AUTOMATED FULL-SYSTEM POWER CHARACTERIZATION

A NEW FRAMEWORK AUTOMATICALLY GENERATES FULL-SYSTEM MULTICORE POWERMARKS, OR SYNTHETIC PROGRAMS WITH DESIRED POWER CHARACTERISTICS ON MULTICORE SERVER PLATFORMS. THE FRAMEWORK CONSTRUCTS FULL-SYSTEM POWER MODELS WITH ERROR BOUNDS ON THE POWER ESTIMATES AND GUIDES THE DESIGN OF ENERGY-EFFICIENT AND COST-EFFICIENT SERVER AND DATA CENTER INFRASTRUCTURES.

Stijn Polfliet
Frederick Ryckbosch
Lieven Eeckhout
Ghent University

Energy and power usage are key design concerns in servers and large-scale data centers, for several reasons. Energy-related costs, for both powering and cooling the servers, have become an important component in the total cost of ownership (TCO) of this class of systems. In fact, with electricity costs trending up, energy-related costs will become an increasing part of the total cost.[1] In addition, as we become more environmentally conscious, improving computer systems' energy efficiency is key to reducing the IT industry's carbon dioxide emissions. Finally, increased power density and temperature can lead to reduced hardware reliability. To improve the energy efficiency and cost efficiency of servers and data centers, we need appropriate tools to understand power usage at the system level.

This article presents a framework to automatically characterize power consumption at the system level in server hardware. The proposed framework lets us automatically generate *powermarks*, or synthetic programs with specific power characteristics. For example, particular powermarks might strive to maximize power consumption at a given CPU usage level through a realizable program. In contrast to prior work in this area, which built CPU-centric single-threaded stressmarks,[2,3] the proposed framework constructs full-system multicore powermarks; besides stressing the CPU, the powermarks also stress main memory, network I/O, and disk I/O. Further, the framework is versatile to the hardware platform of interest because it generates powermarks in a high-level programming language. An evaluation on six hardware platforms illustrates the ability to generate powermarks that exceed the power consumed by a large set of performance benchmarks and existing torture tests.

In a subsequent step, we explore two applications for the powermarks framework. First, we use it to automatically construct full-system power models that estimate a server's total power consumption at a given utilization level. The model's key asset is that it provides meaningful error bounds on the predicted power consumption. The error bounds are determined by powermarks that maximize and minimize power usage through a realizable program. We demonstrate the model's accuracy and applicability on six hardware platforms, and we show that the models can handle simultaneous multithreading (SMT) and chip-level multiprocessing (CMP) architectures.

Second, we use powermarks to guide data center design and dimensioning. In one case

Published by the IEEE Computer Society

study, we studied the impact on the total energy cost of dimensioning the power supply units at the power usage determined by a max powermark rather than the nameplate power consumption. Depending on the hardware platform, we reported energy-cost savings ranging between 34 percent and 63 percent. Going one step further, we evaluated the idea of power capping at a preset power usage level below the max powermark by monitoring the power consumption at the server's power outlet and feeding this back to the power manager running on the server. If power consumption is about to exceed the preset power cap, the power manager artificially reduces the server's usage level by putting the server to sleep at regular intervals so that the effective power usage stays below the power cap. We demonstrate a case study illustrating the effectiveness of power capping with limited impact on overall performance.

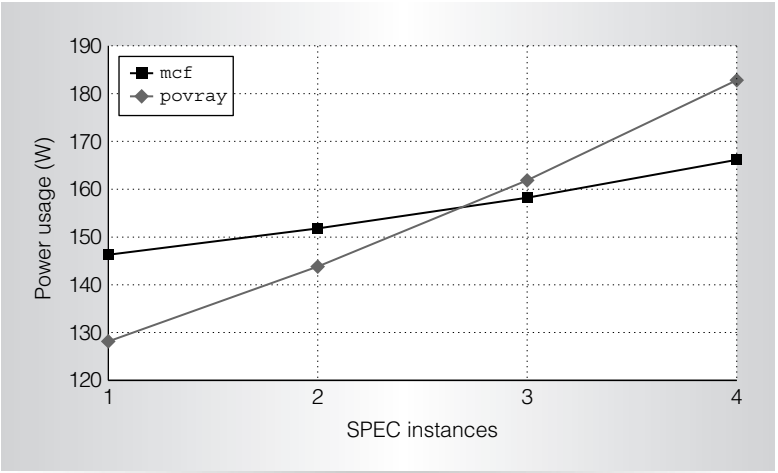

Figure 1. Power consumption as a function of the number of co-run instances for the SPEC CPU2006 `mcf` and `povray` benchmarks on the AMD Opteron Quad-Core processor system.

Powermark framework

There exists substantial prior work in stressmark generation. Several sources report how industry builds stressmarks, often called power viruses, that maximize CPU power consumption, temperature, dI/dt, and so forth.[4-6] Building stressmarks is typically done manually, and is therefore time consuming, tedious, and error prone. To speed up stressmark generation, recent work proposed frameworks for automatically generating stressmarks while focusing on the CPU[2] and both the CPU and memory.[3] These approaches don't provide full-system stressmarks, though, because they focus on the CPU and memory only. Our powermark framework aims to generate powermarks that stress the entire system, including the network interface and the disk alongside the CPU and memory. In addition, our framework targets multicore systems. (For more information on research related to our framework, see the "Related Work in Full-System Multicore Powermarks" sidebar.)

To illustrate that generating multicore stressmarks is a nontrivial extension to single-core stressmarks, we first make several observations before describing our framework in more detail. First, a benchmark that yields higher power consumption than another benchmark when run in isolation doesn't necessarily yield higher power consumption when multiple instances co-run. For example, Figure 1 shows the example benchmarks `mcf` and `povray` on our baseline AMD Opteron Quad-Core system. We observed similar results for other benchmarks. This suggests that a powermark that's optimized for a single core doesn't necessarily imply max power consumption for a multicore processor, and thus, a powermark generator must co-optimize per-core power consumption and overall chip power consumption. The reason is that contention in shared multicore resources affects per-core performance.

A second observation worth noting is that running multiple copies of the same workload doesn't yield maximum power consumption. Memory accesses to a shared state by co-executing threads yield substantially higher power consumption because of cache-coherency traffic. In our framework, memory accesses to a shared state increase power consumption substantially, up to 4.5 W (or 1.9 percent relative to the maximum power) when comparing powermarks that include versus exclude shared-state memory accesses on our baseline system. These two observations imply that a multicore powermark isn't achieved by simply co-executing multiple instances of a single-core powermark—and, in addition, complex interactions

Related Work in Full-System Multicore Powermarks

Several research topics are related to the powermark framework.

