Power-Aware Multi-Core Simulation for Early Design Stage Hardware/Software Co-Optimization

Wim Heirman[*†] Souradip Sarkar[*†] Trevor E. Carlson[*†]
Ibrahim Hur[‡†] Lieven Eeckhout[*]

[*]ELIS Department
Ghent University, Belgium

[†]ExaScience Lab
Leuven, Belgium

[‡]Intel
Leuven, Belgium

ABSTRACT

Stringent performance targets and power constraints push designers towards building specialized workload-optimized systems across a broad spectrum of the computing arena, including supercomputing applications as exemplified by the IBM BlueGene and Intel MIC architectures. In this paper, we make the case for hardware/software co-design during early design stages of processors for scientific computing applications. Considering an important scientific kernel, namely stencil computation, we demonstrate that performance and energy-efficiency can be improved by a factor of 1.66× and 1.25×, respectively, by co-optimizing hardware and software.

To enable hardware/software co-design in early stages of the design cycle, we propose a novel simulation infrastructure by combining high-abstraction performance simulation using Sniper with power modeling using McPAT and custom DRAM power models. Sniper/McPAT is fast — simulation speed is around 2 MIPS on an 8-core host machine — because it uses analytical modeling to abstract away core performance during multi-core simulation. We demonstrate Sniper/McPAT's accuracy through validation against real hardware; we report average performance and power prediction errors of 22.1% and 8.3%, respectively, for a set of SPEComp benchmarks.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization – Performance of Systems**]: Modeling techniques

General Terms

Performance, Experimentation, Design

Keywords

Performance modeling, power modeling, hardware/software co-design, design space exploration, multi-core processor

1. INTRODUCTION

With limited increases in clock frequency because of power constraints, improving next-generation processor performance has become a real challenge. One increasingly attractive way to improve performance within a given power and energy budget is to optimize the system for a specific workload — a paradigm that is broadly adopted for the design of smartphones, tablets, game machines, data centers and supercomputers. Because computer systems are increasingly power and energy-constrained, it is to be expected that workload-optimized system design will become even more prevalent in the future. Notable examples of processors optimized for supercomputing applications are IBM's BlueGene processors and Intel's Many Integrated Core (MIC) architecture. However, optimizing the hardware alone might not be sufficient moving forward. Instead, co-optimizing the workload along with the hardware — hardware/software co-design — can be even more promising, as we experimentally demonstrate in this paper. Hardware/software co-design is already the design paradigm of choice for embedded system design, however, it is not generally employed for high-performance processor design.

A fundamental challenge regarding co-designing hardware and software is how to evaluate design decisions and make trade-offs early in the design cycle. A common approach in architecture design is to employ detailed cycle-accurate simulation. Unfortunately, cycle-accurate simulators are extremely slow, are difficult to scale to large multi-core systems, and take a long time to develop, hence they are inappropriate for early design stages. To make things even worse, making a detailed cycle-accurate simulator power and energy-aware further increases development and evaluation time. Clearly, driving hardware/software co-design through cycle-accurate simulation is particularly problematic.

In this paper, we make the case for architectural simulation at a higher level of abstraction for driving early design stage hardware/software trade-off explorations, while considering both performance and power. Our simulation methodology leverages a mechanistic analytical performance model to abstract away core performance, i.e., core performance is estimated through an analytical model while simulating the uncore (memory hierarchy, interconnection network, etc.) at some level of detail in order to capture inter-core performance interactions. Coupling this high-abstraction performance simulation approach, called interval simulation as implemented in Sniper [6], with high-level power modeling using McPAT [18] and custom DRAM power models, we achieve both good accuracy and speed.

We demonstrate the power of Sniper/McPAT which is a hardware-validated, accurate (for both performance and power), parallel simulator that can run multi-threaded and multi-programmed workloads on multi-core hardware.

Having established the accuracy and speed of our simulation methodology, we perform a number of architecture explorations and we make the case for hardware/software co-design towards higher levels of performance and energy efficiency. To this end, we consider a widely used numerical kernel, namely stencil computation. This case study illustrates the huge potential of hardware/software co-design, which we believe is going to be a crucial design principle towards future processor systems.

More specifically, we make the following contributions in this paper.

- We propose Sniper/McPAT, a high-abstraction simulation methodology for simulating performance and power of large multi-core systems. Sniper is a parallel simulator that benefits from running on multi-core hosts. Further, it employs analytical core modeling to raise the level of abstraction. Both features make up for a fast simulation approach, which enables running more and longer running simulations within a given time budget. With our simulation methodology, we achieve simulation speeds around 2 MIPS when simulating a 16-core target system on an 8-core host system.

- We validate Sniper/McPAT against real hardware and we report good accuracy for both performance and power. We find the average accuracy to be within 22.1% and 8.3% compared to real hardware (Intel Nehalem processor system with 8 cores) for performance and power, respectively, when running benchmarks from SPEComp [2]. These results show that even basic statistics can yield accurate power predictions. Detailed inputs such as activity factors and bit toggles, as done in other power modeling frameworks such as Wattch [5], are not needed in Sniper/McPAT: basic event counts such as the number of cache misses and duty cycles yield good accuracy. Power modeling is part of the public Sniper release, which can be downloaded from `http://snipersim.org`.

