On the Feasibility of SISO Control-Theoretic DVFS for Power Capping in CMPs

Sina Shahhosseini[1*], Kasra Moazzemi[1*], Amir M. Rahmani[1,2], Nikil Dutt[1]

[1] *Department of Computer Science, University of California Irvine, USA*
[2] *Institute of Computer Technology, TU Wien, Vienna, Austria*

Abstract

Power capping are increasingly being deployed in modern processors to meet performance/power requirements of new workloads. These computing systems are limited in their power dissipation, demanding a dependable power management scheme to guarantee the system's efficiency and dependability. Although several ad-hoc and heuristic power management approaches can be found in the literature, their main shortcoming is the lack of formal guarantees to ensure dependability of the processors. Control-theoretic approaches promise flexibility and robustness for DVFS strategies. However, the creation of a responsive yet stable controller requires the often neglected tasks of proper system identification and performance analysis for target applications. This paper presents dependability evaluation of Single-Input Single-Output (SISO) controllers for power capping on single-core as well as multi-core processor architectures. We evaluate responsiveness of different class of applications to computer system control inputs (i.e., DVFS). We illustrate the feasibility of SISO controllers for power capping using the Sniper simulator running PARSEC, SPLASH2 and a set of custom microbenchmarks. Based on our observations, we provide guidelines for developing stable and robust SISO controllers for power capping, show the scenarios where simple classic SISO controllers might not be effective, and identify early symptoms that may result in instability for power capping controllers.

Keywords:
SISO, Control theory, Robust Control, Power Capping, CMP, SPLASH2, PARSEC, PID controller

[*] These authors contributed equally to this work.

1. Introduction

Modern many-core platforms provide high performance but are increasingly constrained by power dissipation. In addition, applications typically exhibit dynamic characteristics (e.g., memory-bound, compute-bound) throughout their execution, resulting in continual changes in the power state of the system. It is essential to control the peak and average power based on application behavior in order to achieve the proper performance with minimum cost [29]. This requires thorough analysis and sophisticated power management methods to control power and provide necessary performance for a diverse set of workloads [13, 14]. Some approaches [18, 17] use analytical models to estimate the average or worst case power consumption of the system based on frequency and voltage level of the system. These methods fail to take into account the effects of workload and input variability during system execution. A promising and well-established approach is the use of control-theoretic solutions based on rigorous mathematical formalisms that can provide bounds and guarantees for system power consumption. In the past, different control methods have been proposed [24, 25] for resource management in the presence of a specific type of workload running on the system. A majority of these methods use Single-Input Single-Output (SISO) controllers. These SISO controllers often deploy proportional Integral (PI), proportional integral derivative (PID), or lead-lag methods. Although these controllers theoretically provide guarantees for stability and robustness, significant care must be taken in their practical implementation to ensure that these properties continue to hold in the implemented designs. For instance, SISO controller can be implemented at the various layers of the abstraction stack (e.g., application, OS, hypervisor or hardware), resulting in different challenges and design trade-offs: software controllers provide ease of implementation and flexibility, while, hardware controllers provide higher responsiveness to sensor measurements. In many cases, the controller configuration needs to be changed to manage power for a new set of applications. Software-based controllers provide such flexibility but are limited on response time to changes in the system, currently in the order of milliseconds. This could pose problems when an application's phase can change faster than the settling time of the controller. In addition, some applications cannot be controlled using classic static controllers and require more advanced solutions. These

are examples of many issues that demand a thorough analysis of application behavior early enough (e.g., at the time of system identification) and well before controller deployment.

In this paper, we present a comprehensive evaluation of SISO control-theoretic methods for power management. We use Intel's Sniper multi-core simulator to control power using DVFS while executing a variety of benchmarks including the PARSEC and SPLASH2 benchmark suites as well as stress-test benchmarks. A preliminary version of the approach has been proposed in [1] with the following contributions:

- We highlighted the need for careful and early system identification and performance evaluation for SISO controller design through several observations.
- We presented an analysis of multiple controller responses and outlined a general robustness classification of workloads based on their computation and communication intensity.
- We identified early symptoms that can cause instability in controller for a single-core processor and show application classes for which simple classic SISO controllers are not effective in managing single-core processors.

We extend this work to consider power management of multi-core platforms executing multithreaded applications by adding the following contributions:

- We extend the study platform to perform VF scaling on multi-core systems. Analysis is done on multi-core systems of different core counts.
- We included multi-threaded benchmark suites (e.g., PARSEC) in our system identification and performance analysis to study multi-threaded applications.
- We provide a benchmark categorization with respect to the applications' level of parallelism, run-time behavior and power consumption to be used in the choice of control design methods.
- We improve our evaluations on the efficiency of control theoretic approaches to provide scalable management to various types of cores and communication schemes.

The rest of this paper is organized as follows. Section II presents the background and motivation for our evaluation of SISO controllers. Section III outlines the general experimental setup. We present our detailed analysis in Section IV. and discuss the insights learned for two benchmark suites and microbenchmarks in Section V. We position our work with related efforts in Section VI, and finally conclude in Section VII.

2. Background and Motivation

Controller design for feedback loops consists of three main stages: **Modelling, Controller Design, Performance Evaluation**. In the *modelling* stage, the system is described either using analytical or statistical (i.e., black-box) methods. A poorly modeled system, which does not properly consider the corner cases, can result in instability of the controller. Thus, it is essential to make sure that the modeling stage captures the holistic dynamics of the system. In our study, we utilize a black-box method based on System Identification Theory [27, 28] for isolating the deterministic and stochastic components of the system to build the model. *Controller design* is a mature field which utilizes many tools that provide off-the-shelf controllers. We use Matlab PID tuner toolbox [30] to design and deploy our controllers. Finally, *performance evaluation* is conducted to ensure key properties of the system including accuracy, overhead, robustness and flexibility. In this work, we argue that the first stage (i.e., modelling) is often over-simplified in existing controllers deployed for computer systems, and the third step (i.e., performance evaluation) is most often neglected which is essential to ensure the robustness and efficiency of the controller as well as the dependability of the systems. In this study, we focus on issues that might arise from poorly performed models and the lessons learned from performance evaluation.

In our study of SISO controllers, we design and deploy PI controllers for power capping. It is important to note that although derivative control law is helpful to add predictability to the controller, stochastic variations in the system output may cause inaccuracy in the controller. This issue becomes more severe in computer systems as they commonly have a significant stochastic component. Therefore, for computer systems PI controllers are preferred over PID controller [21]. PI control benefits from both integral control (zero steady-state error) and proportional control (fast transient response). In most computer systems a first-order PI controller provides rapid response and is sufficiently accurate [21]. In the rest of this paper, when SISO

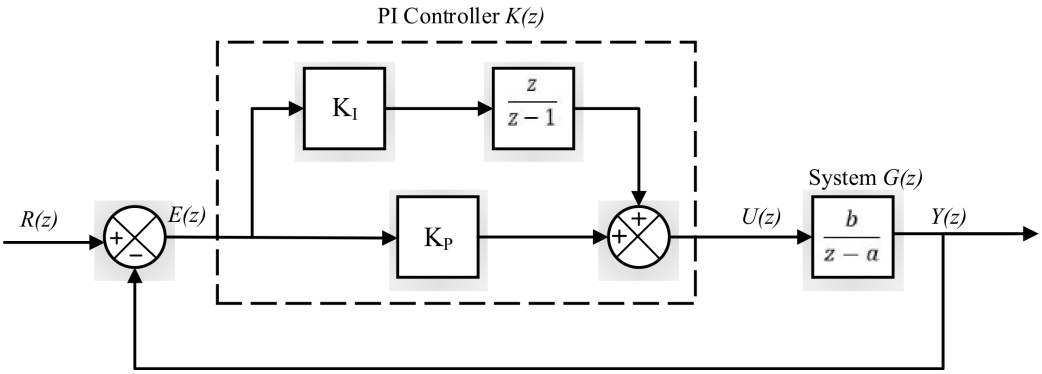

Figure 1: Feedback loop with PI control for a first-order system

controller is mentioned, it would refer to a PI controller. Figure 1 depicts a first-order feedback PI controller modeled in Z-domain. The error *E(z) = R - Y(z)* is the input to the controller. The control input *U(z)* is a sum of the proportional term $K_P \times E(z)$ and the integral term $K_I \times (z/(z-1)) \times E(z)$.

