Portable, Scalable, per-Core Power Estimation for Intelligent Resource Management

Bhavishya Goel*, Sally A. McKee*, Roberto Gioiosa†, Karan Singh‡, Major Bhadauria‡, and Marco Cesati§
*Chalmers University of Technology, SE
†Barcelona Supercomputer Center, ES
‡Cornell University, USA
§University of Rome–Tor Vergata, IT

Email: goelb@student.chalmers.se, mckee@chalmers.se, roberto.gioiosa@bsc.es, cesati@uniroma2.it

***Abstract*—Performance, power, and temperature are now all first-order design constraints. Balancing power efficiency, thermal constraints, and performance requires some means to convey data about real-time power consumption and temperature to intelligent resource managers. Resource managers can use this information to meet performance goals, maintain power budgets, and obey thermal constraints. Unfortunately, obtaining the required machine introspection is challenging. Most current chips provide no support for per-core power monitoring, and when support exists, it is not exposed to software. We present a methodology for deriving per-core power models using sampled performance counter values and temperature sensor readings. We develop application-independent models for four different (four- to eight-core) platforms, validate their accuracy, and show how they can be used to guide scheduling decisions in power-aware resource managers. Model overhead is negligible, and estimations exhibit 1.1%-5.2% per-suite median error on the NAS, SPEC OMP, and SPEC 2006 benchmarks (and 1.2%-4.4% overall).**

I. Introduction

Power and temperature have joined performance as first-order system design constraints. All three influence each other, and together they affect architectural and packaging choices. Power consumption characteristics further influence operating cost, reliability, battery lifetime, and device lifetime. Balancing power efficiency and thermal constraints with performance requires intelligent resource management, and achieving that balance requires real-time power consumption and temperature information broken down according to resource, together with software and hardware that can leverage such information to enforce management policies.

One logical place to institute intelligent resource management with respect to power, performance, and temperature for chip multiprocessor (CMP) systems is at the level of individual cores. Measuring run-time power of a single core is problematic, though. Current chips do not support it. Power meters only report total consumption for everything behind a single power cable, and even if such aggregate data were sufficient, the use of meters becomes completely infeasible as machines scale up: coordinating output and feedback from thousands of meters would require a separate (super) computing system. Cycle-level system simulators provide in-depth information, but are extremely time consuming and prone to error. Power models implemented on top of the architectural abstractions in such simulators are inherently inaccurate [19], and are impossible to verify when attempting to assess new architectural designs. Hardware could be enhanced to measure the current and power draw of a CPU socket, but per-core measurement is difficult when cores share a power plane. Embedding measurement devices on-chip is usually infeasible. Even when measurement facilities exist — e.g., the Intel Core i7 [16] features per-core power monitoring at the chip-level — they are rarely exposed to the user. Indeed, power sensing, actuation, and management support is more often implemented at the blade level with a separate computer monitoring output [14], [21].

On the other hand, temperature sensors are now commonly available per core, and processor architectures generally support some number of performance monitoring counters (PMCs). We use data from these sensors and counters to generate accurate, per-core power estimation models. We create these models precisely because we lack direct information on power consumed.

Previous work uses PMCs to estimate power on uniprocessors and SMPs [18], [10], [24], [22]. Here we estimate power for both single-threaded and multithreaded programs on CMPs, observing median errors (across all benchmarks) of 1.2%, 2.9%, 3.8%, and 4.4% for a quad-core Intel Q6600, an eight-core Intel E5430, a quad-core AMD Phenom, and an eight-core AMD Opteron 8212, respectively. Corresponding overall mean errors are 2.2%, 3.4%, 5.0%, and 4.9%. Results for Intel Core2 Duo and Core i7 platforms are qualitatively similar, and are omitted due to space considerations. We then use our online estimation to help manage workloads and frequencies dynamically.

In this paper we develop a model that uses performance counters and temperature to generate accurate per-core power estimates in real time, with no off-line profiling or tracing. The approach is workload-independent and requires no application modification. We show the effectiveness of our models for driving power-aware scheduling and frequency scaling: schedulers whose decisions are informed by these power models can enforce stricter power budgets, usually with little or no loss in performance. Finally, we

demonstrate portability to four CMP platforms. Studying sources of model error highlights the need for hardware support for power-aware resource management, such as fine-grained power sensors across the chip and more accurate temperature information. Our approach nonetheless shows promise for balancing performance, power, and thermal requirements for platforms from embedded real-time systems to consolidated data centers, and even to supercomputers.

II. Methodology

Building an analytic software model of per-resource power consumption requires some form of machine introspection. Any model must be based on knowledge of how and when a resource is being used. The resource on which we focus here is the processor core, and we leverage nearly ubiquitous performance monitoring counters (PMCs) to glean information about microarchitectural activity.

To build the model, we must 1) choose appropriate PMCs to reflect major activities consuming power, and 2) generate a mathematical model capturing relationships between chosen PMCs and measured system power. Other PMC-based power models use counters chosen *a priori* [22], [24]. Instead, we formalize the relationship between observed counter activity and measured system power with statistical analysis, choosing PMCs that correlate most strongly with measured power. We find that the best set of counters is unique to each system. We sample those PMCs while running microbenchmarks to exercise the microarchitecture and then develop a linear regression model to fit collected data to measured power values. We produce a verifiable system power estimate by adding the "uncore" power (from components external to the cores) to the per-core estimates.

Note that hardware performance counters have limitations. For instance, most Intel platforms only support concurrent sampling of two counters. We multiplex counters, sampling two counters during each half of the time slice. A similar time-slicing strategy would be necessary were we to employ more counters than can be sampled concurrently on any architecture. Many counters report aggregate events at the chip level and not per core, thus they are not suited to our approach. The PC of the instruction triggering a counter overflow is often not the one reported. In some contexts this sample *skid* can be problematic, but in time-based sampling (used to create our models) precision with respect to instructions causing events is unimportant.

A. Choosing PMCs

Current systems include a variety of event counters. We identify those most relevant to our purposes, rank them, evaluate our choices, and create a formal model mapping observed events to measured system power.

Identifying Candidates. To choose appropriate PMCs, we consider common events that represent the heavily used parts of a core's microarchitecture. For instance, monitoring last-level cache misses allows us to track the most power-hungry memory hierarchy activity. Monitoring instructions retired along with their types allows us to follow power usage in the FPUs and integer units. Tracking total instructions retired gives information on overall performance and power. We expect out-of-order logic to contribute to power usage, as well. Even though no counter gives such information directly, monitoring resource stalls can provide insight. Stalls from issue logic may reduce power, but CPU stalls from full reservation stations, load-store queues, or reorder buffers may increase power, since the hardware attempts to extract instruction-level parallelism from the instruction stream. For example, if a fetched instruction stalls, the out-of-order logic tries to find another to execute, examining reservation stations to check each new instruction's dependences and using more dynamic power. We separate available PMCs into the smallest set that covers these contributions to dynamic power, deriving four categories (a reasonable starting place, since our systems monitor two or four counters simultaneously): *FP Units*, *Memory*, *Stalls*, and *Instructions Retired*.