- We use Sniper/McPAT to drive architecture explorations using multi-threaded benchmarks from the SPLASH-2, PARSEC, Rodinia, and SPEComp suites in which we explore the performance and power impact of 3D stacking, multi-core processing, core width, and power settings. We show that 3D stacking, in which the increase in memory bandwidth allows cache size to be traded in for more cores, is an advantageous design point for a broad range of multi-threaded applications.

- We make the case for hardware/software co-design of processor chips for scientific applications. Using stencil computation as a driver numerical kernel, we demonstrate that co-designing the architecture along with the software leads to improvements of a factor 1.66× in terms of performance and 1.25× in terms of energy efficiency.

2. BACKGROUND

Before introducing Sniper/McPAT, we first briefly describe its building blocks, namely interval simulation using Sniper, and power modeling using McPAT.

2.1 Interval simulation

Interval simulation is a recently proposed simulation paradigm for simulating multi/many-core and multi-processor systems at a higher level of abstraction than current practice of detailed cycle-accurate simulation [10]. Interval simulation leverages a mechanistic analytical model to abstract core performance by driving the timing simulation of an individual core without the detailed tracking of individual instructions through the core's pipeline stages. The mechanistic analytical model is constructed from the underlying mechanisms of a superscalar processor core. The foundation of the model is that miss events (e.g., branch mispredictions, cache misses, serializing instructions) divide the smooth streaming of instructions through the pipeline into so-called *intervals*, and that progress between miss events is determined by the amount of processor-exposed instruction-level parallelism (number of inter-instruction dependencies and instruction execution latencies). The branch predictor, memory hierarchy, cache coherence and interconnection network simulators determine the miss events while the analytical model derives the timing of each interval. The cooperation between the mechanistic analytical model and the miss event simulators enables modeling of the tight performance entanglement between co-executing threads on multi-core processors at a higher level of abstraction than detailed cycle-accurate simulators. The implementation of interval simulation in Sniper, a parallel multi-core x86 simulator, results in an accurate, high-abstraction level performance simulator [6].

2.2 McPAT

McPAT [18] is a recently proposed and fully-integrated power, area and timing modeling framework. It models all types of power dissipation and provides an integrated solution for multithreaded and multi-core processors. The timing and area models in this tool are derived from CACTI [27]. The dynamic power model is similar to Wattch [5], while adding short circuit and leakage models.

3. SNIPER/MCPAT SIMULATION

Sniper/McPAT combines Sniper for performance modeling with McPAT and custom DRAM models for power modeling. Sniper, in addition to generating an overall performance estimate, also generates a number of statistics that serve as input for estimating power consumption using McPAT (see Figure 1). Hence, we keep Sniper and McPAT as separate executables. We first run Sniper to obtain performance data and various other statistics. Once the performance simulation is finished, a separate script parses the results and generates an XML input file for McPAT. Finally, the output from McPAT is parsed, and final representations such as total power and/or per-component energy stacks are generated.

3.1 Generating input for McPAT

McPAT's input file consists of two main types of information: architectural parameters and statistics about the

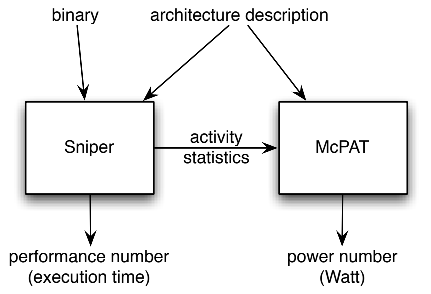

Figure 1: High-level overview of Sniper/McPAT.

activity of various architecture components during application runtime. The architecture parameters, such as the processor's reorder buffer size and issue width, and the size, associativity and latency of each cache level, are used to calculate power and energy costs for each component, for a specific technology node. We picked these parameters using public information [15] when modeling an Intel Nehalem-like system in our setup.

The activity statistics take the form of duty cycles which indicate what fraction of time a component is in use, as well as counts for events that have a per-event energy cost such as a memory access. For high-level components such as hits and misses at the various levels of the memory hierarchy, these values are trivially obtained from Sniper's output statistics. Activities for other components, especially those related to structures inside the core — which interval simulation does not model deliberately in order to raise the level of abstraction and improve simulation speed — need to be estimated. For instance, reads and writes to the reorder buffer (ROB) or register allocation table (RAT) are not explicitly modeled by interval simulation. Instead, we assume a constant usage pattern by each instruction and estimate these statistics as a fixed ratio of the number of instructions, e.g., one ROB read and write per instruction, etc.