Power viruses

Several sources report on power viruses for maximizing CPU power consumption.[1-3] These power viruses are developed manually, which is time consuming, tedious, and error prone. Recently, Joshi et al. proposed a framework for automatically generating synthetic CPU power viruses from an abstract description of the workload.[4] Ganesan et al. generated power viruses that stress the CPU and memory.[5] All of these approaches focus on the CPU and memory only, generate single-threaded power viruses, and do not stress the entire system. Furthermore, transferring these power viruses from one platform to another is hard because they're written in assembly. Our powermark environment, on the other hand, generates multithreaded full-system powermarks, and the framework is easily portable across platforms.

Power benchmarking

Power benchmarking has received increased interest recently. In particular, SPEC released SPECpower ssj2008,[6] a system-level, server-side Java workload that quantifies energy efficiency under varying loads. SPECpower generates and completes a mix of transactions, and the reported throughput is the number of transactions completed per second over a fixed period of time; the workload considers 11 levels of load. Rivoire et al. presented JouleSort, a sort benchmark that reads its input from a file and writes its output to another file on a nonvolatile device.[7] There are three scale categories with 10-Gbyte, 100-Gbyte, and 1-Tbyte records, and the benchmark aims at covering multiple domains, from embedded to mobile to the server domain. These power benchmarks don't strive at maximizing power consumption as the powermark framework does.

Power modeling

Power modeling has garnered significant interest over the past decade. Some efforts focus on simulation approaches for design exploration purposes, such as Wattch.[8] Others have proposed power modeling approaches for real hardware. Isci et al. use hardware performance counters for constructing a CPU component power model.[9] Fan et al. use a more coarse-grained power model that builds on CPU use as the basis for modeling CPU power consumption.[10] Rivoire et al. proposed a framework for estimating full-system power consumption using CPU performance counters.[11] Our full-system power model fits in the category of coarse-grained full-system models and provides bounds on the power estimates.

Synthetic benchmarks

Synthetic benchmarks have a long history, going back to Whetstone and Dhrystone, manually crafted benchmarks that aimed at representing real workloads. Manually building benchmarks is both tedious and time-consuming. More recent work focused on automatically generating synthetic benchmarks from an abstract workload description,[12-14] the primary motivation being to speed up simulation by generating short-running

exist among co-executing threads through shared memory and resources. Further, powermarks are platform specific, so an automated approach to constructing powermarks is desirable. This is our motivation for a framework that can explore this space effectively and efficiently.

Framework overview

Figure 2 illustrates our powermark framework. The central piece in the framework is a workload generator that generates a synthetic benchmark from a set of workload characteristics. The synthetic benchmark is built from a program skeleton; the workload characteristics give the synthetic benchmark its behavioral characteristics of interest.

Because finding the workload characteristics that maximize or minimize power consumption is nontrivial, owing to complex interactions between workload behavior and the target hardware, we employ a genetic algorithm to automatically evolve toward a powermark with desired power characteristics. The genetic algorithm starts off with a number of randomly generated workload profiles, each of which collects several workload characteristics. Each workload profile serves as input to a workload generator, which in turn generates a synthetic benchmark from it with the desired workload characteristics. Each synthetic benchmark then runs on real hardware (or a simulator), and power statistics are collected. If the measured power statistics match the desired power statistics for at least one of the synthetic benchmarks, we've found our powermark and the optimization process ends—the synthetic benchmark is our powermark. If not, we yield to the genetic algorithm to explore the workload space and try out new workload profiles.

Powermark generator

The workload generator takes a workload profile as input and generates a synthetic benchmark. The synthetic benchmark follows a particular skeleton with several

synthetic benchmarks representative of long-running performance benchmarks. Our powermark framework builds on this body of work and generates synthetic benchmarks with specific power characteristics.

References

1. W. Felter and T. Keller, *Power Measurement on the Apple Power Mac G5,* tech. report RC23276, IBM, 2004.
2. M.K. Gowan, L.L. Biro, and D.B. Jackson, ''Power Considerations in the Design of the Alpha 21264 Microprocessor,'' *Proc. 35th Design Automation Conf.,* ACM Press, 1998, pp. 726-731.
3. R. Vishmanath et al., ''Thermal Performance Challenges from Silicon to Systems,'' *Intel Technology J.,* vol. 4, no. 3, 2000; http://download.intel.com/technology/itj/q32000/pdf/thermal_perf.pdf.
4. A.M. Joshi et al., ''Automated Microprocessor Stressmark Generation,'' *Proc. Int'l Symp. High-Performance Computer Architecture,* IEEE Press, 2008, pp. 229-239.
5. K. Ganesan et al., ''System-Level Max Power (SYMPO): A Systematic Approach for Escalating System Level Power Consumption Using Synthetic Benchmarks,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* ACM Press, 2010, pp. 19-28.
6. K.-D. Lange, ''Identifying Shades of Green: The SPECpower Benchmarks,'' *Computer,* vol. 42, no. 3, 2009, pp. 95-97.
7. S. Rivoire et al., ''JouleSort: A Balanced Energy-Efficiency Benchmark,'' *Proc. 2007 ACM SIGMOD Int'l Conf. Management of Data,* ACM Press, 2007, pp. 365-376.
8. D. Brooks, V. Tiwari, and M. Martonosi, ''Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,'' *Proc. 27th Ann. Int'l Symp. Computer Architecture,* ACM Press, 2000, pp. 83-94.
9. C. Isci and M. Martonosi, ''Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data,'' *Proc. 36th Ann. Int'l Symp. Microarchitecture,* IEEE CS Press, 2003, pp. 93-104.
10. X. Fan, W.-D. Weber, and L.A. Barroso, ''Power Provisioning for a Warehouse-Sized Computer,'' *Proc. Int'l Symp. Computer Architecture,* ACM Press, 2007, pp. 13-23.
11. S. Rivoire, P. Ranganathan, and C. Kozyrakis, ''A Comparison of High-Level Full-System Power Models,'' *Proc. 2008 Conf. Power Aware Computing and Systems,* Usenix Assoc., 2008, no. 3; www.usenix.org/event/hotpower08/tech/full_papers/rivoire/rivoire_html.
12. R. Bell Jr. and L.K. John, ''Improved Automatic Testcase Synthesis for Performance Model Validation,'' *Proc. 19th Ann. Int'l Conf. Supercomputing,* ACM Press, 2005, pp. 111-120.
13. A.M. Joshi et al., ''Distilling the Essence of Proprietary Workloads into Miniature Benchmarks,'' *ACM Trans. Architecture and Code Optimization,* vol. 5, no. 2, 2008, doi:10.1145/1400112.1400115.
14. L. Eeckhout et al., ''Statistical Simulation: Adding Efficiency to the Computer Designer's Toolbox,'' *IEEE Micro,* vol. 23, no. 5, 2003, pp. 26-38.

parameterized workload characteristics. The genetic algorithm then aims to search the set of parameterized characteristics that make the resulting powermark satisfy the desired power characteristics.

