Node-level Coordinated Power-Performance Management

Rathijit Sen David A. Wood
Department of Computer Sciences
University of Wisconsin-Madison

Abstract

Pareto-optimal power-performance system configurations can improve performance for a power budget or reduce power for a performance target, both leading to energy savings. We develop a new power-performance management framework that seeks to operate at the pareto-optimal frontier for dynamically reconfigurable systems, focusing on single-node reconfigurations in this paper.

Our framework uses new power-performance state descriptors, Π*-states, that improve upon the traditional ACPI-state descriptors by taking into account interactions among multiple reconfigurable components, quantifying system-wide impact and predicting totally ordered pareto-optimal states.*

We demonstrate applications of our framework for both simulated systems and real hardware. In simulation, we demonstrate (near-)optimal configurations in the presence of a number of reconfigurable options such as core frequency, LLC capacity, offchip bandwidth and memory frequency. On an existing Intel Haswell single-socket server machine, we achieve 48% (average) improvements in energy efficiency using only core frequency management.

1. Introduction

Today, power and energy are among the most critical constraints for system design and use [34, 10]. While compute capability, in terms of number of transistors per chip, has steadily increased (Moore's Law), operating voltage has not reduced in proportion (limited Dennard scaling). This in turn has the unfortunate impact of increasing total energy per computation as well as resource idle/leakage power. With systems getting larger as we step into the ExaScale era, managing power/energy consumption while meeting performance demands is becoming more critical than ever before.

An ideal system would be energy-proportional [10], that is, it would use energy in proportion to the work done. However, workloads vary in their resource requirement characteristics – they may be compute-intensive or cache-intensive or memory-intensive and not necessarily exclusively so. Thus, effectively managing system power/energy consumption requires the ability to dynamically reconfigure resources according to computational requirements. Decades of architectural and circuits research has resulted in a multitude of runtime configuration options, or "knobs" for individual system components. Dynamic voltage and frequency scaling (DVFS) and dynamic frequency scaling (DFS) for cores are well-known techniques [5, 42]. Isci et al. [27] showed that a global power manager can improve performance for a power budget using per-core DVFS as reconfiguration knobs.

Apart from cores, cache and memory also consume significant power and need to be managed for power/energy-efficient performance. Intel's Core Duo processor allows shutting down portions of its last-level cache (LLC) [35]. Intel's Ivy Bridge microarchitecture provides more fine-grained control: a subset of the ways of its set-associative LLC can be turned off [28]. IBM's Power7 enables runtime throttling of memory traffic [44]. Felter et al. [23] demonstrated benefits through power-shifting using core DVFS and memory throttling. Deng et al. [21] and David et al. [19] proposed using DFS/DVFS for main memory. Moreover, to avoid inconsistent/under-performing configurations, resource management must be coordinated. Deng et al. [20] demonstrated coordinated management of core DVFS and memory DVFS.

Resource heterogeneity – a core resource is different from a cache or memory resource – complicates resource management. The Advanced Configuration and Power Interface (ACPI) specification [6] is an open standard that allows devices (resources) to specify discrete operating states identified by alphanumeric names. For example, P0, P1, P2,... represent processor performance states. A central thesis of this work is that *individually ordered lists of operating states for different resources do not identify ordering for combinations of states across resources, required for system-level coordinated management*. ACPI enumerations lack quantification of system-wide power-performance impacts by not accounting for inter-resource interactions or dynamic execution profiles.

In this work we propose a *framework* to address this problem. We use power-performance predictors, using existing and proposed hardware counters, to characterize the expected impact of different configurations and subsequently identify pareto-optimal configurations. We call these augmented state descriptors Π-states. Π-states for our single-node system are computed dynamically from the Π-states of its constituent resources/subsystems. In contrast with ACPI-like state descriptions, Π-states include additional information – slowdown, static power, dynamic energy, amount of work – to "stitch together" component Π-states to form system-wide Π-states.

Our system predicts and exposes a "Π-dashboard" of totally ordered, pareto-optimal Π-states from which the user or operating system selects a desired power-performance profile that causes a "one-shot" transition of the system to the corresponding configuration. The dashboard is updated periodically as execution profiles change over time. Table 2 shows sample predicted Π-dashboards for blackscholes and jbb. The dashboard is constructed from (a subset of) the states lying on the predicted pareto-optimal frontier. Looking ahead, Figure 9 shows the predicted frontiers for a few workloads.

A well-defined coordination interface enhances reusability by separating coordinator implementation from impact-predictor implementation. It also avoids requiring third-party vendors to disclose intellectual property for their specific knobs. Π-states abstract away the implementation details and expose only the (predicted) power-performance impacts of alternate resource configurations. While ACPI-like descriptors are also abstract power-performance descriptors for individual resources, ACPI lacks a general mechanism to "stitch together" the component states at the system level thereby encouraging system-specific solutions for given power-performance objectives.

The main contributions of this work are:

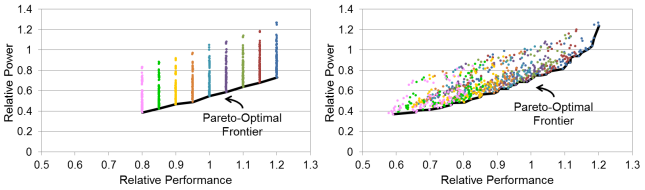
1. We introduce new power-performance state descriptors, Π-states, that enable *runtime identification of the system pareto-optimal frontier*.
2. We propose an user/operating-system interface consisting of a dynamically updating Π-dashboard of predicted pareto-optimal Π-states from which the desired operating point can be selected. This helps to *bridge the gap between high-level power-performance goals and system resource configurations.*
3. We demonstrate 48% (average) improvement in energy efficiency on an existing Intel Haswell server machine using only core frequency management.
4. Contrary to convention, we show that *rapid profiling and reconfiguration are neither useful nor necessary* for long-running workloads. Our best reactive policy reconfigures once every 510 ms.
5. In simulation, we demonstrate coordinated power-performance management involving core DVFS, LLC reconfiguration, offchip bandwidth allocation and memory DVFS. To the best of our knowledge, *no other work has studied coordinated management of these four different types of resources simultaneously.*

The rest of this paper is organized as follows: Section 2 further motivates this work with an example; Section 3 describes the coordination framework; Section 4 demonstrates management of a Haswell server machine for socket frequency settings; Section 5 generalizes Π-states. Section 6 describes a full-system simulation study for knobs such as LLC capacity, offchip bandwidth and memory DVFS[1] that are currently *not publicly available on real machines*; Section 7 summarizes related work; Section 8 concludes the paper.

2. Motivating Example

Table 1 summarizes the knobs that we consider in this paper. We introduce new names for resource operating states, but retain the ACPI convention of alpha-numeric naming. Also, similar to ACPI orderings, state 0 for each knob has the maximum performance capability for that knob and subsequent

[1]The real machines we study allow DRAM frequencies to be selected through the BIOS at boot time, but not through dynamic control.



(a) blackscholes (b) jbb

Figure 1: (Best viewed in color.) Example knob distribution by impact over the state space (810 states). Configuration P4-C0-S0-M2 is at (1,1). The color coding P0 P1 P2 P3 P4 P5 P6 P7 P8 identifies onchip DVFS settings – states with the same color have the same onchip DVFS setting. The black line bounding the state space from below marks the pareto-optimal frontier.

states correspond to lesser performance capabilities. So, P0 is for the highest DVFS level, **C0** is for the largest cache level, **S0** is for the maximum offchip bandwidth and M0 is for the highest memory DVFS level. We choose **P4-C0-S0-M2** as the default/baseline/current configuration.

By changing these knobs, the system can be configured in $9 \times 5 \times 6 \times 3 = 810$ different ways. The knob impacts vary significantly according to the dynamic execution profile. Figures 1 shows knob impacts for two workloads with very different characteristics; blackscholes, having a small working set, is cache, bandwidth, and memory-insensitive whereas jbb is highly sensitive to these resource allocations. Figure 1a for blackscholes shows distinct vertical bands for each Px, implying that only core states affect both power and performance significantly whereas other knobs affect power but not performance. However, the state space for jbb shows more correlated effects, implying that considering just one knob is unlikely to result in optimal system operation.