For other components such as ALUs and the load-store units, McPAT expects an activity factor. We calculate these using the following assumption. We assume these components to achieve a throughput of one instruction per cycle. This means that each instruction that uses a given component, occupies this component for exactly one cycle. The fact that the component is internally pipelined does not matter from a modeling perspective, as each pipeline stage is kept busy by this instruction for just one cycle; even though the activity caused by the instruction might be spread out over multiple clock cycles, it still causes the complete component to be active for one cycle, assuming aggressive clock gating when parts of a component are not used. Thus, the activity factor of the unit can be estimated by taking the number of instructions that use it, implicitly multiplying this number by one active cycle per instruction, and dividing it by the total number of clock cycles the simulation took. We also extended Sniper to output the dynamic instruction mix, which allows us to compute the usage of the respective execution units.

3.2 DRAM power

McPAT does not model DRAM power, although this can represent a significant fraction of total system power [17]. We therefore added a basic first-order DRAM power model. Using Micron's DDR3 System-Power calculator [21], we calculate idle (static plus refresh) power and per-operation energy for reads and writes. Power consumption of the DRAM interface, both for the regular DDR3 case as for 3D stacked memory, is calculated according to the models from [9]. We thus calculate static and dynamic DRAM chip and interface power, and add these to McPAT's power numbers.

3.3 Sniper/McPAT limitations

By using a high-abstraction user-level simulation infrastructure such as Sniper/McPAT, we opt for simulation speed and short simulator development time at the expense of sacrificing some accuracy. Hence, Sniper/McPAT has its limitations and should not be used for detailed microarchitecture studies inside the core of a multi-core processor; however, we believe Sniper/McPAT is an ideal simulation platform for studying system-level design trade-offs at early stages in the design cycle.

We now briefly summarize Sniper/McPAT's limitations. Sniper is a user-level simulator, hence, it does not model system-level code. As we focus on HPC benchmarks, which spend most of their time in user-level code [20], this is a valid trade-off. Second, Sniper is built on top of Pin [19], which is a dynamic binary instrumentation tool. This implies that Pin enables instrumenting correct-path instructions only, i.e., wrong-path execution is not modeled. Again, HPC workloads typically have a high branch prediction accuracy [20], hence, this is a viable assumption. Third, Sniper does not model the internals of a superscalar out-of-order processor core, hence, it is unable to track the occupancy and activity factors inside the core. (Recall the key motivation behind interval simulation is to abstract away these details in order to achieve higher simulation speed.) We therefore need to approximate some of these core internal statistics, as described in the previous section. Yet, in spite of these limitations and simplifying modeling assumptions, Sniper/McPAT is accurate compared against real hardware, as we show in Section 5.

3.4 Simulation speed

Sniper's simulation speed is up to around 2 MIPS when simulating 16-core architectures on an 8-core host machine [6]. To enable power modeling, we extended Sniper to export some additional statistics as described in Section 3.1. All of these changes to the performance simulator do not slow down simulation. Parsing the various statistics files and generating McPAT's input file is straightforward and takes less than a second, as does parsing McPAT's output and generating the final results. The runtime for McPAT itself can be up to a few minutes, mainly taken up by CACTI which searches for the optimum register file and cache organization. Still, compared to performance simulation using Sniper, which takes several hours when running realistic benchmarks, the total increase in simulation time because of power modeling is insignificant.

We can therefore state that, from a simulation time perspective, power modeling is essentially free in Sniper/McPAT. This is in contrast to a more detailed approach such as Wattch, which incurs a slowdown of the SimpleScalar performance simulator in which it is integrated, by about 30% [5]. The reason for this slowdown in Wattch/Simplescalar is that detailed activity factors and the

Component	Parameters
Processor	2 sockets, 4 cores per socket
Core	2.66 GHz, 4-way issue, 128-entry ROB
Branch predictor	Pentium M [28], 17 cycles penalty
L1-I	32 KB, 4 way, 4 cycle access time
L1-D	32 KB, 8 way, 4 cycle access time
L2 cache	256 KB per core, 8 way, 8 cycle
L3 cache	8 MB per 4 cores, 16 way, 30 cycle
Main memory	65 ns access time, 8 GB/s per socket

Table 1: Simulated system characteristics for the dual-socket quad-core Intel Nehalem baseline architecture.

number of bit toggles need to be computed during performance simulation.

3.5 Sniper/McPAT output

McPAT computes chip area and both static and dynamic power, broken down into several architectural components such as the reorder buffer and other core structures, caches, etc. By multiplying total power consumption with the benchmark's execution time as obtained from Sniper's performance simulation, derived metrics such as static and dynamic energy, or energy-delay products can be computed — both for the complete system or broken down per component. In addition, power traces can be computed as well, by running McPAT on a time series of runtime statistics produced by Sniper. Examples will be given in the results section.

4. EXPERIMENTAL SETUP

4.1 Baseline processor configuration

Our baseline processor configuration is configured after a dual-socket, quad-core Intel Nehalem machine. The Nehalem chips consist of four 4-wide out-of-order cores running at 2.66 GHz with private L1 and L2 caches and a shared L3 last-level cache. More details can be found in Table 1.

