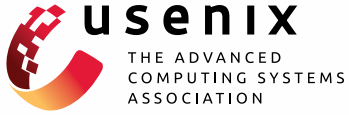

PCP: A Generalized Approach to Optimizing Performance Under Power Constraints through Resource Management

Henry Hoffmann, ***University of Chicago;*** **Martina Maggio,** ***Lund University***

https://www.usenix.org/conference/icac14/technical-sessions/presentation/hoffman

This paper is included in the Proceedings of the 11th International Conference on Autonomic Computing (ICAC '14).

June 18–20, 2014 • Philadelphia, PA

ISBN 978-1-931971-11-9

Open access to the Proceedings of the 11th International Conference on Autonomic Computing (ICAC '14) is sponsored by USENIX.

PCP: A Generalized Approach to Optimizing Performance Under Power Constraints through Resource Management

Henry Hoffmann
University of Chicago, Chicago USA

Martina Maggio
Lund University, Lund Sweden

Abstract

Many computing systems are constrained by power budgets. While they could temporarily draw more power, doing so creates unsustainable temperatures and unwanted electricity consumption. Developing systems that operate within power budgets is a constrained optimization problem: configuring the components within the system to maximize performance while maintaining sustainable power consumption. This is a challenging problem because many different components within a system affect power/performance tradeoffs and they interact in complex ways. Prior approaches address these challenges by fixing a set of components and designing a power budgeting framework that manages only that one set of components. If new components become available, then this framework must be redesigned and reimplemented. This paper presents PCP, a general solution to the power budgeting problem that works with arbitrary sets of components, even if they are not known at design time or change during runtime. To demonstrate PCP, we implement it in software and deploy it on a Linux/x86 platform.

1 Introduction

Modern computing systems are constrained by *dark silicon*, the abundance of transistors enables processors to draw more power than they can safely sustain [9, 39]. For example, the Exynos 5 processor (in the Samsung Galaxy S4 phone) has a 5.5W peak power that is nearly $2\times$ the maximum sustainable heat dissipation, limiting peak speed to less than 1 second [36]. At the other end of the spectrum, the next generation of exascale supercomputers is predicted to be constrained by an operating budget of approximately 20 MW [2]. In addition, Microsoft was recently fined for not using *enough* power and violating an agreement with a utility company [12]. Executing in these systems requires solving a constrained optimization problem: maintaining the power budget, while maximizing performance within the power constraint.

Many separate components contribute to total power consumption and various techniques have been proposed to manage individual components. For example, management systems exist for core allocation [27], dynamic voltage and frequency scaling (DVFS) [30, 42], processor idling [11], cache [1], DRAM [7, 47], and disk [25].

Of course, different applications have different needs and coordinated management of several components provides better performance within a power budget [19, 29]. Examples that coordinate components include those that handle cores and DVFS [33, 45], DVFS and memory speed [6, 10, 26], and thread scheduling and DVFS [44]. Unfortunately, prior multi-component management approaches are not general in terms of the components under control. Instead, they *fix* a specific set of components, whose interactions are known at design time and they do not permit this set of components to change. If new components become available or existing components are disabled, these power management approaches will either (1) deliver poor performance or (2) require redesign and reimplementation. Thus, *there is a need for a general approach to multi-component power management that continues to deliver maximum performance for a given power budget even as new components become available or existing components are disabled.*

Challenges

A generalized power management system must address three challenges:

Unknowns: Prior approaches rely on rigorous models for either the specific machine under control (*e.g.,* a particular smartphone [37]) or for a specific application and platform (*e.g.,* a web browser on a mobile device [38]). A generalized power management system, however, must either construct its models on the fly or compensate for inaccuracies and unknowns in general models.

Interaction: System components interact to produce a

complex (often nonlinear) effect on power and performance. If individual components are controlled separately, their interaction can lead to suboptimal behavior, even when these separate controllers are individually optimal [16]. Thus, a generalized power management system must coordinate all available components even if they are not known at design time or vary at runtime.