Ranking Counters. We create microbenchmarks to exercise different parts of the machine. We use no code from test benchmark suites, since the model must be application independent. We explore the space spanned by the four categories and attempt to cover common cases as well as extreme boundaries. The resulting counter values have large variations ranging from zero to several billion. For example, CPU-bound benchmarks have few cache misses, and integer benchmarks have few FP operations. The microbenchmarks have large *for* loops and *case* statements branching to different code nests. We compile the microbenchmarks with no optimization to prevent redundant code removal, and run them simultaneously on each core. All behaviors should fall within the space defined by our categories, making the approach applicable to both current and future applications.

While running the microbenchmarks (once, at system startup), we collect power, and performance data for the PMCs in the four categories. We use a Watts Up Pro power meter to measure system power[1] and *pfmon* [13] to collect PMC data. We order PMCs according to Spearman's rank correlation to measure the relationship between each counter and power, selecting the top counter per category.

Even though we select event categories that attempt to capture different kinds of microarchitectural activity, we're analyzing a cohesive hardware component for which all counters track inter-related events. This means that all counters chosen will necessarily contribute some redundant information. To assess this, we select several top-ranked PMCs in each category, and analyze the correlations among the PMCs themselves. If the top-ranked counters correlate strongly with other counters, we investigate counters that

[1]Note that we leave the meter plugged in to gauge model accuracy during normal operation, but when not testing accuracy, the meter is no longer required once the model is formed.

correlate less strongly with both power and other candidate counters, and evaluate accuracy of several alternate models. For the systems we study here, correlation of observed event rates and measured power is the strongest criterion for choosing PMCs: alternative models that try to leverage less redundant information (but perhaps less information overall) yield poorer models. In the absence of a mathematical formula for evaluating information redundancy versus power contributions of a set of events, this sanity check helps ensure that we're generating high quality models.

Note that model accuracy depends on the PMCs available on a given platform. If available PMCs do not sufficiently represent the microarchitecture, model accuracy will suffer. For example, the AMD Opteron 8212 supports no single counter giving total floating point operations. Instead, separate PMCs track different types of floating point operations. We therefore choose the one most highly correlated with power. Model accuracy would likely improve if a single PMC reflecting all floating point operations were available.

B. Forming the Machine Model

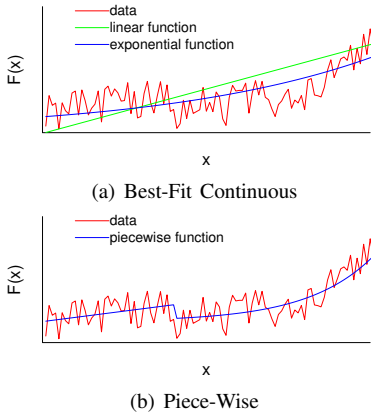
Having identified events that contribute significantly to consumed power, we create a formalism to map observed microarchitectural activity and measured temperatures to measured power draw. We re-run the microbenchmarks sampling just the chosen PMCs, collecting power and temperature data at each sampling interval. We normalize each time-sampled PMC value, e_i, to the elapsed cycle count to generate an *event rate*, r_i. We then map rise in core temperature, T, and the observed event rates, r_i, to core power, P_{core}, via a piece-wise model based on multiple linear regression, as in Equation 1. We apply linear weights to transformations of event rates, as in Equation 2.

$$\hat{P}_{core} = \begin{cases} F_1(g_1(r_1), \cdots, g_n(r_n), T), & if \text{ condition} \\ F_2(g_1(r_1), \cdots, g_n(r_n), T), & else \end{cases} \quad (1)$$

$$\text{where} \quad r_i = e_i/(cycle\ count), T = T_{current} - T_{idle}$$

$$F_n = p_0 + p_1 * g_1(r_1) + ... + p_n * g_n(r_n) + p_{n+1} * T \quad (2)$$

We use transformations to increase linearity or to achieve constant variance in the regression equation (to make the model more amenable to linear regression and to improve estimation accuracy). Since Spearman's rank correlation is independent of frequency distributions, transformations can be linear, inverse, logarithmic, exponential, or square root. We choose a piece-wise linear function because we observe significantly different behavior for low PMC values. This lets us retain the simplicity of linear regression and to capture more detail in the core power function. We examine counter values plotted against power measurements to find opportunities for transformations, as well as to identify candidate locations for splitting the data to form the piece-wise model. For example, for the (hypothetical) data in Figure 1(a), neither a linear nor exponential transformation fits well. By breaking the data into parts, we create the better-fitting piece-wise combination in Figure 1(b).



(a) Best-Fit Continuous

(b) Piece-Wise

Figure 1. Examples of Poor Continuous and Better Piece-Wise Approximation Functions

We use a least squares estimator to determine weights for function parameters, yielding the piece-wise linear model in Equation 3 (Figure 2). The first part of the equation estimates power when the L2 counter is very low, and the second estimates power for the rest of the space.

C. Applying the Model

System power is based on measuring the power supply's current draw from the power outlet when the machine is idle. When we cannot find published values for idle processor power, we sum power draw when powered off and power saved by turning off cdrom, floppy, and hard disk drives; removing all but one memory DIMM; and disconnecting fans. We subtract idle core power from idle system power to get *uncore* (baseline without processor) power. Change in the uncore power itself (due to DRAM or hard drive accesses, for instance) is included in the model estimations. Including temperature as a model input accounts for variation in uncore static power. We always run in the active power state (C0): experimenting with other power states is future work.

III. Experimental Results

We evaluate our models by estimating per-core power for the SPEC 2006 [29], SPEC-OMP [28], [4], and NAS [6] benchmark suites. We use gcc 4.2 to compile our benchmarks for a 64-bit architecture with optimization flags enabled, and run all benchmarks to completion. We use the *pfmon* utility from the *perfmon2* library [13] to access hardware performance counters. Table I gives system details. Idle processor temperature is 36°C for both the Intel Q6600 and AMD Phenom platforms. Idle system power is 141.0W for the Intel Q6600, and 84.1W for the AMD Phenom. We subtract 38.0W [1] idle processor power to obtain an uncore power of 103W for the Q6600, and subtract 20.1W [2] idle processor power to obtain an uncore power of 64.0W

$$\hat{P}_{core} = \begin{cases} 7.699 + 0.026 * log(r_1) + 8.458 * r_2 + -3.642 * r_3 + 14.085 * r_4 + 0.183 * T, & r_1 < 10^{-6} \\ 5.863 + 0.114 * log(r_1) + 1.952 * r_2 + -1.684 * r_3 + 0.110 * log(r_4) + 1.044 * T, & r_1 \geq 10^{-6} \end{cases} \quad (3)$$

$$where \quad r_i = e_i/2200000000 \ (1s = 2.2B \ cycles), T = T_{current} - T_{idle}$$

Figure 2. Piece-Wise Linear Function for Core Power

for the Phenom. Idle processor temperature for the Intel E5430 system is 45°C, and uncore power is 122.0W. Idle processor temperature for the AMD 8212 is 33°C, and uncore power is 90W. Table II lists chosen counters. We use the *sensors* utility from the *lm-sensors* library to obtain core temperatures, and we use a Watts Up Pro power meter [12] to gather power data. This meter is accurate to within 0.1W, and it updates once per second. We can thus only verify our power estimates at the granularity of one second.