4.2 Benchmarks

Results in this paper combine applications from four benchmark suites: SPLASH-2 [31], PARSEC [3], Rodinia [7] and SPEComp [2], compiled with GCC 4.3.2. The input sizes used are `simsmall` and `simlarge` for PARSEC and `train` from SPEComp (medium). For SPLASH-2 and Rodinia, we used the inputs defined in [12]. We only enable performance modeling during the parallel section of the benchmark (the region of interest, ROI); the sequential parts at the beginning and end of a benchmark execution are not included in any timing or energy results given in this paper.

In addition to these standardized benchmark suites, we also include a scientific kernel, namely stencil computation, for driving a hardware/software co-design case study. Stencil computation is part of the Berkeley dwarfs [1] (Structured Grid), and is a widely used kernel in many applications, such as physical dynamics simulations. This particular application models heat transfer across a 2-D grid; we will refer to this application as the `heat` benchmark. A more detailed description can be found in Section 7.1.

5. HARDWARE VALIDATION

In this section, we evaluate the accuracy of Sniper/McPAT against real hardware. We specifically focus on power consumption here; Sniper has been hardware-validated for performance previously, by Carlson et al. [6].

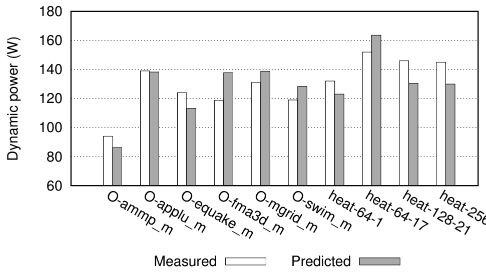

Figure 2: Measured versus predicted dynamic power consumption using Sniper/McPAT.

5.1 Setup

Our validation experiments compare McPAT's *Peak Dynamic Power* prediction against that of a dual-socket, 45 nm Intel Nehalem processor based server machine (an IBM x3650 M2). Total system power was measured using a Racktivity RC0816 [24] Power Distribution Unit (PDU) with integrated power metering. This PDU performs real-time true RMS measurements of the server's 230V AC power supply and allows us to read out the server's power consumption once every second; the reported number is the average power consumed over the second. We run our benchmarks on real hardware and measure total system power. Since we only have one power measurement per second, we selected only benchmarks for which the ROI was longer than two seconds to make sure we always have at least one valid power measurement from the ROI. Out of the extensive set of benchmarks that we considered in this work (see Section 4.2), only six of the benchmarks from the SPEComp suite and the `heat` benchmark run longer than two seconds; the other benchmarks from SPLASH-2, PARSEC, Rodinia and SPEComp did not run long enough to obtain meaningful power numbers. Each benchmark is run for two minutes. During the first minute we allow the system to reach a thermal equilibrium. We then report the average power consumption during the second minute for those samples that completely overlap with the benchmark's ROI. Finally, for validation purposes of Sniper/McPAT's dynamic power estimates, we subtract the system's idle power so our numbers do not include power consumed by disks, motherboard, network interfaces, etc., but account for processor and DRAM dynamic energy only.

5.2 Results

Figure 2 compares measured against predicted dynamic power consumption using Sniper/McPAT, for the selected SPEComp benchmarks and several configurations of the `heat` benchmark. The numbering scheme for the configurations of `heat` consists of the tile size and number of steps (s) computed per tile. These parameters can be used to tweak the memory access behavior of this application, as we will explain in more detail in Section 7.1. As expected, power consumption is benchmark-dependent. Sniper/McPAT captures the trend fairly accurately with an average absolute error of 8.3% and a maximum error of 16% for SPEComp's `fma3d`. We also re-evaluated Sniper's performance accuracy and found the average absolute error for this set of benchmarks to be 22.1%, which is in line with the results reported by Carlson et al. [6].

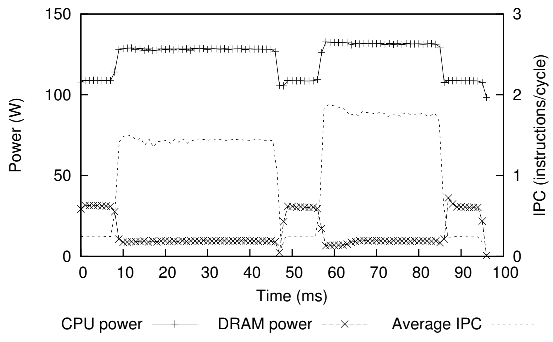

Figure 3: Simulated execution behavior for `fft`.

Figure 3 plots a trace of CPU and DRAM dynamic power consumption over time for the `fft` benchmark, along with an IPC trace (averaged across all eight cores). Execution phases with low IPC are memory-intensive. Hence, CPU power consumption is relatively low while DRAM power consumption is high. During compute-intensive, high-IPC phases, CPU power is high while DRAM power is relatively low. This illustrates that Sniper/McPAT's power predictions track resource usage as expected.