Dynamic variability and correlated effects across knobs make identifying optimal configurations for the current execution a challenging problem. Mere enumeration of possible states for individual knobs, as in the ACPI [6] approach, is insufficient because it does not quantify power-performance impacts at the system level or take into account correlated effects across different knobs. So, it is difficult to answer questions such as: which system configuration performs the best for a given power budget? which system configuration has the minimum energy-delay (ED) or ED^2 product?

Knobs	State names
Onchip DVFS	P0, P1, P2, P3, **P4**, P5, P6, P7, P8
Cache (LLC) Capacity	**C0**, C1, C2, C3, C4
SERDES Bandwidth	**S0**, S1, S2, S3, S4, S5
Memory DVFS	M0, M1, **M2**

Table 1: Dynamic reconfiguration knobs that we study. See Section 6 for more details. To the best of our knowledge, *no other work has studied simultaneous coordination across all of these four resource types.* We highlight the default settings in bold font.

Π− states	Relative Perf.	Relative Power	Configuration
Π_0	1.19	0.76	P0-C4-S5-**M2**
Π_1	1.14	0.69	P1-C4-S5-**M2**
Π_2	1.09	0.64	P2-C4-S5-**M2**
Π_3	1.04	0.58	P3-C4-S5-**M2**
Π_4	1.00	0.54	**P4**-C4-S5-**M2**
Π_5	0.96	0.50	P5-C4-S5-**M2**
Π_6	0.91	0.45	P6-C4-S5-**M2**
Π_7	0.86	0.42	P7-C4-S5-**M2**
Π_8	0.80	0.38	P8-C4-S5-**M2**

(a) blackscholes (small working set). Only Px varies.

Π− states	Relative Perf.	Relative Power	Configuration
Π_0	1.19	1.29	P0-**C0**-**S0**-M0
Π_1	1.16	1.26	P0-**C0**-**S0**-M1
Π_2	1.14	1.12	P0-**C0**-S1-M0
Π_3	1.11	1.06	P1-**C0**-S1-M0
Π_4	1.08	0.99	P0-C1-S1-M0
Π_5	1.05	0.93	P1-C1-S1-M0
Π_6	1.02	0.88	P2-C1-S1-M0
Π_7	1.00	0.83	P3-C1-S1-M0
Π_8	0.98	0.81	P3-C1-S1-M1
Π_9	0.96	0.77	P2-C1-S2-M0
Π_{10}	0.93	0.74	P5-C1-S1-M0
Π_{11}	0.92	0.71	P2-C1-S4-M0
Π_{12}	0.90	0.68	P3-C1-S3-M1
Π_{13}	0.88	0.65	P3-C1-S4-M1
Π_{14}	0.87	0.63	**P4**-C1-S4-M0
Π_{15}	0.84	0.60	P5-C1-S3-M1
Π_{16}	0.81	0.56	P6-C1-S3-M1
Π_{17}	0.80	0.54	P6-C1-S4-M1

(b) jbb (cache-, bandwidth-, memory-sensitive). All knobs vary.

Table 2: Sample (predicted) Π-dashboards, for two different applications, showing Π-states that indicate predicted speedup and power relative to the current configuration (P4-C0-S0-M2). Pareto-optimal Π-states are totally ordered: $\Pi_0 > \Pi_1 > ....$ The default ordering is: $i < j \implies (\Pi_i.Perf > \Pi_j.Perf) \wedge (\Pi_i.Power > \Pi_j.Power)$.

Fortunately, these apparently diverse problems can be reduced to a single problem of identifying the set of pareto-optimal states, that is, the pareto-optimal frontier. A state is pareto-optimal, with respect to power-performance, if there is no other state that improves performance without increasing power consumption or reduces power consumption without reducing performance. Figure 1 shows the pareto-optimal frontiers for blackscholes and jbb. Appendix A proves that the best states for most power-performance objectives for throughput-oriented systems lie on the pareto-optimal frontier. *The challenge is to identify the pareto-optimal frontier for the system at runtime from the individual reconfiguration knobs of interacting components.*

Quantifying system impact is a necessary precondition for comparing configurations leading to identification of pareto-optimal ones. In Figures 1a and 1b, a configuration at $(1,y)$ has the same performance as the baseline configuration, but at y-times power (saves energy if $y<1$) whereas one at $(x,1)$ has x-times performance at the same power (saves energy if $x>1$, by employing a "race to halt" policy). For example, configuration $(1,0.54)$ for blackscholes implies 46% power savings for the same performance; configuration $(1.08,0.99)$ for jbb implies 8% more performance for (almost) the same power. However, this kind of information is missing in the ACPI-like description of Table 1 (and Table 4). Selecting an arbitrary configuration without quantitative impact information can be severely suboptimal since the state spaces span over a wide range of power and performance as seen in Figures 1 and 9.

Π-dashboards enable selection of a variety of power-performance profiles for the system. Referring to Table 2a, Π_0 represents the highest performance state, Π_4 represents a state with 46% power savings for the same performance, Π_5 represents a state with 5% better performance for 7% less power, and so on. The dashboard attempts to bridge the gap between high-level power-performance goals and system resource configurations – *a mapping capability that is largely missing in today's systems.*

Trial-and-error approaches to dynamically finding the best configuration are not attractive since reconfiguring some of these knobs incurs cost in time and energy e.g., >150 μsec for bandwidth reconfiguration (see Appendix D). LLC warmup/writeback can take up to several msecs depending on the capacity and offchip bandwidth. Accumulated overheads and suboptimality of repeated trials may significantly reduce the benefits of reconfigured system operation.

3. Management Framework

We logically partition the system into subsystems and a coordinator. Figure 2 shows a logical structure of the system. Subsystems control internal resources, track properties of interest such as activity counts, temporal locality, bandwidth requirements of the current execution profile and predict their power-performance characteristics for a number of feasible alternate configurations. The coordinator combines the subsystem predictions to determine system-wide impacts, constructs the dashboard, then communicates the selected combination back to the subsystems. The coordinator may be implemented as a software routine (ISR) that runs on one or more cores, or as a specialized unit such as a PCU in modern systems [39].

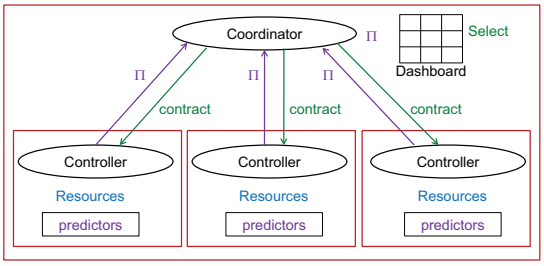


Figure 2: High-level view of the management framework.

Execution time is logically partitioned into intervals. Like most predictors, this work uses past execution behavior to predict behavior in subsequent intervals.

- **Training Interval**: Each subsystem uses hardware monitors to observe its own execution characteristics that it then

uses to predict Π-states for its reconfiguration options.

- **Configuration Selection Interval**: Subsystems communicate the formulated Π-states to the coordinator which then composes them to determine system-wide power-performance impact. It determines and totally orders pareto-optimal states to form a Π-dashboard. The user/OS selects the desired power-performance profile. The coordinator communicates decisions back to the subsystems.
- **Local Adaptation Interval**: Subsystems reconfigure themselves to conform to the selected operating point.

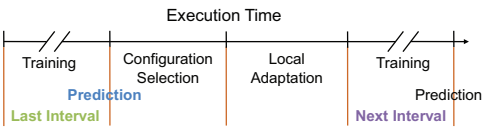


Figure 3: Training, Selection and Adaptation intervals

Figure 3 illustrates execution intervals. Training intervals are much longer than selection and adaptation intervals in order to make overheads negligible. Reconfigurations can be frequent (e.g., once per few tens of milliseconds for DVFS) or spaced apart (e.g., once per several hundred milliseconds for cache resizing). Our best reactive policy for the real machine study in Section 4 reconfigures once every 510 milliseconds. Short training intervals enable more reactivity than longer intervals but consider less execution history.

4. Real Machine Study

We demonstrate the potential for our approach on real hardware by improving the energy efficiency, measured as performance-per-watt, using only processor frequency control. This shows the management framework, described in Section 3, in action for a single reconfiguration knob. The system is a single-socket quad-core Haswell-based Xeon E3-1275 server with 32 GB memory, henceforth referred to as HS. HS runs RHEL with kernel version 2.6.32 and allows 15 frequency steps from 0.8 – 3.5 GHz and a turbo boost region (3.501 – 3.8 GHz).