We test our models for both multithreaded and single-threaded applications on our four platforms. We assess model error by comparing our system power estimates to power meter output (which our power meter limits to a one-second granularity). Then we incorporate the power model into a proof-of-concept, power-aware resource manager (a user-level meta-scheduler) designed to maintain a specified power envelope. The meta-scheduler manages processes non-invasively, requiring no modifications to the applications or OS. It does so by suspending/resuming processes and, where supported, applying dynamic voltage/frequency scaling (DVFS) to alter core clock rates. For these experiments, we degrade the system power envelope by 5%, 10%, or 15%. Lower envelopes render cores inactive, and we do not consider them. Finally, we incorporate the model in a kernel scheduler, implementing a pseudo power sensor per core. The device does not exist in hardware, but is simulated by the power model module. Reads to a pseudo-device retrieve the power estimate computed most recently.

A. Computation Overhead

If computing the model is expensive, its use becomes limited to course timeslices. In this experiment we study the overhead of our power model in order to evaluate its use in an OS task scheduler. The scheduler we use is specifically tailored to High Performance Computing (HPC) applications, which require that the OS introduce little or no overhead. In most cases, it delivers better performance with better predictability, and it reduces OS noise [8], [9]. The model's overhead depends on 1) the frequency with which the per-core power is updated, and 2) the complexity of the operations required to calculate the model. We calculate overhead by measuring execution time for our kernel scheduler running with and without reading the PMCs and temperature sensors. We vary the sample period from one minute to 10 msec. We time applications for five runs and take the average (differences among runs are within normal execution time variation for real systems). Table III gives measured times for the scheduler with and without evaluating the model. These data show that computing the model adds no measurable overhead, even at 10ms timeslices.

B. Estimation Error

In our experiments, we run multithreaded benchmarks with one thread per core, and single-threaded benchmarks with an instance on each core. Data are calculated per core, and errors are reported across all cores. Figure 3 through Figure 6 show percentage error for the NAS, SPEC-OMP, and SPEC 2006 applications on all systems. Figure 7 through Figure 10 show standard deviation of error for each benchmark suite on the Intel Q6600, Intel E5430, AMD Phenom, and AMD Opteron platforms. The occasional high standard deviations illustrate the main problem with our current infrastructure: instantaneous power measurements once per second do not reflect continuous performance counter activity since the last meter reading.

Estimation error ranges from 0.3% (*leslie3d*) to 7.1% (*bzip2*) for the Intel Q6600 system, from 0.3% (*ua*) to 7.0% (*hmmer*) for the Intel E5430 system, from 1.02% (*bt*) to 9.3% (*xalancbmk*) for the AMD Phenom system, and from 1.0% (*bt.B*) to 10.7% (*soplex*) for the AMD Opteron 8212 system. Our model exhibits median error of 1.6%, 1.6%, and 1.1% for NAS, SPEC-OMP, and SPEC 2006, respectively, on the Q6600. Corresponding errors per suite are 3.9%, 3.5%, and 2.8% on the E5430, 4.5% 5.2%, and 3.5% on the Phenom, and 2.6%, 3.4%, and 4.8% on the 8212. On the Intel Q6600, only five (out of 45) applications exhibit median error exceeding 5%; on the Intel E5430, only six exhibit error exceeding 5%; on the AMD Phenom, eighteen exhibit error exceeding 5%; and on the AMD Opteron 8212, thirteen exhibit error exceeding 5%. These errors are not excessive, but lower is better: identifying causes of error and refining the model are ongoing work. For instance, the Opteron 8212 temperature sensor data appear inaccurate: we measure different idle temperatures for different cores. We achieve good results despite the inaccuracies by underestimating idle temperature and using that value consistently for model training, testing, and dynamic calculation. Prediction errors are not clustered, but are spread throughout application execution. Monitoring PMCs at a finer granularity reveals that our meter's one-second resolution for delivering *instantaneous* power measurements prevents capturing counter activity between samples.

Figure 11 shows model coverage via Cumulative Distribution Function (CDF) plots for the suites. On the Q6600,

	Intel Q6600	Intel Xeon E5430	AMD Phenom 9500	AMD Opteron 8212
Cores/Chips	4, dual dual-core	8, dual quad-core	4	8, quad dual-core
Frequency (GHz)	2.4	2.0, 2.66	1.1, 2.2	2.0
Process (nm)	65	45	65	90
L1 Instruction	32 KB 8-way	32 KB 8-way	64 KB 2-way	64 KB 2-way
L1 Data	32 KB 8-way	32 KB 8-way	64 KB 2-way	64 KB 2-way
L2 Cache	4 MB 16-way shared	6 MB 16-way shared	512 KB 8-way exclusive	1024 KB 16-way exclusive
L3 Cache	N/A	N/A	2 MB 32-way shared	N/A
Memory Controller	off-chip, 2 channel	off-chip, 4 channel	on-chip, 2 channel	on-chip, 2 channel
Main Memory	4 GB DDR2-800	8 GB DDR2-800	4 GB DDR2-800	16 GB DDR2-667
Bus (MHz)	1066	1333	1100, 2200	1000
Max TDP (W)	105	80	95	95
Linux Kernel	2.6.27	2.6.27	2.6.25	2.6.31

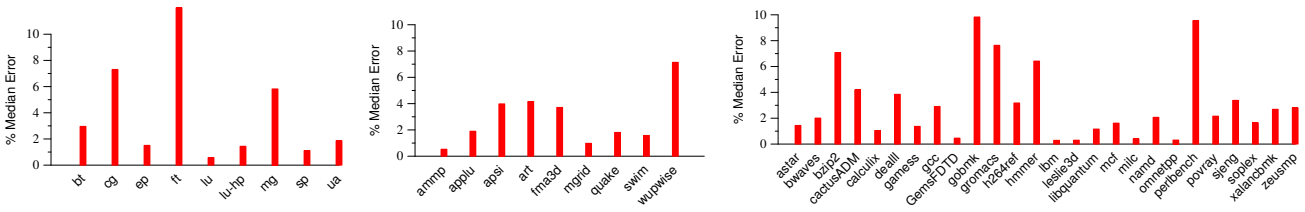
Table I
MACHINE CONFIGURATION PARAMETERS

Category	Intel Q6600	Intel E5430	AMD Phenom 9500	AMD Opteron 8212
Memory	L2_LINES_IN	LAST_LEVEL_CACHE_MISSES	L2_CACHE_MISS	DATA_CACHE_ACCESSES
Instructions Executed	UOPS_RETIRED	UOPS_RETIRED	RETIRED_UOPS	RETIRED_INSTRUCTIONS
Floating Point	X87_OPS_RETIRED	X87_OPS_RETIRED	RETIRED_MMX_AND_FP_INSTRUCTIONS	DISPATCHED_FPU:OPS_MULTIPLY
Stalls	RESOURCE_STALLS	RESOURCE_STALLS	DISPATCH_STALLS	DECODER_EMPTY