The synthetic benchmark skeleton consists of several compute-intensive threads along with one disk thread and one network thread. The network thread sends out network packets with a parameterized sleep time between packets. The disk thread similarly reads at random locations on the disk with parameterized sleep time between disk reads. The compute-intensive threads can be configured through multiple parameters, such as the number of threads, the size of memory accessed by an individual thread only, the size of memory accessed by all threads to invoke cache-coherency traffic, the strides with which the program traverses these regions of memory, the instruction mix, and the interinstruction dependencies to control instruction-level parallelism (ILP). The reason for having separate disk and

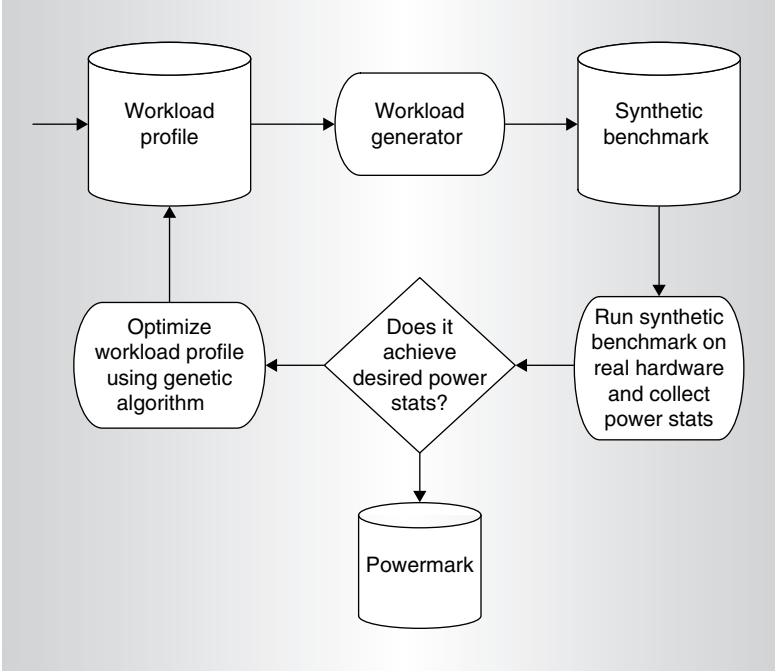

Figure 2. The powermark framework. The framework consists of three steps: workload generation, collection of power stats, and automated exploration.

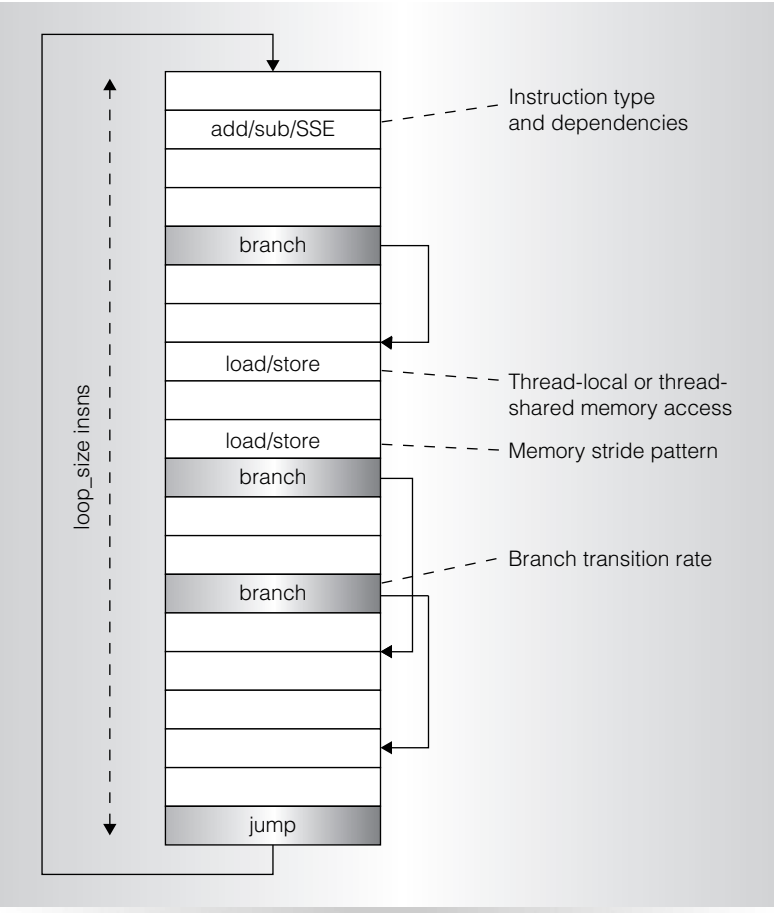

Figure 3. The loop body of a CPU-intensive thread. The loop of instructions is determined by characteristic dimensions.

network threads is to stress both the disk and the network while simultaneously stressing the CPU and memory. We run the powermark for 30 seconds at least.

The CPU threads consist of a loop of instructions that is iterated (see Figure 3). The characteristic dimensions determine the instructions in the loop. For each instruction in the loop, we determine its type, dependencies, memory stride access pattern (in case of a load or store), and branch behavior (in case of a branch). There are 43 characteristics in total (see Table 1). The workload generator uses characteristic dimensions for the CPU threads. Each dimension consists of several value-probability pairs, and the sum of these pairs equals one within a dimension.

There are various dimensions to consider.

- The program mix dimension consists of three value-probability pairs: one set for the probability of arithmetic operations, one for memory operations, and one for branch operations. This dimension lays out a CPU thread's instruction mix.
- The arithmetic mix dimension holds 18 value-probability pairs, each representing one type of arithmetic instruction, such as integer addition, floating-point multiplication, single instruction, multiple data (SIMD), and division.

Table 1. Workload characteristics that serve as input to the synthetic benchmark generator.