Equation 1 describes a simple discrete PI control form that can later be transformed to transfer function. Note that to compute the current control input *u(k)*, the controller needs to have the current value of the error *e(k)* along with the past value of the error *e(k-1)* and the past value of the control input *u(k-1)*. The PI control law has the form:

$$u(k) = u(k-1) + (K_P + K_I)e(k) - K_P e(k-1) \tag{1}$$

It is important to note that a power management controller designed for only a specific class of applications might not perform well in managing power for other types of workloads. The merit of a controller is measured in terms of four properties: Accuracy, Overhead, Robustness and Flexibility. Thus, a designer's major concern is to evaluate how well a controller satisfies these properties while executing different types of workloads (e.g., compute-bound or memory-bound). The dependability evaluation presented in this work offers designers a better insight on how to properly model (i.e., identify) their system and what kind of considerations they need to take into account when designing controllers for processors.

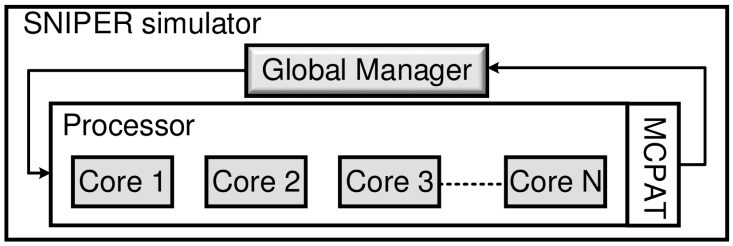

Figure 2: Simulation framework overview

3. Experimental Setup

In this section, we describe our framework and the experimental setup. A workload categorization is presented in order to clarify the performance analysis performed on the SISO controllers. Understanding the behavior of applications is essential in evaluation of identified systems and implemented controllers.

3.1. Simulation Framework

For processor architectural simulation, we use the Sniper [26] simulator which provides micro-architectural details of power and performance of variety of processors. This architectural simulator enables evaluation of single-core and multi-core processors with different communication mechanisms such as bus and Network-on-Chip (NoC). A series of additions was made to this simulator in order to enable run time closed-loop power capping which are discussed in the rest of this section.

3.1.1. Framework Overview

In our work, in order to enable run time power capping using PI controllers, a mechanism called "Global manager" is added to this simulator to manage the DVFS settings at run-time based on computer system response to application behavior. Figure 2 represents overview of this framework. By default, the global manager is invoked every 2.5 ms (common software controller epoch) to obtain the state of the computational cores and determine the next level for their frequency. In addition, MCPAT [3] is used to capture and estimate power consumption.

3.1.2. Architecture Configuration

We model the Gainestown (Nehalem-EP) 45nm microarchitecture for our evaluations. This model simulates an in-order core with the issue width of

Workload	Domain	**Problem size**
Barnes	High-Performance	32768 particles
Ocean-Contiguous	High-Performance	1024*1024 matrix
Ocean Non-contiguous	High-Performance	1024*1024 matrix
FMM	High-Performance	32768 particles
Radiosity	Graphics	room
Raytrace	Graphics	Car -m64
Water-NSQ	High-Performance	2197 Molecules
Water-SP	High-Performance	2197 Molecules
Volrend	Graphics	head

Table 1: SPLASH-2 benchmark list and their problem size

two. In addition each core has 64 KB of L1 cache for data and instruction. L2 and L3 cache for each core has 256 KB and 8192 KB capacity respectively. Data access time for L1 and L2 cache is 4 and 8 clock cycles and for L3 cache, it is 30 clock cycles. Also, DRAM latency is 45 clock cycles and DRAM bandwidth is 7.6 GB/s. In case of multicore simulations, framework is tuned to simulate homogeneous cores controlled by the global manager.

3.2. Benchmark Categorization

Two sets of workloads have been utilized in our work in order to make a comprehensive study of capabilities of SISO controllers for power capping regarding the wide variety of applications behavior. There have been many efforts to construct benchmark suites that can comprehensively represent real world software execution. For example, SPEC [36] workloads include different high performance computing applications. ALPBench [35] is a suite of multimedia workloads. Minebench [37] includes benchmarks for Data Mining Workloads. In the recent years comprehensive benchmark suites like SPLASH2 [34] and PARSEC [33] gained a lot of attention as they cover many domains and in addition they scale well for multi-core systems. In our studies we use these two benchmark suites (PARSEC and SPLASH-2) and provide a detailed analysis on the effects of application behavior on controllability of the system. In addition, a set of micro-benchmarks are devised to stress various parts of a system that are further explained in discussion section.

Workload	**Application Domain**	Parallelization
Blackscholes	Financial Analysis	Data-parallel
Bodytrack	Computer Vision	Pipeline
Canneal	Engineering	Data-parallel
Dedup	Enterprise Storage	Pipeline
Facesim	Animation	Data-parallel
Ferret	Similarity Search	Pipeline
Fluidanimate	Animation	Data-parallel
Freqmine	Data Mining	Data-parallel
Raytrace	Visualization	Data-parallel
Streamcluster	Data Mining	Data-parallel
Swaptions	Financial Analysis	Data-parallel
Vips	Media Processing	Data-parallel
X264	Media Processing	Pipeline

Table 2: PARSEC benchmark list and their Application Domain [32]

3.2.1. SPLASH2

The SPLASH-2 suite is one of the most widely used collections of multithreaded workloads [34]. Table 1 represents a detailed description of workloads included in this benchmark suite. Parallel machines were not as common as nowadays and were mostly used for scientific objectives when SPLASH-2 benchmark suite was released. This fact is reflected in high performance nature of the workloads included in SPLASH-2 benchmarks.

3.2.2. PARSEC

PARSEC as one of the emerging multithreaded benchmark sets contains applications that have been designed to take advantage of multiprocessor computers with shared memory [33]. Applications included in the benchmark suite are composed of programs from a wide range of application domains (e.g., engineering, machine learning, storage, finance, Etc.) in order to capture the increasingly diverse ways in which computers are used. Containing applications with different parallel programming models that are geared toward common CMPs, brings out the possibility of using PARSEC benchmark suite to test the performance of a diverse set of computer systems including embedded systems.

3.2.3. Benchmark Comparison

PARSEC benchmark suite was designed and maintained with the goal of capturing recent trends in computing. Low-degree of similarities between SPLASH-2 and PARSEC indicated that these developments might have an impact on workloads that fundamentally alters their characteristics [32]. For example, PARSEC includes many workloads that follow the pipeline programming model. On the other hand, SPLASH-2 set is mostly composed of high-performance and graphics applications. Furthermore, the input dataset prepared for PARSEC workloads are tuned towards CMPs. In contrast, SPLASH-2 input set are optimized for large shared memories in high performance systems.

