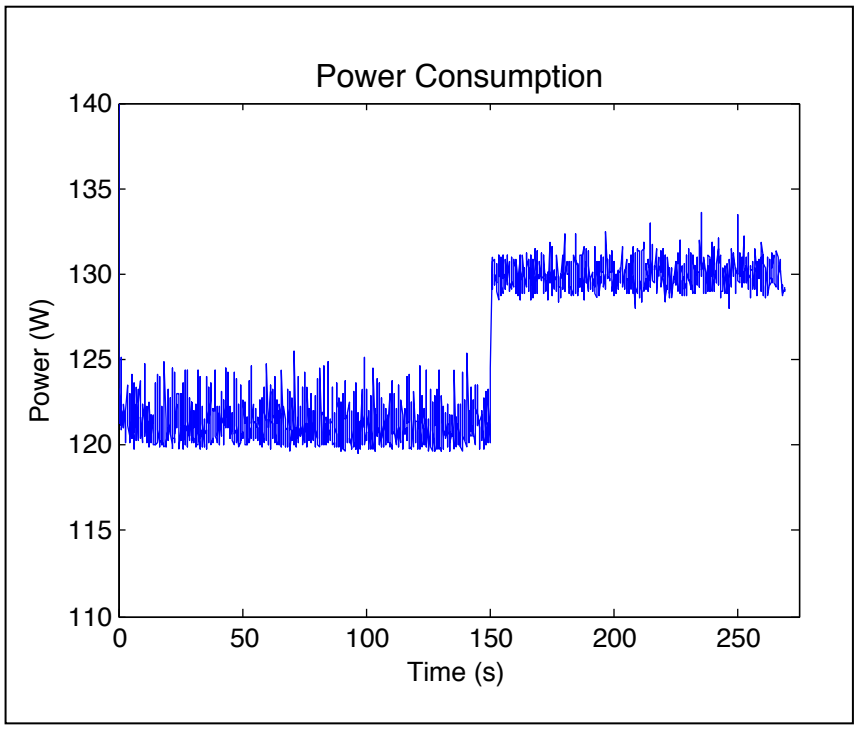

Figure 16: Power consumption of a soft power capped workload (ferret). The initial low power mode uses 2 active cores at 1.60 GHz and the high power mode uses 4 active cores at 1.60 GHz. α (fraction of instructions executed in the high power mode) is 0.6.

It is important to evaluate what happens in the system during the switch to see if there is overhead from the migration of threads, which we did not take into account in our theoretical models. The switch itself could cause a momentary spike

in power. We see in Figure 16 that that is not the case: the power increases to its final range without a temporary increase during or after the switch. Another possibility for overhead could stem from cached data. A thread may be using cached data from the memory in its original core, but when it is migrated to the new core, it suddenly experiences cache misses because the copy of the data has not been brought to the new core's cache yet. In order to investigate this possibility, we plot in Figure 17 the fraction of the number of instructions retired comprised by cache misses for the workload in the soft power capped mode, (ferret), as well as the cache misses for the workload run exclusively in the low and high power modes (2 and 4 active cores, 1.60 GHz).

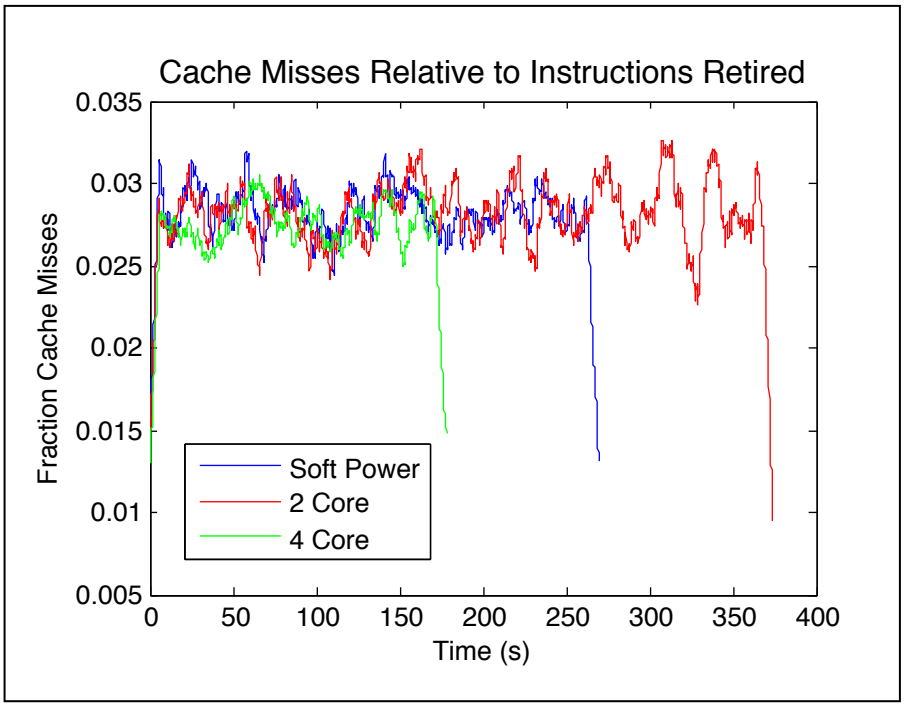

Figure 17: Cache misses as a fraction of total instructions retired for the soft power capped workload, and the workload running exclusively in the low and high power modes (2 and 4 cores at 1.60 GHz).

We observe a large degree of similarity in all three cases and that, for the soft power capped case, its behavior remains constant throughout its duration. In other words, there does not seem to be a spike in cache misses as a result of the switching itself. Therefore there is not detectable cache-related overhead as a result of the migration of core.

Figure 18, Figure 19, Figure 20, and Figure 21 depict the theoretical and experimental average power, runtime, total energy consumption, and workload size respectively, of *ferret* with the aforementioned low and high power modes as a function of α (the fraction of instructions executed in the high power mode). We can see that our experimental data matches very well with our theoretical predictions, further justifying the accuracy of the models without a term dedicated to overhead.

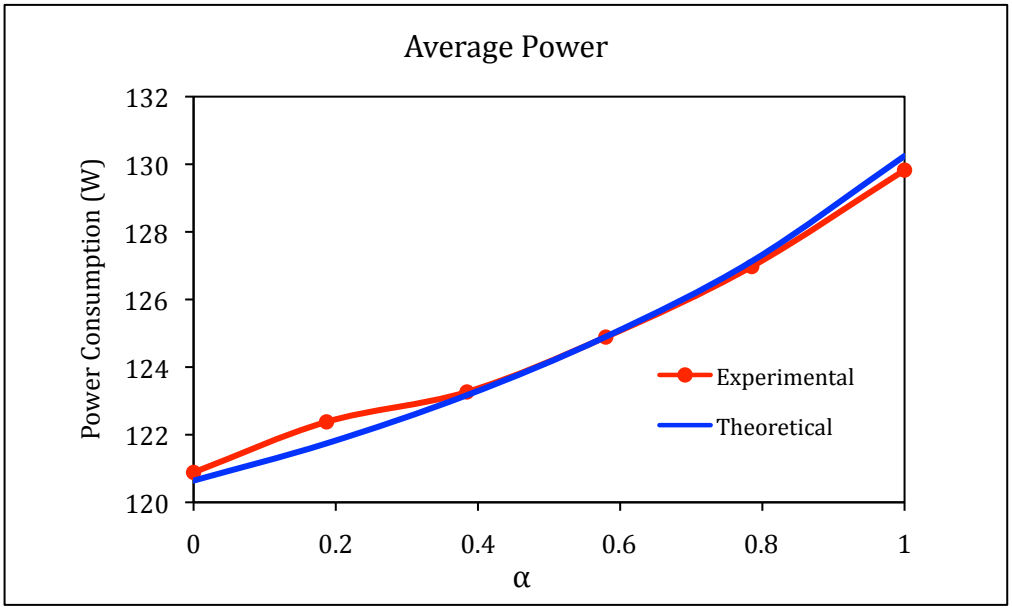

Figure 18: The theoretical and experimental average power values for the workload ferret are shown as a function of α, which is defined as the fraction of instructions in the workload executed in the high power state. The low power state uses 2 active cores at a frequency of 1.60 GHz, and the high power state uses 4 cores, also at 1.60 GHz.

