Dependability Evaluation of SISO Control-Theoretic Power Managers for Processor Architectures

Sina Shahosseini[1], Kasra Moazzemi[1], Amir M. Rahmani[1,2], and Nikil Dutt[1]
[1]Department of Computer Science, University of California, Irvine, USA
[2]Institute for Computer Technology, TU Wien, Vienna, Austria
e-mail: {sshahhos, moazzemi, amirr1, dutt}@uci.edu

***Abstract*—Dynamic power managers are increasingly being deployed in modern embedded 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 power management 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) for power management on processor architectures. We also analyze the effect of frequent application phase changes on the responsiveness of controllers. We evaluate responsiveness of different class of applications to computer system control inputs such as DVFS. We illustrate the feasibility of hardware and software SISO controllers for power management using the Sniper simulator running SPLASH2 and microbenchmarks. Based on our observations, we provide guidelines for developing stable and robust SISO controllers for power management, show the scenarios where simple classic SISO controllers might not be effective, and identify early symptoms that may result in instability for power management controllers.**

***Keywords*—SISO, Control theory, Robust Control, Dynamic Power Management, SPLASH2, PID controllers**

I. 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 behaviour in order to achieve the proper performance with minimum cost [25]. This requires thorough analysis and sophisticated power management methods to control power and provide necessary performance for a diverse set of workloads. Some approaches [14], [13] 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 [20], [21] 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 tradeoffs: 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 behaviour 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 SPLASH2 benchmark suite as well as stress-test benchmarks. The main contributions of this paper are:

- We highlight the need for careful and early system identification and performance evaluation for SISO controller design through several observations
- We present a detailed analysis of multiple controller responses for the SPLASH2 benchmark suite and outline a general robustness classification of workloads based on their computation and communication intensity
- We identify early symptoms that can cause instability in a controller and lay down guidelines for stable and robust SISO controller design for power management in computer systems, and show application classes for which simple classic SISO controller is not effective.

The rest of this paper is organized as follows. Section II

978-1-5386-2844-7/17/$31.00

presents the background and motivation for our evaluation of SISO controllers. Section III outlines the general system design. We present our detailed analysis in Section IV. and discuss the insights learnt for two benchmark suites in Section V. We position our work with related efforts in Section VI, and finally conclude in Section VII.

II. 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 [23], [24] 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 [26] 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 management. 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 [17]. 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 [17]. In the rest of this paper, when SISO 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)*. It is this memory inherent in the PI controller that makes it dynamic (in contrast to the static PI controller). The PI control law has the form:

$$u(k) = u(k-1) + (K_P + K_I)e(k) - K_P e(k-1) \quad (1)$$

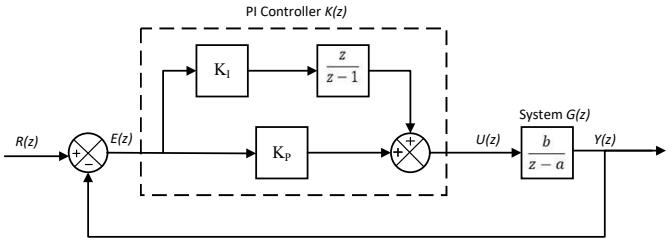


Fig. 1: Feedback loop with PI control for a first-order system

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.

III. Experimental Setup

In this section, we describe our framework and the experimental setup. A categorization of benchmarks is presented in order to clarify the performance analysis performed on the SISO controllers.

A. Simulation Framework

For processor architectural simulation, we use the Sniper [22] simulator which provides micro-architectural details of power and performance; and we model the Gainestown (Nehalem-EP) 45nm microarchitecture for our evaluations. Table I shows the configuration of this architecture. In addition, McPAT [2] is used to capture and estimate power consumption. In this work, in order to enable run time power management using PI controllers, a mechanism called "Global manager" is added to this simulator to manage the DVFS settings at runtime based on computer system response to application behaviour. 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.