The Linux acpi-cpufreq module includes the following policies, called governors, that decide the operating frequency. The root user can dynamically change the governor.

- **PowerSave** (**S**): Sets all cores to the lowest frequency.
- **OnDemand** (**O**): Periodically samples (default: 10 ms interval) cores to adjust frequencies based on core utilization.
- **UserSpace** (**U**): The root user can set the socket frequency (all cores together[2]) to any of the allowed frequencies.
- **Performance** (**P**): Sets all cores to the highest frequency.

Turbo boost happens only when the highest frequency (3.501 GHz) is selected (explicit in **O** or **U** mode, automatic in **P** mode). To further distinguish between modes, we constrain **U** mode to exclude **S** or **P** mode frequencies, i.e., it operates in the range of 1.0 – 3.5 GHz.

[2]This interface does not allow per-core settings for HS. All cores transition to the highest frequency of any core in the socket.

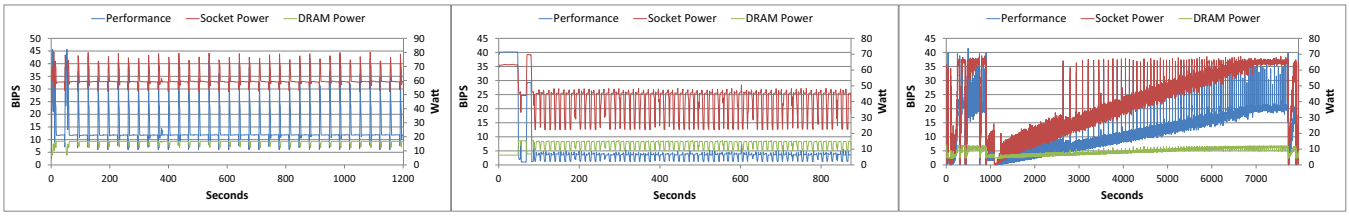
We evaluate the system using a total of 17 workloads – 14 from SPECOMP2012 [3], plus graph500 [1], hpcg [2] and jbb2013 [4]. HyperThreading (2 hardware threads per core, i.e., 8 hardware threads per socket) is enabled for all studies. We measure socket power and DRAM power (RAPL Mode 1) using an additional software thread that reads available RAPL counters at 1 second intervals. HS has a socket TDP of 84W and a remarkably low socket power of ∼0.27W when idle. DRAM idle power is ∼4.3W. We consider system power as the sum of socket power and DRAM power.

Figure 4 shows example power-performance traces for applu (SPECOMP2012), graph500 and jbb2013 in **P** mode. applu performs several iterations, each with a memory-intensive portion followed by a compute-intensive portion; the performance and power spikes indicate iteration boundaries. The DRAM power drops during the compute-intensive part of each iteration due to less memory accesses. graph500 runs 64 iterations of bfs after initialization. Both applu and graph500 exhibit long-term periodic behavior in both performance and power readings, with periods of tens of seconds corresponding to iteration lengths. jbb2013 dynamically increases parallelism as it runs and shows much less periodicity. graph500 and jbb2013 are run to completion whereas applu (all of SPECOMP2012) and hpcg are run for the first 1200 seconds.

HS exhibits significant opportunities in improving BIPS/Watt (equivalently, Instructions/nanoJoule) by changing frequency settings alone. BIPS changes between 1.18x (swim) to 4.86x (bwaves) in going from **S** to **P** modes whereas power changes between 2.52x (swim) to 5.67x (botsalgn), leading to a BIPS/Watt range of 1.29x (imagick) to 2.14x (swim) between best and worst values for that workload over all frequencies. applu, graph500 and jbb show a ratio of 1.84x, 1.67x and 1.37x respectively (also see Figure 6).

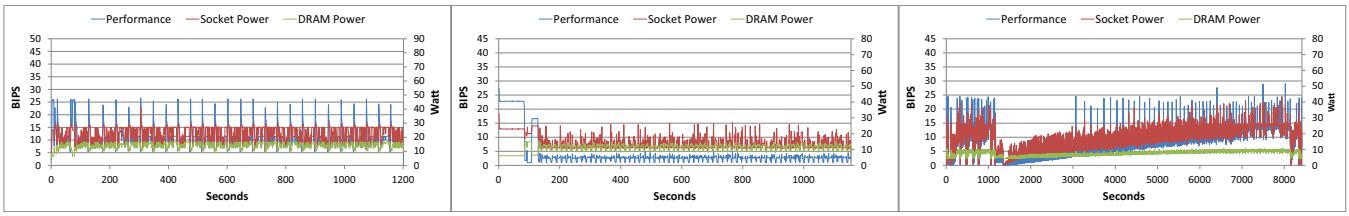
In order to exploit the improvement potential, we implement a simple reactive, **R(t)**, mode of operation. For this system, there are two subsystems (see Figure 2), one for DRAM and one for the socket. The socket predictor sets the frequency to 0.8 GHz, 2.1 GHz and 3.5 GHz in three consecutive intervals of t ms each and observes the power, performance, memory read and write bandwidths for each setting. It then interpolates (piecewise linear) the effects for the other frequencies. The DRAM predictor is invoked every $12t$ ms and adjusts a computed linear regression between DRAM power and read and write bandwidths (two variables) based on current readings. The regression is reset every 17 observations ($204t$ ms) to react faster to phase changes. The coordinator reads the socket predictions and DRAM predictions every $51t$ ms (immediately after the $3t$ socket sampling), composes the predictions, computes the Π-dashboard and selects the best frequency. Figure 5 shows power-performance traces for applu (SPECOMP2012), graph500, jbb2013 in **R(10)** and the improvement in BIPS/Watt.

There are two main challenges in implementing the interpolant for the socket predictor: 1) getting successive sample



(a) applu. Instructions/nanoJoule = 0.18. (b) graph500. Instructions/nanoJoule = 0.10. (c) jbb2013. Instructions/nanoJoule = 0.24.

Figure 4: Power-Performance traces in P-mode on HS. Higher Instructions/nanoJoule implies more energy efficiency.



(a) applu. Instructions/nanoJoule = 0.30. (b) graph500. Instructions/nanoJoule = 0.17. (c) jbb2013. Instructions/nanoJoule = 0.32.

Figure 5: Power-Performance traces in R(10)-mode on HS. Higher Instructions/nanoJoule implies more energy efficiency.

points that show non-decreasing performance and power with increasing frequency and 2) getting sample points with acceptable measurement noise/jitter. The first issue arises when the workload exhibits local phase behavior. The second issue arises with rapid sampling that makes the jitter in the energy measurements seem to be higher than that in the timing measurements leading to occasionally unrealistic power calculations. We disregard samples if either decreasing values are found or if power readings differ in more than 10x between the three samples and the coordinator transitions to 3.5 GHz. While other default actions are possible, we choose to penalize ourselves when we are not confident about the interpolation. We see samples discarded more frequently in **R(1)** (geomean: 14.2%) than in **R(4)** or **R(10)** (geomean: $< 2\%$ for both). swim is particularly hard hit in **R(1)** with $\sim$82% of samples discarded mostly due to the first issue.

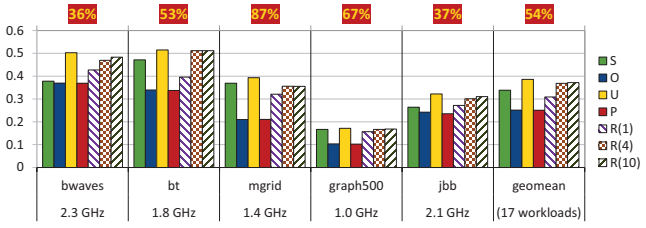


Figure 6: BIPS-per-Watt on HS with different policies.