4. Evaluation

In this section, we evaluate two often-neglected important aspects in the design of a controller: **System Identification** and **Performance Analysis**. For system identification, we show examples of both well identified systems and poorly modeled systems with some hints about what kind of behavior in the model may result in imprecise controller design. These evaluations are done for both single-core systems and CMPs with demonstration of effect of increasing number of cores. This can be valuable for cases where a controller must be implemented in hardware and changing its configuration is costly. For performance analysis, we evaluate various types of controllers for SPLASH2 and PARSEC workloads and highlight the pros and cons of each method. In addition, the trend of controlling behavior is analyzed for multithreaded applications running on different size of CMPs. Furthermore, as part of our evaluation we categorize the workloads based on measurement of their power consumption and instruction per second (IPS) and then analyze the settling time (all time measurements done by epoch units), maximum overshoot and controllability of each class of application.

4.1. System Identification

After defining the controlled system, the first step would be to generate test wave forms from training applications for system identification [28]. Ideal training applications represent the behavior of applications to be executed on the real system [27]. A test waveform contains a series of samples for controller inputs and outputs for a training application, and should exercise as many input permutations as possible. Once the training data is collected,

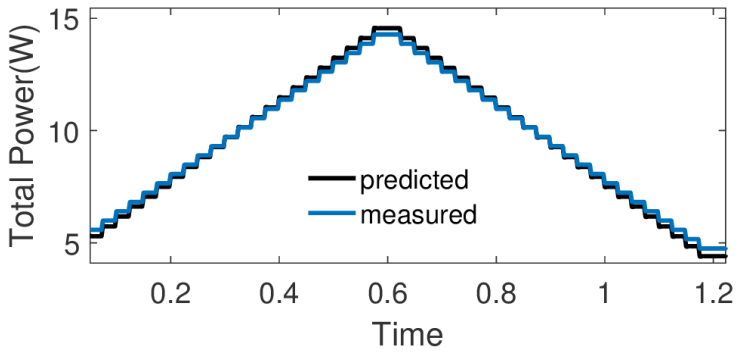

Figure 3: CPU bound microbenchmark with well identified model

the model can be created. During this stage, the system dynamics is exercised often by applying a staircase waveform to the control input (e.g., operating frequency). Such staircase would stimulate system behavior in response to various levels of control input. In our work, we change CPU frequency from 1 GHz to 3.3 GHz with steps of 100 MHz. In this method, training sets use varying frequency (e.g., a set of out-of-phase staircase signals for the control inputs) in order to isolate the deterministic and stochastic aspects of the system. Voltage level is assumed to be fixed in this simulation. This model is then evaluated to predict the expected data from the identified system. Abnormal behavior from this model can raise a flag that the controller to be designed from this model might be inaccurate. In our work, we used Matlab's system identification toolbox for this process [31]. Below, we showcase some of these scenarios.

Figures 3 and 4 demonstrate positive and negative examples that designers have to look for in system identification phase. These two figures show the result of system identification of two hand-tuned workloads. More precisely, they show how well a model can predict the system's output running an application when operating frequency is changed in a staircase form over time. Figure 3 shows that the predicted model for a performance regulated benchmark that closely fits the measured model. On the other hand, Figure 4 shows a memory bound benchmark that lacks the ability to fit into the expected model. Although the controller changes the frequency levels, this change does not have a clear correlation to system output due to the system stalling for memory accesses instead of executing instructions for a majority of simulation cycles.

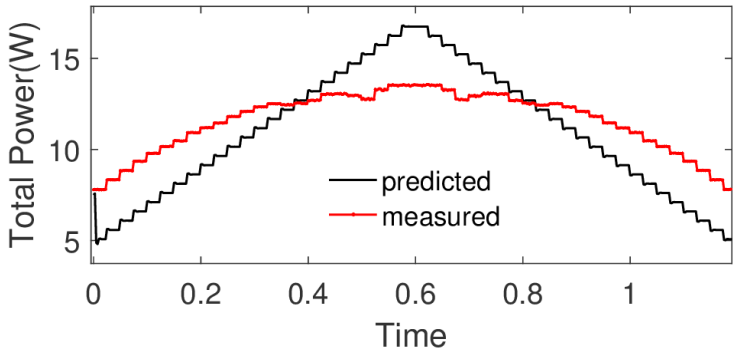

Figure 4: Memory bound microbenchmark model with limited tracking range

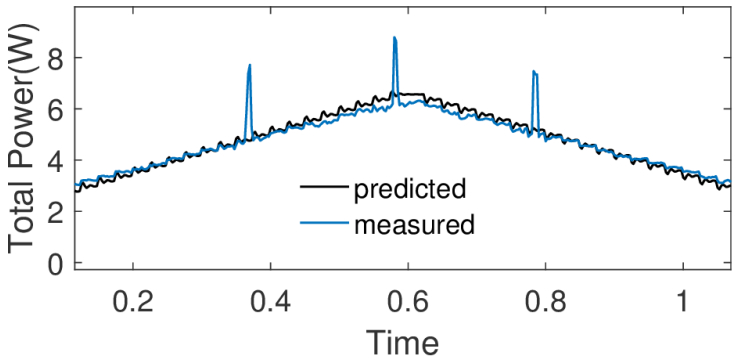

Figure 5: Barnes workload well identified model with noise

4.1.1. Single-Core Models

Next, we investigate a selected set of models that show a series of stochastic behavior that can manifest in the system identification stage. System identification results shown in this section are performed for a single core processor running one thread of each benchmark. Majority of these benchmarks are selected from SPLASH-2 benchmark suite to isolate the effect of off-chip memory accesses. Section 4.1.2 focuses on system identification for CMP models that mostly utilizes PARSEC benchmarks optimized for these architectures.

Figure 5 shows a part of the Barnes workload that closely fit the expected model while demonstrating spikes at certain points of time. These spikes can be the result of a change in the workload execution behavior which is common

in many real-time applications. As the duration of these spikes are very short and the model can rapidly respond to such changes, they are considered as the stochastic part of the system dynamics which should be isolated from the deterministic part, and would not cause any issue in the performance of the system. In contrast, Figure 6 demonstrates part of an identified model for the Raytrace workload that exhibit a long period of underestimation. There are restrictions (such as level of aggressiveness and transient state) that can be applied during controller design stage which can mitigate these abnormal conditions, motivating the need to consider these issues upfront.

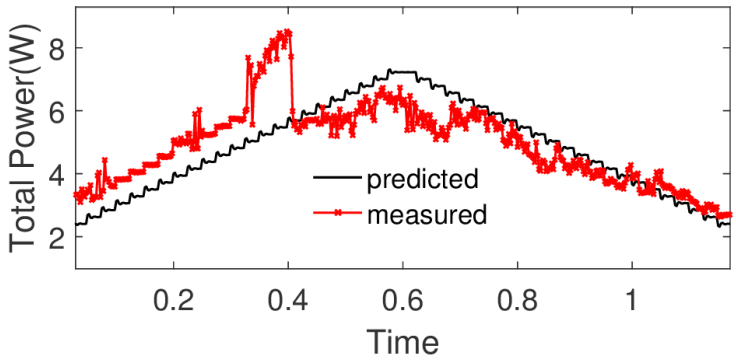

Figure 6: Raytrace workload model exhibiting error in prediction

4.1.2. CMP Models

After identifying models for single-core processors, we aim to identify models for CMP systems. This change expands the exploration space for controller design. On one side, exceeding number of cores can be a challenge for system identification phase and controller design process. On the other hand, the multithreaded behavior of applications over different cores can add a large noise when controlling the whole system complex with solely one SISO controller. To address some of the mentioned challenges we study the capability of SISO controllers in power capping of CMPs with following scenarios for the system identification phase:

- We demonstrate a single threaded application running on a multi-core processor. One core is executing the application thread and other cores simply have idle power consumption. The goal of this experiment would be to see the static effect of idle cores on system identification.