Characteristic	Description
Network thread sleep time	Time between network packets
Disk sleep time	Time between disk reads
Number of threads	Number of CPU-intensive threads
Loop size	Number of instructions in the main loop of the CPU-intensive thread
Size of thread-local memory	Size of memory region accessed by an individual thread
Size of thread-shared memory	Size of memory region accessed by all threads
Memory stride profile	Stride and probability for traversing memory regions
Dependency distance	Minimum dependency distance counted as the number of dynamically executed instructions between two dependent instructions
Program instruction mix	Percentage of arithmetic, memory, and branch operations
Memory mix profile	Percentage of load-local, load-shared, store-local, and store-shared operations
Arithmetic mix profile	Percentage of instruction types out of 18 types including integer, floating point (single precision, double precision), and Streaming SIMD Extensions (SSE) instructions
Branch transition rate	Transition rate and probability

- The dependency distance dimension represents the likelihood of observing a particular interinstruction dependency distance in the synthetic benchmark; we define dependency distance as the number of dynamically executed instructions between a write and a read to a register or memory location. We keep track of 20 dependency distances.
- The memory mix dimension holds four value-probability pairs, with each pair representing one of the following: load operations on thread-local memory, store operations on thread-local memory, load operations on thread-shared memory, or store operations on thread-shared memory.
- The memory stride dimension determines the stride value with which local or shared memory is traversed—that is, a stride value of s means that if memory is read at location A, the next read will be at location $A + s$. We can specify multiple stride values with respective likelihoods of occurrence.
- The branch dimension represents the branch transition rate, or the number of times a branch switches between taken and not taken, divided by the number of times the branch is executed. We can specify multiple transition rates with corresponding likelihoods of occurrence.

As we mentioned earlier, we generate synthetic benchmarks in a high-level programming language (C in our case), and not assembly. This lets us more easily deploy our framework across platforms.

Generating synthetic benchmarks with specific behavioral characteristics comes with challenges that we wouldn't encounter when generating synthetic benchmarks at the assembly level. One issue relates to generating SIMD instructions such as Streaming SIMD Extensions (SSE) and its further enhancements in the x86 architecture. To force the compiler to generate SIMD instructions—which we found to be a significant contributor to the overall CPU power consumption (1.9 W per core on our baseline)—we use vectorizable data types. Further, we generate C code such that the compiler can't optimize the code away through dead code elimination or copy propagation. This is done by letting each thread return a value to the main thread that it computes on the basis of all the computed values in the thread. Finally, we want the desired code and branch characteristics as we generate the synthetic benchmark to be preserved after compilation. Therefore, we force the compiler to allocate variables in registers (by using the `-O1` flag and not `-O0`), and not to do loop hoisting or if-conversion. This is done by using specific compiler flags when compiling the powermarks. For the same reason, we randomize all variables when beginning to execute the synthetic benchmark to avoid constant folding and copy propagation by the compiler.

Power monitoring

The next step is to run the synthetic benchmark on real hardware or on a simulator. In our setup, we run the synthetic benchmark on real hardware platforms and measure power consumption at the power outlet. We use the Racktivity RC0816 device,[7] which measures power usage in real time. Because the power measurement device is connected to a server's power outlet, it measures full-system power consumption. The Racktivity power-monitoring device measures power consumption at a one-second time granularity within 0.1 W of accuracy. We measure maximum (or minimum) power consumption observed during the course of the entire program execution. We keep room temperature constant at 24 degrees Celsius. We calculated measurement variability across multiple runs and found the variability to be less than 0.1 percent.

Automated exploration

To drive the search process, we employ a genetic algorithm that effectively avoids local optima. The genetic algorithm searches the workload space by varying the workload characteristics in the abstract workload description, and it optimizes these characteristics toward a powermark.

As we mentioned earlier, the genetic algorithm starts from a random set of workload profiles. A workload profile characterizes a

Table 2. Torture tests designed to stress the hardware platform.

Torture test	Optimized for	Language
burnBX	Stressing cache/memory interfaces	x86 assembly
burnK6	AMD K6	x86 assembly
burnK7	AMD K7	x86 assembly
burnMMX	Stressing cache/memory interfaces	x86 assembly
burnP5	Intel Pentium	x86 assembly
burnP6	Intel Pentium Pro, II, Celeron	x86 assembly
MPrime	All CPUs, all instruction-set architectures (ISAs)	C

workload through its characteristic dimensions. The collection of these workload profiles is called a *generation*; there are 20 workload profiles in our setup. We evaluate these workload profiles according to the objective function, also called the *fitness function*—that is, a synthetic benchmark is generated and is run on real hardware and power stats are collected. A new population, an *offspring*, which is a subset of these workload profiles, is probabilistically selected. The likelihood for a workload profile to be selected is determined by the workload profile's fitness function—that is, a fitter workload profile is more likely to be selected. The offspring consists of 10 workload profiles. We also retain the single-best workload profile across generations.

Selection alone can't introduce new workload profiles in the search space, so we perform mutation and crossover to build the next generation of 20 workload profiles. We perform crossover, with probability *pcross*, by randomly exchanging parts of two selected workload profiles from the current offspring. This means that we exchange a number of characteristic dimensions among two workload profiles to create new workload profiles. The mutation operator prevents premature convergence to local optima by randomly altering parts of a workload profile, with probability *pmut*. Mutation changes some characteristic dimensions in a workload profile to create a new one. The generational process continues until a specified termination condition has been reached.

In our experiments, we specify the termination condition as the point where there's little or no improvement in the objective function across successive generations or we've iterated 50 generations. In our setup, we set *pcross* and *pmut* to 0.4 and 0.2, respectively; we determined these parameters experimentally. The genetic algorithm's end result is an abstract workload configuration for which the corresponding synthetic benchmark stresses the objective function the most—this is our powermark. In this article, we consider two objective functions: max power and min power.

Powermark evaluation

Evaluating powermarks is troublesome. The main difficulty is that we can't prove that a powermark that's optimized to maximize power consumption effectively maximizes power consumption. In other words, you can't prove that there does not exist some other program that consumes even more power than the powermark. This is true for both manually and automatically generated powermarks.

Nevertheless, to assess our framework, we compare the power consumed by the generated powermarks against a broad range of existing performance benchmarks (SPEC CPU2006 and PARSEC) as well as a set of *torture tests*. When running the single-threaded SPEC CPU2006 benchmarks, we run as many copies of a benchmark as there are hardware thread contexts on the hardware platform. For the multithreaded PARSEC benchmarks, we run as many threads as there are hardware threads. The torture tests are synthetic benchmarks designed to torture, or stress, the hardware platform (see Table 2). We consider six hardware platforms in our evaluation (see Table 3). Figure 4 shows the power the various torture tests consume along with the maximum power consumed by one of the SPEC

Table 3. Hardware platforms under evaluation.