Core	configuration
Issue width	2(In Order)
L1$/$D size(KB)	32/32
L2 size(KB)	512

TABLE I: Simulation core configuration

B. Benchmark Categorization

In this work, we used two sets of workload: the SPLASH2 comprehensive benchmark suite; and two custom microbenchmarks exercising *computation-bound* and *memory-bound* workloads. The purpose of these two micro-benchmarks is to stress specific parts of the architecture to observe the ability of the controllers to track the power reference. This categorization is made based on the volume of memory access and CPU utilization of each workload. Table II shows the list of the benchmarks used from the SPLASH2 suite and their corresponding input sizes.

Workload	Input size
Barnes	32768
Ocean-Contiguous	1024*1024 matrix
Ocean-Nocontiguous	1024*1024 matrix
FMM	32768
Radiosity	room
Raytrace	Car -m64
Water-NSQ	2197 Molecules
Water-SP	2197 Molecules
Volrend	head

TABLE II: Benchmark list and their input size

IV. 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 behaviour in the model may result in imprecise controller design. 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 workloads and highlight the pros and cons of each method. 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, maximum overshoot and controllability of each class of application.

A. System Identification

After defining the controlled system, the first step would be to generate test waveforms from training applications for system identification. Ideal training applications represent the behavior of applications to be executed on the real system. 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, 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 behaviour 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 behaviour 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 [27]. Below, we showcase some of these scenarios.

Figures 2 and 3 show the result of system identification of two microbenchmarks. More precisely, they show how well a model can predict the system's output running a microbenchmark when operating frequency is changed in a staircase form over time. Figure 2 shows that the predicted model for CPU bound benchmark closely fits the measured model. On the other hand, Figure 3 shows that the memory bound benchmark lacks the ability to fit into the expected

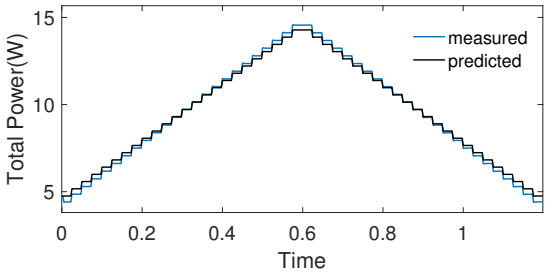


Fig. 2: CPU bound microbenchmark well identified model

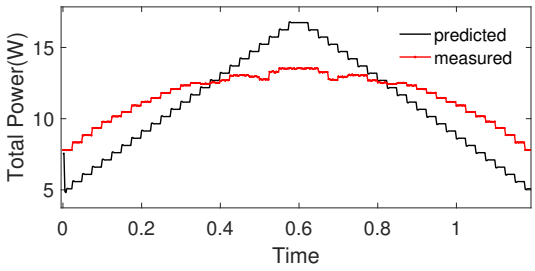


Fig. 3: Memory bound microbenchmark model with limited tracking range

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.

The next set of models show some stochastic behaviours that can manifest in the system identification stage. Figure 4 shows a section 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 behaviour which is common in many realtime 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 5 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. The two important notes from system identification is to evaluate the responsiveness of the controller to control inputs and grasp a better understanding of stochastic and deterministic behavior of application.