Table II
PMCS SELECTED FOR EACH PLATFORM

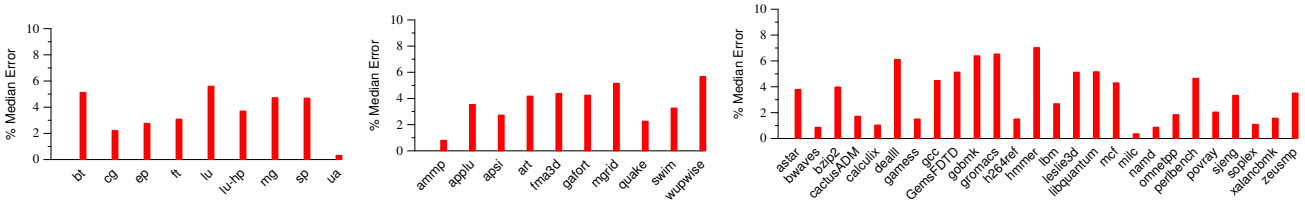
Benchmark	baseline	model (10msec)	model (100msec)	model (1sec)
ep.A serial	35.68	35.57	36.04	35.59
ep.A OMP	4.84	4.89	4.77	4.74
ep.A MPI	4.77	4.72	4.73	4.75
cg.A serial	5.82	5.83	5.83	5.83
cg.A OMP	1.95	1.95	1.95	1.95
cg.A MPI	2.19	2.20	2.20	2.20
ep.B serial	146.53	146.52	145.52	146.77
ep.B OMP	19.45	19.33	19.35	19.54
ep.B MPI	18.95	19.41	19.12	19.18
cg.B serial	559.58	560.50	560.11	560.10
cg.B OMP	91.29	92.64	96.90	89.92
cg.B MPI	79.11	79.18	79.18	79.05

Table III
SCHEDULER BENCHMARK TIMES FOR SAMPLE NAS APPLICATIONS ON THE AMD OPTERON 8212 (SEC)



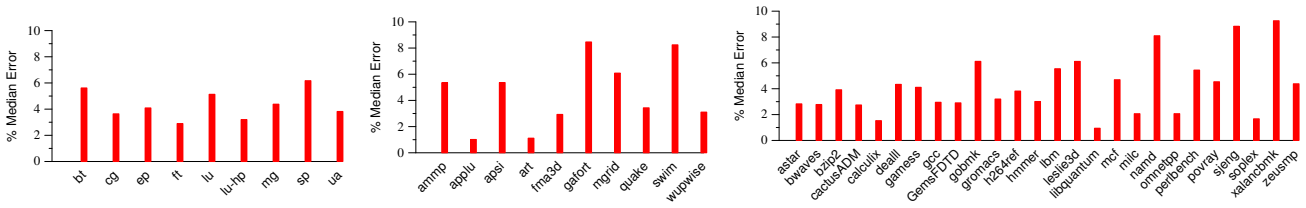
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3. Median Estimation Error for Intel Q6600 system



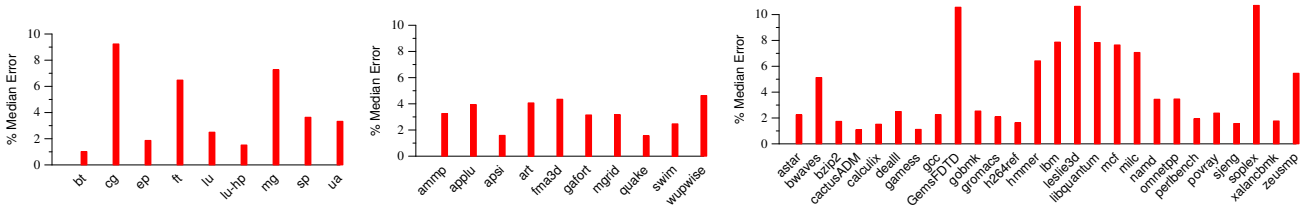
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 4. Median Estimation Error for Intel E5430 system



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 5. Median Estimation Error for the AMD Phenom 9500



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 6. Median Estimation Error for the AMD Opteron 8212

85% of estimates have less than 5% error, and 98% have less than 10%. On the E5430, 73% have less than 5% error, and 99% less than 10%. On the Phenom, 59% have less than 5% error, and 92% less than 10%. On the 8212, 37% have less than 5% error, and 76% have less than 10%. The vast majority of estimates exhibit very small error.

Figure 12 through Figure 15 compare measured to estimated power for our experimental platforms. Values are calculated as the mean of power over an application's execution. These data show that our models do not consistently under- or over-estimate power across benchmark suites. Figure 16 shows that estimates closely follow measured data throughout *gcc* execution on the Intel Q6600, illustrating why the model underestimates power by 0.7W on average for this application. We validate by running identical workloads on all cores, so errors on individual cores are likely to be similar and cumulative (instead of canceling each other).

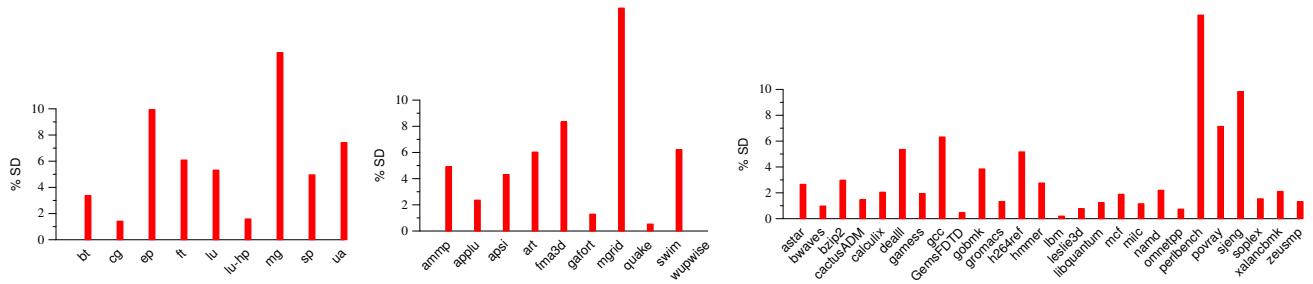
C. Live Power Management

We use per-process power estimates to enforce a given power budget for multiprogrammed workloads on each CMP. Table IV presents three sets of multiprogrammed workloads exhibiting varying levels of *computational intensity*, or the ratio of instructions retired to cache misses.

We experiment with two example policies for our meta-scheduler. The policies themselves are not particularly interesting, but they try to optimize for different goals, and together they provide a proof of concept with respect to the use of our power models in resource management. When the power envelope is breached on a system supporting DVFS, the meta-scheduler reduces the frequency of the core running the process selected by the policy. When DVFS is unavailable or exhausted, the meta-scheduler suspends the selected process. When there is power slack, the meta-scheduler resumes the process selected by the policy, or if all processes are running and DVFS is supported, it increases the frequency of the core running the selected process. The *max inst/watt* policy attempts to achieve the most energy-efficiency under a given power envelope by selecting the process with the fewest instructions committed per watt of power consumed when the power budget is exceeded, and the one with the most instructions committed per watt when there is slack. The *per-core fair* policy tries to allocate available power equitably per core. When the envelope is breached, the policy selects the process with highest average consumed power. When there is slack, the policy selects the process with lowest average consumed power.