Figure 6 compares the energy efficiency with different modes of operation. The best **U**-mode frequency for each workload is shown below. The percentage numbers at the top show gains in energy efficiency over **P**-mode with that frequency. Our main findings are:

- *The potential rewards for selecting optimal configurations are significant*: 28.6% (imagick) to 113.7% (swim) over **P** (geomean: 53.9% over **P**, 13.9% over **S**). The **O** and **P** modes are suboptimal for this metric for every workload.
- *There is no single best static setting*: 0.8 GHz (swim), 1.0 GHz (applu, graph500, hpcg), 1.4 GHz (mgrid), 1.8 GHz (bt, botsspar), 2.0 GHz (ilbdc, smithwa, kdtree), 2.1 GHz (md, nab, botsalgn, fma3d, jbb), 2.3 GHz (bwaves, imagick).
- *Rapid profiling and reconfigurations are neither useful nor necessary*: both **R(10)** (geomean: 48.3% over **P**, 9.8% over **S**) and **R(4)** (geomean: 47.1%, 8.9%) improved over **R(1)** (geomean: 23.2%, -8.8%). **R(10)** improved by 2.8% over **R(4)** for bwaves.

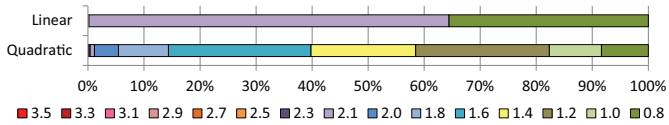


Figure 7: R(10) freq. distribution for applu on HS(0.8–3.5 GHz).

One of the shortcomings of the piecewise linear interpolation used by the socket predictor is that for the performance-per-watt metric, only the sample frequencies (0.8/2.1/3.5 GHz) can be chosen[3]. We also evaluated quadratic interpolation for an alternative socket impact predictor *without needing to change the coordinator*. Figure 7 shows the frequency distribution for both schemes for applu. While Linear fluctuates mostly between 0.8 and 2.1 GHz, resulting in $\sim$68% improvement over **P**-mode, Quadratic selects more frequencies in between resulting in $\sim$78% improvement. mgrid showed similar improvements.

We re-evaluated results using the same #instructions (min. across all policies from the fixed-time runs) for each workload. In terms of **U**-mode gains over **P**-mode, applu changed

[3]**Proof sketch**: Perf(f)=af+b and Pwr(f)=cf+d $\implies$ Perf(f)/Pwr(f) is monotonic in f. So, the maxima will occur among the end points of the interval.

from 84% to 80% whereas botsspar changed from 47% to 52%. geomean changed < 2% for all policies. The best **U**-mode frequency changed for three workloads. All trends remained the same.

With respect to the knobs in Table 1, this study dealt with just one knob (onchip DVFS). The approach of trying out various settings becomes cumbersome with more knobs not only due to the large number of combinations, but also because resources like large caches and SERDES incur significant reconfiguration overhead. For efficient prediction, diverse knobs (as in this work) may benefit from different, specialized predictors [27, 40]. Moreover, with third party components [12], a generic coordinator may not even know what knobs are available and how to predict for them.

What is needed is a well-defined interface that components can use at run time to communicate their operating states, identified using their own predictors, to the coordinator. This interface must be generic, like ACPI, but more expressive to allow the coordinator to determine pareto-optimal state combinations. We propose such an interface using Π-states.

5. Π-states

Let p denote a typical subsystem and *sys* denote the larger system having a coordinator. Π-state i for subsystem p, denoted by $\Pi_i(p)$ is a 4-tuple $(\ell_i(p), d_i(p), s_i(p), \boldsymbol{w}_i(p))$ where:

- $\ell_i(p)$ is the relative sℓowdown (=new time/baseline time) by p, *for its own operations*.
- $d_i(p)$ is the *d*ynamic energy predicted to be used by p.
- $s_i(p)$ is the *s*tatic power predicted to be used by p.
- $\boldsymbol{w}_i(p)$ is a vector of ***w***ork request summaries generated by p. The type of requests and summaries are pre-determined. An example is a request vector (#fetches,#writebacks) that depends on the LLC capacity in that state (see Section 6.1).

The description may include additional meta-data such as transition cost, predictor confidence and thermal headroom.

Subsystem p sends a descriptor packet $C(p)$ to the coordinator that includes the Π-state descriptions and information on how to adjust them, if needed, while composing them to get system-wide descriptions. In Section 4 the DRAM state was parametrized by read and write bandwidths that were instantiated by the coordinator from the particular socket state being considered. We further separate two aspects for more fine-grained modeling control:

1. *Amount of work*: For example, #reads, #writes.
2. *Rate of work*: For example, dependence on total time. Queueing delays usually change with work arrival/service rates that in turn depend on total time. To facilitate this, we allow $\ell_i(p)$ of $\Pi_i(p)$ to be encoded by a (small) lookup table (LUT) that is indexed by $\ell(sys)$. Any discrete function of a single variable can be encoded by a LUT.

5.1. Π-state Composition

The coordinator composes subsystem Π-states to generate system Π-states. Let $t(p)$ denote the contribution by p to the baseline system time, $t(sys)$, so that $t(sys) = \sum_p t(p)$ (see Section 6.3 for an example implementation of how non-overlapping contributions to system time are estimated).

For each combination of subsystem Π-states, the coordinator creates a tuple $(\ell(sys), d(sys), s(sys), \boldsymbol{w}(sys))$ where

$$\ell(sys) = \frac{\sum_p t(p)\ell(p)}{\sum_p t(p)} \quad (1)$$

$$d(sys) = \sum_p d(p) \quad (2)$$

$$s(sys) = \sum_p s(p) \quad (3)$$

$\boldsymbol{w}(sys)$ is currently unused, but would be set if this node generates work requests for other nodes in a cluster.

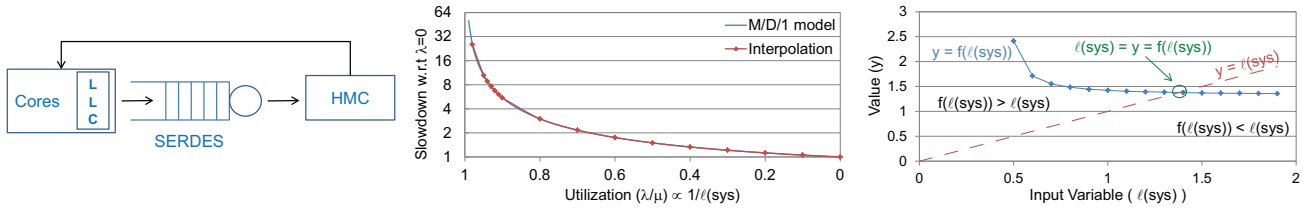
In the above, Equation 1 determines system slowdown relative to the current configuration; smaller values of $\ell(sys)$ indicate better performance, and vice versa. This new value of system time can affect work arrival rates, so individual contributions must be re-adjusted (discussed below). Equations 2 and 3 determine system dynamic energy and static power consumptions as the sum of the uses by the subsystems.

We model a closed system with timing feedback. Figure 8a shows an example. This makes it possible for $\ell(p)$ in Equation 1 to be a function of $\ell(sys)$. We need to find a value for $\ell(sys)$ (and hence, $\ell(p)$ for all p) that is consistent with Equation 1, that is, LHS=RHS. It is a solution, or fixed point, of the equation LHS-RHS=0 and characterizes a point of timing equilibrium in the feedback system.

While a possible solution approach is to rewrite the equation in closed functional form (e.g., a polynomial) and use online solvers, there are two challenges with this approach: the functions must be known in closed form and they may be non-linear (e.g., queueing models) requiring online solvers with non-trivial computation costs.

We mitigate both difficulties by constraining that for each Π-state, $\ell(p)$ be *monotonically decreasing* (see Appendix B for a definition) in $\ell(sys)$. The intuition is that as total execution time increases (larger $\ell(sys)$), waiting and service times for each unit of work decrease or stay the same, in turn limiting execution time increase. As an example, Figure 8b shows an M/D/1 queueing response function (larger $\ell(sys) \implies$ lower utilization $\implies$ less slowdown, that is, smaller $\ell(p)$) and interpolation points for a 16-entry LUT.

If all $\ell(p)$ are monotonic, decreasing, then the fixed point will be unique if it exists. To see this, assume that for an example combination of Π-states, the RHS of Equation 1 is denoted by $f(\ell(sys))$. Figure 8c plots $f(\ell(sys))$ for different values of $\ell(sys)$. If all $\ell(p)$ are monotonic, decreasing, then f is also monotonic, decreasing (see Appendix B). The dashed line passing through the origin represents potential fixed points. The actual fixed point (LHS=RHS) is where the dashed line intersects the curve. It is obvious that the intersection point will be unique if it exists. The constraint on $\ell(p)$ is sufficient, but not necessary for the unique solution.