Processor	Full description	Nameplate power (in W)
Opteron Quad-Core	AMD Opteron Quad-Core 2350 2.0 GHz	380
Opteron Dual-Core	2× Dual-Core AMD Opteron Processor 2212 HE	600
Athlon	AMD Athlon XP 3000+	350
Core i7	Intel Core i7 CPU 920 @ 2.67 GHz	500
Pentium 4	Intel Pentium 4 CPU @ 3.20 GHz	230
Atom	Intel Atom CPU 330 @ 1.60 GHz	60

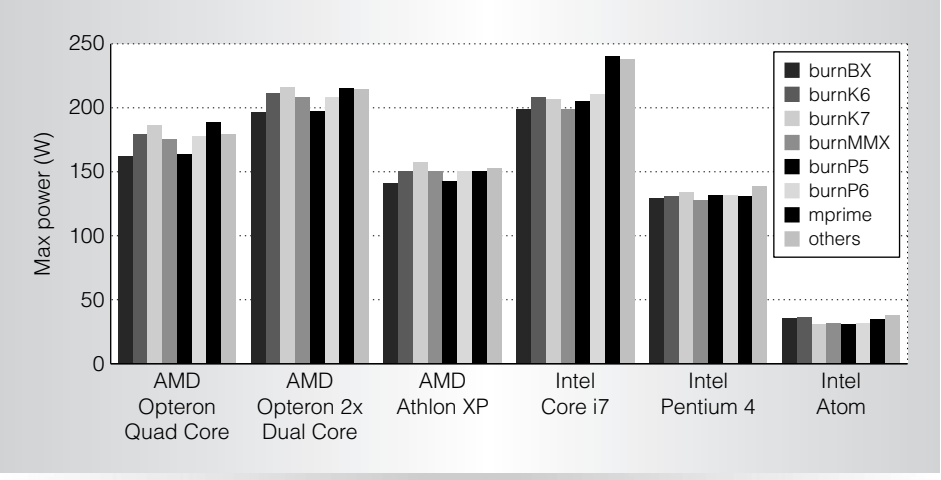

Figure 4. Power consumption for the torture tests versus SPEC CPU and PARSEC (labeled ''others'') on six hardware platforms.

CPU2006 and PARSEC benchmarks (called ''others'' in the legend). The torture tests generate the largest power consumption for four of the six hardware platforms, albeit by only a small margin compared to the performance benchmarks. For the Intel Atom and Pentium 4, two SPEC CPU benchmarks (`dealII` and `libquantum`) exceed the torture tests.

Figure 5 quantifies by how much the powermarks exceed the maximum power consumed by the torture tests and the performance benchmarks shown earlier (labeled ''maximum power virus''). We consider two flavors of powermarks: a CPU powermark that maximizes CPU and memory power but doesn't stress the other system components, and a full-system powermark that maximizes power consumption of the entire system (including the disk and the network). The CPU powermark increases maximum system power consumed by up to 4 percent for the Intel Core i7 and Atom platforms. The full-system powermark increases maximum power consumed over the CPU powermark by up to slightly less than 3 percent for the dual-socket AMD Opteron system.

We obtained the largest max power deltas for the most recent processors, namely the AMD Opteron, Intel Core i7, and Intel Atom. This is likely a result of aggressive clock gating in these processors: the workload significantly impacts what logic is active at a given point in time, so maximizing processor activity maximizes power consumption. The older processors (the AMD Athlon XP and Intel Pentium 4) presumably don't implement such aggressive clock gating, so a program might not have as much impact on the amount of power the processor consumes. Although the increase in

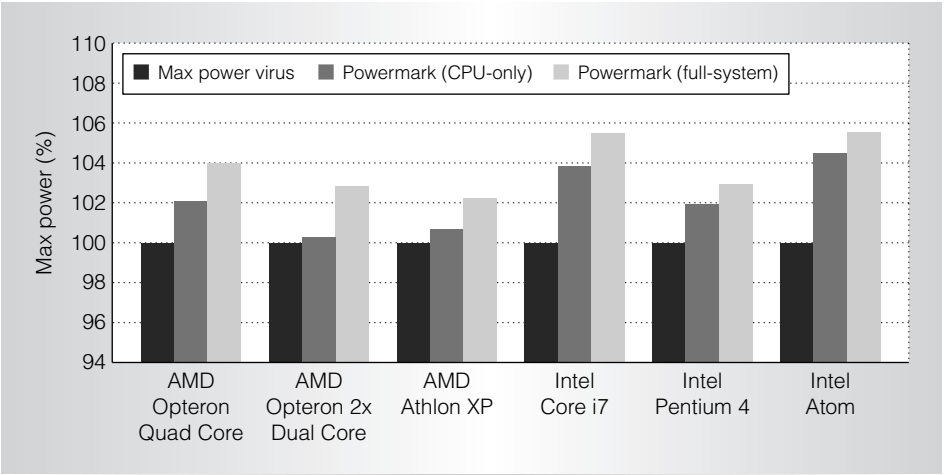

Figure 5. Power consumption of the powermarks versus the max torture test and performance benchmarks. This graph compares power consumed by the maximum power virus against the automatically generated powermarks that maximize CPU power only and total system power.

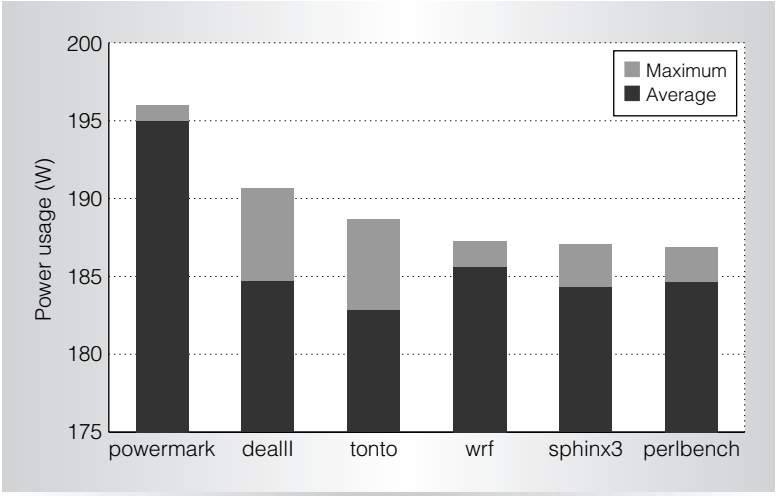

Figure 6. Average and maximum power consumption for the powermark compared to the top five power-hungry performance benchmarks on the AMD Opteron Quad-Core processor platform. The gap between maximum and average sustained power consumption is smaller for the powermark than for the performance benchmarks. The performance benchmarks are sorted by decreasing maximum power consumption.

max power consumption might seem small in absolute terms between a CPU-centric powermark and a full-system powermark, we believe it's significant because the disk and network are powered while running a CPU powermark, and reading from a disk increases power consumption by only 3.1 W in our setup.