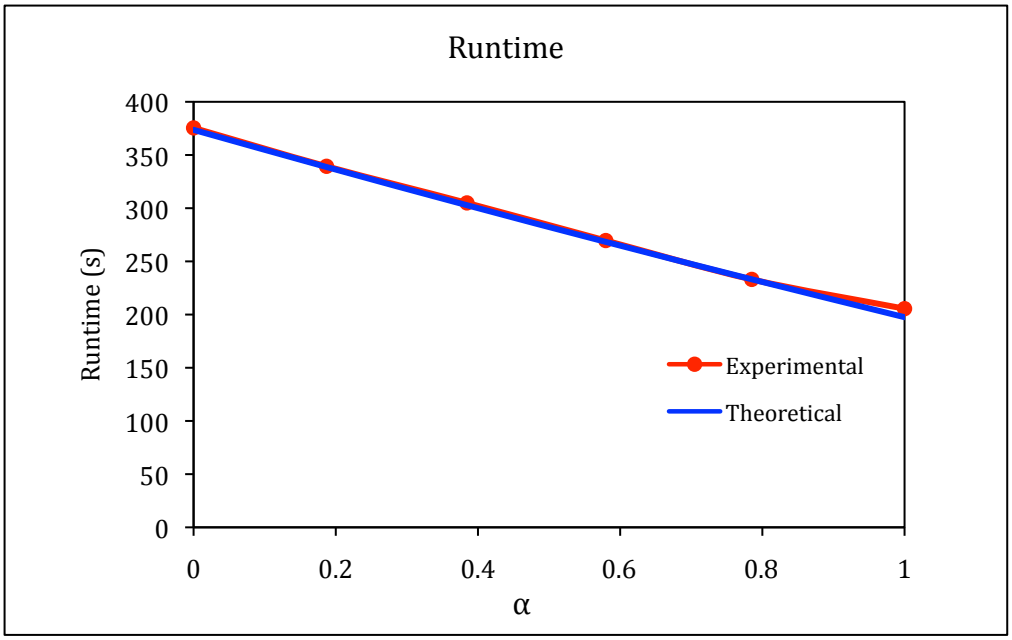

Figure 19: The theoretical and experimental runtime values for the workload ferret are shown as a function of α.

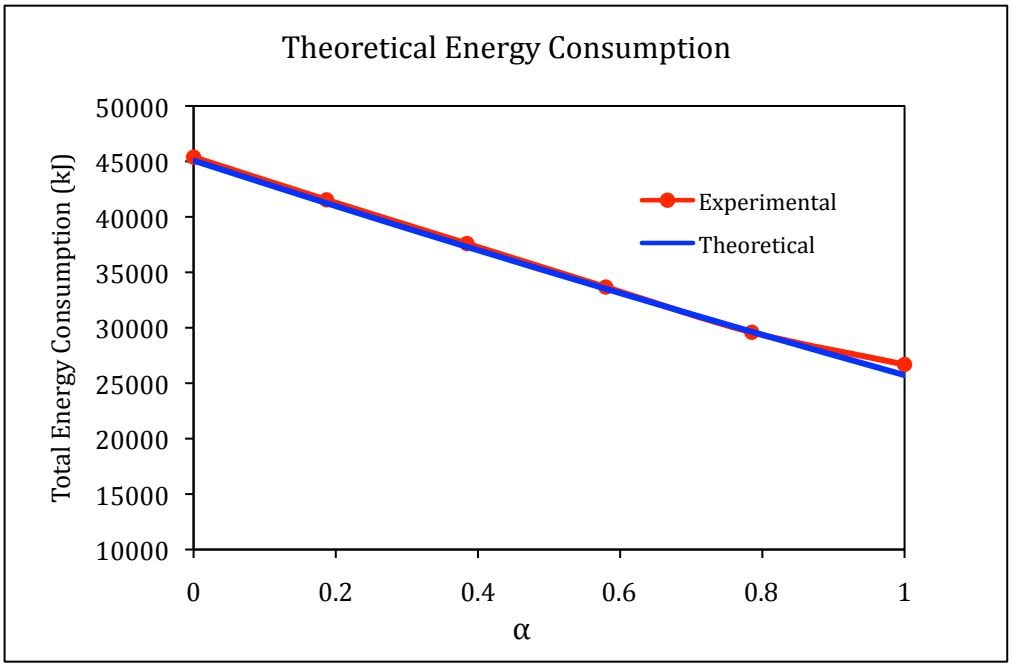

Figure 20: The theoretical and experimental total energy consumption values for the workload ferret are shown as a function of α.

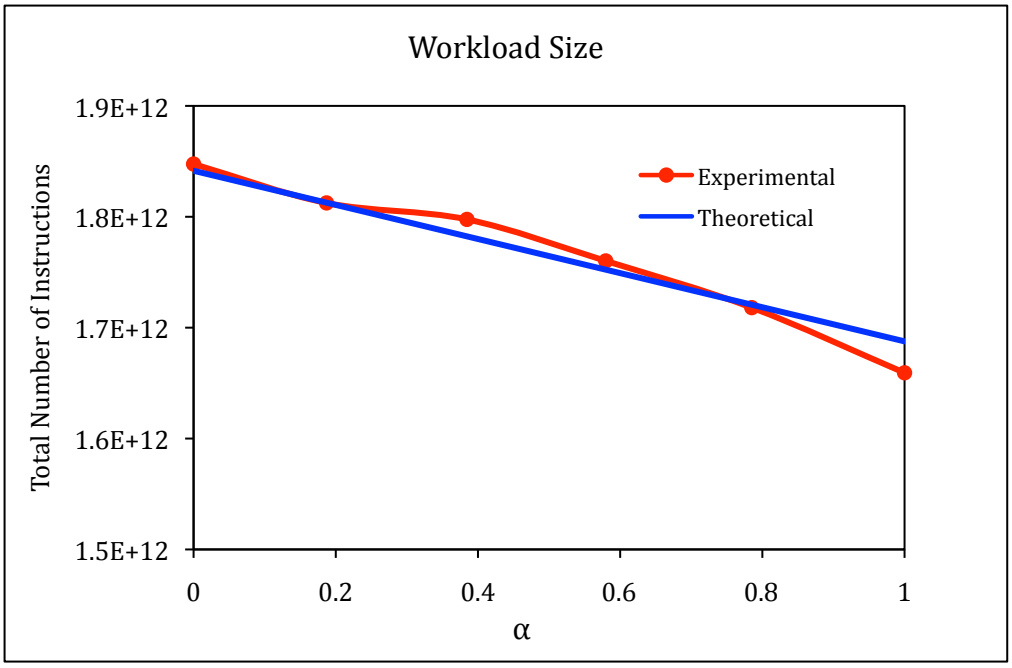

Figure 21: The theoretical and experimental total number of instructions values in the workload ferret are shown as a function of α.

The theoretical models prove to be highly accurate in predicting the behavior of soft power capping as a function of α for this implementation. Additionally, we verify the two benefits of soft power capping described in Chapter 3 with our experimental data in Figure 22, which shows the power-performance tradeoffs provided by soft power capping by switching between 2 and 4 cores at 1.60 GHz. Our experimental data confirms that the set of power-performance tradeoffs become more fine-grained with soft power capping, and we can see that given the correct value for α, alternating between 2 and 4 core is faster than using 3 cores for the entire duration, for the same power budget. In Figure 23 we show the power-performance tradeoffs provided by soft power capping when switching between 2 and 4 cores at 2.67 GHz.

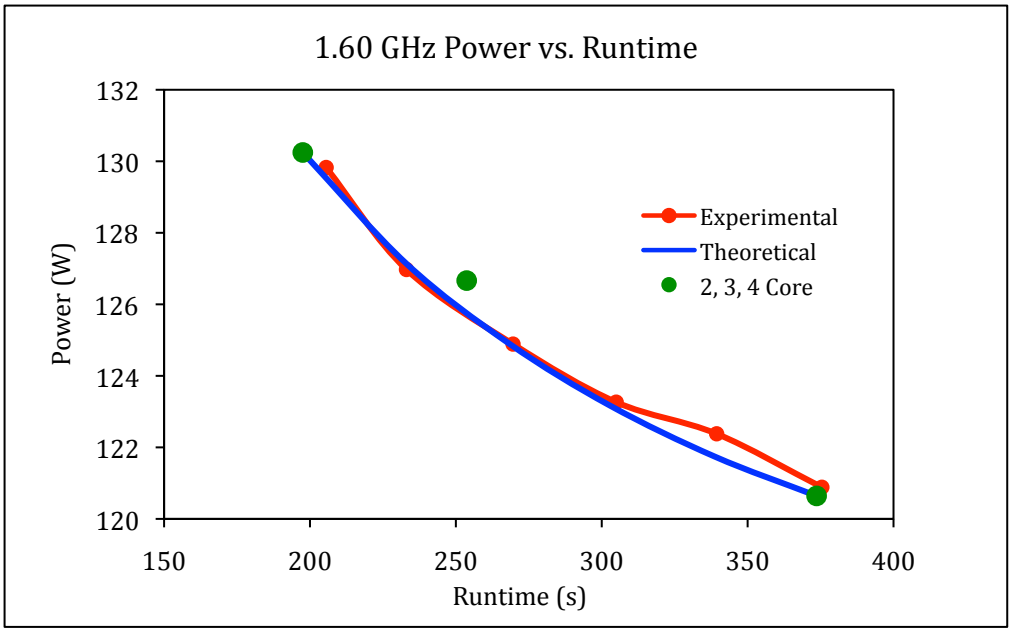

Figure 22: Predicted and experimental power vs. runtime of ferret, for α from 0 to 1. The experimental runtime and power consumption of ferret in three static modes (2, 3, and 4 active cores, all at 1.60 GHz) are also shown.