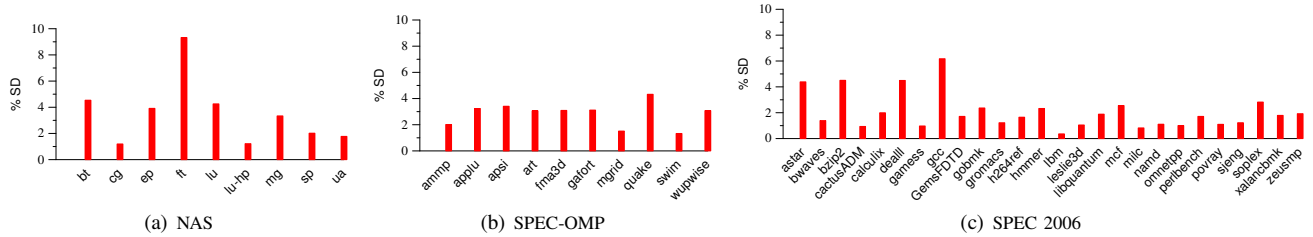
We perform two sets of experiments. For the first set, we use the 2.2 GHz AMD Phenom and 2.4 GHz Intel Q6600 systems. We selectively suspend/resume processes to remain within the system power envelope. For the second set, we use the AMD Phenom and Intel E5430 systems, using DVFS to vary frequencies between 1.1 and 2.2. GHz, and between 2.0 and 2.67 GHz, respectively. We form separate power models for the lower frequencies. The power manager can thus better implement policy decisions by estimating power for either frequency. Since the E5430 platform supports DVFS per (quad-core) chip, and not per core, we independently vary frequencies among the two chips. If all core frequencies have been reduced but the power envelope is still breached, we suspend processes to reduce power. We compare against runtimes with no enforced power envelope.

Figure 17 and Figure 18 show examples of the two policies with different power envelopes for the Intel Q6600 and AMD Phenom systems. Measured and estimated power correlate well. We follow the power envelope, and we do so



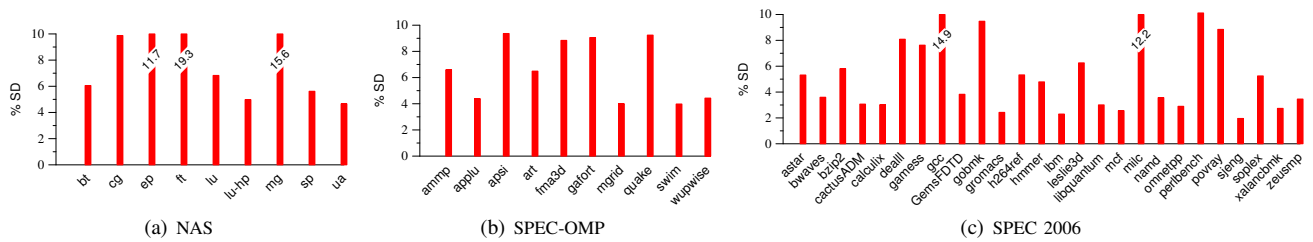
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 7. Standard Deviation of Error for Intel Q6600 system



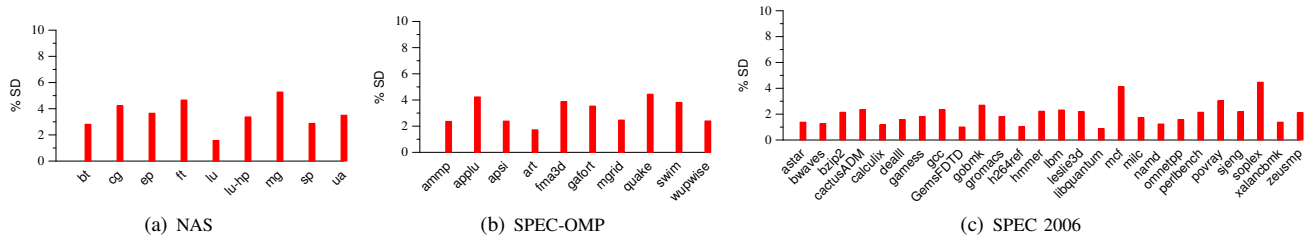
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 8. Standard Deviation of Error for Intel E5430 system



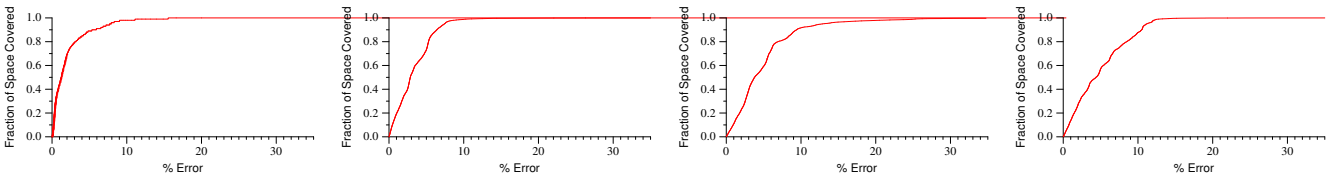
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 9. Standard Deviation of Error for the AMD Phenom 9500



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 10. Standard Deviation of Error for the AMD Opteron 8212

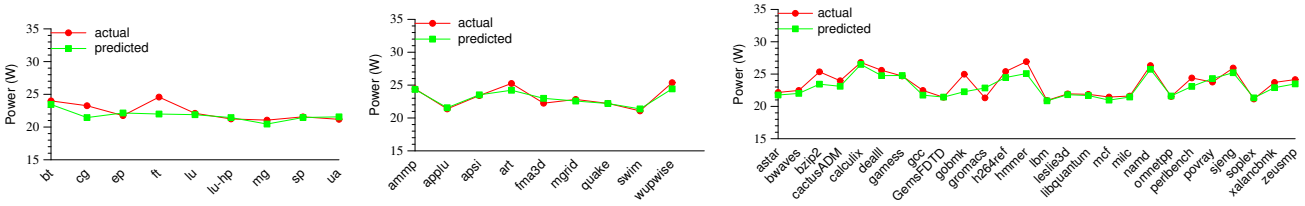


(a) Intel Q6600 (b) Intel E5430 (c) AMD 9500 (d) AMD 8212

Figure 11. Cumulative Distribution Function (CDF) Plots Showing Fraction of Space Predicted (y axis) under a Given Error (x axis) for Each System

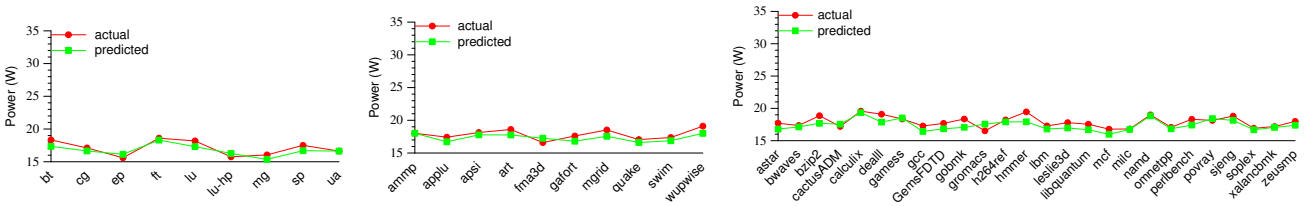
Benchmark Set	Intel Q6600 and AMD Phenom	Intel E5430
CPU bound	ep, gamess, namd, povray	calculix, ep, gamess, gromacs, h264ref, namd, perlbench, povray
Moderate	art, lu, wupwise, xalancbmk	bwaves, cactusADM, fma3d, gcc, leslie3d, sp, ua, xalancbmk
Memory bound	astar, mcf, milc, soplex	applu, astar, lbm, mcf, milc, omnetpp, soplex, swim