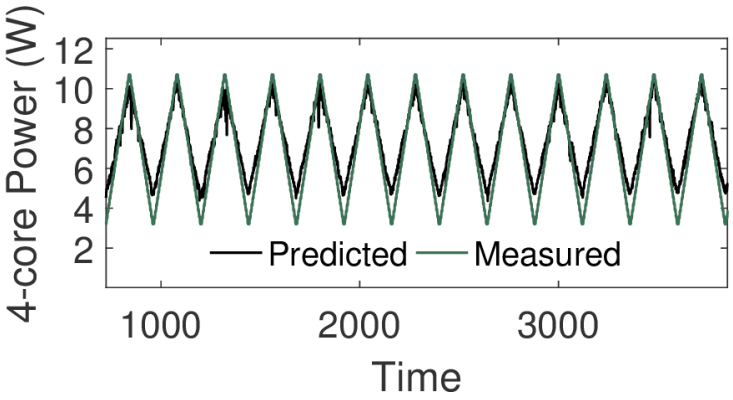

Figure 7: System identification for a 4-core system using one thread of Freqmine benchmark executing on one core. Model fits general flow with a static shift at the bottom.

- We analyze the power model for CMPs using DVFS. We demonstrate the variety of dynamic power changes on a 4-core processor based on run-time behavior of each application thread. In addition, we evaluate the accumulated total power for the whole system in respect to power consumption of each core and platform communication mechanism.

- We assess the system identification capability for CMP systems with low number of homogeneous cores.

- We illustrate the trend of increase in inaccuracy of system identification stage as the number of cores grows larger.

The first transition from a single-core processor to a multi-core processor system identification model is extending the architecture while keeping the same configuration for the software execution. This would reveal some of the key points to look after in design process of a controller for the multi-core systems. In order to evaluate this case, we evaluate same benchmarks on a 4-core platform while restricting the number of threads to one. Figure 7 shows the result of system identification for Freqmine benchmark. While the application is running on only one of the four cores, the other cores consume power in their idle state. This constant power usage would manifest itself with slight shift at the bottom of each staircase period. If the model can fit the general trend, this shift can easily be eliminated in control design step.

Furthermore, we take a look at general power actuations during black-box system identification for a 4-core multi-processor. Figure 8 shows power

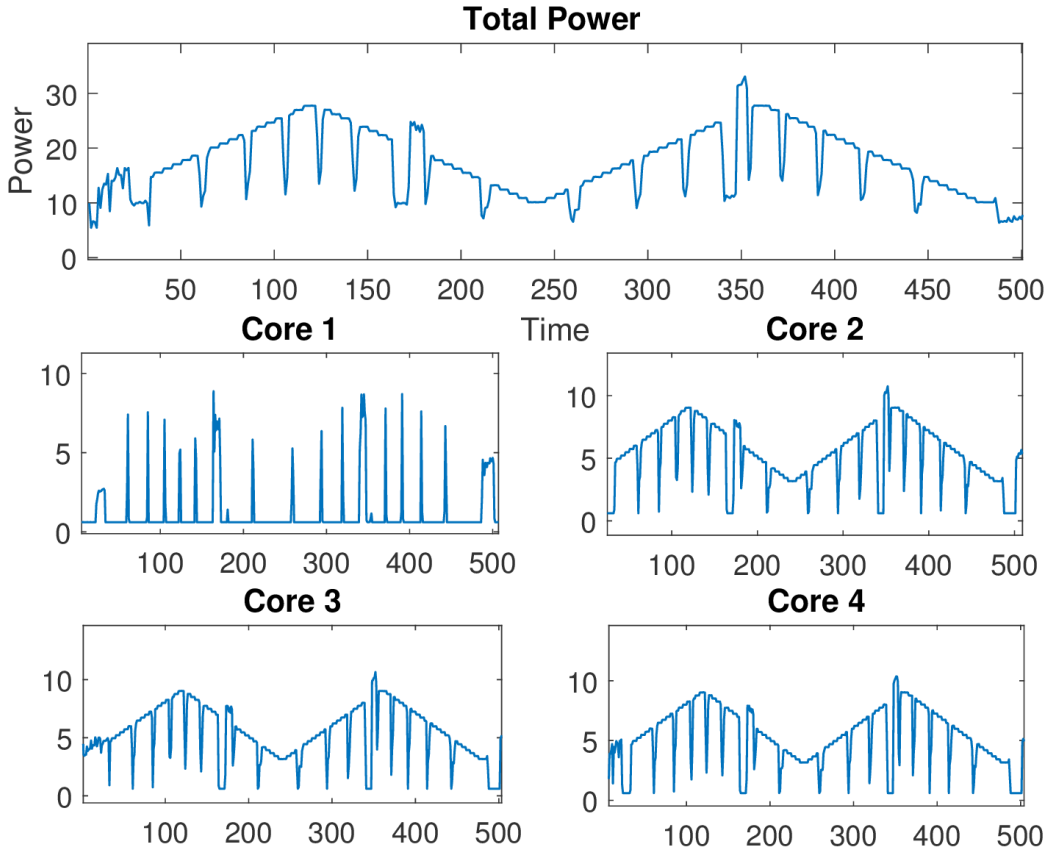

Figure 8: Power usage of a 4-core system while tuning Bodytrack benchmark. Top figure represents Total power of the whole system and the rest are break down of each core power.

measurement while execution of Bodytrack benchmark on a homogeneous 4-core system. Top part of Figure 8 shows the staircase model of total power of the whole system that shows a good response to changes in system frequency. This model is able to rapidly recover from spikes and slopes that are caused by application stochastic behavior on each individual core. Four bottom system identification models in Figure 8 show individual core response to frequency changes. We can observe that each core depending on the running a thread shows unique run-time behavior but at the end we are concerned with aggregated total power model.

Next step is to analyze how well this model can fit the predicted model. Figure 9 demonstrates the system identification of the 4-core system that