Optimization: A power manager must not exceed the power budget, yet must also deliver the best possible performance for a given budget. A generalized approach must not sacrifice too much performance for generality.

Contributions

This paper addresses the above challenges to produce PCP, short for Power Constraints with maximum Performance, a machine-level power management system that is general with respect to the components it manages. PCP uses feedback control to ensure the power budget is not exceeded. The controller is augmented with an *estimator*, which addresses the challenge of unknowns (see Section 3.3), and a *translator*, which addresses the challenges of interaction and optimization (see Section 3.4). The estimator dynamically tailors control to a particular application, the controller then produces a generic control signal, and the translator turns this signal into a configuration of available components that meets the power budget while providing close to maximum performance.

We implement PCP and test it on a Linux/x86 system. The results show that PCP is:

General: PCP has low overhead (as shown in Section 4.2). Furthermore, components can be added and removed at runtime and PCP automatically adapts to the new conditions. (See Section 4.3).

Accurate: PCP meets the power budgets with low errors. Across all machines and all benchmarks, PCP keeps the average power within 2% of the budget. PCP maintains good results across a range of power budgets from small to large. For small budgets (5% of maximum power), PCP's average error is slightly over 1% of the budget. For all other targets, PCP's average error is less than 1%. (See Section 4.4).

Efficient: Given a power budget, PCP achieves close to the maximum possible performance. We compare PCP to an oracle and find it achieves close to optimal performance for a range of power targets. For the smallest target, the average performance is 92% of the oracle. For all other targets, performance is greater than 95% of the oracle. (See Section 4.5).

The paper is organized as follows. Section 2 compares PCP with prior research. Section 3 discusses PCP's design. Section 4 evaluates PCP, providing experimental evidence for our claims. Finally, Section 5 concludes the paper.

2 Related Work

We describe related work building systems that solve constrained optimization problems in the power and performance dimensions.

Some approaches provide performance guarantees and minimize power consumption. Examples exist at the datacenter-level [21, 40], and machine-level. At the machine-level, techniques provide performance and minimize power by managing DVFS in the processor [45], assignment of cores to an application [27], caches [1], DRAM [47], and disks [25]. Other approaches coordinate multiple components within a machine. For example, Li et al. manage memory and processor [26], while Dubach et al. demonstrate a method for coordinating a large collection of microarchitectural features [8]. Bitirgen et al. coordinate DVFS, cache, and memory bandwidth [4]. METE is an adaptive control system which manages cores, DVFS, and off-chip bandwidth to provide performance guarantees [35]. Gu and Nahrstedt provide a general approach to guarantee performance for multimedia applications in a distributed environment [14]. Hoffmann et al. provide a general technique for coordinating components to meet performance constraints [20]. These solutions provide performance guarantees, but do not guarantee power consumption.

Other approaches guarantee power consumption while maximizing performance subject to this constraint. Such techniques are important for avoiding power overload [9, 39]. Datacenter-level solutions include those proposed by Wang et al. [41] and Raghavendra et al. [31]. These solutions are hierarchical and require some machine-level power management scheme. Machine-level systems for guaranteeing power have been developed to manage DVFS for a processor [23], per-core DVFS in a multicore [22], processor idle-time [11], and DRAM [7]. Other approaches coordinate management of multiple components including processor and DRAM [6, 10], processors and core allocation [33], and combining DVFS and thread scheduling [32, 44].

Like many of the above approaches, PCP uses control theory to manage the dynamic behavior of the machine. Control theory is a general *technique* [15], but implementations are often specific to a particular machine. Some prior approaches provide generality by implementing control systems at the middleware layer [13, 24, 46]. The middleware handles the complicated construction of the control system, but the application writer must be aware of the components on the system. PCP differs as it does not require application programmer input at all.