Table IV
EXPERIMENTAL MULTIPROGRAMMED WORKLOADS



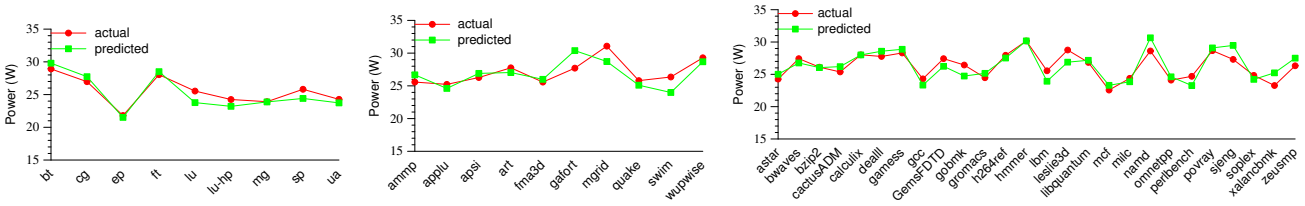
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 12. Estimated vs. Measured Error for Intel Q6600 system



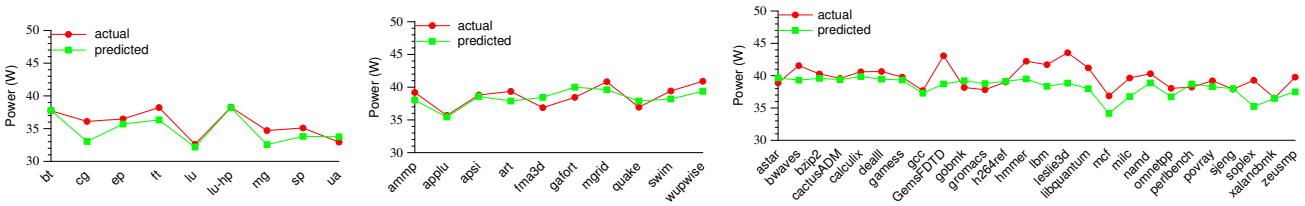
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 13. Estimated vs. Measured Error for Intel E5430 system



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 14. Estimated vs. Measured Error for the AMD Phenom 9500



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 15. Estimated vs. Measured Error for the AMD Opteron 8212

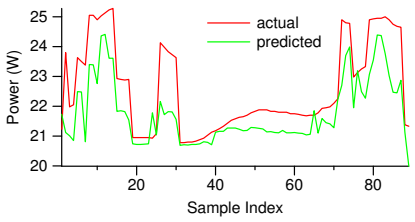
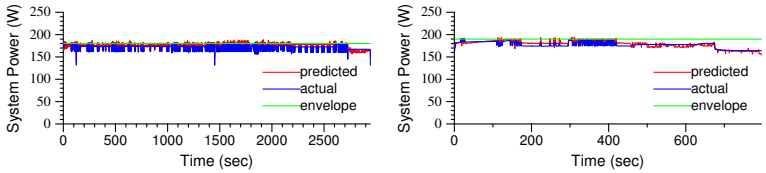
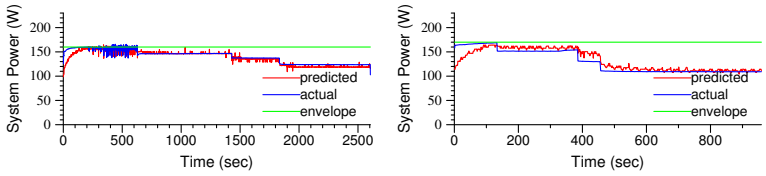


Figure 16. Detail of Estimated vs. Measured Error for *gcc* on the Intel Q6600



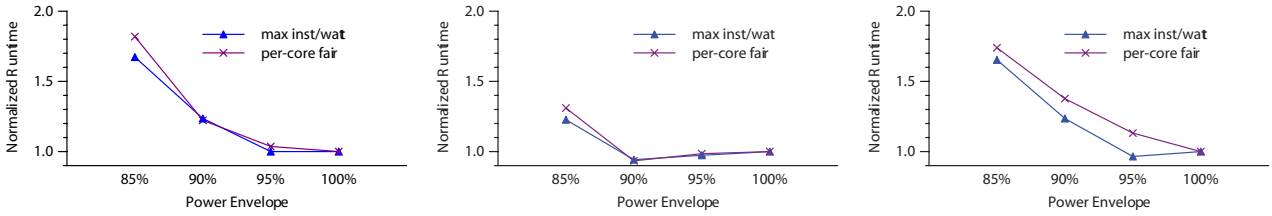
(a) Moderate, Max Inst/Watt, 90% (b) CPU-bound, Per-Core Fair, 95%

Figure 17. Estimation Error for Different Workloads and Envelopes on the Intel Q6600



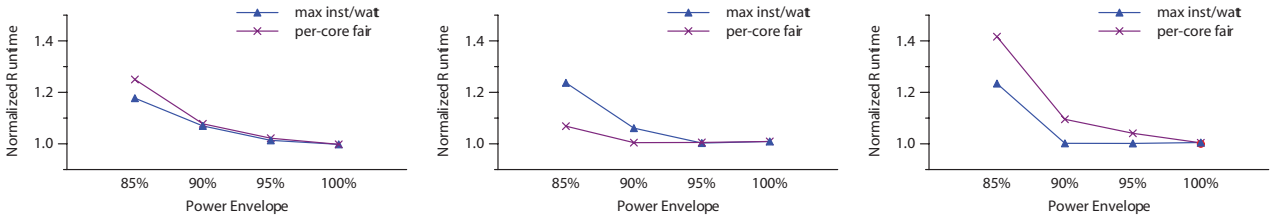
(a) Moderate, Max Inst/Watt, 90% (b) CPU-bound, Per-Core Fair, 95%

Figure 18. Estimation Error for Different Workloads and Envelopes on the AMD Phenom



(a) CPU-bound (b) Memory-bound (c) Moderate

Figure 19. Runtimes for Workloads on the Intel Q6600



(a) CPU-bound (b) Memory-bound (c) Moderate

Figure 20. Runtimes for Workloads on the AMD Phenom

entirely on the basis of our estimation-based power manager. This obviates the need for a power meter, and is a useful tool for software control of total system power. Figure 19 and Figure 20 show normalized runtimes for the two sample policies and all power envelopes on both systems. This meta-scheduler only suspends/resumes processes. Performance sometimes improves for the 95% and 90% power envelopes, since with no power envelope, all processes compete for cache and memory bandwidth. When power envelopes are enforced, the meta-scheduler throttles processes to conserve power, making more cache and memory bandwidth available to the executing processes. This can speed application execution. This effect disappears for the 85% envelope: even though there is less competition, processes cannot execute at maximum speed without breaching the power envelope.