Finally, we wanted to verify whether computing power consumption at the end of the performance simulation using aggregate performance and activity statistics yields any differences compared to computing power consumption at smaller time granularities. We therefore ran McPAT once per one millisecond time interval to obtain a power trace over time. Computing the total energy consumption from this power trace yielded the same total energy within two percent as obtained from aggregate performance and activity statistics. This is because the power models in McPAT are mostly linear; moreover, the thermal response of the chip is slow enough such that short bursts in activity do not significantly affect overall temperature.

6. ARCHITECTURAL EXPLORATION

Using our validated simulation framework, we now perform a design space exploration in which we compare four architecture design points. We consider the processor configuration inspired by the dual-socket Nehalem system as described in Table 1 as a starting point, and asked ourselves the following question:

> *Given a technological advancement by two technology nodes, from 45 nm to 22 nm, how can we best use the available improvements in transistor density and energy efficiency?*

The first architecture considered is a conservative integration, in which we integrate the eight cores of the dual-socket quad-core Nehalem machine onto a single chip. Along with an increase in clock frequency from 2.66 GHz to 3.059 GHz, and cache size (from 256 KB to 512 KB for L2, and from 8 MB to 32 MB for L3), this forms our *8-core* design point.

In addition to this conservative scaling option, we also explore several more drastic modifications. The three alternate architecture design points that we consider in this trade-off study all have 16 cores (twice the number of cores), with each core having half the L2 cache size (256 KB instead of 512 KB). Other modifications are as follows:

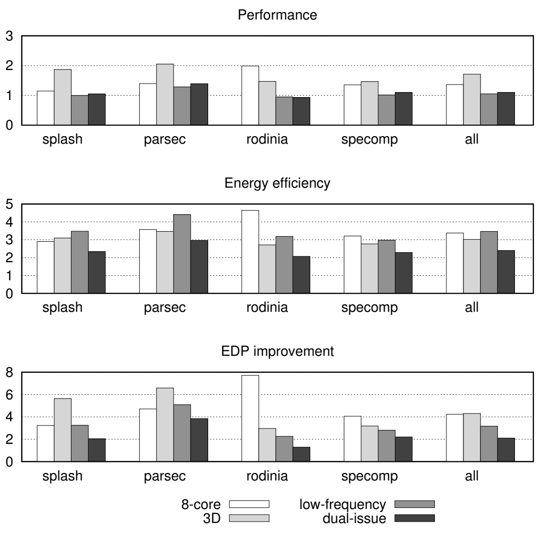

Figure 4: Average improvements per benchmark suite for the four 22 nm architecture design points over the 45 nm Nehalem baseline machine in terms of performance, energy efficiency and energy-delay product (EDP).

- The *3D* design point does not integrate an L3 cache but uses 3D stacked memory instead, which has a higher memory bandwidth and slightly shorter memory access time compared to regular DDR3 memory. This architecture results in a slightly bigger chip and nearly twice the power budget.
- The *low-frequency* design point reduces clock frequency and operating voltage which enables integrating 16 cores in a smaller power envelope. We assume a 16 MB L3 cache (2 times 8 MB per 8 cores) in order to reduce off-chip memory bandwidth pressure.
- The *dual-issue* design point replaces the 4-wide out-of-order cores with 16 less aggressive dual-issue cores. Reducing cache sizes compared to the 8-core architecture allows for integrating twice the number of cores at a slight increase in chip area.

The simulation parameters for these architectures are described in Table 2, along with chip area estimates and maximum observed power consumption for any of the benchmarks, as reported by Sniper/McPAT.

Note that the relatively high power consumption of the *3D* design point is caused by the fact that, for most of the benchmarks, the processor cores are much more active: compared to the other architectures, the time spent waiting for DRAM is lower here. Power consumption in 3D-stacked DRAM itself is slightly lower than that of the other architectures, mainly because long-distance off-chip communication on the DRAM bus is replaced by much more efficient short-range communication using through-silicon vias (TSVs).

6.1 Results

Figure 4 summarizes the average improvements per benchmark suite for the four 22 nm architecture design points over the 45 nm baseline architecture in terms of performance, energy efficiency and energy-delay product (EDP). (Energy

Parameter	Nehalem	8-core	3D	low-frequency	dual-issue
Sockets per system	2	1	1	1	1
Cores per socket	4	8	16	16	16
Core frequency	2.66 GHz	3.059 GHz	3.059 GHz	1.8 GHz	3.059 GHz
Core voltage	1.2 V	1.2 V	1.2 V	1.025 V	1.2 V
Issue width	4	4	4	4	2
ROB size	128	128	128	128	32
L2 cache size (per core)	256 KB	512 KB	256 KB	256 KB	256 KB
L3 cache size	8 MB per chip	32 MB	—	8 MB per 8 cores	8 MB per 8 cores
Memory bandwidth	8 GB/s	8 GB/s	128 GB/s	8 GB/s	8 GB/s
Memory latency	65 ns	65 ns	50 ns	65 ns	65 ns
Technology node	45 nm	22 nm	22 nm	22 nm	22 nm
Chip area	2× 243 mm^2	151 mm^2	181 mm^2	208 mm^2	187 mm^2
Maximum observed power	2× 99 W	80 W	130 W	58 W	102 W

Table 2: Simulated system characteristics used in the architectural exploration study.