PCP is a novel machine-level power control approach that coordinates multiple components. PCP is distinguished by the fact that the components it manages are not known at design time and may change during run-

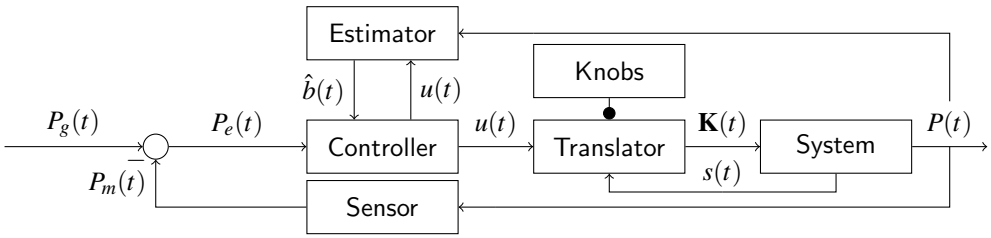

Figure 1: Block diagram representing the overall control and scheme.

time.

3 Runtime Control Framework

The PCP runtime is illustrated in Figure 1. The runtime begins with a simple model of the System under control, including the available components, or Knobs, affecting power and performance. A Controller measures power consumption and produces a generic control signal indicating available overhead in the power budget. An Estimator dynamically tunes the controller to the specific application and system under control and accounts for approximations in the initial model. A Translator turns the generic control signal into a performance-optimal component configuration that achieves the power budget. Each of these modules is described below.

3.1 System

The System block in Figure 1 describes available system components, from the controller's perspective. PCP uses a discrete time, linear model of the system where the index t measures time (in discrete intervals). The system has n separate components, or *knobs*, that affect power and performance. The system's input is a *configuration*, or vector of values $\mathbf{K}(t) \in \mathbb{R}^n$, each of which represents a setting for one of the knobs. The system has a *baseline* power consumption b representing the lowest possible power. We denote the system's power consumption as $P(\mathbf{K}(t))$, measured as a proportion of b. The speed of the system is $S(\mathbf{K}(t))$. The system's outputs are the current *power consumption* $P(t)$ and the current *performance* measure $s(t)$ (for speed). $P(t)$ is the average power consumption in the interval between $t-1$ and t:

$$P(t) = bP(\mathbf{K}(t-1)) \tag{1}$$

This simple linear model is used to derive a controller in Section 3.2. The estimator (Section 3.3) compensates for the inherent approximations in the model.

PCP requires power and performance feedback, but is agnostic about the source. Power feedback can come from a model or direct power measurements. Performance feedback can come from any number of appropriate hardware counters, the only requirement is that the metric increase with increasing speed.

3.2 Controller

The Controller in Figure 1 eliminates the power error $P_e(t) = P_g(t) - P_m(t)$, between the power budget, or goal $P_g(t)$, and the actual measured power $P_m(t)$. To do so, the controller computes a control signal. In a typical controller, this signal would directly specify settings for the available knobs $\mathbf{K}$; however, such an approach does not generalize because it ties the controller's design directly to the system under control. Instead, PCP computes a generic control signal $u(t)$, independent of the specific knobs in the system. $u(t)$ represents an allowable power consumption over the baseline b; *i.e.,* $u(t) = P(\mathbf{K}(t))$. For example, $u(t) = 1.5$ indicates that the controller allows 50% greater power consumption than baseline. The translator maps this control signal into knob settings (see Section 3.4). Therefore, given $P_e(t)$ and b at time t, the controller calculates a new signal $u(t)$. This generic signal is derived following standard practice resulting in the Integral controller:

$$u(t) = u(t-1) + P_e(t)/b \tag{2}$$

3.3 Estimator

The b parameter has a profound effect on PCP's behavior. Although PCP models it as a constant, we know that its true value varies as a function of the hardware and the application (or even phases within the application). Therefore, b represents a key *unknown* corresponding to the first challenge listed in Section 1. The Estimator in Figure 1 is responsible for overcoming this unknown by continually estimating its true value as $\hat{b}(t)$. Adding the estimator makes the control scheme *adaptive*, in the sense that it automatically adjusts, not only to the feedback, but also to varying operating points and unknowns.