It's insightful to look at how the powermark's average and maximum power consumption compares to the top five power-hungry performance benchmarks (see Figure 6). Our power monitor quantifies power consumption at a one-second time granularity. The maximum power consumption reported in this graph is the maximum power consumption observed at a one-second granularity over the course of a benchmark's entire execution; the average number is the average power consumption over the entire benchmark execution. The gap between the maximum and average power consumption is smaller for the powermark than for the performance benchmarks. This means that the powermark achieves high power consumption over a long period of time whereas the performance benchmarks exhibit power peaks only (and the peaks are lower than for the max powermark).

The powermark framework also provides an opportunity to study how workload characteristics affect power consumption. For illustrative purposes, we compared the characteristics of the powermarks generated for the low-end Intel Atom versus the high-end Intel Core i7 processor system. We found that the Core i7 powermark exhibits five times more ILP compared to the Atom powermark, and accesses much larger thread-local and thread-shared spaces. This reflects the

fact that the Core i7 is a wide, superscalar, out-of-order processor with large caches, whereas the Atom is a narrow, in-order processor with much smaller caches.

Furthermore, we observe a larger fraction of branches for the Core i7 powermark, whereas the Atom powermark exhibits a larger fraction of arithmetic and memory operations. Remarkably, most of the arithmetic operations are SIMD (SSE) operations: 70 percent on the Core i7 and 55 percent on the Atom. In addition, the loads executed in the Core i7 powermark issue independent parallel memory accesses to stress the memory hierarchy to the fullest.

Finally, Figure 7 illustrates how fast the genetic algorithm converges. The randomly generated synthetic benchmarks at start-up consume 179.2 W, whereas the powermark consumes 196.3 W, an increase of 17.1 W. The genetic algorithm converges to this maximum power consumption relatively quickly in 30 generations. This evolutionary search took 10 hours of wall clock time.

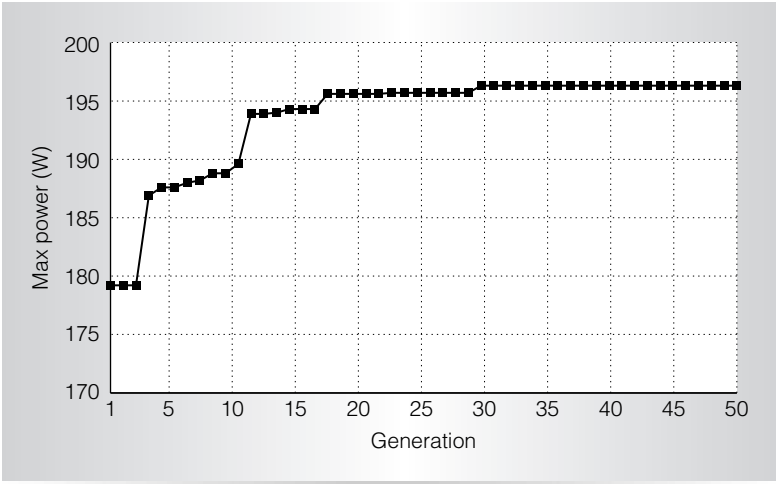

Figure 7. Convergence of the genetic algorithm on the AMD Opteron Quad-Core system. The genetic algorithm quickly converges to a powermark that maximizes power consumption in 30 generations.

Full-system power modeling

A first application that we explored for the powermark framework was to build system-level power models in an automated way. The idea is to generate two powermarks—one that maximizes power consumption and one that minimizes power consumption.

We conjectured that the average between the maximum powermark and the minimum powermark would correspond to the average power consumption we would see across a large set of workloads. We found this to be true within 2.8 percent (see Figure 8). The vertical axis shows power consumption, and

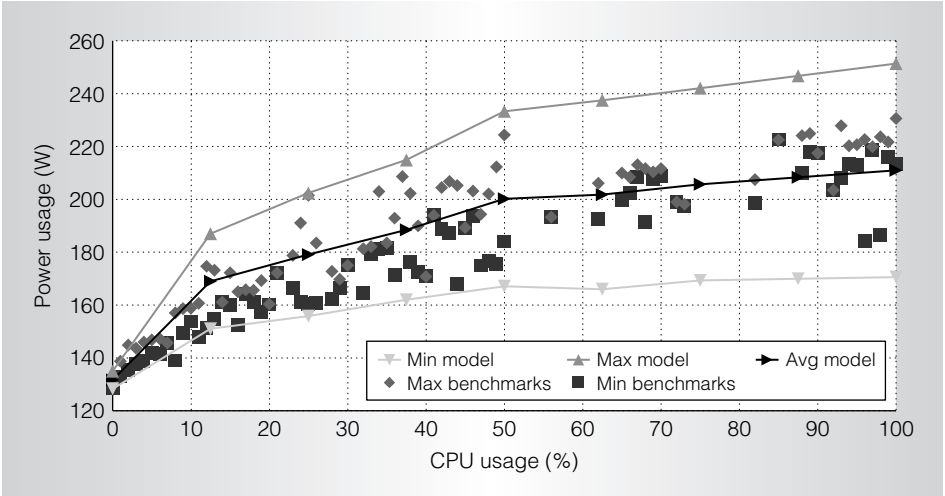

Figure 8. Validating the power model on the Intel Core i7 system. The maximum and minimum power consumption of the performance benchmarks fall within the maximum and minimum bounds predicted by the model across different CPU usage levels. We employ multicore processing on the Intel Core i7 below 50-perent CPU usage and engage SMT beyond 50-percent CPU usage.