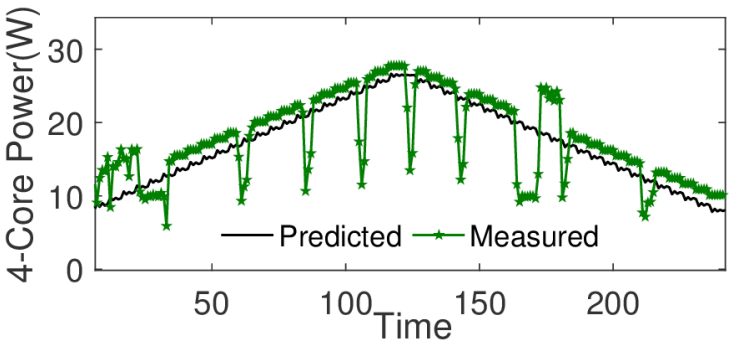

Figure 9: 4-core system identification for bodytrack benchmark

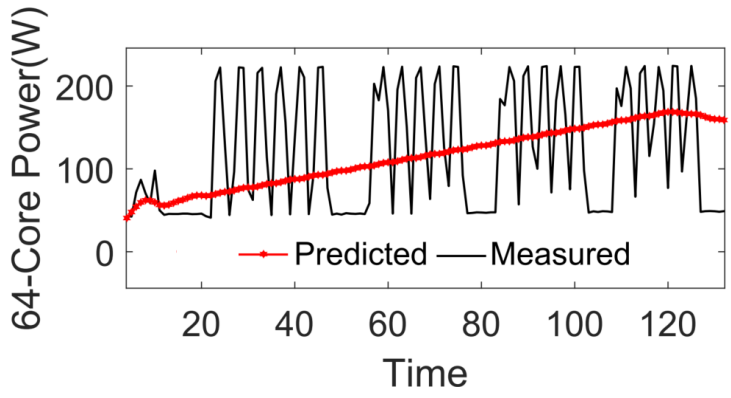

Figure 10: 64-core system identification for bodytrack benchmark

we have seen previously using *Bodytrack* benchmark. This model shows a promising trend for design of a SISO controller for the 4-core CMP. Majority of PARSEC benchmarks show similar results for system identification stage using a 4-core system. We were curious to see if it is possible to identify larger systems for an accurate control design. We extended the simulated architecture to a 64-core network on chip system. Figure 10 shows system identification for 64 thread of the same benchmark application (*Bodytrack*) on the 64-core platform. It can be easily inferred from Figure 10 that this model cannot be easily identified and the controller designed from this model might lead to unresponsive system.

To demonstrate the trend of decrease in accuracy of system identification, we picked one of the benchmarks (*Swaptions*) that showed well-fitted model

Benchmark	4-cores	8-cores	16-cores	32-cores
Fit to estimation	78.99%	40.7%	9.529%	3.004%

Table 3: Fit to estimation data trend with increase in number of computing cores while executing one thread of *Swaptions* Benchmark on each core.

for 4-core platform for further evaluation. We extended the simulation to 8, 16 and 32 cores to see the fit to model trend. Table 3 shows the decrease in the ability to fit the predicted model while increasing the number of processing cores. The two important notes from system identification stage is to evaluate the responsiveness of the controller to control inputs and grasp a better understanding of stochastic and deterministic behavior of application.

To give better insight regarding the decrease in accuracy of system identification when moving form small number of cores (4-cores) to a platform with large number of cores (64-cores) a cross-validation of residuals has been done. Figure 11 demonstrates both cross-correlation and autocorrelation evaluation for bodytrack benchmark. Residual is the stochastic component (e.g., disturbance, noise, etc.) of the system output, which is not supposed to be included in the model. When validating the model, the model output is compared to noisy system outputs. Therefore we expect the residual to be pure noise. To verify this, the residual is analyzed for correlation. If there is no correlation between the residual and itself or any inputs, the model is accurate enough. *Confidence* can be used to specify a range. In this work, commonly used 99 percent boundaries have been set for the confidence. A confidence level is the probability with which the true output will fall into the range of confidence boundaries. After an estimated system dynamics is produced using system identification techniques, it is cross-validated using different data sets. Cross-correlation is a standard method of estimating the degree to which two series are correlated. In our case, cross-correlation (bottom part of Figure 11) is evaluated for power as the output of the system based on the frequency as the input of the identified system. We can observe that the 4-core system model can retain in the confidence boundaries while the larger system model is outside these boundaries for all samples. The cross-correlation is similar in nature to the convolution of two functions. In an autocorrelation, which is the cross-correlation of a signal with itself, there will always be a peak at a lag of zero. The top part of Figure 11 compares auto-correlation of residuals for these two systems. Similar to the previous part, only the 4-core system identification can stay in the boundaries. These

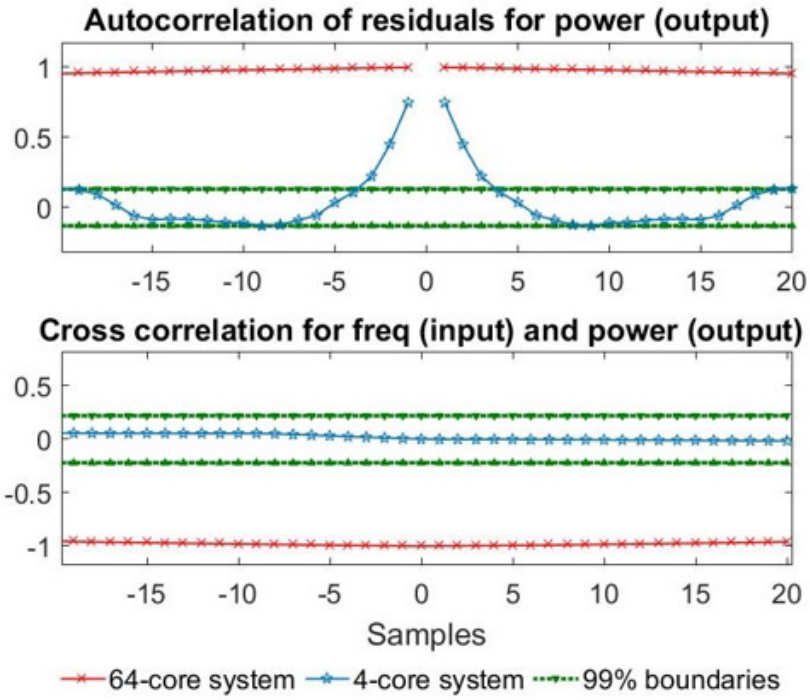

Figure 11: Auto/Cross-correlation of residuals for 4-core and 64-cor systems.

results show controllers for a system with large number of cores are often infeasible to design due to the lack of a sufficiently accurate system dynamics model.

4.2. Performance Analysis

After the system identification stage, controller design is performed by using the Matlab PI tuner [30]. Typically there are three ways that designers choose to design a controller for a computer system. The first set of methods take a statistical average of metrics gathered from system identification phase to represent the general case. The second scenario involves designing a controller for a system that runs predefined workloads (i.e., application specific) such as a smart watch or industrial plant machines. In this case, designers have the opportunity to tune the controller based on the application at hand for better accuracy. Table 4 shows these workload specific control parameters (i.e., gains) used to control the system running each

benchmark (i.e., optimal application specific parameters extracted from Matlab). Finally, the third scenario uses a worst case configuration that performs conservatively for all benchmarks and is more robust against disturbances, however suffers from slow settling time. It should be mentioned that despite all these methods and vast variety in off-the-shelf controllers, there are some applications that cannot be controlled with a simple SISO controller and that would either require more advanced controllers (e.g., non-linear, adaptive, self-tuning) or different/more configuration knobs. We describe these scenarios in Subsection 4.2.4.

Workload	K_P	K_I	**Multithreaded**	K_P	K_I
Barnes	114	229	Blackscholes	10.08	20.17
Ocean-Contiguous	156	226	Bodytrack	12.50	25.10
FMM	114	229	Facesim	8.7	5.4
Radiosity	184	369	Ferret	23.2	46.4
Raytrace	244	247	Fluidanimate	20.17	40.35
Water-NSQ	139	228	Freqmine	68.1	136.01
Water-SP	175	250	Swaptions	39.1	40.2
Volrend	141	282	X264	137.1	47.85
Average	180	240	Average	38.86	45.18

Table 4: CPU core configuration for Nehalem-EP

4.2.1. Customized case

For many systems using control-theoretic power managers, we may have design time knowledge regarding the workloads to be executed. This enables control designers to customize the power manager based on these pre-defined applications. System identification and controller design stages are performed individually on each application. Table 4 shows these workload specific K_P and K_I configurations. Figure 12 shows proper behavior of the *Water_nsq* benchmark in tracking the 7 Watts power reference. Ability to track a specific reference would be essential later on when DVFS manager wants to set a power reference to optimize energy efficiency. Examples from multithreaded applications are discussed in Section 4.2.3. We observe similar trends for all other workloads except the two benchmarks discussed in Section 4.2.4.