PCP estimates b using a Kalman filter [43] based on the time-varying model:

$$\begin{aligned} b(t) &= b(t-1) + \delta b(t) \\ P(t) &= u(t-1)\,b(t-1) + \delta P(t) \end{aligned} \tag{3}$$

which describes the variation of b and P subject to both disturbance and noise (respectively δb and δP).

Denoting the system power variance and measurement variance as $q_b(t)$ and r_b, the Kalman filter formulation is

$$\begin{aligned} \hat{b}^-(t) &= \hat{b}(t-1) \\ e_b^-(t) &= e_b(t-1)+q_b(t) \\ k_b(t) &= \frac{e_b^-(t)\,u(t)}{[u(t)]^2\,e_b^-(t)+r_b} \\ \hat{b}(t) &= \hat{b}^-(t)+k_b(t)\,[P(t)-u(t)\hat{b}^-(t)] \\ e_b(t) &= [1-k_b(t)\,u(t-1)]\,e_b^-(t) \end{aligned} \tag{4}$$

where $k_b(t)$ is the Kalman gain for the system power consumption, $\hat{b}^-(t)$ is the *a priori* estimate of $b(t)$ and $\hat{b}(t)$ is the *a posteriori* one, and $e_b^-(t)$ is the *a priori* estimate of the error variance while $e_b(t)$ is the *a posteriori* one. PCP uses a Kalman filter because it produces a statistically optimal estimate of the system's parameters, and is provably exponentially convergent [5]. Note that the only required parameter is r_b, the measurement noise, which is a property of the feedback mechanism.

3.4 Translator

The Translator in Figure 1 converts the generic control signal $u(t)$ into a component configuration $\mathbf{K}(t)$ at time t. The control signal is a continuous signal which must be translated into discrete knob settings. This problem is challenging because any given power signal may be achieved by multiple combinations of components. The Translator must find the combination that meets the budget while maximizing performance.

PCP schedules configurations over the τ time units between now and the next power measurement. PCP assigns τ_c time units to each configuration where $\sum_c \tau_c = \tau$. The translator schedules these configurations to maximize performance while guaranteeing the power budget, specified by the control signal, is not exceeded. Denoting a configuration of knobs as $K_c(t)$ with corresponding speed $S(K_c(t))$ and power $P(K_c(t))$, then PCP solves the following optimization problem:

$$\text{maximize} \quad \sum_c \tau_c \cdot S(K_c(t)) \tag{5}$$

$$\text{subject to} \quad \sum_c \tau_c \cdot P(K_c(t))/\hat{b}(t) \leq u(t) \tag{6}$$

$$\sum_c \tau_c = \tau \tag{7}$$

$$t \geq \tau_c \geq 0, \quad \forall\, c \tag{8}$$

Equations 5–8 assign values for all τ_c to maximize performance (Equation 5) subject to the constraints that the power signal is not exceeded (Equation 6). The final two constraints (Equations 7 and 8) ensure that the time spent in all configurations is non-negative and does not exceed the time of the next measurement. This type of optimization problem is similar to others that have arisen for real-time systems which minimize energy while meeting a performance constraint. Several heuristic solutions have been developed that apply to both performance and power constraints and provide near-optimal behavior in practice [17].

We note that as components enter and leave the system, the power and performance of various configurations may change. This is especially true if different groups of components interact in non-linear ways. PCP accounts for this variability by continually estimating $S(K_c(t))$ and $P(K_c(t))$ online. Both quantities are estimated using Kalman filter formulations similar to that of Equation 4 (these formulations are omitted for space). This estimation allows PCP to adapt to changing sets of components at runtime. In practice, PCP first estimates the performance and power consumption of the previous configuration using Kalman filters, and then solves Equations 5–8 using the new estimate to determine the resource allocation for the next time step.

3.5 Summary

PCP meets power budgets while optimizing performance. Its generalized system model is independent of any particular set of hardware components. PCP's controller ensures that power budgets are met. The estimator overcomes the challenge of handling unknowns, while the translator handles the challenges of component interaction and performance optimization. PCP can incorporate new knobs as they become available at runtime, and it can be deployed on new systems without redesign.