(a) Closed system model (b) M/D/1 queueing model response function (c) Equation 1 fixed point for an example Π-state

Figure 8: (a) Closed system model with feedback from the memory subsystem, (b) example approximation of queueing response using piecewise linear interpolation, (c) representation of unique fixed-point (consistent/equilibrium state) for $\ell(sys)$.

For each combination of subsystem Π-states, the coordinator repeatedly computes the RHS until $\ell(sys)$ reaches (close to) a fixed point (see Appendix C for the algorithm), then creates a new system Π-state $(\ell(sys), d(sys), s(sys), \boldsymbol{w}(sys))$.

In case $\ell(p)$ is not monotonically decreasing, the iterations may not converge to a fixed point or the fixed point may not be unique. It is relatively straight-forward to detect non-convergence, and select a point with smallest difference between old and new values of $\ell(sys)$. However, the chosen configuration can be suboptimal in that case.

5.2. Π-state Pareto-dominance and Ordering

Pareto-dominance is important in identifying optimal system states. Pareto-dominated states are not considered for selection as they are suboptimal in comparison with other states.

Let $t(sys)$ denote the baseline system time. $\Pi_i(sys)$ pareto-dominates $\Pi_j(sys)$ if either of the following conditions hold:

1. $\ell_i(sys) < \ell_j(sys)$, $Power_i(sys) \leq Power_j(sys)$
2. $Power_i(sys) < Power_j(sys)$, $\ell_i(p) \leq \ell_j(p)$

where $Power_i(sys) = \left(\frac{d_i(sys)}{t(sys)*\ell_i(sys)} + s_i(sys)\right)$ is the total system power consumption (dynamic+static) in $\Pi_i(sys)$.

Qualitatively, Condition 1 implies that state i has better performance than state j for the same or less power consumption while Condition 2 implies that state i has lower power consumption than state j for the same or better performance.

Pareto-optimal Π-states are totally ordered by their predicted power-performance values according to:

$$i < j \implies \ell_i(sys) < \ell_j(sys), Power_i(sys) > Power_j(sys)$$

Qualitatively, this implies that both performance and power consumption in state j are less than that in state i. Performance is inversely proportional to $\ell(sys)$.

We use a 2-pass algorithm for ordering Π-states. First, the Π-states are binned into a small number (=50) of bins based on their $\ell(sys)$ values with only one state retained per bin. Then, starting from Π_0, states that have at least 2% less power than the previous state are retained.

The ordered list of pareto-optimal Π-states forms the dashboard (e.g., Table 2). The dashboard can also be visualized in a graphical form as shown in Figures 9 (Section 6.4). The blue curve connects the predicted pareto-optimal Π-states, with Π_0 at the upper end of the curve. (Predicted) pareto-dominated states (not shown) lie above the curve.

6. Simulation Study

Table 3 describes the 8-core CMP that we study through *full-system* simulations using GEMS [30] augmented with timing-first processor models, Wattch [14] and CACTI 5.3 [41].

We assume an 8-banked 32-way L3 cache that is dynamically re-configurable in capacity (2/4/8/16/32 MB) and uses the PLRU replacement policy. We use a simple XOR-based hashing function to distribute lines more uniformly among the L3 cache sets [17] and conservatively assume a constant access latency (in cycles) for all configurations.

We model our memory system based on the new Hybrid Memory Cube (HMC) technology [37, 31]. Compared with traditional DDR3 memory, HMC offers *new power-saving modes by partially shutting down its serial (SERDES) interconnect*. Its stacked memory has a logic layer base that we believe can implement simple predictors. Table 4 describes the available knobs and their possible settings in detail. Onchip DVFS level **P4** (=1.00) corresponds to 2132 MHz, 0.9V. Each Px level in Table 4 describes frequency relative to **P4**. The highest onchip frequency is $1.20 * 2132 = 2558$ MHz. With respect to SERDES configurations, 'F', 'H', 'Q' and 'N' denote full-width, half-width, quarter-width and no-width (powered-down) links (see Table 5, Appendix D for more details). We assume a maximum memory DVFS (overclocking) of 20%.

For this study we use the following workloads:

Multithreaded: Wisconsin commercial [7] (apache, jbb, oltp, zeus), PARSEC [11] (blackscholes, bodytrack, fluidanimate, freqmine, swaptions) with "simlarge" inputs, SPEComp [9] (ammp, equake, fma3d, gafort, swim, wupwise) with "ref" inputs. Each workload uses 8 threads and runs for a fixed amount of work (e.g. #transactions or loop iterations [8]).

Multiprogrammed: Combinations of SPEC CPU2006 [26] (astar, bwaves, cactusADM, bzip2, gcc, lbm, libquantum, milc, omnetpp, soplex). Each workload consists of 8 programs, equally divided among the benchmarks in the combination.

Each simulation run starts from a mid-execution checkpoint that includes cache warmup. **Last** and **Next** intervals consist of ∼250M instructions each. Baseline (**C0**) MPKI ranges from < 0.01 (blackscholes) to ∼34 (mcf-libquantum).

We will now describe specific predictors that we have used for this study; however, other predictors can be used as well. While the choice of predictors will affect the prediction accu-

Core configuration	8 cores, each 4-wide out-of-order, 128-entry window, 32-entry scheduler, YAGS and NoSQ predictors				
L1 Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP, separate L1I and L1D				
L2 Cache	private 256KB 8-way per core, 6 cycle access latency, PLRU, ITRS-LOP				
L3 Cache*	shared, configurable 2MB 2-way - 32MB 32-way, 8 banks, 14 cycle access latency, PLRU, ITRS-LOP, serial				
Coherence protocol	MESI (Modified, Exclusive, Shared, Invalid), directory				
On-Chip Interconnect	2D Mesh, 16B bidirectional links				
On-chip frequency*	2132-2665 MHz	**Technology generation**	32nm	**Temperature**	340K-348K
Off-Chip Interconnect*	SERDES, details in Appendix D				
Main Memory*	4GB stacked memory (2x2GB stacks), details in Appendix D				

Table 3: System configuration. Reconfigurable components are marked with a *.

Onchip DVFS	LLC	SERDES bandwidth	Mem. DVFS
P0=1.20	**C0**=32MB	**S0**=F F F F	M0=1.20
P1=1.15	C1=16MB	S1=H H H H	M1=1.10
P2=1.10	C2=8MB	S2=Q Q Q Q	**M2**=1.00
P3=1.05	C3=4MB	S3=Q Q Q N	-
P4=1.00	C4=2MB	S4=Q Q N N	-
P5=0.95	-	S5=Q N N N	-
P6=0.90	-	-	-
P7=0.85	-	-	-
P8=0.80	-	-	-

Table 4: Knob descriptions. Default state is P4-C0-S0-M2.

racy and computational complexity, our Π-framework is orthogonal to this choice. For the following discussion, assume that the current Π-state is described by $(\ell_{cur}, d_{cur}, s_{cur}, \boldsymbol{w}_{cur})$.

6.1. Socket/Onchip Subsystem

The onchip subsystem includes the cores and the LLC. We assume that the cores and LLC are in the same power plane as in the SandyBridge [39] microarchitecture. With 9 P and 5 C states, this subsystem generates $9 \times 5 = 45$ $\Pi(onchip)$ states. Let $(\ell_{new}, d_{new}, s_{new}, \boldsymbol{w}_{new})$ describe a new Π-state.

Using published data [25] for voltage-frequency pairs for the Pentium M, we assume that $(V_{new} - V_{cur}) \propto (f_{new} - f_{cur})$ and that every 200MHz change in frequency is accompanied by a 50mV change in voltage. Thus, $V_{new} = V_{cur} + 50mV \left(\frac{f_{new} - f_{cur}}{200MHz}\right)$. Let $\gamma = \frac{f_{new}}{f_{cur}}$.