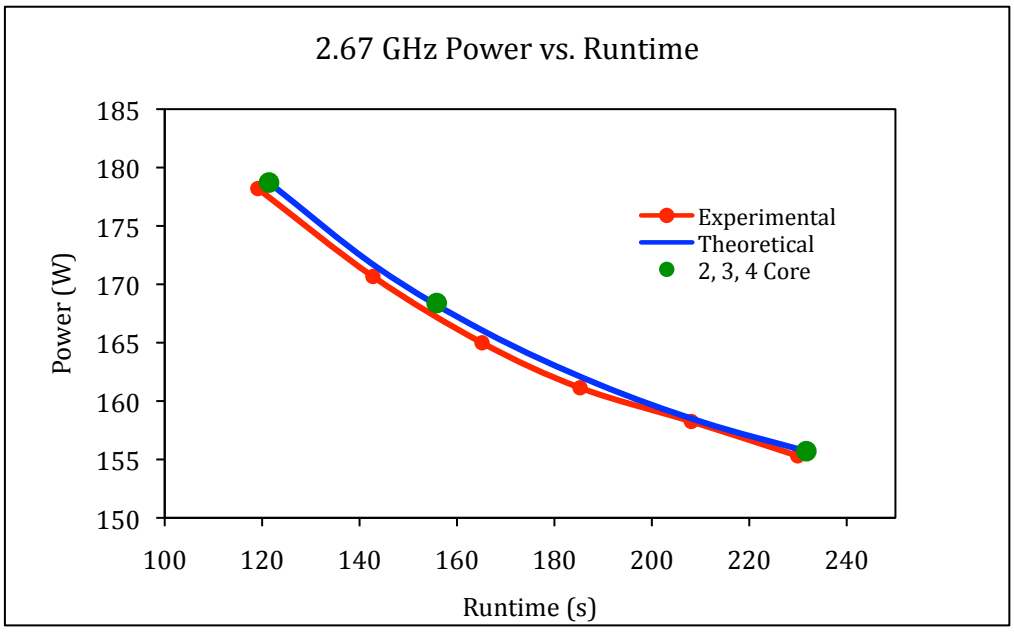

Figure 23: Predicted and experimental power vs. runtime of ferret at 2.67 GHz, for α from 0 to 1. The experimental runtime and power consumption of ferret in three static modes (2, 3, and 4 active cores, all at 2.67 GHz) are also shown.

We show an updated version of the power-runtime Pareto frontier for *ferret* in Figure 24. We add our experimental soft power capping data for switching between 2 and 4 cores at both frequencies (1.60 GHz and 2.67 GHz). The 1.60 GHz 2-4 core switching dominates existing points but the 2.67 GHz 2-4 core switching implementation is dominated by existing configurations. This result is in line with our original observation about the power-runtime Pareto frontier: that it is advantageous to select as many active cores as possible given the power constraint, and after that select the highest frequency. Switching between 2 and 4 cores achieves an "average" active core number of less than 4, so for a high frequency that configuration is not advantageous.

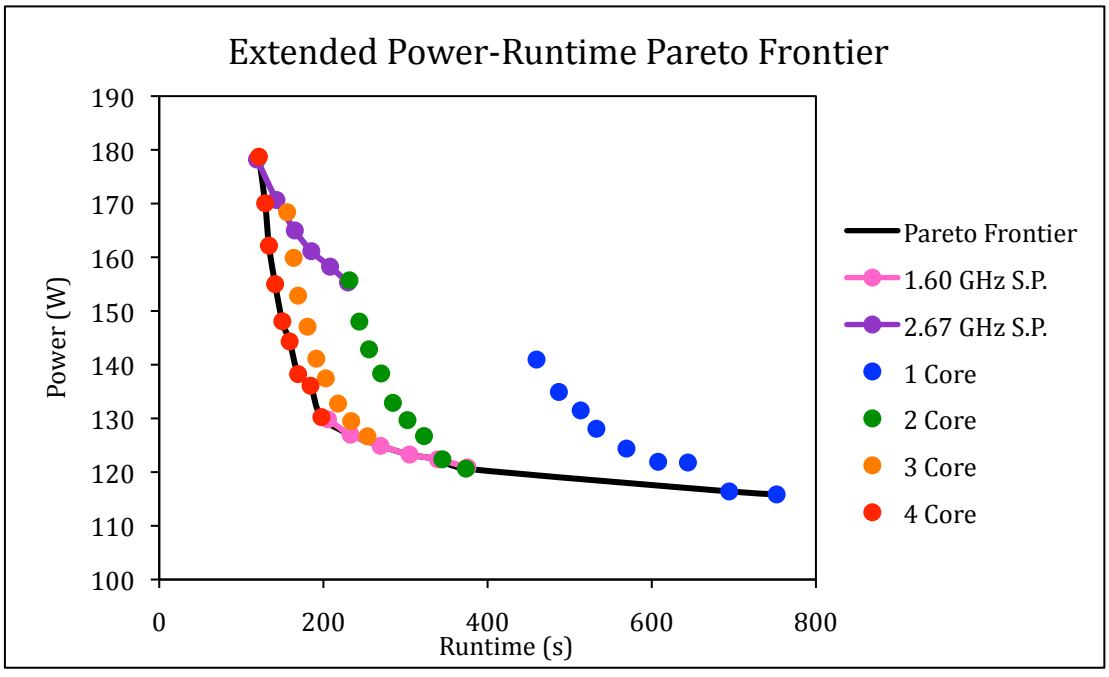

Figure 24: The power-runtime Pareto frontier for ferret, updated to include the soft power capping results at 1.60 GHz and 2.67 GHz (switching between 2 and 4 cores)

However, for power budgets under 4 cores at 1.60 GHz, switching between 2 and 4 cores at 1.60 GHz is advantageous, because there are only a few discrete settings made available to us by hard power capping in this power range. Let's say we have a power cap of 125 W. With our original set of modes available to use by hard power capping, we are just under the 126.7 W needed to run at 3 cores at 1.60 GHz. The closest setting that meets our budget is running at 2 cores at 1.73 GHz, which consumes 122.4 W of power and has a total runtime of 345 seconds. However, if we use soft power capping, we can solve for α such that we are using up all of our allotted power to achieve the best performance. We find that the best fit for this case is $\alpha = 0.6$ when we are switching between 2 and 4 cores at 1.60 GHz, which leads to a runtime of 270 seconds. This change results in a 21.7% improvement in runtime for a total power consumption that still fits the budget at 124.9 W. The finer-grained set of power-performance tradeoffs along the power-runtime Pareto frontier offered by soft power capping in cases such as these make it an attractive control knob.

Chapter 5: An Improved Implementation of Soft Power Capping

5.1: Energy Credits

The previous framework for soft power capping described in Chapter 4, in which the control algorithm changed the number of active cores a single time after a certain amount of time has passed, has its disadvantages. First, it requires oracle knowledge of the duration of the workload in each static mode (2 core and 4 core, in our previous examples) to select the correct switching point in order to achieve the desired average power consumption. Second, it is coarse-grained and leads to instantaneous violations of the average power cap that are long in duration, which could be disadvantageous depending on the motivation for capping the power. Finally, it relies on data from previous experiments rather than feedback from the system to determine its behavior. Because workload behavior can change from run to run, it is unable to ensure that it is actually satisfying the average power budget desired by the user. We observe that a soft power capping that relies on feedback from the system will meet the power budget with more accuracy because it can adapt its behavior based on data from its current power consumption.

The main challenge of implementing soft power capping is determining when the system should switch between the two settings, given that the workload could terminate at any time. The system may spend time in the high power mode, planning to make up for it in the low power mode later. If the workload terminates too soon, however, the average power cap will be violated. Because we want to guarantee that the power budget is satisfied, it is preferable to err on the side of

using less power than the budget than more. It therefore makes sense to build up a certain amount of "credit" in the low power mode, spending it later in the high power mode, to ensure that the average power consumption does not exceed the budget.

We define the amount of credit that has been built up in terms of energy.. We take the following equation to determine our current "energy balance", where $E_{previous}$ is the energy balance calculated from the previous time sample, P_{cap} is the average power budget we are trying to pinpoint, $P_{measured}$ is the current system power, and $\Delta time$ is the change in time since the previous sample. We define *energy credit* to be the threshold for $E_{balance}$ that triggers a transition from the low power mode to the high power mode.

(5) $$E_{balance} = E_{previous} + (P_{Cap} - P_{Measured}) \cdot \Delta time$$

When the system is in the low power mode, $E_{balance}$ grows, because its current power consumption is smaller than the budget. Conversely, when the system is in the high power mode, $E_{balance}$ shrinks, because the current power consumption is higher than the budget. In the low power mode, when $E_{balance}$ reaches the energy credit, the control algorithm then switches the system to be in the high power mode so that it spends the balance that has just been built up. In the high power mode, when $E_{balance}$ reaches 0 J, the control algorithm switches the system back to the low power mode to start accumulating the balance again.