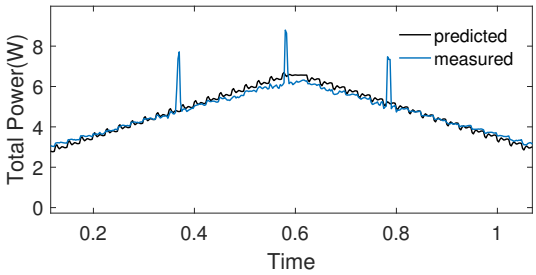


Fig. 4: Barnes workload well identified model with noise

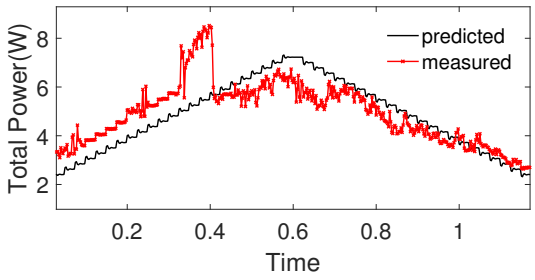


Fig. 5: Raytrace workload model exhibiting error in prediction

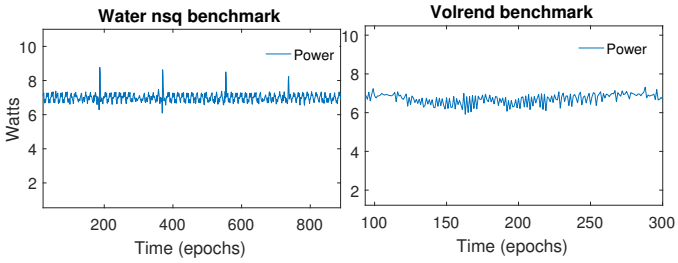


Fig. 6: Examples of well-tuned controllers with proper behaviour (a) Water NSQ benchmark (b) Volrend, the power reference is set to 7W

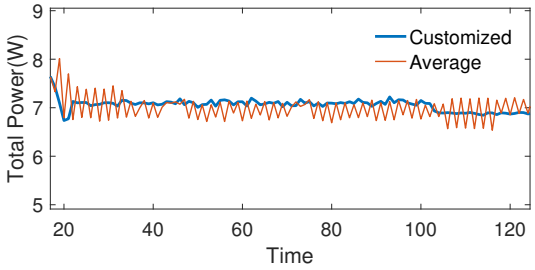


Fig. 7: Fmm benchmark with average and customized case

B. Performance Analysis

After the system identification stage, controller design is performed for instance by using the Matlab PI tuner [26]. 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. For our workloads this average is represented with $K_P = 180$ and $K_I = 240$. 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 III shows these workload specific K_P and K_I 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 suffering from slow settling time. In our experiments, the CPU-bound benchmark shows the most conservative behaviour with $K_P = 77$ and $K_I = 154$ control gains. It should be mentioned that despite all these methods and vast variety for 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 such scenarios below.

Workload	K_P	K_I
Barnes	114	229
Ocean-Contiguous	156	226
Ocean-Nocontiguous	110	363
FMM	114	229
Radiosity	184	369
Raytrace	244	247
Water-NSQ	139	228
Water-SP	175	250
Volrend	141	282
Average	180	240

TABLE III: CPU core configuration for Nehalem-EP

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 III shows these workload specific K_P and K_I configurations. Figure 6 shows proper behavior of the *Water_nsq* and *Volrend* benchmarks 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. We observe similar trends for all other workloads except the two benchmarks discussed in Section IV-B3.

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 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 [19] 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 7 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 8 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.

3) Corner case: So far we evaluated both dynamic and static methods to design and deploy a PI controller for power management. 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

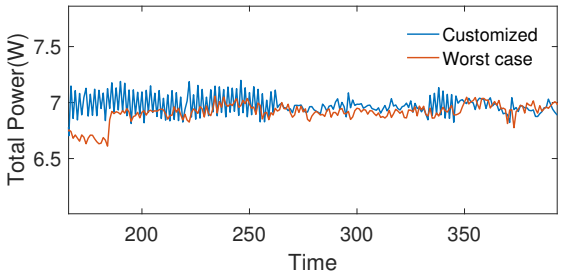
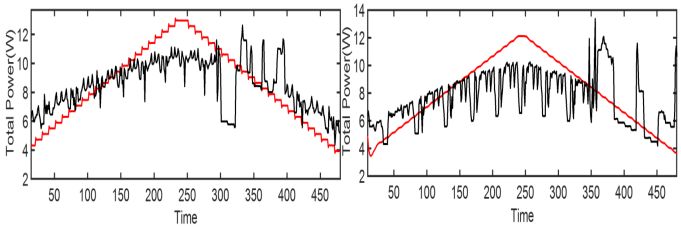