4.2.2. Average and worst case

Many general-purpose systems do not have the flexibility to accommodate customized controllers either due to variety of system workloads or because

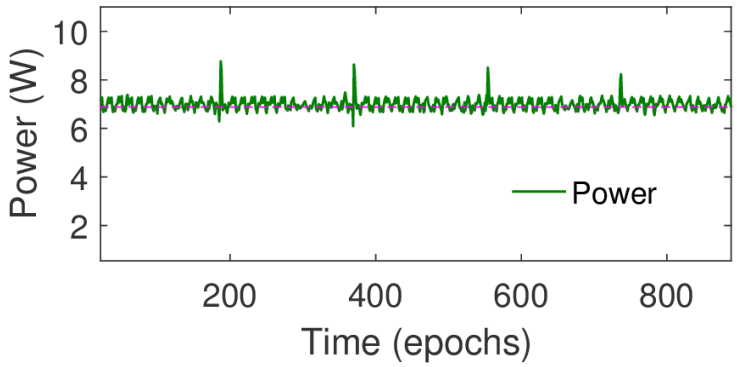

Figure 12: Example of well-tuned controller for *Water-NSQ* benchmark following 7W power reference

the controller cannot be easily reconfigured. In these situations, designers choose a representative configuration that can meet their requirements. Here two commonly reported control strategies use a statistical average case of predicted applications [23] or use a worst case scenario that can respond with slower speed but which provide larger margins of guarantees.

As an example for the average case, Figure 13 shows the difference between the customized controller and average case controller for the *FMM* benchmark in tracking the 7 Watts power reference. Both cases can keep the power close to the reference but the customized controller minimizes the tracking error with minimal deviations from the reference, while the average controller oscillates over the power reference. This is due to the fact that fine-grain step of the average case controller is larger than what this workload requires. For the worst case, Figure 14 shows the comparison between the customized case and worst case for the Raytrace workload. As expected, the worst case scenario has slower settling time due to smaller steps (smallest K_P and K_I) but after reaching 7 Watts, it can reliably follow the power reference.

4.2.3. CMP Controllers

Nowadays, most of computer systems including embedded systems utilize Chip Multi-Processors (CMPs) or Heterogeneous Multi-Processors (HMPs). The advantage of using multiple cores on a single die is that these multiprocessors become available commodity for parallel applications. In comparison to majority of SPLASH-2 benchmarks evaluated in previous section which are designed for high performance computers, PARSEC applications are optimized to take advantage of CMPs. Table 4 specifies the customized control

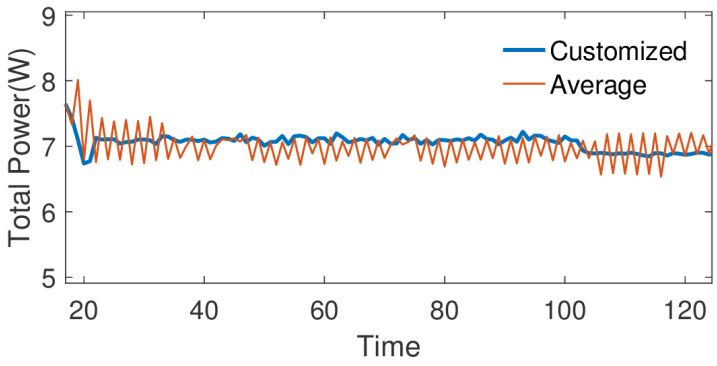

Figure 13: FMM benchmark with average and customized case

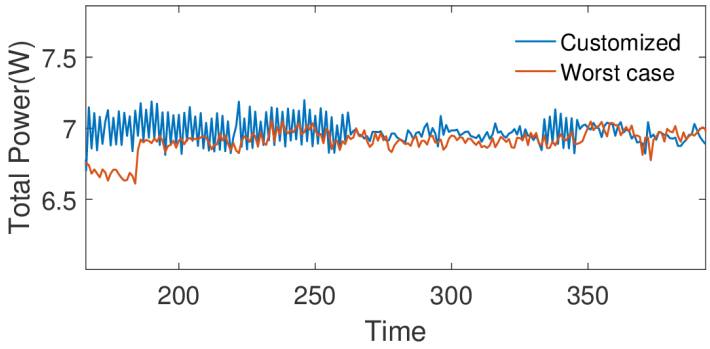

Figure 14: Raytrace benchmark with average and customized case

parameters for some of these benchmarks.

To show some of the selected controllers designed for CMPs, we identify all PARSEC benchmarks for a 4-core system using bus communication and share memories. Insights from system identification phase were presented in Section 4.1.2. Here, we select some of these applications for control design using methods discussed in the beginning of Section4.2. Figure15 shows power reference tracking (20 Watts) for a customized controller for *Swaptions* benchmark running on a 4-core system. As shown here the SISO controller designed from a well-identified model has no trouble controlling the total power of a 4-core system.

Figure 16 compares two controllers designed for *Facesim* benchmarks. First controller is a custom designed controller for this benchmark which shows rapid conversion to 20 Watts power reference with low overshoot. Sec-

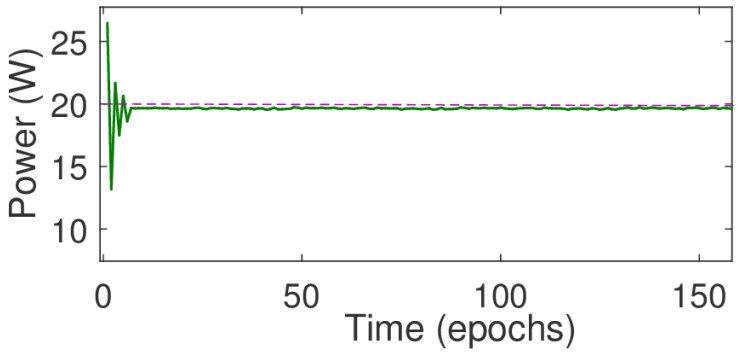

Figure 15: 4-core controller tracking 20 Watts for *Swaptions* benchmark.

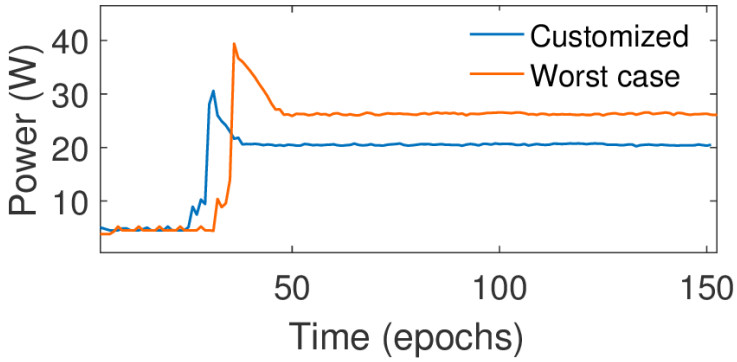

Figure 16: Comparison of customized and worst case controller for 4-core system tracking 20 Watts for *Facesim* benchmark.

ond controller is a worst case controller that follows the same trend but slower and more sluggish. Also there is a bigger overshoot and steady-state error.