4 Evaluation

4.1 Experimental Setup

We describe the systems and applications used in our evaluation.

Systems

To demonstrate PCP, we deploy it on a real Linux/x86 system, a single socket Intel E5-1650 with 6 cores, 1 memory controller and 12 DVFS settings from 1.2 — 3.2 GHz. It supports hyper-threading and TurboBoost. In addition, it allows suspension of the current application to enter a low-power idle state. Idling consumes 85% of the lowest measured active power (85W to 100W). The highest measured power consumption is 235W. To measure power consumption the machine supports hardware power measurement at 1 ms intervals [34].

Benchmarks

Our benchmarks consist of the 13 PARSECs [3] plus Dijkstra and STREAM [28]. PARSEC has a mix of im-

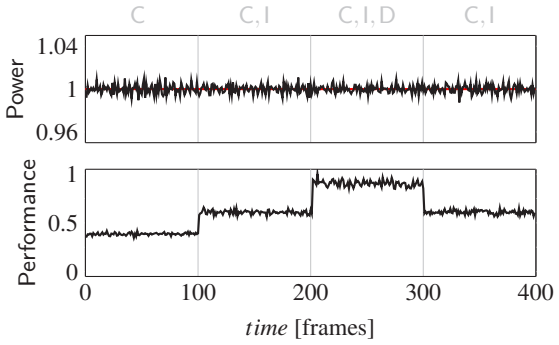

Figure 2: Experiment changing actuators.

portant, emerging multicore workloads. Dijkstra is a parallel implementation of single-source shortest paths on a large, dense graph developed for this paper. STREAM tests memory performance. These last two benchmarks test PCP's ability to handle resource limited applications. Dijkstra has some parallelism, but does not scale well. STREAM is generally memory limited, but needs sufficient compute resources to saturate the available memory bandwidth.

Although PCP supports a range of performance feedback metrics (see Section 3), in this paper we measure application performance as instructions per second and collect this data with hardware performance counters. In general, however, PCP could work with any performance metric that increases with increasing application speed and decreases as application performance decreases; *e.g.,* application specific performance metrics [18].

4.2 Overhead

We evaluate PCP's overhead by running all benchmark applications with PCP, but disabling its ability to actually change component settings. Thus, the runtime executes and consumes resources, but cannot have any positive effect. We compare execution time and power consumption to the application running without our runtime. For most applications, there is no measurable difference. The largest difference in performance is 2.2%, measured for the *fluidanimate* benchmark, while the power consumption is within the run to run variation for all benchmarks ($< 1\%$). We find this overhead acceptable. All measurements in this section include the overhead and impact of the runtime system.

4.3 Changing Actuators

To demonstrate the claim of generality, *we add and remove components at runtime and demonstrate that PCP system automatically adapts to the new conditions*. We run the raytrace benchmark for 400 time units, where each unit is a frame. During execution, the set of available knobs changes. We distinguish between four different intervals of 100 time units. During the first interval, PCP only adjusts the number of cores. During the second interval, idle time becomes available. In the third interval, DVFS is also available. In the last interval, DVFS is disabled (causing the performance loss that occurs at time 300).

Figure 2 shows the results. The power consumption is very close to the setpoint (1 in the figure); the percentage difference is less than 1%. As different actuators become available, PCP takes advantage of them to improve performance without exceeding the power budget. The figure shows performance normalized to the maximum achievable for the power budget.

4.4 Stability and Accuracy

We demonstrate PCP's stability and accuracy by running each benchmark on our target system and telling PCP to maintain a power budget. For this experiment, we set the power budget halfway between the minimum and maximum achievable power. This middling power budget is one of the toughest to meet in practice because there are many combinations of components which can achieve the same power. PCP must determine the highest performance configuration at runtime.