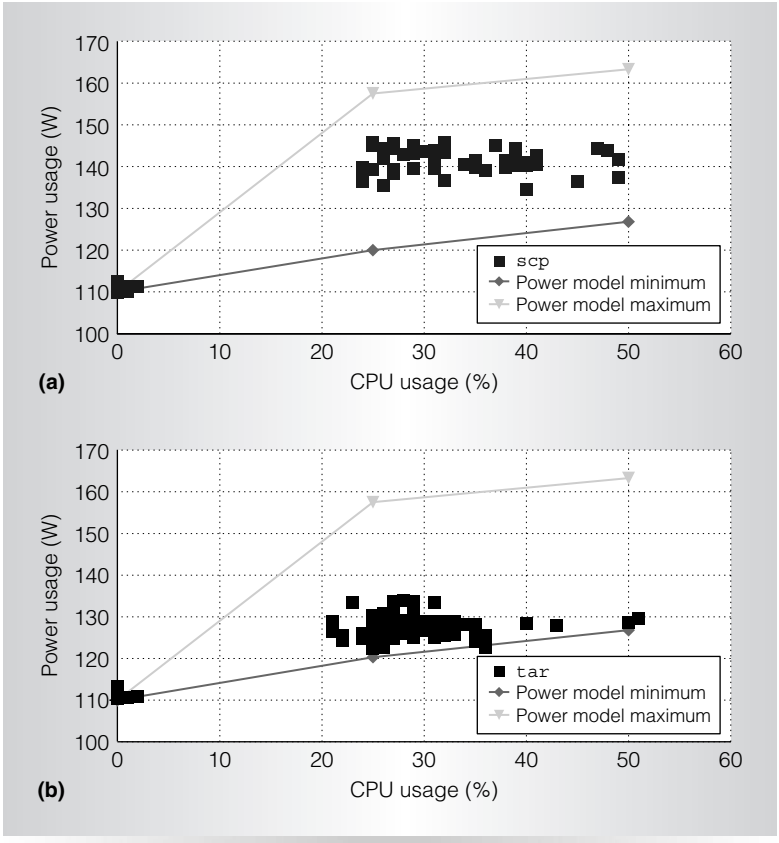

Figure 9. Validating the power model considering disk I/O intensive workloads: `scp` on the left and `tar` on the right on the AMD Opteron Quad-Core system. Disk-intensive workloads fall within the maximum and minimum bounds predicted by the power model.

the horizontal axis shows CPU usage. We considered both SPEC CPU2006 and PARSEC as our performance benchmarks. For SPEC CPU, we ran 1-core, 2-core, 3-core, and 4-core workloads by running multiple copies concurrently. For PARSEC, we ran up to eight threads: we ran each thread on an individual core for up to four cores, and beyond four threads, we scheduled up to two threads per core through SMT execution.

We report the maximum and minimum power consumption (at a one-second granularity) observed across those benchmarks. The performance benchmarks all fall within the bounds reported by the system-level power model. Interestingly, we see a knee in the curves at 50-percent CPU usage, or at 4 cores, because the Intel Core i7 has four cores, with each core running two hardware SMT threads. Power consumption increases at a faster pace during multicore execution relative to SMT execution.

Figure 9 shows similar results for two disk I/O intensive workloads, namely `scp` and `tar`: power consumption never exceeds the error bounds determined by the model. This approach's key feature is that the model was constructed in an automated way and is easily applied on different platforms. Additionally, the power model provides bounds that are determined systematically using powermarks, in contrast to prior work that obtains error bounds through statistics on a set of workloads.[8,9]

Data center dimensioning

A second application we explored was to dimension the data center using powermarks. A server's nameplate power consumption is typically overly high compared to what the server is really consuming—for example, the nameplate power consumption typically mentions what the server might consume with all the options installed, including maximum memory, disk, and PCI cards.[1] The nameplate power consumption for the Intel Core i7, for example, is rated at 500 W, but our system-level powermark consumes around 250 W. (See Table 3 for the nameplate power consumption for all of the hardware platforms.)

This suggests that dimensioning the data center for the system-level powermark's power consumption rather than the nameplate power consumption can lead to significant cost reductions. In fact, some data center infrastructure providers charge their clients on the basis of the nameplate power consumption, not the maximum realizable power consumption. Figure 10 shows the energy cost reduction by dimensioning the maximum realizable power consumption; normalized energy cost is shown if the system is dimensioned by the nameplate power consumption versus the powermark's power consumption. Dimensioning using the powermark can reduce total energy costs by between 34 percent and 63 percent, depending on the hardware platform.

Going one step further, we realize that only a minority of the programs ever reach a power consumption level close to the

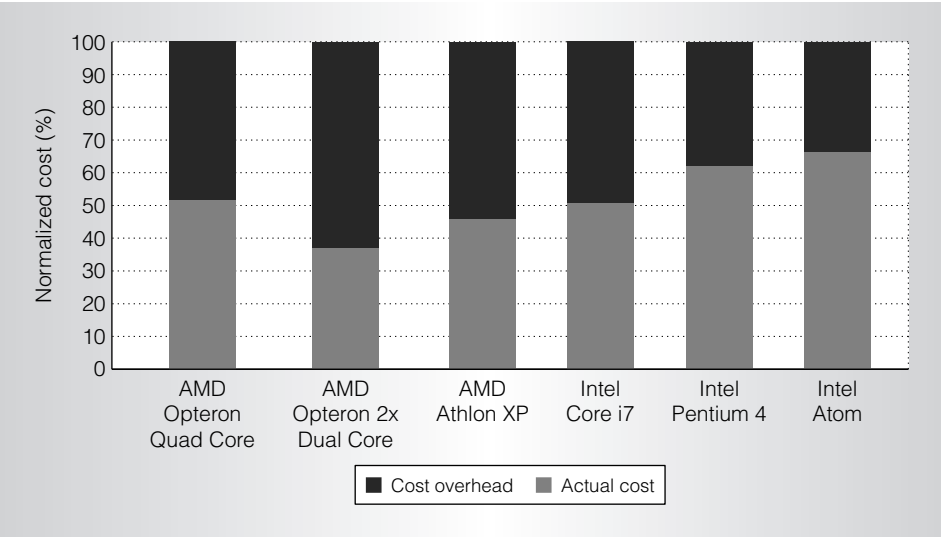

Figure 10. Normalized energy cost for a server dimensioned for the nameplate power consumption versus the maximum power consumption derived from a system-level powermark. The cost overhead that can be saved ranges between 33 and 62 percent by dimensioning the server based on the maximum powermark.

max powermark. This suggests that it might be cost efficient to dimension the system for a power consumption level below its max powermark, and to have a monitoring and feedback system that lowers processor activity if actual power consumption exceeds the designed power level, so that in practice the system (almost) never exceeds the designed lower level. This reduces costs significantly while penalizing performance by only a small margin.

We evaluated this idea using the user-space Linux `cpulimit` tool, which lets us control the CPU usage of a process. In our experiment, we considered the AMD Opteron Quad-Core and ran all of the SPEC CPU2006 and PARSEC benchmarks one after another, setting the threshold at 10 percent below the maximum powermark (see Figure 11). The policy is such that if the actual power consumption exceeds the threshold—which it can for a short period of time given current power supplies—we run the processor at a usage level of 80 percent for five minutes. (Tuning the various parameters in this scheme is out of this article's scope.) The performance penalty is limited to 5.3 percent, while reducing costs by 10 percent compared to dimensioning with respect to the maximum powermark, or 58 percent in total compared to the nameplate power.