By ensuring that $E_{balance}$ remains positive, we can theoretically guarantee that the average power consumption is always less than or equal to the required power cap, no matter when the workload terminates. Another benefit of this approach is its

simplicity: the energy balance equation remains the same regardless of the current operating mode, energy credit size, and power budget.

5.2: Energy Credit Sizing

For the nine largest benchmarks in the PARSEC 2.0 suite (*blackcsholes, bodytrack, facesim, ferret, fluidanimate, freqmine, raytrace, streamcluster, and swaptions*) we tested the performance of five different energy credit sizes: 10 J, 20 J, 30 J, 40 J, and 50 J. We tested these energy credits on three different iterations of soft power capping: switching between 1 and 4 active cores at 1.60 GHz for the same budget allocated to 2 active cores at 1.60 GHz, switching between 1 and 4 active cores at 1.60 GHz for the same budget allocated to 3 active cores at the same frequency, and switching between 2 and 4 active cores at 1.60 GHz for the same budget allocated to 3 active cores at that frequency. In doing so we are evaluating the second application of soft power capping (better performance than existing modes available by hard power capping). We chose switching between 1 and 4 and 2 and 4 cores because there is symmetry in thread allocation at 1, 2 and 4 cores. We use 1.60 GHz because as we learned in Chapter 4, alternating the number of active cores is only beneficial when there is not enough power to run at 4 cores.

Figures 25 and 26 show soft power capping applied to the same workload for different energy credit sizes, power budgets, and types of mode switching. Figure 25 shows the power consumption of *blackscholes* when given an energy credit of 10 J, switching between 1 and 4 active cores, for the power budget of 2 static cores, and

Figure 26 shows the power consumption given an energy credit of 50 J, switching between 2 and 4 active cores, for the power budget of 3 active cores.

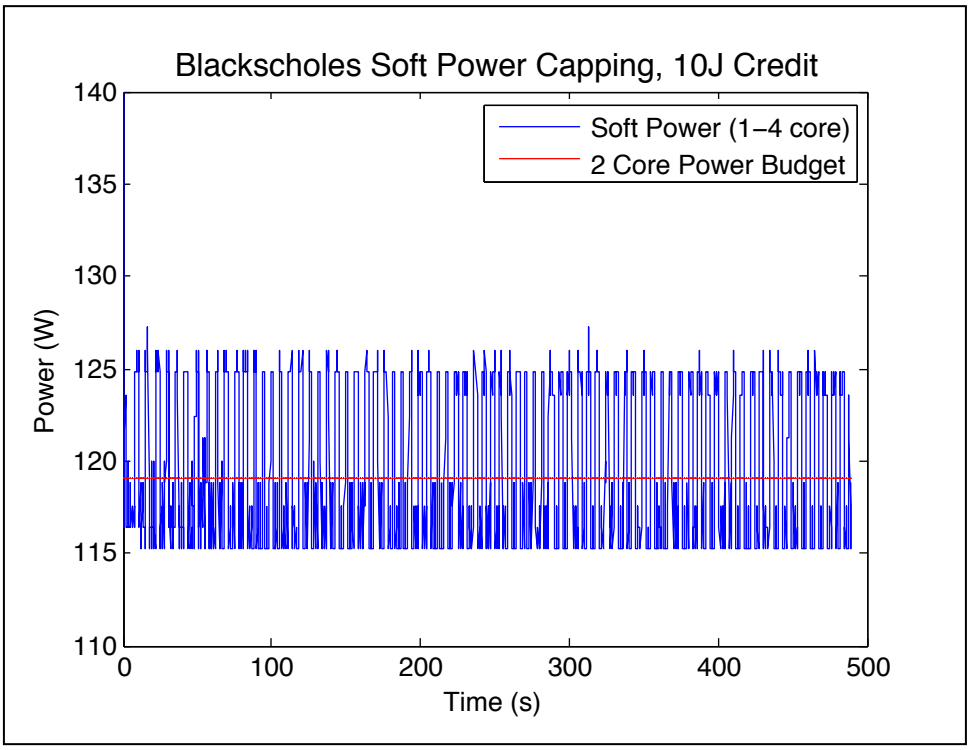

Figure 25: The power consumption of blackscholes using the energy credit implementation of soft power capping. The low power mode uses 1 active core at 1.60 GHz and the high power mode uses 4 active cores at 1.60 GHz. The energy credit size is 10 J and the power budget is the power budget consumed by 2 cores at 1.60 GHz.

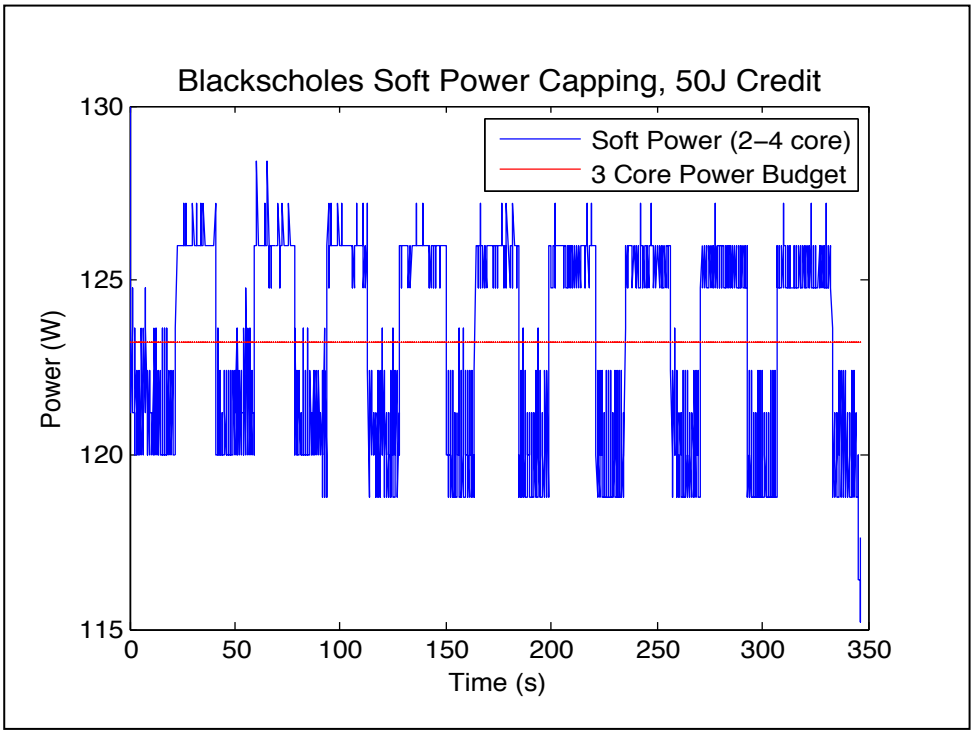

Figure 26: The power consumption of blackscholes using the energy credit implementation of soft power capping. The low power mode uses 2 active core at 1.60 GHz and the high power mode uses 4 active cores at 1.60 GHz. The energy credit size is 50 J and the power budget is the power budget consumed by 3 cores at 1.60 GHz.

Figure 27 gives the runtime speed up achieved by soft power capping for the nine workloads with respect to the energy credit size for the 1 – 4 core switching given the power budget of 2 cores. Table 1 shows the difference in power consumption between the soft power capped workloads and the static workloads. Figure 28 and Table 2 show the runtime speedup and difference in power consumption for the 1 – 4 core switching implementation given the power budget of 3 cores, and Figure 29 and Table 3 show the runtime speedup and difference in power consumption for the 2 – 4 core switching implementation given the budget of 3 cores.