We quantify the stability and accuracy of PCP by measuring the power consumption at each control interval and calculating the mean absolute percentage error (MAPE):

$$MAPE = \frac{1}{n}\sum_{k=1}^{n} \begin{cases} p_m(k) > p_\ell : & \left|\frac{p_m(k)-p_\ell}{p_m(k)}\right| \cdot 100\% \\ p_m(k) \leq p_\ell : & 0 \end{cases} \quad (9)$$

Low MAPE (expressed as a percentage) indicates that the power budget is stable and at or below the desired value. Note that we do not penalize PCP for operating below the power budget, but this may result in suboptimal performance. We quantify performance in the next section. Finally, when calculating MAPE, we subtract idle power from the measured power. Doing so quantifies the contribution of the control system to accuracy and stability. Leaving the idle power in the calculation would result in lower MAPE values.

The results are illustrated in Figure 3. The figure shows each benchmark on the x-axis and the MAPE (as a percentage) on the y-axis. The measured errors are quite low in general, and all are under 8%. These results indicate that PCP is stable and accurate across a range of benchmark applications. Those benchmarks with higher MAPE tend to have multiple phases of computation and the errors occur when PCP is optimizing

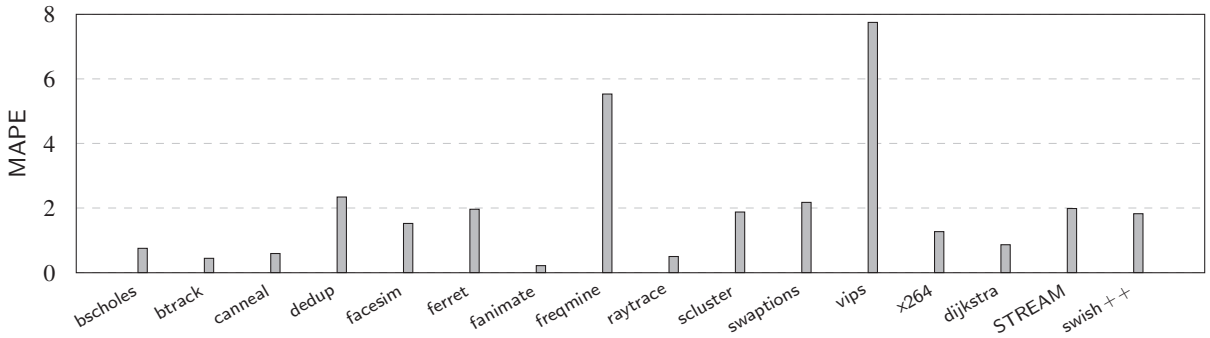

Figure 3: Mean average percentage error.

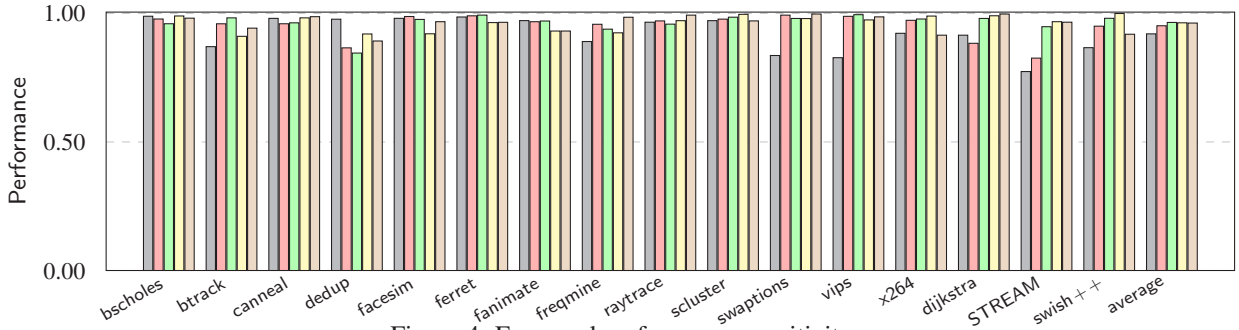

Figure 4: Error and performance sensitivity.

across phases. Even for these applications, however, PCP produces good results, which are consistent with the low observed error in Figure 2.