- $\ell_{new} = \frac{1}{\gamma}$. This assumes that the onchip contribution to execution time is inversely proportional to frequency. Note that ℓ_{new} is a constant for each $\Pi(onchip)$ and thus satisfies the monotonically decreasing condition.
- $d_{new} = d_{cur} \left(\frac{V_{new}}{V_{cur}}\right)^2 \gamma$. This uses the intuition that dynamic energy is proportional to $V^2 f$. We obtain constants for LLC activation energy from CACTI. A minor limitation of the current predictor is that it ignores changes in dynamic energy for cache line replacements.
- $s_{new} = s(CPU) + s(LLC) + \Delta s_v + \Delta s_T$ where Δs_v and Δs_T respectively denote the effect of voltage and temperature on static power. We consider Δs_v, Δs_T only for frequencies above **P4**. We assume linear scaling for Δs_v [15] and 3 °C contributed by every 200MHz [25] change in frequency to calculate Δs_T. We use constants from CACTI for $s(CPU)$, $s(LLC)$ and linear interpolation to determine Δs_T (CACTI provides values at temperature intervals of 10K).
- $\boldsymbol{w}_{new}$ is obtained using an online cache miss-rate predictor [40] and assuming that the ratio of fetches to writebacks remains the same across configurations.

6.2. Offchip Subsystem

The offchip subsystem includes the offchip interconnect (SERDES) and main memory (HMC). With 6 S and 3 M states, this subsystem generates $6 \times 3 = 18$ $\Pi(offchip)$ states. Let $(\ell_{new}, d_{new}, s_{new}, \boldsymbol{w}_{new})$ describe a new Π-state.

- $\ell_{new} = \frac{SERDES\_delay + HMC\_delay}{current\_offchip\_delay}$ where
  - $SERDES\_delay = \frac{2\mu - \lambda}{2\mu(\mu - \lambda)}$ (standard M/D/1 model) where μ is the service rate (Table 5, 1 roundtrip transaction in 6-flit-time + expected inter-quadrant delay) and λ is the arrival rate (affected by onchip $\boldsymbol{w}$ and $\ell(sys)$).
  - $HMC\_delay = \frac{28.44ns}{\gamma}$ where $\gamma = \frac{f_{new}}{f_{cur}}$ (1.0, 1.1 or 1.2). This assumes that memory queueing delays are negligible due to the high parallelism offered by the HMC.

  Note that ℓ_{new} is monotonically decreasing in $\ell(sys)$ for each $\Pi(offchip)$. ℓ_{new} is encoded by a 16-entry LUT. All delays are normalized so that $\ell_{new} = 1$ when $\ell_{sys} = 1$ for the current configuration.
- $s_{new} = s(SERDES) + s(HMC)$ where $s(SERDES)$ is obtained from Table 5 and $s(HMC)$ = current HMC refresh power$\times\gamma^2$.
- d_{new} = current HMC background energy$\times\gamma^2$ + (#reads+#writes)$\times$current HMC energy/access$\times\gamma^2$.
- $\boldsymbol{w}_{new}$ is not set (no further requests generated).

6.3. Non-overlapping subsystem time, $t(p)$

Equation 1 (Section 5.1) assumes that $t(p)$ is available. We use online linear regression between ΔLLC misses and ΔCPI (obtained with simple performance counters at ∼1 msec intervals) to estimate non-overlapping $t(onchip)$ and $t(offchip)$ so that $t(sys) = t(onchip) + t(offchip)$ where $t(offchip) = coefficient \times \#misses$. We use a statically fixed value (from a run with maximum observed $\frac{t(offchip)}{t(sys)}$) for $coefficient$ if online regression fails to determine it.

While $\ell(offchip)$ in Subsection 6.2 quantifies expected

slowdown for a single request, $t(offchip) = coefficient \times$ *#misses* in the above paragraph quantifies the offchip contribution to system time taking total misses and parallelism into account (multiple out-of-order cores, lockup free caches, etc.). To capture both effects, we scale $t(offchip)$ by the ratio of the predicted *#misses* for that Π-state relative to the current *#misses* before applying Equation 1.

6.4. Analysis of Trends

Comparing predictions against full state-space simulated values for all workloads is impractical. Instead, our goals are: (i) to evaluate system impact quantification accuracy at the predicted pareto-optimal frontiers, (ii) to analyze reasons for prediction errors and (iii) to discuss implications for power-efficient performance.

Figure 9 show, for a few representative workloads, the predicted pareto-optimal curve and the actual (simulated) power-performance for those configurations. The distance of × from the curve quantifies the predicted suboptimality in the default/baseline/current configuration for that workload – an operating point that is "vertically below" is predicted to reduce power at the same performance, whereas a point that is "horizontally right" is predicted to increase performance at the same power consumption level. Some workloads such as blackscholes, freqmine do not have a point "horizontally right" to × indicating that with the available knobs, power can be saved even after maximizing performance.

Each graph in Figure 9 also includes two tables:

Avg. Error: the average of absolute values of relative errors for **Predicted** values with respect to **Last Interval (LI)** simulation (top row) and **Next Interval (NI)** simulation (bottom row) for the predicted configurations (also see Figure 3).

Pareto: This has two columns.

1. **Dom.**: the fraction of predicted configurations pareto-dominated (hence, suboptimal) by other predicted configurations during **LI** simulation and **NI** simulation.
2. **Inv.**: the fraction of pairwise orderings among non-dominated points for **LI** simulation and **NI** simulation that were inverted with respect to the predicted ordering. There are $\frac{(n-k)*(n-k-1)}{2}$ pairwise orderings for n total points having k dominated points.

For example, during **LI** simulation for apache, 3 of 21 predicted pareto-optimal states were dominated by some of the 18 other states and of the 153 ($= \frac{18\times17}{2}$) pairwise-orderings for those 18 states, 1 predicted ordering was violated/inverted. 0 Dom., 0 Inv., and ∼0% Error identify the best predictions. Some dominated points are marginally suboptimal (e.g., astar-gcc). Order inversions are rare.

Avg. **LI** errors (only model effects) are mostly <10% (except zeus, mcf-libquantum). One of the sources of error arises from the M/D/1 model not exactly describing the bursty memory traffic. Phase changes are an important source of prediction error. This is demonstrated in astar-bwaves, bodytrack, mcf-bwaves, omnetpp-lbm, swim where the predicted values are quite good compared to **LI** simulation but has more errors for **NI** simulation. We expect the system to employ continuous monitoring to check predicted versus actual power-performance and revert to the baseline configuration in case of a misprediction that violates required power budgets or performance goals. The monitor should employ a guard mechanism to provide better energy security [13].

The point marked **ED** represents the state predicted to have the minimum energy-delay product; the corresponding resource configurations are also shown. Notice that the predicted **ED** configurations are different for different workloads. This reinforces our point that mere enumeration of configuration knobs is not sufficient for determining optimal configurations; one needs to additionally consider dynamic workload characteristics and inter-component interactions. This is true for other power-performance objectives as well.

While the high-performance states P0, **C0**, M0 appear in predicted minimum **ED** configurations of some workloads, **S0** does not appear in any of them. However, **S0** is important for overall best performance (e.g., apache, jbb). This suggests that very high offchip bandwidths may not be result in energy-optimal operations as the extra power needed may outweigh the performance benefits. The default configuration (**P4**-**C0**-**S0**-**M2**) does not appear in the predicted minimum **ED** configuration for any workload, but default knob values **P4**, **C0**, **M2** appear for 9/24, 2/24 and 16/24 workloads respectively.

The data also reveals distinct classes of workload performance characteristics: (i) **cache and memory insensitive** – e.g., ammp, blackscholes, bodytrack, swaptions, (i) **cache insensitive but memory sensitive** - e.g., equake, gafort, (iii) **cache and memory sensitive** – e.g., apache, jbb, zeus. Workloads in classes (i) and (ii) do not benefit from large caches whereas workloads in class (iii) do. Workloads in class (i) do not benefit from high bandwidth or fast memory whereas workloads in classes (ii) and (iii) do. Runtime classification is important for reducing the number of Π-state combinations to consider (Section 6.5). This can be achieved by requiring subsystems to report Π-states that differ by at least a minimum margin in power-performance from each other. With this scheme, workloads that are performance-insensitive to some subsystem will have only one Π-state generated by that subsystem.