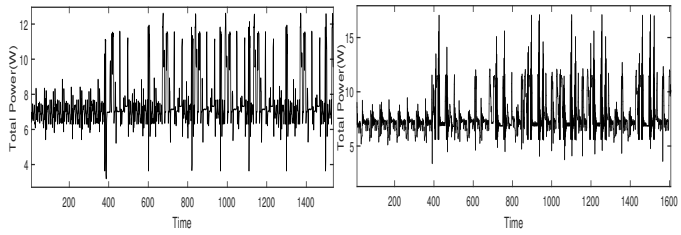
Fig. 8: Raytrace benchmark with average and customized case

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 9 and 10 show the system identification, and controller deployment phases of the system running two benchmarks (i.e., *Ocean* with contiguous and noncontiguous workloads) that are not controllable using solely DVFS. In the following section, we analyze these two benchmarks in more detail to elaborate the reasons behind their abnormal behaviour.



(a) Ocean Contiguous workload (b) Ocean Nocontiguous workload

Fig. 9: System identification of uncontrollable workloads



(a) Ocean Contiguous workload (b) Ocean Nocontiguous workload

Fig. 10: Performance analysis of uncontrollable workloads while trying to track 7 Watts reference

V. Discussion

In this section, we discuss the reliability and performance of SISO controllers in power management 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 in the SPLASH2 benchmark suite (all except the two Ocean workloads). In addition, hand tuned controllers were able to meet the second set of requirement 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 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 two benchmarks that exhibited abnormal behaviour in tracking power references with high standard deviation. Figure 10 shows the behaviour of these two benchmarks. Our first reasoning behind this behaviour 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 two 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.

Our next solution to this issue was to investigate these two benchmarks in more detail. We looked at a few measurable metrics and what stood out was the average power consumption. The results in Table IV shows the average power consumption of each benchmark while tracking 7-Watt power reference. As we can see, only the two irregular benchmarks (Ocean-Contiguous and Ocean-Nocontiguous) 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 behaviour 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 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 IV. 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 behaviour 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 behaviour 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. Such cases either require more advanced controllers (e.g., non-linear, adaptive, self-tuning) or different/more configuration knobs such as memory bandwidth that can effect the system's power more efficiently.

VI. Related Work

There is a large body of literature on heuristic-based approaches for resource management [7], [8], [9], [10]. There have been a variety of techniques that are used for power and resource management in processors. Heuristics such as [1], [3], [4] uses a rule-based method in contrast to threshold methods [5], [6]. Predictive methods [11], [12], [13] that typically benefit from nested loops have been used to manage

Workload	Average power (Watts)	IPS
Barnes	6.3289	2.21E+09
Ocean-Contiguous	7.8306	1.07E+09
Ocean-Nocontiguous	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 IV: Comparison of average power and IPS

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 [15], [16], [18], [19], [20], [21] that provide formal and efficient means to address robustness and testability for managing computer systems. Most these methods target a specific class of applications. In this work we analyze an ample set of workloads in detail to point out benefits and shortcomings of SISO controllers for power management. We present guidelines that can help designers select the proper configuration for their system controller.

VII. 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 management schemes that provide guarantees on the system's efficiency and dependability. One of the viable solution to this issue is the use of control-theoretic methods with formal guarantees. We experimentally 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. Moreover, an in-depth analysis has been performed on multiple controller configurations using SPLASH2 benchmark suite.

Acknowledgment

We acknowledge financial support by the Marie Curie Actions of the European Union's H2020 Programme.

References

[1] 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.

[2] 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.

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

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

[5] 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).

[6] 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).

[7] 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.

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

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

[25] 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.

[26] 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"

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