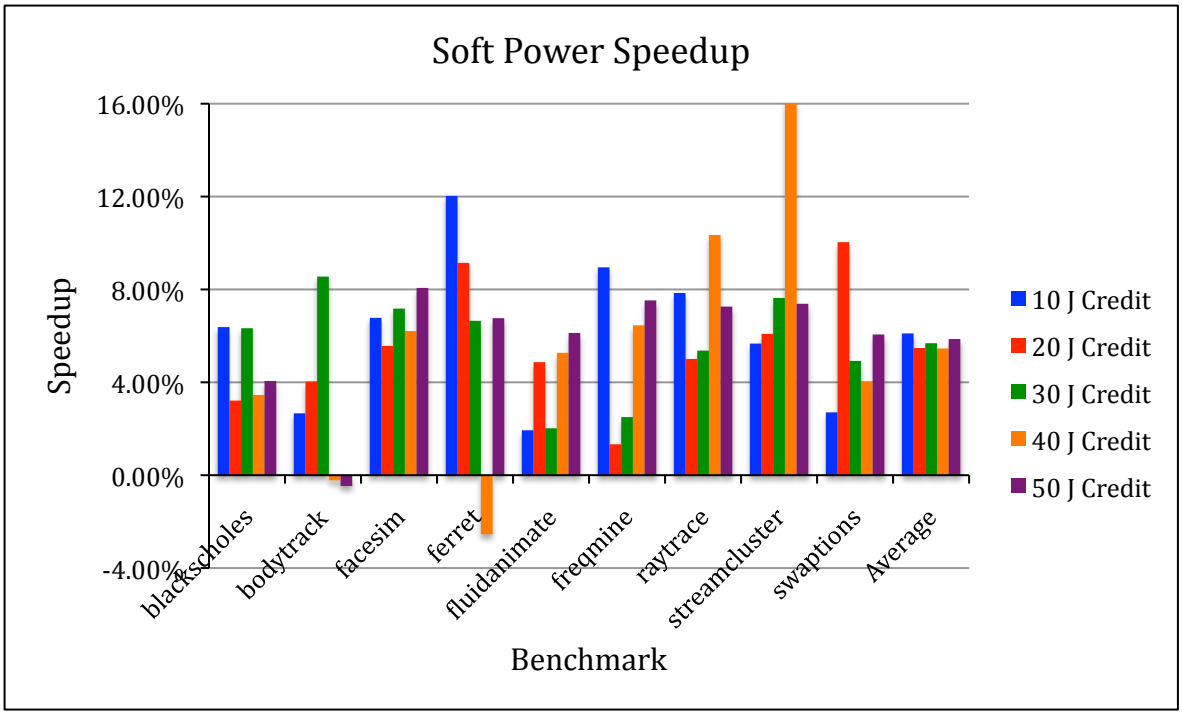

Figure 27: Speedup of soft power capped workloads switching between 1 and 4 active cores at 1.60 GHz (compared to the runtime of the workloads running on 2 cores at 1.60 GHz) with respect to energy credit size.

Table 1: Percent difference in power consumption of soft power capped workloads switching between 1 and 4 active cores at 1.60 GHz (compared to the power consumption of the workloads running on 2 cores at 1.60 GHz) with respect to energy credit size. The average for each credit size is bolded.

	10 J Credit	**20 J Credit**	**30 J Credit**	**40 J Credit**	**50 J Credit**
blackscholes	-0.25%	-0.29%	-0.22%	-0.12%	-0.06%
bodytrack	-0.28%	-0.33%	-0.16%	-0.29%	-0.18%
facesim	-0.20%	-0.28%	-0.39%	-0.36%	-0.46%
ferret	-0.08%	0.05%	-0.05%	-0.02%	0.01%
fluidanimate	-0.36%	-0.29%	-0.18%	-0.06%	-0.10%
freqmine	-0.20%	-0.16%	-0.14%	-0.11%	-0.09%
raytrace	-0.22%	-0.27%	-0.23%	-0.13%	-0.08%
streamcluster	-0.32%	-0.44%	-0.21%	-0.23%	-0.26%
swaptions	-0.18%	-0.12%	-0.19%	-0.13%	-0.18%
Average	**-0.23%**	**-0.24%**	**-0.20%**	**-0.16%**	**-0.16%**

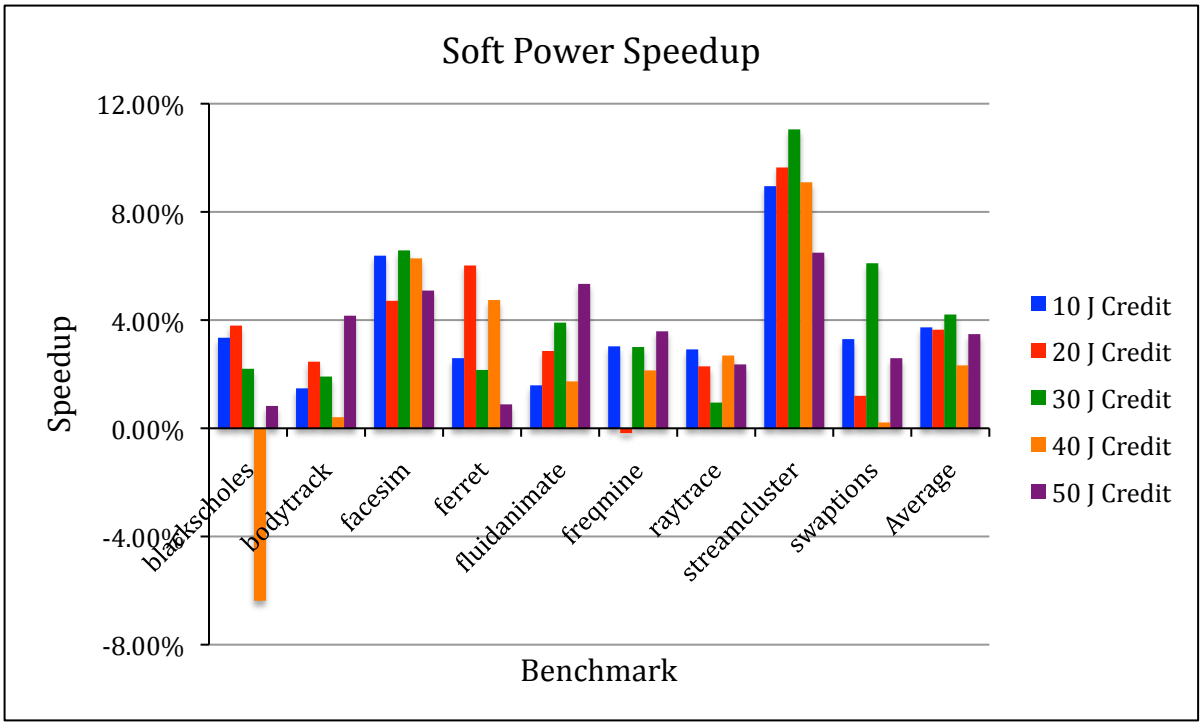

Figure 28: Speedup of soft power capped workloads switching between 1 and 4 active cores at 1.60 GHz (compared to the runtime of the workloads running on 3 cores at 1.60 GHz) with respect to energy credit size.

Table 2: Percent difference in power consumption of soft power capped workloads switching between 1 and 4 active cores at 1.60 GHz (compared to the power consumption of the workloads running on 3 cores at 1.60 GHz) with respect to energy credit size, with the averages for each size in bold.

	10 J Credit	**20 J Credit**	**30 J Credit**	**40 J Credit**	**50 J Credit**
blackscholes	0.01%	0.06%	-0.06%	0.04%	0.09%
bodytrack	-0.26%	-0.07%	-0.07%	-0.11%	-0.08%
facesim	-0.32%	-0.24%	-0.45%	-0.31%	-0.42%
ferret	0.01%	0.06%	0.15%	0.05%	0.30%
fluidanimate	-0.22%	-0.05%	-0.08%	-0.16%	-0.14%
freqmine	-0.04%	0.03%	-0.02%	-0.07%	0.00%
raytrace	-0.04%	-0.10%	0.10%	-0.04%	0.08%
streamcluster	-0.36%	-0.30%	-0.32%	-0.10%	-0.12%
swaptions	0.00%	-0.13%	-0.02%	0.06%	0.02%
Average	**-0.14%**	**-0.08%**	**-0.09%**	**-0.07%**	**-0.03%**