4.5 Error and Performance Sensitivity

We further test PCP's efficiency by measuring the sensitivity of performance to the power target. We run each of our benchmarks with 5 different power targets. Each power target is x% between the minimum and maximum achievable where $x \in \{5, 25, 50, 75, 95\}$. For each power target, we deploy the application on the machine and PCP adjusts resource usage to keep power consumption at or below the target. We measure the error and the performance for each target and report results in Figure 4.

The figure shows the normalized performance (lower plot) on the respective y-axes for each of the 5 power targets. Benchmark name is shown on the x-axis. The results demonstrate PCP's efficiency across a range of power targets. In general, the lowest performance is achieved for the the 5% target; however, the average performance of 0.92 is still close to optimal. All other targets have an average performance greater than 0.95. The worst single achieved performance comes from STREAM at the 5% target, which has a normalized performance of 0.77. This low result comes from the time taken to learn to allocate STREAM the minimum resources on this machine. The near-optimal performance across a range of benchmarks and power targets demonstrate PCP's efficiency.

Although the accuracy results are omitted for space, they are consistent with those reported in Figure 3. The highest MAPE values are achieved for the 50% target. All other targets are lower in general.

5 Conclusion

This paper presents PCP, a power budgeting system, which guarantees power consumption while maximizing performance subject to the power budget. PCP's distinguishing feature is its generality; *i.e.,* the design is independent of any particular set of components that could be used to manage power/performance tradeoffs. PCP overcomes the four challenges of handling dynamics, accounting for unknowns, managing component interaction, and optimizing performance. We have implemented PCP and deployed it on several hardware platforms and demonstrated its generality, portability, accuracy, and performance. Widespread adoption of PCP would make it easy to quickly develop power management systems as new hardware components become available.

Acknowledgements

Henry Hoffmann is grateful for support from the U.S. Dept. of Energy (under DOE DE-AC02-06CH11357).

References

[1] R. Balasubramonian et al. “Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures”. In: *MICRO*. 2000.

[2] K. Bergman et al. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead*. 2008.

[3] C. Bienia et al. “The PARSEC Benchmark Suite: Characterization and Architectural Implications”. In: *PACT*. 2008.

[4] R. Bitirgen et al. “Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach”. In: *MICRO*. 2008.

[5] L. Cao and H. M. Schwartz. “Exponential convergence of the Kalman filter based parameter estimation algorithm”. In: *Intl. Journal of Adaptive Control and Signal Processing* 17.10 (2003).

[6] J. Chen and L. K. John. “Predictive coordination of multiple on-chip resources for chip multiprocessors”. In: *ICS*. 2011.

[7] B. Diniz et al. “Limiting the power consumption of main memory”. In: *ISCA*. 2007.

[8] C. Dubach et al. “A Predictive Model for Dynamic Microarchitectural Adaptivity Control”. In: *MICRO*. 2010.

[9] H. Esmaeilzadeh et al. “Dark silicon and the end of multicore scaling”. In: *ISCA*. 2011.

[10] W. Felter et al. “A performance-conserving approach for reducing peak power consumption in server systems”. In: *ICS*. 2005.

[11] A. Gandhi et al. “Power capping via forced idleness”. In: *Workshop on Energy-Efficient Design*. Austin, TX, 2009.

[12] J. Glanz. “Data Barns in a Farm Town, Gobbling Power and Flexing Muscle”. In: *The New York Times* (September 23, 2012).

[13] A. Goel et al. “SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit”. In: *2nd USENIX Windows NT Symposium*. 1998.

[14] X. Gu and K. Nahrstedt. “Dynamic QoS-aware multimedia service configuration in ubiquitous computing environments”. In: *ICDCS*. 2002.

[15] J. L. Hellerstein et al. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[16] J. Heo and T. Abdelzaher. “AdaptGuard: guarding adaptive systems from instability”. In: *ICAC*. 2009.

[17] H. Hoffmann. “Racing vs. Pacing to Idle: A Comparison of Heuristics for Energy-aware Resource Allocation”. In: *HotPower*. 2013.

[18] H. Hoffmann et al. “Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments”. In: *ICAC*. 2010.