Considering less knobs than those available can result in significant suboptimality. For example, if only DVFS knobs are considered [20,21,23,19,27], states Px-**C0**-**S0**-**M2** would be chosen for blackscholes but they use ∼71%-112% more power than states Px-C4-S5-**M2** (corresponding Px values) at practically the same performance levels.

6.5. Coordinator computation time

The iterative solver (see Appendix C) used 1-13 iterations to reach within 1% of the fixed point if it existed. Π-state composition for 810 states and subsequent ordering took <1 msec for all workloads. Referring to the discussion in Section 4 we see that relatively long profiling (and reconfiguration) inter-

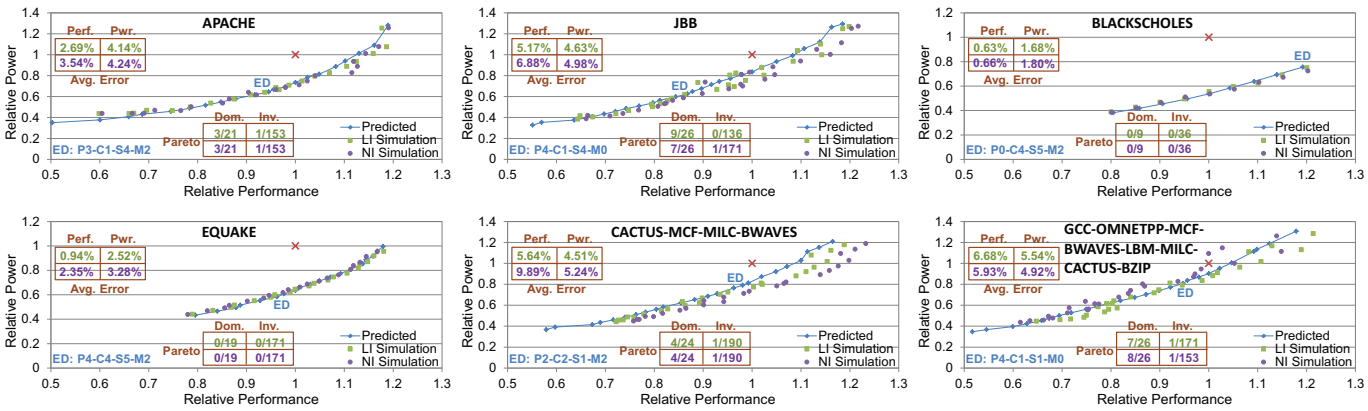


Figure 9: Simulated (LI, NI) power-performance for predicted pareto-optimal configurations. × (1,1) marks the default/baseline/current ("You are here") configuration (P4-C0-S0-M2). ED refers to the predicted configuration for minimum EDP. Section 6.4 describes the Avg. Error and Pareto Tables.

vals are preferred. The **R(10)** mode reconfigures once every 510 msec. In this context, we consider the computation time to be quite reasonable.

We also observe that many workloads exhibit long-term variation and periodic behavior. For example, applu (Figure 4) shows a ∼35 sec periodicity in **P** mode; this increases to ∼54 sec at the best user-mode frequency of 1 GHz. graph500 exhibits 12.3 sec (**P**-mode) to ∼18 sec (at 1 GHz) periods. We expect long training intervals that track considerable execution history to work well with such workloads.

The coordinator computation time increases with the total number of compositions which is the product of the number of exposed Π-states for each subsystem. While this may limit scalability for systems with a lot more configuration knobs, the following techniques can reduce the computation time:

Parallel computation: Compositions are independent of each other and can be evaluated in parallel.

Minimum separation: Subsystems can expose only Π-states that differ by at least a minimum pre-determined margin. So, for example, a memory-insensitive workload will result in only a single offchip Π-state getting generated. This, in effect, achieves a runtime classification of workload performance characteristics (also see Section 6.4).

Greedy selection: A subset of Π-states can be chosen greedily as demonstrated by CoScale [20]. The approach in CoScale is to arrange operating states in decreasing order of marginal utility ($\Delta power/\Delta performance$) and greedily pick states till a maximum slack is not violated. Although theoretically the greedy strategy can have up to a 2-approximation factor [18], CoScale demonstrates a tighter approximation in practice. CoScale, however, neither generates a pareto-optimal frontier nor does it consider non-DVFS knobs such as cache capacity or bandwidth allocations.

7. Related Work

There exists a variety of flavors of the power/energy management problem. The *power-budgeting* problem seeks to partition a maximum power budget among resources to maximize performance [27]; the *energy-minimization* problem seeks to find configurations that minimize energy consumption (equivalently, maximizes performance/watt); the *min-EDP* problem seeks to minimize the energy-delay product (EDP) [24] so that configurations that reduce energy but cause unacceptable delays are not chosen. Π-dashboards enable state selection for these and other problems.

A number of papers have studied simultaneous reconfigurations of multiple components within a node: cores and caches [33, 29, 22]; cores and memory [23, 20]. However, an integrated study with cores, caches and memory is missing.

The Advanced Configuration and Power Interface (ACPI) [6] allows processors/devices to expose performance (P) or power-saving (C) states that software can select to request state transition for that processor/device. However, unlike our Π-states, they do not quantify power-performance impact, nor do they provide an ordering across configurations with multiple devices. The upcoming Haswell microarchitecture [16] proposes to extend C-states to include latency requirements, past history and runtime hints from devices.

Eckert et al. [22] proposed new processor P-states and L2 P-states (power-gating and drowsy modes), but did not provide a framework for optimal system configuration selection.

Researchers have recognized the need for coordination for small- as well as large-scale systems [38,43,36,20]. We agree that coordination is important and necessary. We also make our coordinator agnostic to the knob implementation details.

Modern systems, such as those with the Sandy Bridge microarchitecture [39], include a centralized power-control unit (PCU) that collects telemetry information from functional blocks and performs control actions. Our proposed coordinator can be colocated with/implemented by such a PCU.

8. Conclusions

This paper introduced new Π-state descriptors and a retargetable framework for determining pareto-optimal power-

performance states in CMPs with reconfigurable resources. Π-states improve upon ACPI descriptors by quantifying impact and predicting a total order among configurations involving multiple knobs. The coordinator interfaces with subsystem controllers but is otherwise agnostic of specific reconfiguration knobs or impact predictors. While this work dealt with single-node systems, our long-term vision is to extend coordination using Π-state descriptors to larger multi-node and hierarchical systems such as in datacenters. We hope that this work will encourage future research into augmenting the ACPI standard to enable easy and efficient identification of the pareto-optimal frontier at the system level.

APPENDIX

A. Pareto-optimality & Power-Perf. Metrics

Consider a state y that is not on the pareto-optimal frontier. So there exists at least one other state x such that $\Pi_x.Perf \geq \Pi_y.Perf$ and $\Pi_x.Power \leq \Pi_y.Power$ with at least one of the inequalities being strict. Obviously then, the highest performing state with/without a (maximum) power cap and the lowest power state with/without a (minimum) performance bound must lie on the pareto-optimal frontier.

Since in this work, $performance \propto \frac{1}{delay}$, the above condition also means that $\Pi_x.delay \leq \Pi_y.delay$ and $\Pi_x.Power \leq \Pi_y.Power$ with at least one of the inequalities being strict. Since energy is power multiplied by time (delay), it implies that the lowest energy point with/without a delay cap must lie on the pareto-optimal frontier. Since the state corresponding to the highest performance-per-watt is the same as the state with the lowest energy, that state will also be on the pareto-optimal frontier. Moreover, according to the above condition, states corresponding to the minimum energy-delay (ED) product or ED^2 product or, in fact, any $ED^n, n \geq 0$ must also lie on the pareto-optimal frontier.

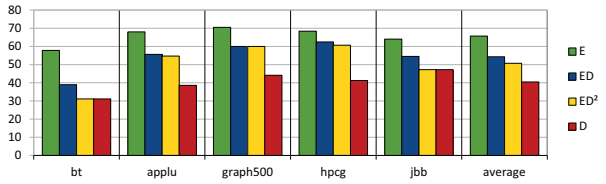


Figure 10: Avg. Socket thermal headroom (°C) on HS.

Selecting different metrics changes not only the operating power-performance envelope, but also the thermal characteristics. Figure 10 shows the average socket thermal headroom on the Haswell server for several metrics (E: minimizes energy, D: minimizes delay) for the best (pareto-optimal) user-mode frequency settings for that metric. The average is taken over 17 workloads. Unsurprisingly, the chip gets hotter as one moves through E → ED → ED^2 → D operating points.