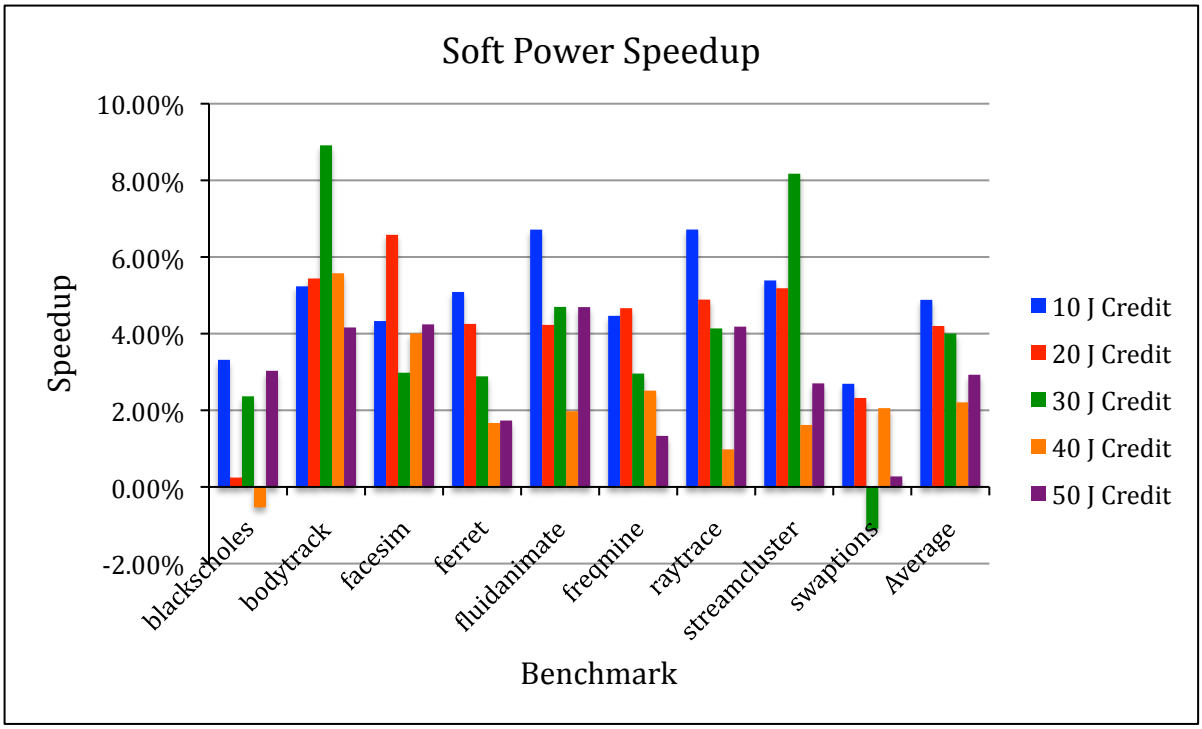

Figure 29: Speedup of soft power capped workloads switching between 2 and 4 active cores at 1.60 GHz (compared to the runtime of the workloads running on 3 cores at 1.60 GHz) with respect to energy credit size.

Table 3: Percent difference in power consumption of soft power capped workloads switching between 2 and 4 active cores at 1.60 GHz (compared to the power consumption of the workloads running on 3 cores at 1.60 GHz) with respect to energy credit size, with the averages in bold.

	10 J Credit	**20 J Credit**	**30 J Credit**	**40 J Credit**	**50 J Credit**
blackscholes	-0.04%	-0.02%	0.13%	0.06%	0.05%
bodytrack	-0.44%	-0.13%	-0.35%	-0.26%	-0.28%
facesim	-0.48%	-0.44%	-0.34%	-0.34%	-0.15%
ferret	-0.07%	-0.02%	0.06%	0.13%	0.12%
fluidanimate	-0.33%	-0.33%	-0.25%	-0.21%	-0.23%
freqmine	-0.08%	-0.09%	-0.06%	-0.01%	-0.10%
raytrace	-0.15%	-0.09%	-0.14%	-0.14%	0.01%
streamcluster	-0.34%	-0.35%	-0.38%	-0.31%	-0.24%
swaptions	-0.12%	-0.10%	0.12%	0.09%	0.04%
Average	**-0.23%**	**-0.17%**	**-0.13%**	**-0.11%**	**-0.09%**

The maximum speedup achieved by soft power capping through the use of energy credits is 16.1% compared to the static core case. The results indicate that there is a definitive performance improvement as a result of using soft power capping across all workloads and all energy credit sizes. Additionally, we can see that the difference between the average power and the cap is negligible, indicating that use of energy credits allows for a workload to precisely meet the power budget.

The choice of energy credit size is an interesting design decision and we can see that different workloads perform better with different credit sizes. One benefit of choosing a small energy credit size, such as 10 J, is that we don't run the risk of building up a large credit, only to have the workload terminate before we spend it. Figure 26 illustrates this situation. However, a larger credit would be the better choice when migration of threads happens frequently enough to create overhead. As discussed in Chapter 4, for a single mode transition we do not notice any overhead from switching, so it seems likely that in general, a smaller credit would perform better due to the problem of building up unspent credit. This is reflected in the average improvement with respect to energy credit in Figures 27, 28, and 29: we can see that the smaller credits tend to perform slightly better. However this relative advantage for smaller credits would have a significantly reduced impact when used on longer-running workloads.

It is also possible that different energy credits have different average throughputs, resulting in the better performance of one credit size over another. For example, consider the case where there is a slight spike in power observed during a mode transition. In this case, a large credit would perform better, because that spike

in power would use up a relatively smaller amount of the credit than would be the case for a small credit. These types of variance are explored in Section 5.3, in which we describe an adaptive technique for energy credit sizing.

5.3: An Adaptive Energy Credit-Sizing Algorithm

Although the results from Section 5.2 are promising, especially given the correct energy credit size, it is not realistic to assume a server would be able to run a workload multiple times to identify the best energy credit size. As a result, we propose an adaptive algorithm, which predicts and selects the best energy credit size from 10, 20, 30, 40, and 50 J based on a search in the beginning of the workload. It iterates through one cycle of each energy credit and computes which one will minimize runtime.

We assume in the following derivation that the n, the total number of instructions, is fixed regardless of energy credit. Although the number of instructions varies somewhat depending on the mode of execution for a workload, as discussed in Chapter 3, we ignore these relatively small variations here. We also assume that the average throughput of each mode is fixed for a given credit size.

We know that the total runtime for a workload is given by the following equation, where t_{ins} is the average time per instruction:

$$t_{total} = n \cdot t_{ins}$$

We can describe t_{ins} with the following relationship, where r_L and r_H are the times spent in the low and high modes for each energy credit cycle, and μ_L and μ_H are the average throughputs for those modes:

$$t_{ins} = \frac{r_L + r_H}{\mu_L \cdot r_L + \mu_H \cdot r_H}$$

Because n is assumed constant, minimizing t_{total} is equivalent to minimizing t_{ins}. As a result we calculate r_L, r_H, μ_L, and μ_H for each energy credit size in our search period, and select the credit that minimizes t_{ins}.

Figure 30 shows the values for the average time per instruction (t_{ins}) with respect to energy credit size computed online by our algorithm for the workload blackscholes, given a 3 core power budget, and low and high power modes of 2 and 4 cores, respectively (all at 1.60 GHz). We can see that t_{ins} is smallest at 40 J, so our algorithm selected 40 J to be the credit size after the search period. Figure 31 shows the power consumption over time for this example, where the search period in the beginning can be identified.

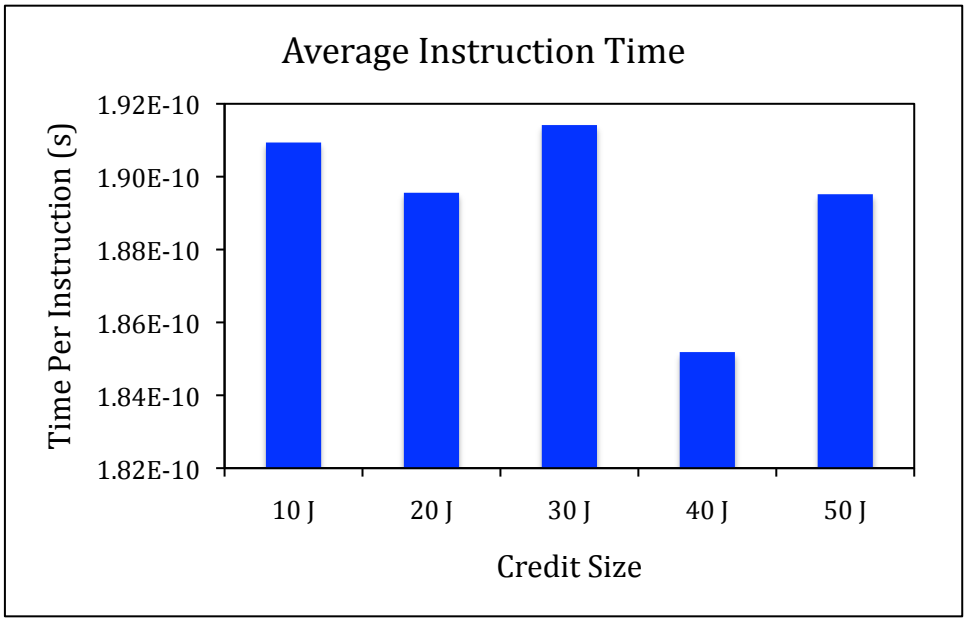

Figure 30: Average time per instruction for each energy credit size during the sweep of the workload blackscholes for a 3 core power budget and low and high power modes of 2 and 4 cores, all at 1.60 GHz.