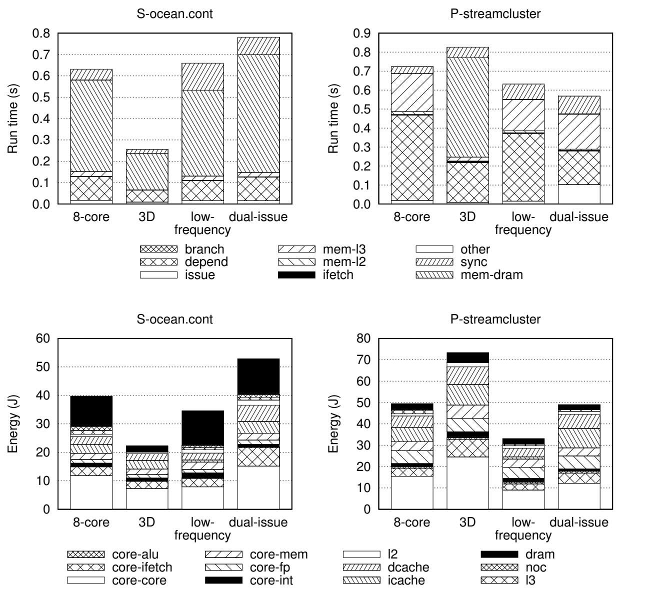

Figure 5: Time and energy stacks for `ocean.cont` from SPLASH-2 and `streamcluster` from PARSEC.

consumption in these results includes both dynamic and static energy consumption as reported by Sniper/McPAT.) Whereas the *3D* design point yields the highest absolute improvement in performance, its power consumption is rather high so it does not lead to the best architecture when energy consumption is taken into account. Instead, when optimizing for energy, the *low-frequency* design point is the optimum configuration for this set of benchmarks.

The high performance of the *3D* design point is especially apparent for benchmarks that are DRAM bandwidth bound. One such example is `S-ocean.cont`, see Figure 5 (left column) for cycle and energy stacks. For other applications with moderate working set sizes, the *3D* architecture suffers from the absence of an L3 cache. `P-streamcluster` for instance (Figure 5, right column) has a working set that does fit in the L3 cache for the *8-core*, *low-frequency* and *dual-issue* design points; on the *3D* architecture on the other hand, this benchmark needs to go out to the stacked DRAM, which — even though it is faster and more energy-efficient than regular DRAM — is still more expensive than using data that can be found in on-chip caches.

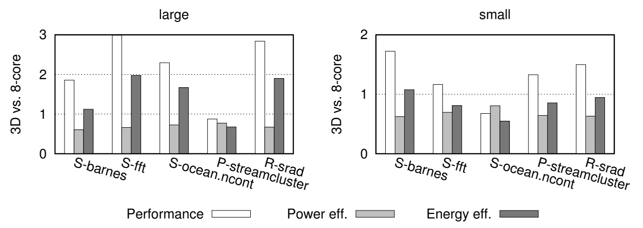

Figure 6: Improvement of the 3D design point over the 8-core design point in terms of performance, power and energy efficiency, using large (left) versus small (right) inputs: Different input sizes lead to different conclusions.

Out of the four benchmark suites, Rodinia is the one that stands out by having poor performance on all of the 16-core architectures. Even though Rodinia is written with GPUs in mind, which have many small cores, the Rodinia benchmarks do not seem to parallelize very well on a multicore CPU environment. One problem is that the data sets are not very large, which makes them fit in the caches — removing the benefit the *3D* design point had. Also, its fine-grained parallelism using short inner loops does not work well in a context where threads are heavy-weight and intercore synchronization is relatively expensive, as is the case in the GNU OpenMP runtime that was used.

6.2 Dependence on input size

It is well understood that a benchmark's input size may have a major impact on performance, and simulations using a small input may therefore not be a reliable proxy for larger inputs. This is because working set size often scales with input size, which may lead to very different cache behavior. Therefore, simulations need to be done using larger and more realistic input sizes, which further emphasizes the need for a fast simulator because of increased runtimes.

This not only applies to performance, but also to power and energy consumption. As an example, Figure 6 plots performance, power and energy efficiency of the *3D* design point relative to the *8-core* design point for a selection of benchmarks, using both large and small inputs. For some benchmarks, such as `S-barnes`, the conclusion as to which architecture is more power or energy-efficient is the same irrespective of the input size; however, for other benchmarks, the conclusion is reversed. For instance, using the small input, one would conclude that *3D*'s energy efficiency is worse compared to *8-core* for all of the selected benchmarks. However, when using the large input, *3D* is considerably more energy-efficient for three out of five of the applications considered. This shows that architecture studies should take caution when using reduced input sizes.