[19] H. Hoffmann et al. “Self-aware computing in the Angstrom processor”. In: *DAC*. 2012.

[20] H. Hoffmann et al. “A Generalized Software Framework for Accurate and Efficient Managment of Performance Goals”. In: *EMSOFT*. 2013.

[21] T. Horvath et al. “Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control”. In: *Computers, IEEE Transactions on* 56.4 (2007).

[22] C. Isci et al. “An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget”. In: *MICRO*. 2006.

[23] C. Lefurgy et al. “Power capping: a prelude to power shifting”. In: *Cluster Computing* 11.2 (2008).

[24] B. Li and K. Nahrstedt. “A control-based middleware framework for quality-of-service adaptations”. In: *IEEE Journal on Selected Areas in Communications* 17.9 (1999).

[25] X. Li et al. “Performance directed energy management for main memory and disks”. In: *Trans. Storage* 1.3 (2005).

[26] X. Li et al. “Cross-component energy management: Joint adaptation of processor and memory”. In: *ACM Trans. Archit. Code Optim.* 4.3 (2007).

[27] M. Maggio et al. “Controlling software applications via resource allocation within the Heartbeats framework”. In: *CDC*. 2010.

[28] J. D. McCalpin. “Memory Bandwidth and Machine Balance in Current High Performance Computers”. In: *IEEE TCCA Newsletter* (1995).

[29] D. Meisner et al. “Power management of online data-intensive services”. In: *ISCA* (2011).

[30] T. Pering et al. “The simulation and evaluation of dynamic voltage scaling algorithms”. In: *ISLPED*. 1998.

[31] R. Raghavendra et al. “No ”power” struggles: coordinated multi-level power management for the data center”. In: *ASPLOS*. 2008.

[32] K. K. Rangan et al. “Thread motion: fine-grained power management for multi-core systems”. In: *ISCA*. 2009.

[33] S. Reda et al. “Adaptive Power Capping for Servers with Multithreaded Workloads”. In: *Micro, IEEE* 32.5 (2012).

[34] E. Rotem et al. “Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge”. In: *Hot Chips*. 2011.

[35] A. Sharifi et al. “METE: meeting end-to-end QoS in multicores through system-wide resource management”. In: *SIGMETRICS*. 2011.

[36] Y. Shin et al. “28nm High- Metal-Gate Heterogeneous Quad-Core CPUs for High-Performance and Energy-Efficient Mobile Application Processor”. In: *ISSCC*. 2013.

[37] A. Shye et al. “Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures”. In: *MICRO*. 2009.

[38] N. Thiagarajan et al. “Who killed my battery?: analyzing mobile browser energy consumption”. In: *WWW*. 2012.

[39] G. Venkatesh et al. “Conservation cores: reducing the energy of mature computations”. In: *ASPLOS*. 2010.

[40] A. Verma et al. “Server workload analysis for power minimization using consolidation”. In: *USENIX Annual technical conference*. 2009.

[41] X. Wang et al. “MIMO Power Control for High-Density Servers in an Enclosure”. In: *IEEE Transactions on Parallel and Distributed Systems* 21.10 (2010).

[42] M. Weiser et al. “Scheduling for reduced CPU energy”. In: *Mobile Computing* (1996).

[43] G. Welch and G. Bishop. *An Introduction to the Kalman Filter*. Tech. rep. TR 95-041. UNC Chapel Hill, Department of Computer Science, 2006.

[44] J. A. Winter et al. “Scalable thread scheduling and global power management for heterogeneous many-core architectures”. In: *PACT*. 2010.

[45] Q. Wu et al. “Formal online methods for voltage/frequency control in multiple clock domain microprocessors”. In: *ASPLOS*. 2004.

[46] R. Zhang et al. “ControlWare: A middleware architecture for Feedback Control of Software Performance”. In: *ICDCS*. 2002.

[47] H. Zheng et al. “Mini-rank: Adaptive DRAM architecture for improving memory power efficiency”. In: *MICRO*. 2008.