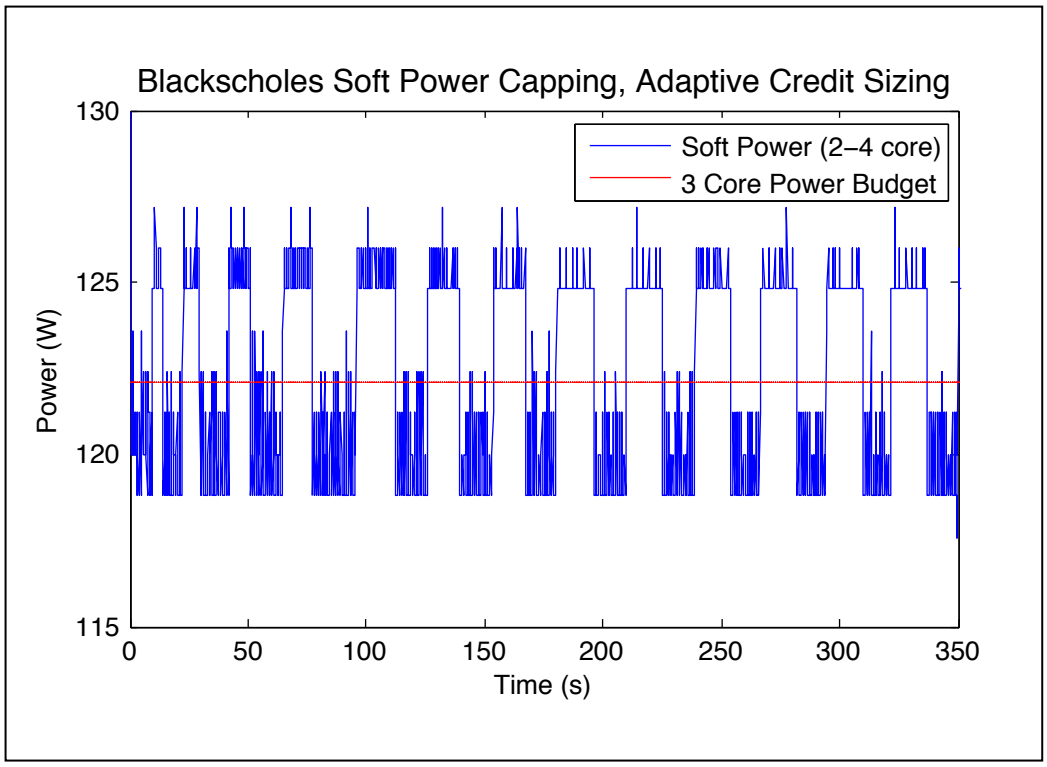

Figure 31: Power consumption of blackscholes executed with the adaptive credit sizing algorithm. During an initial sweep of all of the credits (10, 20, 30, 40, and 50 J), The search period computes the average time per instruction for each energy credit and selects the one producing the smallest time.

The performance improvements of each of the 9 workloads executed with the adaptive energy credit algorithm compared to the static 2, 3 core base cases are detailed in Tables 4, 5, 6, along with the energy credits selected for each workload and the power difference from the base case. The speedups achieved for each credit are also represented graphically in Figures 32, 33, and 34.

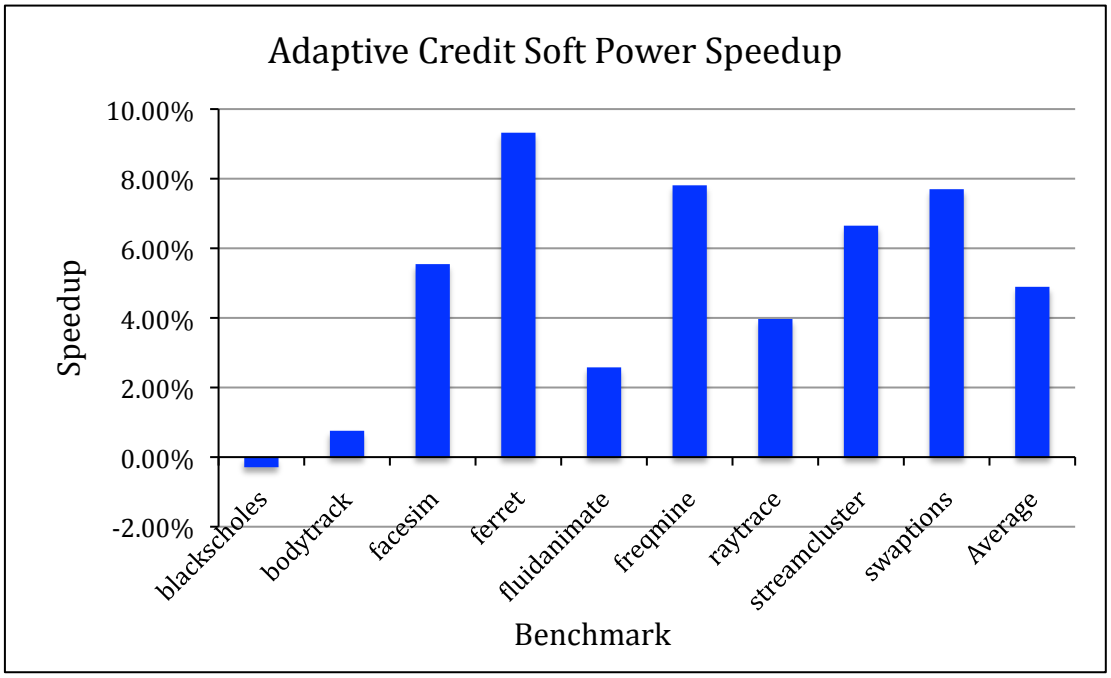

Figure 32: Speedup of soft power capped workloads switching between 1 and 4 active cores at 1.60 GHz (compared to the runtime of the workloads running on 2 cores at 1.60 GHz) using the adaptive credit sizing algorithm.

Table 4: Results from soft power capping with the adaptive energy credit algorithm, for the 1-4 core switching given a budget of 2 active cores.

	Credit Chosen (J)	Speedup	Power Difference
blackscholes	40	-0.29%	-0.21%
bodytrack	50	0.76%	-0.08%
facesim	10	5.55%	-0.34%
ferret	50	9.32%	0.04%
fluidanimate	30	2.58%	-0.26%
freqmine	40	7.81%	-0.11%
raytrace	20	3.97%	-0.07%
streamcluster	30	6.65%	-0.22%
swaptions	40	7.70%	-0.05%
Average	**34.44**	**4.89%**	**-0.14%**

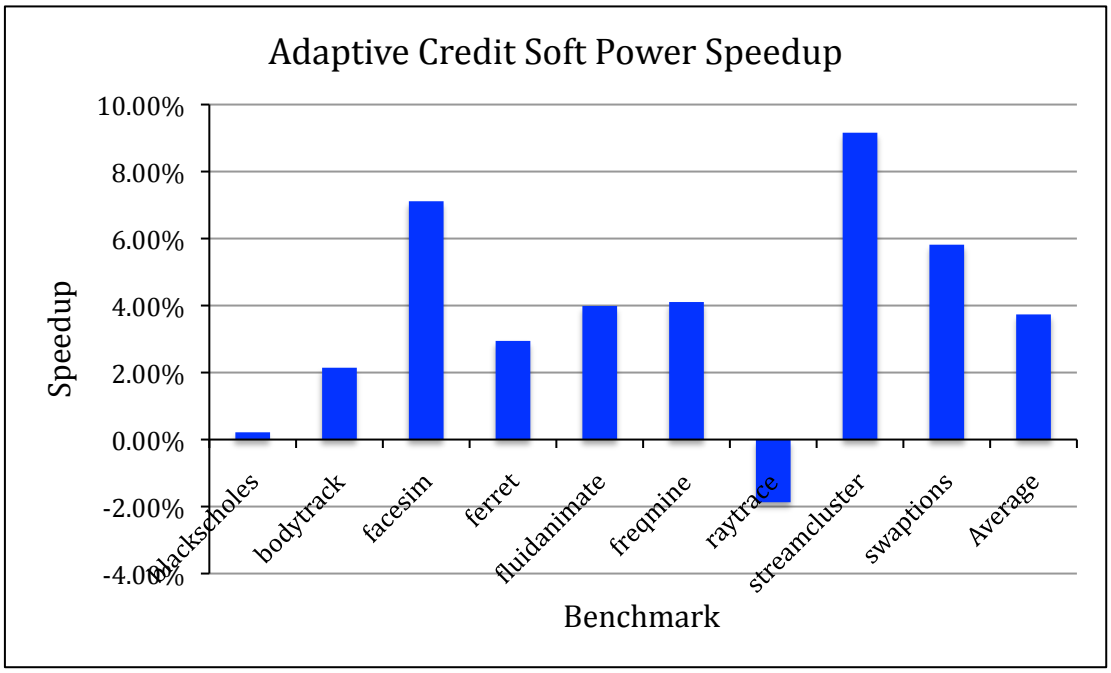

Figure 33: Speedup of soft power capped workloads switching between 1 and 4 active cores at 1.60 GHz (compared to the runtime of the workloads running on 3 cores at 1.60 GHz) using the adaptive credit sizing algorithm.

Table 5: Results from soft power capping with the adaptive energy credit algorithm, for the 1-4 core switching given a budget of 3 active cores.