7. HARDWARE/SOFTWARE CO-DESIGN

We now go one step further and we use Sniper/McPAT to drive a case study in which we optimize both hardware and software. We do this for an important scientific kernel, namely stencil computation. Our kernel implementation allows for trading off data locality with redundant computation, which enables finding an optimum software configuration for a given hardware setting, and vice versa.

7.1 Heat transfer application

Our stencil computation benchmark models heat transfer across a regular 2D grid over a number of time steps. The heat transfer equation involves stencil computation in which temperature at a given point in time at each grid location is a linear combination of the temperatures of that location and its neighbors at the previous time step.

A naive implementation of the heat transfer equation computes a single time step at a time, and iterates over the complete grid to apply the stencil operation at each grid location (data element). This implementation has very poor data reuse because each data element is used only once per time step. By the time a data element is used again — at the next time step — the processor will have touched all other data elements, and hence, when simulating a large grid, likelihood for seeing a hit in the cache is small.

To improve memory locality, the algorithm has been reorganized to compute multiple time steps (s) on a small tile of the complete grid. This allows the tile's data elements to be reused multiple times, before moving on to the next tile. Additionally, when the application is parallelized by distributing tiles across cores, synchronization can be postponed by allowing each core to do more independent work before shared data at the tile boundaries needs to be communicated. A disadvantage is that at the edges of a tile, data from a neighboring tile about future time steps is needed — which is yet to be computed. This can be solved at the expense of doing some redundant work, by overlapping the edges of the tiles and letting the overlap decrease (by the width of the stencil) at each iteration. This way, the computation evolves — when time is represented as a dimension perpendicular to the grid plane — in a pyramid-shaped fashion, see Figure 7.

The heat benchmark was implemented using Intel Thread Building Blocks (TBB). Tasks are spawned which each simulate s time steps for a single tile, which TBB's work-stealing scheduler distributes over the available cores. There are many more tiles than cores so even in the presence of non-homogeneous memory access latencies the load balance of this application is typically very good.

7.2 Roofline model

The roofline model [30] provides a useful framework for high-level reasoning about algorithm efficiency and performance on a particular hardware platform. Figure 8 illustrates the roofline model for this particular application on our baseline architecture. The horizontal axis denotes the *arithmetic intensity* which is defined as the number of

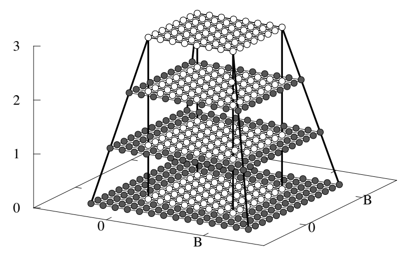

Figure 7: Illustration of three iterations of the heat transfer simulation, applied to an 8×8 tile. To satisfy the data dependencies up to the third step of the stencil, redundant computations (on the darker dots) are performed at the boundaries of the tile. From [11].

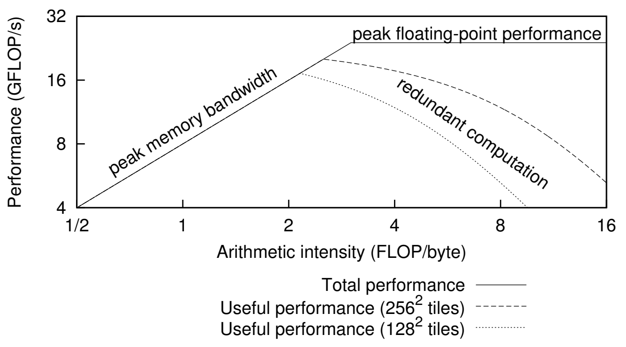

Figure 8: Expected performance profile of the heat benchmark using the roofline model [30], for the 8-core configuration with 128^2 and 256^2-sized tiles.

floating-point operations performed per byte loaded from memory. Initially, at low arithmetic intensities the problem is essentially memory bandwidth bound. Improving data reuse increases arithmetic intensity and increases performance, up to the peak floating-point performance of the machine.

In this kernel, arithmetic intensity can be raised by increasing the number of time steps computed per tile. Unfortunately, this optimization also increases redundant work, which causes useful performance to fall back once the amount of redundant computation becomes too high (dotted lines in Figure 8). The crossover point at which redundant work offsets the additional benefit of increased locality, moves towards higher values of s (higher levels of arithmetic intensity, to the right in Figure 8) for larger tile sizes. In addition, introducing too much redundant work will also flatten the useful performance curve, in the sense that an increasing fraction of the available machine's peak performance will be used for doing redundant work.