As part of our future work, we plan to incorporate the full-system power-modeling framework presented in this article into a data center simulation infrastructure that we're currently developing. The end goal is to be able to predict performance and power of large server and data center infrastructures running real-life commercial workloads. This will be an important tool for optimizing servers and data centers for minimum TCO while taking into account performance, power, cooling, hardware purchase cost, infrastructure cost, maintenance cost, and so on. MICRO

Acknowledgments

We thank the anonymous reviewers for their thoughtful comments and suggestions. We thank Kristof De Spiegeleer for providing the Racktivity power measurement device, and we thank Lieven Lemiengre and Wouter Kampmann for their early work on the powermark framework. We also thank Pradip Bose and Jeff Stuecheli for early discussions on power virus generation. Stijn Polfliet is supported through a doctoral fellowship by the Agency for Innovation by

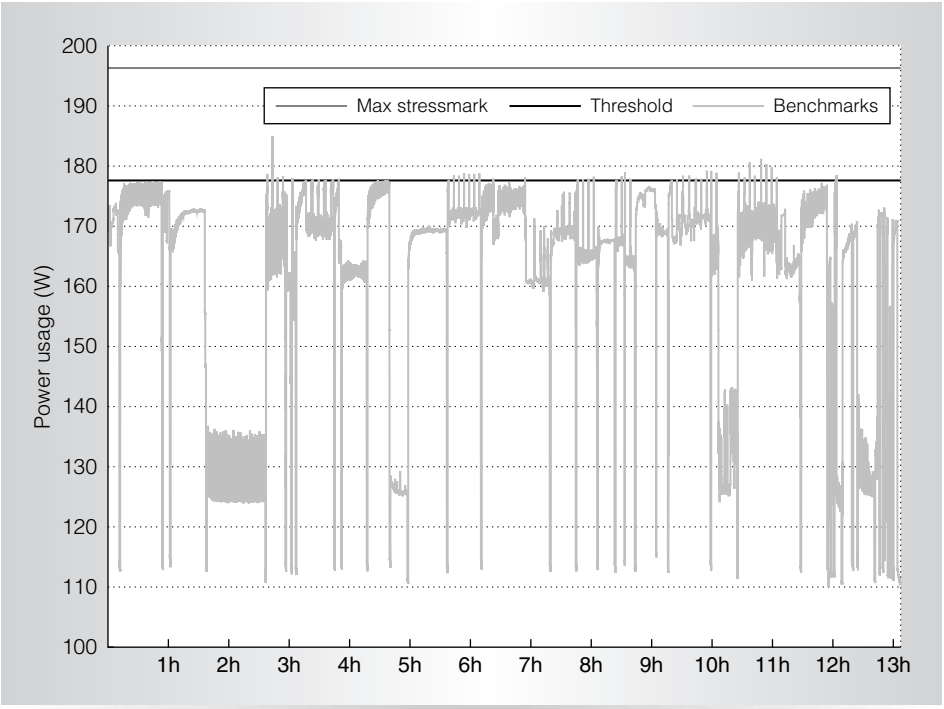

Figure 11. Dimensioning the server on a threshold power consumption that's 10 percent below the max powermark.

Science and Technology (IWT). Frederick Ryckbosch is supported through a doctoral fellowship by the Research Foundation–Flanders (FWO). Additional support is provided by FWO projects G.0232.06, G.0255.08, and G.0179.10, the UGent-BOF projects 01J14407 and 01Z04109, and the European Research Council under the European Community's Seventh Framework Program (FP7/2007-2013)/ERC Grant agreement no. 259295.

References

1. L.A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines,* Morgan and Claypool, 2009.
2. A.M. Joshi et al., ''Automated Microprocessor Stressmark Generation,'' *Proc. Int'l Symp. High-Performance Computer Architecture,* IEEE Press, 2008, pp. 229-239.
3. K. Ganesan et al., ''System-Level Max Power (SYMPO): A Systematic Approach for Escalating System Level Power Consumption Using Synthetic Benchmarks,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* ACM Press, 2010, pp. 19-28.
4. W. Felter and T. Keller, *Power Measurement on the Apple Power Mac G5,* tech. report RC23276, IBM, 2004.
5. M.K. Gowan, L.L. Biro, and D.B. Jackson, ''Power Considerations in the Design of the Alpha 21264 Microprocessor,'' *Proc. 35th Design Automation Conf.,* ACM Press, 1998, pp. 726-731.
6. R. Vishmanath et al., ''Thermal Performance Challenges from Silicon to Systems,'' *Intel Technology J.,* vol. 4, no. 3, 2000; http://download.intel.com/technology/itj/q32000/pdf/thermal_perf.pdf.
7. Racktivity, Racktivity RC0816, http://www.racktivity.com/media/pdf/Racktivity_EnergySwitch_ES1008_datasheet.pdf.
8. X. Fan, W.-D. Weber, and L.A. Barroso, ''Power Provisioning for a Warehouse-Sized Computer,'' *Proc. Int'l Symp. Computer Architecture,* ACM Press, 2007, pp. 13-23.
9. S. Rivoire, P. Ranganathan, and C. Kozyrakis, ''A Comparison of High-Level Full-System Power Models,'' *Proc. 2008 Conf. Power*

Aware Computing and Systems, Usenix Assoc., 2008, no. 3; www.usenix.org/event/hotpower08/tech/full_papers/rivoire/rivoire_html.

Stijn Polfliet is a doctoral student in the Electronics and Information Systems Department at Ghent University. His research interests include computer architecture and systems, with a specific emphasis on managing power consumption in high-end computer systems. Polfliet received his MS in computer science and engineering from Ghent University.

Frederick Ryckbosch is a doctoral student in the Electronics and Information Systems Department at Ghent University. His research interests include computer architecture and systems, especially simulation, modeling, and optimization of high-end computer systems. Ryckbosch received his MS in computer science and engineering from Ghent University.

Lieven Eeckhout is an associate professor in the Electronics and Information Systems Department at Ghent University. His research interests include computer architecture and the hardware/software interface, with a focus on performance analysis, evaluation, and modeling and workload characterization. Eeckhout received his PhD in computer science and engineering from Ghent University. He's a member of IEEE and the ACM.

Direct questions or comments about this article to Lieven Eeckhout, ELIS—Ghent University, Sint-Pietersnieuwstraat 41, B-9000, Gent, Belgium; leeckhou@elis.ugent.be.