4.2.4. Corner cases

So far we evaluated both dynamic and static methods to design and deploy a PI controller for power capping. Using lessons learned from these evaluations, designers can choose the suitable method for their system. However, it is important to note that the appropriateness and feasibility of these methods depend on the system being *controllable*. The controllability property guarantees that the controller can always keep the plant within a set of boundaries around the reference. In other words, if the controller is not provided with proper means (actuators or configuration knobs), it would be unable to reach the desired reference. Figures 17 and 18 show the system identification, and controller deployment phases of the system running *Ocean* benchmark. Both

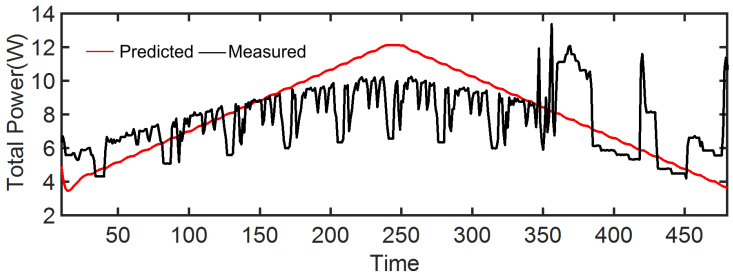

Figure 17: Ocean Non-Contiguous workload. System identification of uncontrollable workloads

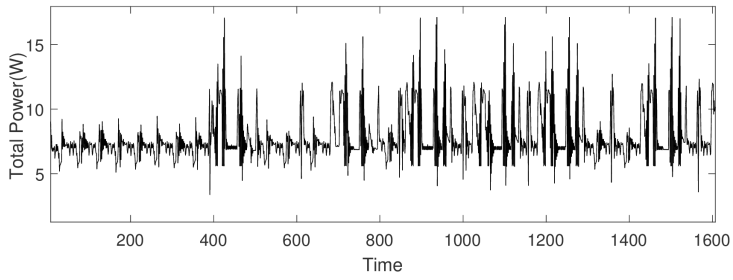

Figure 18: Ocean Non-Contiguous workload. Performance analysis of uncontrollable workloads while trying to track 7 Watts reference

implementations of *Ocean* benchmark (contiguous and non-contiguous) show similar behavior. These applications are not controllable using solely DVFS actuation. In the following section, we analyze these benchmarks in more details to elaborate of the reasons behind their abnormal behavior.

5. Discussion

In this section, we discuss the reliability and performance of SISO controllers in power capping of different class of workloads based on the evaluations done in the previous section. Performance analysis done on the deployed controllers showed stability for majority of the workloads for single-core and 4-core CMPs. In addition, hand tuned controllers were able to meet the second set of requirements which are maximum 30% overshoot and settling time less than 150ms. In some cases, controllers using the statistical average were not able to meet the overshoot requirements. The reason for more frequent power overshoots in average case is that it does not have the fine grain tuning that some of the workloads require. Although the worst case configuration

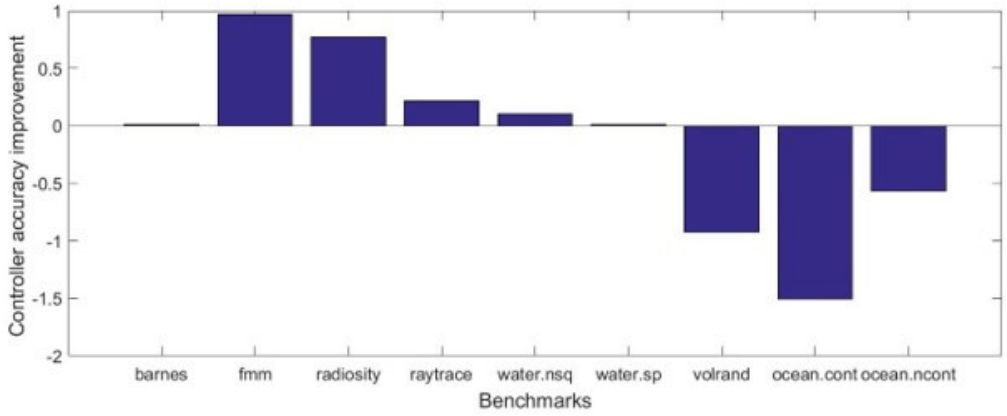

Figure 19: Accuracy improvement from software controller to a faster hardware controller

was not as rapid as the customized controllers, overall it proved to be a reliable controller. Therefore, for scenarios where the computer system is designed to execute an application with similar computing characteristics, the average case can be a valid candidate; and for systems sensitive to changes in power levels that can tolerate some degree of performance overhead, worst case controllers can be deployed.

In our experiments, we observed few benchmarks that exhibited abnormal behavior in tracking power references with high standard deviation. Figure 18 shows the behavior of one of these benchmarks. Our first reasoning behind this behavior was that slow response time of a software controller is longer than the periods of time that these workloads change their application phases. This can cause a late response (change in frequency) to a phase that is already passed which can exacerbate the current power state. In order to check this issue we moved our software SISO controller mechanism to the hardware level with 10× faster sampling and DVFS epochs (from 2.5ms to around 0.25ms). Contrary to our expectations, the experiment showed that a faster controller did not have much improvement on these cases. Although we were able to capture power violations at an earlier stage, the responses of our controllers were not able to mitigate this issue. Figure 19 shows controller accuracy improvement when migrating from software controllers to hardware controllers. For most of controllable benchmarks, faster hardware controller shows small (less one percent) increase in accuracy but for the corner cases this faster response causes ripple effect and reduction in accuracy.

Our next solution to this issue was to investigate these benchmarks in more detail. We looked at a few measurable metrics and what stood out was the average power consumption. The results in Table 5 shows the average power consumption of each benchmark in SPLASH-2 benchmark suite while tracking 7-Watt power reference. As we can see, only the two irregular benchmarks (Ocean-Contiguous and Ocean Non-contiguous) have the average power consumption higher than 7 Watts which results in many power violations. Taking into account the inability to track the power reference and the high average power indicated that there might be a barrier that prevents the application behavior to rapidly follow changes in the CPU frequency. At this stage, these two benchmarks were suspected to be memory-bound compared to the rest of the workloads that are CPU-bound. In order to verify this hypothesis, we measured the instruction per second (IPS) of all similar high performance workloads in our benchmark set and tailored the two microbenchmarks that stress CPU and memory modules. The average IPS gathered from each workload is reported in Table 5. We could clearly observe that IPS for irregular workloads were far less than the other workloads in the SPLASH2 benchmark suite. Memory-bound microbenchmarks exhibited similar behavior with an average power higher than reference power and an IPS less than one half of other benchmarks' average IPS. This experiment validated the theory that the abnormal behavior of the two Ocean benchmarks is due to their high volume of memory accesses which prevents the changes in CPU frequency to have a direct effect on power reference. In order to enable controller to respond better to memory-bound applications, we have increased the order of the controller three times. Our evaluations show, compared to first order controllers, second, third and fourth order controllers had [-2, 2] percent difference in controller performance which is neither sufficient nor computationally effective. Such cases either require more advanced controllers (e.g., MIMO adaptive, self-tuning) or different/more configuration knobs such as memory bandwidth that can effect the system's power more efficiently.