For the second set of experiments we vary power usage by choosing between frequencies of 1.1 and 2.2 GHz on the AMD Phenom system and between 2.0 and 2.67 GHz on the Intel 5430 system. If DVFS is insufficient, the meta-scheduler suspends processes. Figure 21 and Figure 22 show normalized runtimes for the two policies and all power envelopes on the AMD Phenom and E5430 systems. Note the different scales of the y axes: performance losses for shrinking power envelopes on the Phenom are much smaller with DVFS. For that machine, the performance-oriented policy maintains performance in all cases. On the Intel E5430, we lose under 4% performance for all but the CPU-bound workload under the strictest power envelope. The performance differences for the policies shows that DVFS alone does not maintain the power envelope (except in the

case of the memory bound workloads), and that the policies obviously suspend different processes for the CPU-bound and moderate workloads. Memory-bound workloads benefit most from the policy that tries to allocate power equally among cores. Likewise, those same workloads suffer little or no performance loss with stricter power budgets when DVFS and scheduling are used to enforce power budgets. Lower frequency slows computation, but since it does not impact memory performance, the non-CPU-bound workloads suffer little (and sometimes benefit).

IV. Related Work

Much previous work leverages performance counters for power estimation. Most does not report model error, and none presents results as extensively as we do here. Our work is based on the approach of Singh et al. [27], [26]; we augment that work by incorporating temperature in the models, exploiting frequency scaling, and validating the models on several more platforms.

Joseph and Martonosi [18] use PMCs to estimate power in a simulator (SimpleScalar [5]) and on real hardware (a Pentium Pro). Their hardware supports two PMCs, while they require twelve. They perform multiple benchmark runs to collect data, forming the model offline. Multiplexing twelve PMCs would require program behavior to be relatively constant across the entire sampling interval. They cannot estimate power for 24% of the chip, and so they assume peak power for those structures.

Contreras and Martonosi [10] use five PMCs to estimate power for different frequencies on an XScale system (with an in-order uniprocessor). Like Joseph and Martonosi, they gather data from multiple benchmark runs. They derive power weights for frequency-voltage pairs, and form a parameterized linear model. Their model exhibits low percentage error, like ours, but they test on only seven applications, and their methodology is only demonstrated for a single platform. This methodology, like that of Joseph and Martonosi, is not intended for on-line use.

Economou et al. [11] use PMCs to predict power on a blade server. They profile the system using application-independent microbenchmarks. The resulting model estimates power for the CPU, memory, and hard drive with 10% average error. Wu et al. [30] also use microbenchmarks to develop a model using a variable number of PMCs to estimate active power of Pentium 4 functional units. They measure CPU power with a clamp-on ammeter. Such invasive approaches are neither applicable to systems in the field, nor scalable to HPC clusters. Rajamani et al. [24] develop online power and performance models on a Pentium M system. They track a small number of PMCs (chosen *a priori*), and develop their model using existing kernels (as opposed to microbenchmarks specifically designed to exercise the counters). They present two power-management strategies (one targeting performance and the other power savings) that leverage their models for dynamic p-state control. Their policies are effective in meeting power and performance requirements. No percentage errors are reported.

Merkel and Bellosa [22] use PMCs to estimate power per processor in an eight-way SMP, shuffling processes to reduce overheating of individual processors. They do not state which PMCs they use, nor how they are chosen. Their goal is not to reduce power, but to distribute it. Their estimation method suffers less than 10% error in a linux kernel implementation.

Lee and Brooks [20] predict power via statistical inference models. They build correlations based on hardware parameters, using the most significant parameters to train their model. They profile their design space a priori, and then estimate power on random samples. This methodology requires training on the same applications for which they want to estimate power, and so their approach depends on having already sampled all applications of interest.

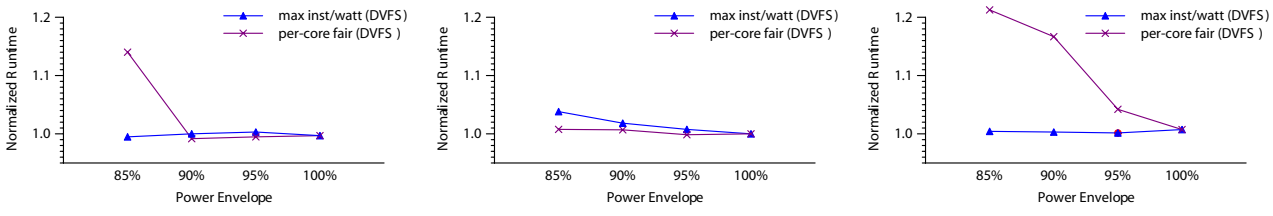
Isci et al. [17] analyze power management policies for a given power budget, performing experiments on the Turandot [23] simulator. They assume on-core current sensors to obtain per-core power, while we propose a technique to model per-core power without additional hardware. They leverage per-core and domain-wide dynamic voltage/frequency scaling (DVFS) to retain performance while remaining within a power budget. Here we explore both suspending threads and adapting frequencies to make our approach applicable to more systems.

Instead of applying DVFS per core, Rangan et al. [25] study thread placement and migration among independent voltage and frequency domains. This *thread motion* permits much finer-grain control over power management and delivers better performance than conventional DVFS for a given power budget. Grochowski et al. [15] study four methods to control energy per instruction (EPI), finding that a combination of VFS and asymmetric cores offers the most promising approach to balancing latency and throughput in an energy-constrained environment. Annavaram et al. [3] leverage this work to throttle EPI by scheduling multithreaded tasks on an asymmetric CMP, using more EPI in periods of low parallelism to make better use of a fixed power budget.

Banikazemi et al. [7] present a power-aware meta-scheduler. They exploit the built-in power monitoring hardware on an IBM blade server with eight Intel Xeon 5345 cores (four dual-core chips). They use PMC data to calculate cache occupancy, miss ratios, and CPI, deriving a performance model from which they estimate system power.

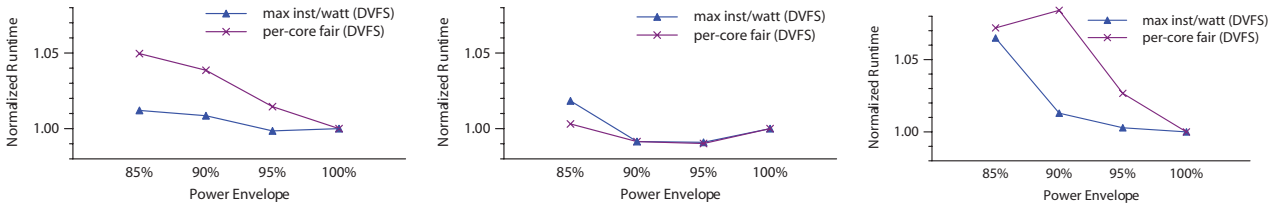
V. Conclusions and Future Work

We derive analytic, piece-wise linear power models mapping PMC values and temperature readings to power, and demonstrate their accuracy for the SPEC 2006, SPEC-OMP and NAS benchmark suites. We write microbenchmarks to exercise the PMCs in order to characterize the machine, and



(a) CPU-bound (b) Memory-bound (c) Moderate

Figure 21. Runtimes for Workloads on the AMD Phenom (with DVFS)



(a) CPU bound (b) Memory bound (c) Moderate

Figure 22. Runtimes for Workloads on the Intel E5430 system (with DVFS)

run those microbenchmarks with a power meter plugged in to generate data for building the models. We select the PMCs that correlate most strongly with measured power for our regression models. Because they are built on microbenchmark data, and not actual workload data, the resulting models are application independent. We apply the models to 45 benchmarks (including multithreaded applications) on four CMPs containing dual- or quad-core chips totaling four or eight cores. We omit data for Core2 Duo and Core i7 platforms: results are qualitatively and quantitatively similar. Previous work focuses largely on uniprocessor power models (that do not incorporate temperature), and each methodology targets a single architecture. In spite of our generality, estimation errors are consistently low across six different systems (data for four of which we include here). We observe overall median errors per machine between 1.2% and 4.4%.