B. Monotonically decreasing functions

A function $f(x)$ is monotonically decreasing in x if $x_1 > x_2 \implies f(x_1) \leq f(x_2)$. We require only monotonic ($f(x_1) \leq f(x_2)$), *not strictly monotonic* ($f(x_1) < f(x_2)$). Any function that evaluates to a constant is monotonically decreasing as well. Any affine combination, with non-negative coefficients, of monotonically decreasing functions is also monotonically decreasing. The RHS of Equation 1 is such an affine combination, the p^{th} coefficient being $\frac{t(p)}{\sum_p t(p)}$.

C. Iterative convergence to fixed-point

At the fixed point, $x = f(x)$ or, equivalently, $x - f(x) = 0$. Our iterative method uses the observation that x and $f(x)$ always bracket the fixed point (See Figure 8c; $f(x) > x$ for x less than the fixed point and $f(x) < x$ for x larger than the fixed point.

Our iteration method is similar to the standard bisection method of finding roots (in this case we are finding the root of the equation $x - f(x) = 0$). Let x_0 denote the initial value. If $f(x_0)$ is not already the fixed point, then either $x_0 < f(x_0)$ or $x_0 > f(x_0)$. Let lb and ub denote the lower and higher of the two values respectively. By our previous observation, the interval $[lb, ub]$ must contain the fixed point if it exists.

Let $m = \frac{lb+ub}{2}$. If m is not the fixed point and $f(m) > m$, then the interval $[m, min(ub, f(m))]$ will contain the fixed point if it exists. Otherwise, if $f(m) < m$, then the interval $[max(lb, f(m)), m]$ will contain the fixed point if it exists. In either case, the length of the search interval is reduced in every iteration. We terminate the iterations on reaching reasonably close (within 1%) of the fixed-point, or when $lb > 2$ (we are considering up to slowdown=2), or when the interval is very small (fixed point not found/does not exist).

D. HMC reconfiguration model

Our system uses 2 memory channels, each with a 2GB HMC stack. We use 4-link devices with 64-byte access block size. While the spec [31] defines static full-width and half-width configurations, we assume that each link can be dynamically reconfigured to these widths by transitioning a subset of its lanes to down mode. We also add a quarter-width configuration. Table 5 describes these configurations.

We assume that each device supports DVFS [21, 19] with overclocking of 10% and 20% accompanied by a linear decrease in tRC and a linear increase in Vdd. Vdd changes affect active, background and refresh power [32].

Config.	Bandwidth		Rd req		Wr req		Rd resp		Wr resp		Pwr
	Lanes	GB/s	Flits	ns	Flits	ns	Flits	ns	Flits	ns	W
Full(F)	16x2	50.0	1	0.64	5	3.2	5	3.2	1	0.64	0.8
Half(H)	8x2	25.0	1	1.28	5	6.4	5	6.4	1	1.28	0.4
Quarter(Q)	4x2	12.5	1	2.56	5	12.8	5	12.8	1	2.56	0.2
None(N)	0	0	1	∞	5	∞	5	∞	1	∞	0

Table 5: Offchip Link Configurations (single link).

References

[1] "Graph 500 reference implementation v2.1.4," http://www.graph500.org/referencecode.

[2] "High performance conjugate gradients v2.1," https://software.sandia.gov/hpcg/download.php.

[3] "SPEC OMP2012," http://www.spec.org/omp2012/.

[4] "SPECjbb2013," http://www.spec.org/jbb2013/.

[5] "Enhanced Intel SpeedStep technology for the Intel Pentium M processor," 2004.

[6] "Advanced configuration and power interface specification, revision 5.0," 2011.

[7] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood, "Simulating a $2M commercial server on a $2K PC," *IEEE Computer*, vol. 36, no. 2, pp. 50–57, 2003.

[8] A. R. Alameldeen and D. A. Wood, "IPC considered harmful for multiprocessor workloads," *IEEE Micro*, vol. 26, no. 4, pp. 8–17, 2006.

[9] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady, "SPEComp: A new benchmark suite for measuring parallel computer performance," ser. WOMPAT '01, 2001, pp. 1–10.

[10] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," vol. 40, no. 12, 2007.

[11] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, 2011.

[12] B. Black, "Die stacking is happening," MICRO 2013 Keynote.

[13] P. Bose, A. Buyuktosunoglu, J. Darringer, M. Gupta, M. Healy, H. Jacobson, I. Nair, J. Rivers, J. Shin, A. Vega, and A. Weger, "Power management of multi-core chips: Challenges and pitfalls," in *DATE*, 2012, pp. 977–982.

[14] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," ser. ISCA '00, 2000, pp. 83–94.

[15] J. A. Butts and G. S. Sohi, "A static power model for architects," ser. MICRO 33, 2000.

[16] R. Chappell, B. Toll, and R. Singhal, "Intel® next generation microarchitecture codename Haswell: New processor innovations," *Intel Developer Forum*, 2012.

[17] R. Cypher, "Apparatus and method for determining stack distance including spatial locality of running software for estimating cache miss rates based upon contents of a hash table," *US7366871*, 2008.

[18] G. B. Dantzig, "Discrete-variable extremum problems," *Operations Research*, vol. 5, no. 2, pp. 266–288, 1957.

[19] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," ser. ICAC '11, 2011, pp. 31–40.

[20] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and memory system DVFS in server systems," ser. MICRO 45, 2012.

[21] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: active low-power modes for main memory," ser. ASPLOS '11, 2011, pp. 225–238.

[22] Y. Eckert, S. Manne, M. J. Schulte, and D. A. Wood, "Something old and something new: P-states can borrow microarchitecture techniques too," ser. ISLPED '12, 2012, pp. 385–390.

[23] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," ser. ICS '05, 2005, pp. 293–302.

[24] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.

[25] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, "Thermal response to DVFS: analysis with an Intel Pentium M," ser. ISLPED '07, 2007, pp. 219–224.

[26] J. L. Henning, "SPEC CPU2006 benchmark descriptions," vol. 34, no. 4, pp. 1–17, 2006.

[27] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," ser. MICRO 39, 2006.

[28] S. Jahagirdar, V. George, I. Sodhi, and R. Wells, "Power management of the third generation Intel Core micro architecture formerly codenamed Ivy Bridge," in *Hot Chips 24*, 2012.

[29] K. Ma, X. Wang, and Y. Wang, "DPPC: Dynamic power partitioning and capping in chip multiprocessors," ser. ICCD '11, 2011, pp. 39–44.

[30] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, pp. 92–99, 2005.

[31] HMC Consortium, "Hybrid memory cube specification 1.0," 2013.

[32] MICRON, "TN-41-01: Calculating memory system power for DDR3," 2007.

[33] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," ser. PACT '08, 2008, pp. 177–186.

[34] T. N. Mudge, "Power: A first class design constraint for future architecture and automation," ser. HiPC '00, 2000, pp. 215–224.

[35] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar, "Power and thermal management in the Intel Core Duo processor," *Intel Technology Journal*, vol. 10, 2006.

[36] L. Nogueira, L. M. Pinho, and J. Coelho, "A feedback-based decentralised coordination model for distributed open real-time systems," *Journal of Systems and Software*, vol. 85, no. 9, pp. 2145 – 2159, 2012.

[37] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hot Chips 23*, 2011.

[38] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: coordinated multi-level power management for the data center," ser. ASPLOS XIII, 2008, pp. 48–59.

[39] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the Intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[40] R. Sen and D. A. Wood, "Reuse-based online models for caches," in *SIGMETRICS*, 2013.

[41] T. Shyamkumar, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," Hewlett Packard Labs, Tech. Rep. HPL-2008-20, 2008.

[42] "Intel Turbo Boost technology in Intel Core microarchitecture (Nehalem) based processors," 2008.

[43] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller, "SHIP: Scalable hierarchical power control for large-scale data centers," ser. PACT '09, 2009, pp. 91–100.

[44] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. Rubio, F. Rawson, and J. Carter, "Architecting for power management: The IBM POWER7 approach," in *HPCA*, 2010, pp. 1 –11.