6. Related work

There is a large body of literature on heuristic-based approaches for resource management [8, 9, 10, 11, 12]. There have been a variety of techniques that are used for power and resource management in processors. Heuristics such as [2, 4, 5] uses a rule-based method in contrast to threshold meth-

Workload	Average power (Watts)	IPS
Barnes	6.3289	2.21E+09
Ocean-Contiguous	7.8306	1.07E+09
Ocean Non-contiguous	7.5408	1.36E+09
FMM	6.9943	3.55E+09
Radiosity	6.7472	2.82E+09
Raytrace	6.4023	2.71E+09
Water-NSQ	6.9931	3.14E+09
Water-SP	6.9846	2.97E+09
Volrend	6.7491	3.30E+09
Compute-bound ubench	6.7207	4.18E+09
Memory-bound ubench	7.1668	1.18E+09

Table 5: Comparison of average power and IPS

ods [6, 7]. Predictive methods [15, 16, 17] that typically benefit from nested loops have been used to manage resources in computer systems. There are shortcomings to ad-hoc and heuristic-based approaches in addressing some of the important attributes of a controllable system. For example lack of guarantees, the need for exhaustive training and close to reality models. Furthermore, there have been control theory based methods [19, 20, 22, 23, 24, 25] that provide formal and efficient means to address robustness and testability for managing computer systems. The most successful of these concurrently coordinate and control multiple goals and actuators in a non-conflicting manner by adding an ad-hoc component to a controller or hierarchical loops. In [38] the authors provide a guide for designing MIMO formal controllers for tuning architectural parameters in processors to enhance coordination, and demonstrate the coordinating management of multiple goals for unicore processors. In addition, authors in [39] demonstrate design methodology of these controllers for multicore platforms. In this work we analyze an ample set of workloads in detail to point out benefits and shortcomings of SISO controllers for power capping for both unicore and multicore processors.

7. Conclusion

In this paper, we evaluated the dependability of SISO control-theoretic power managers and presented guidelines for system designer on how to properly model their system and verify the performance of the designed controller.

The growing significance of power and thermal issues in these system calls for robust power capping schemes that provide guarantees on the system's efficiency and dependability. One of the viable solutions to this issue is the use of control-theoretic methods with formal guarantees. We showed that in the design phase of the existing power managers, system identification and controller performance analysis phases are often neglected. In this work, we took a closer look at issues that might arise from these two steps and provided guidelines to designers on how to design and deploy more reliable controllers. Our evaluations include power capping methods for both single-core and multi-core processors. Moreover, an in-depth analysis has been performed on multiple controller configurations using SPLASH2 and PARSEC benchmark suites.

Acknowledgment

We acknowledge financial support by the Marie Curie Actions of the European Union's H2020 Programme, as well as the National Science Foundation under NSF grant CCF-1704859.

References

[1] Sina Shahosseini, Kasra Moazzemi, Amir M. Rahmani, Nikil Dutt. Dependability Evaluation of SISO Control-Theoretic Power Managers for Processor Architectures. In Proc. of NORCHIP, 2017.

[2] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In Proc. of MICRO, 2006.

[3] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In Proc. of MICRO, 2009.

[4] Ashutosh S. Dhodapkar and James E. Smith. 2002. Managing multi-configuration hardware via dynamic working set analysis. In Proc. of ISCA, 2002.

[5] Eiman Ebrahimi, Onur Mutlu, Chang Joo Lee, and Yale N. Patt. In Proc. of MICRO, 2009.

[6] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. 2012. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45).

[7] Hwisung Jung, Peng Rong, and Massoud Pedram. 2008. Stochastic modeling of a thermally-managed multi-core system. In Proceedings of the 45th annual Design Automation Conference (DAC '08).

[8] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No "power" struggles: coordinated multi-level power management for the data center. SIGOPS Oper. Syst, 2008.

[9] Seungryul Choi and Donald Yeung. 2006. Learning-Based SMT Processor Resource Distribution via Hill-Climbing. SIGARCH Comput. Archit. News, 2006.

[10] Priyanka Tembey, Ada Gavrilovska, and Karsten Schwan. 2010. A case for coordinated resource management in heterogeneous multicore platforms. In Proc. of ISCA, 2010.

[11] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In Proc. of MICRO, 2011.

[12] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. 2008. Power capping: a prelude to power shifting. Cluster Computing 11, 2 (June 2008), 183-195.

[13] Hamid Nejatollahi and Mostafa E. Salehi. Voltage scaling and dark silicon in symmetric multicore processors. The Journal of Supercomputing, Springer Nature. 2015.

[14] Hamid Nejatollahi and Mostafa E. Salehi. Reliability-Aware Voltage Scaling of Multicore Processors in Dark Silicon Era. Journal of Advances in Parallel Computing. 2018.

[15] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. 2008. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In Proce. of MICRO, 2008.

[16] Christophe Dubach, Timothy M. Jones, and Edwin V. Bonilla. 2013. Dynamic microarchitectural adaptation using machine learning. ACM Trans. Archit, 2013.

[17] Divya Mahajan, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, and Hadi Esmaeilzadeh. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration. In Proc. of ISCA, 2016.

[18] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. In Proc. of ASPLOS, 2011.

[19] A. M. Rahmani et al., "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In Proc. of ISLPED, 2011.

[20] Yefu Wang, Kai Ma, and Xiaorui Wang. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. SIGARCH Comput. Archit. News, 2009.

[21] J. L. Hellerstein , Yixin Diao, Sujay Parekh, Dawn M. Tilbury. ,Feedback Control of Computing Systems .John Wiley Sons, 2004

[22] A. Bartolini, M. Cacciari, A. Tilli and L. Benini, "A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores. In Proc. of DATE, 2009.

[23] Asit K. Mishra, Shekhar Srikantaiah, Mahmut Kandemir, and Chita R. Das. 2010. CPM in CMPs: Coordinated Power Management in Chip-Multiprocessors. In Proc. of SC, 2010.

[24] Q. Wu, P. Juang, M. Martonosi, L. S. Peh and D. W. Clark, "Formal control techniques for power-performance management. In Proc. of MICRO, 2005.

[25] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. 2004. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. SIGARCH Comput. Archit, 2004.

[26] Trevor E. Carlson et al. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In Proc. of SC, 2011.

[27] L. Ljung, System Identification: Theory for the User. Prentice Hall PTR, 1999.

[28] L. Ljung, Black-box models from input-output measurements, in Proc. of IMTC, 2001

[29] Rahmani, A.M. and Liljeberg, P. and Hemani, A. and Jantsch, A. and Tenhunen, H., The Dark Side of Silicon . Springer, Switzerland 1st Ed., 2016.

[30] MathWorks, "Designing PI Controllers with PID Tuner" Tech. Rep., 2017."https://www.mathworks.com/help/control/getstart/designing-pid-controllers-with-the-pid-tuner-gui.html"

[31] MathWorks, " System Identification Toolbox" Tech. Rep., 2017."https://www.mathworks.com/products/sysid.html"

[32] C. Bienia, S. Kumar and Kai Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors," 2008 IEEE International Symposium on Workload Characterization, Seattle, WA, 2008, pp. 47-56.

[33] C. Bienia, S. Kumar, J. P. Singh and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," 2008 International Conference on Parallel Architectures and Compilation Techniques (PACT), 2008

[34] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," Proceedings 22nd Annual International Symposium on Computer Architecture, 1995

[35] Man-Lap Li, R. Sasanka, S. V. Adve, Yen-Kuang Chen and E. Debes, "The ALPBench benchmark suite for complex multimedia applications," IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005., 2005, pp. 34-45.

[36] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. SIGARCH Comput. Archit, 2007

[37] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik and A. Choudhary, "MineBench: A Benchmark Suite for Data Mining Workloads," 2006 IEEE International Symposium on Workload Characterization, San Jose, CA, 2006.

[38] R. P. Pothukuchi, A. Ansari, P. Voulgaris and J. Torrellas, "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, 2016, pp. 658-670

[39] T. R. Muck, B. Donyanavard, K. Moazzemi, A. M. Rahmani, A. Jantsch and N. D. Dutt, "Design Methodology for Responsive and Robust MIMO Control of

Heterogeneous Multicores," in IEEE Transactions on Multi-Scale Computing Systems.