	Credit Chosen (J)	Speedup	Power Difference
blackscholes	50	0.22%	0.12%
bodytrack	30	2.14%	-0.06%
facesim	20	7.11%	-0.38%
ferret	40	2.94%	0.20%
fluidanimate	40	3.99%	-0.04%
freqmine	50	4.11%	-0.05%
raytrace	10	-1.87%	-0.06%
streamcluster	40	9.16%	-0.26%
swaptions	50	5.82%	-0.04%
Average	**36.67**	**3.74%**	**-0.06%**

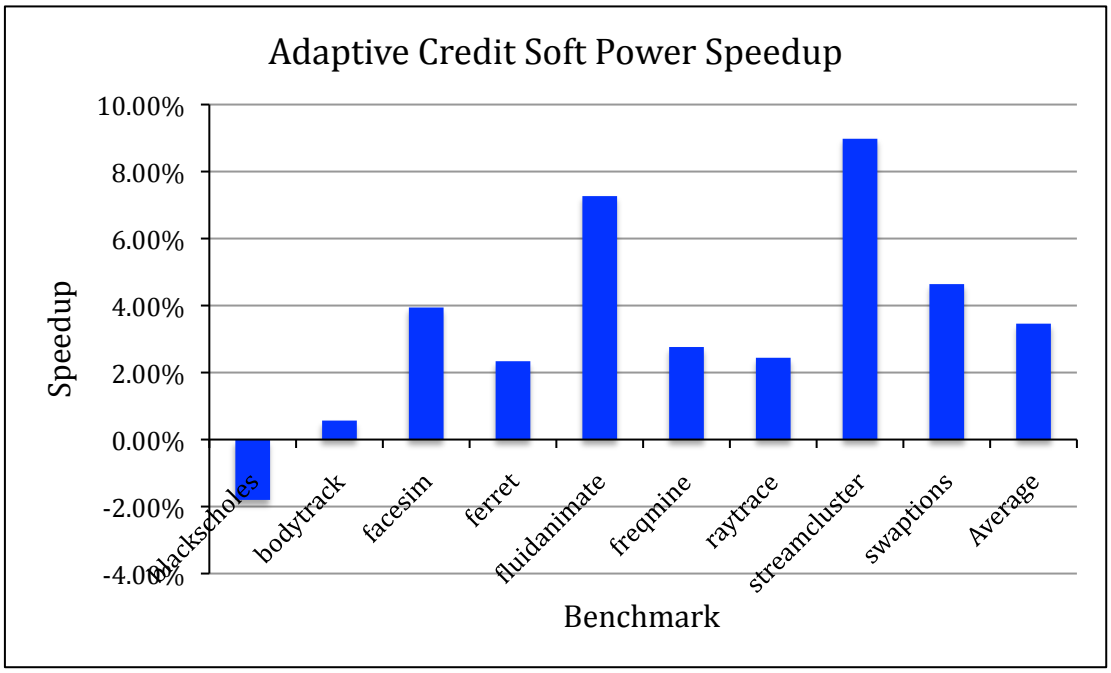

Figure 34: Speedup of soft power capped workloads switching between 2 and 4 active cores at 1.60 GHz (compared to the runtime of the workloads running on 3 cores at 1.60 GHz) using the adaptive credit sizing algorithm.

Table 6: Results from soft power capping with the adaptive energy credit algorithm, for the 2-4 core switching given a budget of 3 active cores.

	Credit Chosen (J)	Speedup	Power Difference
blackscholes	40	-1.80%	0.07%
bodytrack	*	0.57%	-0.23%
facesim	30	3.94%	-0.31%
ferret	40	2.34%	0.02%
fluidanimate	30	7.27%	-0.30%
freqmine	50	2.76%	0.02%
raytrace	20	2.44%	-0.12%
streamcluster	50	8.98%	-0.25%
swaptions	50	4.64%	-0.08%
Average	**38.75**	**3.46%**	**-0.13%**

**This workload terminated before it completed the sweep of all 5 energy credits.*

The adaptive credit-sizing algorithm provides a maximum performance improvement of 9.32%, with the average performance improvements ranging from 3.46% to 4.89%. These gains are comparable to the average gains achieved by the static energy credit allocation, which range between 2.21% and 6.11%. We can see that the average credit size chosen by our algorithm is in the mid to high 30s. However, the static energy credit allocation strategy has the best performance relative to the base case at 10 J for two of our three soft power capping configurations (1-4 core switching for the 2 core budget, and 2-4 switching for the 3 core budget) at 6.11% and 4.88% respectively. The second best performance for the third configuration (1-4 core switching for the 3 core budget) is also at 10 J, with an average speedup relative to the base case of 3.73%.

These findings suggest that there is an advantage of using a smaller credit size that our algorithm is not taking into account. Our algorithm optimizes the time per instruction, however, it does not take into account the possibility described earlier, in which a large credit is built up but never spent due to termination of the workload. Given a linear accumulation and spending of the energy balance, we can expect to have a balance of half of our energy credit size left over after termination. Consequently, our expected leftover balance given an unknown termination point during a 10 J credit run of a workload is 5 J, whereas our expected leftover balance during a 50 J credit run of a workload is 25 J. Assuming that the average power consumption in each mode is the same regardless of energy credit, we are therefore "missing out" on 20 J of high power mode execution when we use a 50 J credit compared to the 10 J credit. An interesting future direction for this work could

therefore be biasing the algorithm toward smaller energy credit sizes for relatively short workloads, so that it can be a more accurate predictor of which credit is optimal. For long workloads, the algorithm could include recalibration periods in which the sweep was performed again after some amount of time, and the optimal credit was recalculated and reselected.

Chapter 6: Conclusions

Due to increasing costs of power-related expenses in computing systems, power capping has become a common and effective technique for lowering total operating costs of datacenters. There are powerful which allow the satisfaction of a power budget that is subject to change while optimizing overall performance, such as DVFS and sleep states. However, most of the focus today is on *hard power capping*, in which the instantaneous power consumption of the system must always be under a fixed budget. We propose and formalize the concept of *soft power capping*, in which the instantaneous power consumption of the system may be violated as long as the *average* power consumption remains at or below the budget.

We present a theoretical framework for predicting the performance of a soft power capped workload given two modes of operation (one consuming more power than the budget and one consuming less). In this work, our control knob for soft power capping is the number of active cores in the system, but our theoretical framework is extendable to any other knob, given two modes of execution. We propose two applications of soft power capping. First, it allows the user to choose between an infinite set of power-performance tradeoffs, in comparison with the discrete set made available by hard power capping. Second, soft power capped workloads can outperform their hard power capped counterparts even when their power consumptions are the same.

We implement the proposed soft power capping method and compared our experimental results to the results predicted by our models. The models, when

given empirical data from workloads running in fixed modes of operation, were shown to be highly accurate in predicting the performance of a soft power capped workload.

We improved upon the proposed soft power capping strategy discussed in Chapter 4 using the concept of *energy credit*. For this method of soft power capping, a workload builds up "credit" in a low power mode. Once that credit reaches a certain threshold, the workload is then allowed to spend it in the high power mode. We present the experimental results from using the energy credit based approach to soft power capping for multiple energy credit sizes, where we see a maximum improvement of 16.1% compared to non soft-power capped configurations. We also confirm that the energy credit approach allows us to meet our power budget precisely, with all deviations being smaller than 0.5%.

Finally, we propose an adaptive energy credit-sizing algorithm, which predicts and selects the optimal energy credit size during runtime. This algorithm, used in conjunction with soft power capping, produces average improvements (relative to the hard power capped base case) between 3.46% and 4.89%, with a maximum improvement demonstrated of 9.32%.

The performance improvements made possible by soft power capping make it an attractive control mechanism for computing systems. Our energy credit-based approach to soft power capping is simple and flexible and provides concrete performance improvements, making it an effective power management protocol, especially when used in conjunction with the adaptive energy credit-sizing algorithm.

References

1. C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO*, 2006.

2. Yefu Wang, Kai Ma, and Xiaorui Wang. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. In *Proceedings of the 36th annual international symposium on Computer architecture* (ISCA '09). ACM, New York, NY, USA, 314-324.

3. Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO-44 '11). ACM, New York, NY, USA, 175-185.

4. Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. *SIGPLAN Not.* 46, 3 (March 2011), 199-212.

5. C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, June 2009.

6. D. Filani, J. He, S. Gao, et al. Dynamic Data Center Power Management: Trends, Issues, and Solutions. IntelTechnology Journal, page 59, February 2008.

7. Bing Shi, Yufu Zhang, and Ankur Srivastava. 2010. Dynamic thermal management for single and multicore processors under soft thermal constraints. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design* (ISLPED '10). ACM, New York, NY, USA, 165-170.

8. Ryan Cochran and Sherief Reda. 2010. Consistent runtime thermal prediction and control through workload phase detection. In *Proceedings of the 47th Design Automation Conference* (DAC '10). ACM, New York, NY, USA, 62-67.

9. C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, June 2009.

10. A. Gandhi, R. Das, J. Kephart, M. Harchol-Balter, and C. Lefurgy. Power Capping Via Forced Idleness. In Proceedings of Workshop on Energy-Efficient Design, 2009.

11. Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. *SIGARCH Comput. Archit. News* 35, 2 (June 2007), 13-23.

12. Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. 2009. Optimal power allocation in server farms. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems* (SIGMETRICS '09). ACM, New York, NY, USA, 157-168.