We leverage our power model to perform live power management on CMP systems with and without DVFS. We suspend and resume processes based on per-core power usage, ensuring that a given power envelope is not breached. Where supported, we also scale core frequencies to remain under the power envelope. Our power managers enforce power budgets with little or no performance loss.

Estimating per-core power is challenging, since some resources are shared among cores (e.g., caches, the DRAM memory controller and off-chip memory). Current machines only permit sampling a few PMCs simultaneously per core, which necessitates multiplexing the counters for models requiring more inputs. This affects the size of the sampling interval for applying the model, something we have just begun to study. We have incorporated the model into an HPC Linux kernel scheduler, and are in the process of experimenting with different scheduling policies. Having a unified infrastructure permits a fair comparison of power-aware thread scheduling with techniques like thread motion. Tracking PMC values at finer timeslices than that for which our meter can measure power emphasizes the need for accurate power sensors on chip: our model is sufficient to be useful, but much finer resource management would be enabled by making more hardware information available.

As numbers of cores and power demands continue to grow, efficient computing requires better methods for managing individual resources. The per-core power estimation methodology presented here extends previously published models in both breadth and depth, and represents a promising tool for helping meet those challenges, both by providing useful information to resource managers and by highlighting opportunities for improving hardware support for energy-aware resource management. Such support is essential for fine-grained, power-aware resource management.

VI. Acknowledgments

This work is supported in part by the Ministry of Science and Technology of Spain (TIN-2007-60625, JCI-2008-3688), by the European Commission (IST-004408, ICT-249059), and by the United States National Science Foundation (CCF-0702616). Opinions expressed herein are solely the authors', and in no way reflect those of the funding agencies.

References

[1] Intel Core 2 Quad Q6600 (Power consumption, temperature, overclocking). http://www.behardware.com/articles/651-2/intel-core-2-quad-q6600.html, Jan. 2007.

[2] Energy Consumption: The Processor and Cool'n'Quiet Mode. www.tomshardware.com/reviews/amd-power-cpu,1925-7.html, Dec. 2008.

[3] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law through EPI throttling. In *Proc. 32nd IEEE/ACM International Symposium on Computer Architecture*, pages 298–309, June 2005.

[4] V. Aslot and R. Eigenmann. Performance characteristics of the SPEC OMP2001 benchmarks. In *Proc. European Workshop on OpenMP*, Sept. 2001.

[5] T. Austin. SimpleScalar 4.0 release note. http://www.simplescalar.com/.

[6] D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Report NAS-95-020, NASA Ames Research Center, Dec. 1995.

[7] M. Banikazemi, D. Poff, and B. Abali. PAM: A novel performance/power aware meta-scheduler for multi-core systems. In *Proc. IEEE/ACM Supercomputing International Conference on High Performance Computing, Networking, Storage and Analysis*, number 39, Nov. 2008.

[8] E. Betti, M. Cesati, R. Gioiosa, and F. Piermaria. A global operating system for HPC clusters. In *Proc. IEEE International Conference on Cluster Computing*, page 6, Aug. 2009.

[9] C. Boneti, R. Gioiosa, F. Cazorla, and M. Valero. A dynamic scheduler for balancing HPC applications. In *Proc. IEEE/ACM Supercomputing International Conference on High Performance Computing, Networking, Storage and Analysis*, number 41, Nov. 2008.

[10] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 221–226, Aug. 2005.

[11] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proc. Workshop on Modeling, Benchmarking, and Simulation*, June 2006.

[12] Electronic Educational Devices. Watts Up PRO. `http://www.wattsupmeters.com/`, May 2009.

[13] S. Eranian. Perfmon2: a flexible performance monitoring interface for Linux. In *Proc. 2006 Ottawa Linux Symposium*, pages 269–288, July 2006.

[14] M. Floyd, S. Ghiasi, T. Keller, K. Rajamani, F. Rawson, J. Rubio, and M. Ware. System power management support in the IBM POWER6 microprocessor. *IBM Journal of Research and Development*, 51(6):733–746, 2007.

[15] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. In *Proc. IEEE International Conference on Computer Design*, pages 236–243, Oct. 2004.

[16] Intel Corporation. Intel Core(TM) i7 Processor. `http://www.intel.com/products/processor/corei7/`, Dec. 2008.

[17] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proc. IEEE/ACM 40th Annual International Symposium on Microarchitecture*, pages 347–358, Dec. 2006.

[18] R. Joseph and M. Martonosi. Run-time power estimation in high-performance microprocessors. In *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 135–140, Aug. 2001.

[19] N. Kim, T. Austin, T. Mudge, and D. Grunwald. *Challenges for Architectural Level Power Modeling*, pages 317–337. Kluwer Academic Publishers, 2002. ISBM: 0-306-46786-0.

[20] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. 12th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 185–194, Oct. 2006.

[21] H.-Y. McCreary, M. Broyles, M. Floyd, A. Geissler, S. Hartman, F. Rawson, T. Rosedahl, J. Rubio, and M. Ware. Energyscale for IBM POWER6 microprocessor-based systems. *IBM Journal of Research and Development*, 51(6):775–786, 2007.

[22] A. Merkel and F. Bellosa. Balancing power consumption in multicore processors. In *Proc. ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 403–414, Apr. 2006.

[23] M. Moudgill, P. Bose, and J. Moreno. Validation of Turandot, a fast processor model for microarchitecture exploration. In *Proc. International Performance, Computing, and Communications Conference*, pages 452–457, Feb. 1999.

[24] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. Application-aware power management. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, pages 39–48, Oct. 2006.

[25] K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: Fine-grained power management for multi-core systems. In *Proc. 36th IEEE/ACM International Symposium on Computer Architecture*, June 2009.

[26] K. Singh. *Prediction Strategies for Power-Aware Computing on Multicore Processors*. PhD thesis, Cornell University, 2009.

[27] K. Singh, M. Bhadauria, and S. McKee. Real time power estimation and thread scheduling via performance counters. *Proc. Workshop on Design, Architecture and Simulation of Chip Multi-Processors*, Nov. 2008.

[28] Standard Performance Evaluation Corporation. SPEC OMP benchmark suite. `http://www.specbench.org/hpg/omp2001/`, 2001.

[29] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite. `http://www.specbench.org/osg/cpu2006/`, 2006.

[30] W. Wu, L. Jin, and J. Yang. A systematic method for functional unit power estimation in microprocessors. In *Proc. 43rd ACM/IEEE Design Automation Conference*, pages 554–557, July 2006.