Trading off the amount of redundant computation against the increase in data locality is the main challenge in optimizing the heat transfer algorithm in particular, and stencil computations in general. This problem is difficult to solve analytically or through intuition because of the complex interplay between redundant work versus improving locality and how this affects energy consumption and performance. This reinforces the need for a fast simulation methodology

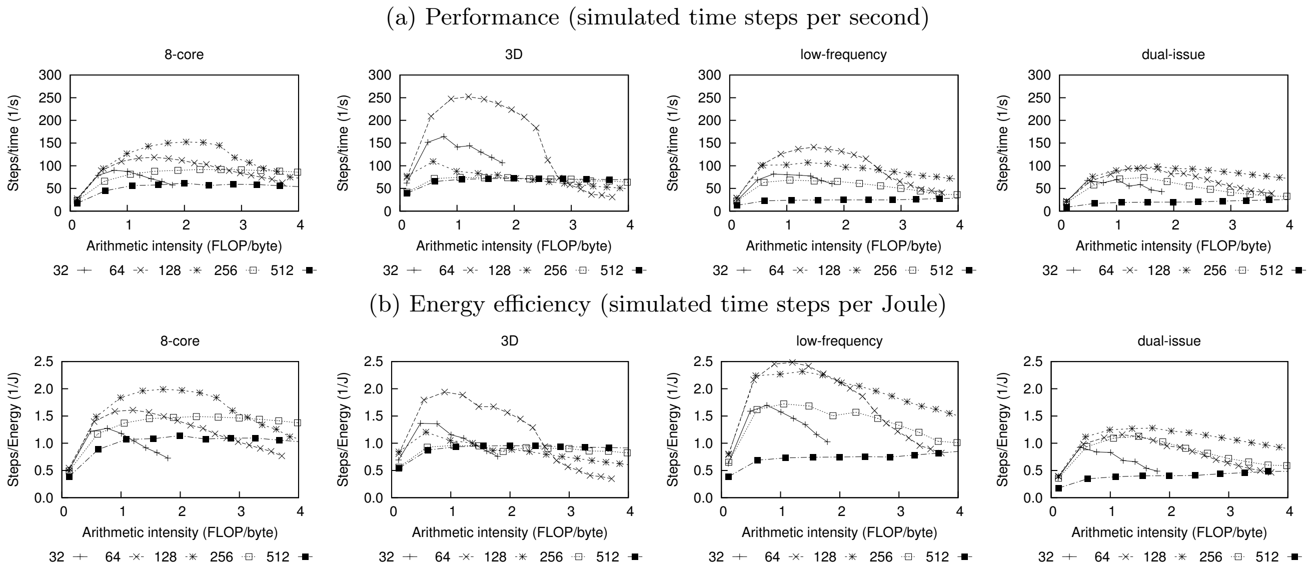

Figure 9: Hardware/software co-design for the heat benchmark when optimizing for performance and energy efficiency: four architecture design points are considered while varying software parameters such as tile size (see legend) and arithmetic intensity (horizontal axes).

for performance and power to explore these trade-offs and hardware/software interactions.

7.3 Co-design analysis

Figure 9 plots the simulation results for hardware/software co-design for the heat benchmark application. The grid domain is 4096×4096 elements. This domain is split up into square tiles measuring between 32 and 512 data points on each side. The complete domain is 128 MB in total and does not fit in the last-level cache for any of the architectures considered, so tiles always have to be loaded from main memory. The number of time steps performed on each tile before moving on to the next one varies between 1 and 65 steps. The graphs plot the achieved number of time steps per second, or per Joule of consumed energy, as a function of arithmetic intensity. The performance graphs in Figure 9(a) follow the basic roofline model from Figure 8: initially, increasing the number of time steps improves performance, which later falls back once the amount of redundant computation becomes too high. Additionally, each architecture has an optimal tile size which maximizes data reuse while still fitting in the cache. For example, the *8-core* design point reach a performance level of around 150 simulated time steps per second, using a tile size of 128×128. The working set of this application is two tiles worth of data, corresponding to the previous and current time steps. At one double-precision floating-point number or 8 bytes per element, the working set of 256 KB for the 128^2-sized tile fits in a core's private 512 KB L2 cache, whereas for the larger 256^2 tiles it does not — which makes performance significantly lower than that predicted by the roofline model. The three other architectures, which have an L2 cache of only 256 KB, reach their optimal performance using the smaller 64^2 tiles.

If we consider the effect of arithmetic intensity on energy rather than performance, we see that generally the optimum has shifted towards the left, meaning that less redundant work — and a slightly increased access rate to main memory — is preferred. This is because the ratio between access times of caches and DRAM is very high, making the performance aspect of more redundant work cheap when compared to extra DRAM accesses. On the other hand, when comparing the energy cost of DRAM accesses versus that of extra computation, the ratio is lower which makes the relative cost of the redundant work higher. Note that this extra cost consists of not just extra floating-point operations, which are in themselves very cheap (in the order of 0.5 nJ per double-precision operation, vs. 100 nJ per DRAM access[1]) but also the cost of extra instructions flowing through all stages of the out-of-order pipeline, extra cache accesses, etc. (In Figure 5, floating-point ALU energy represents only a small fraction of total core energy.)

When considering performance alone, the *3D* architecture clearly wins for this benchmark. The additional bandwidth of the 3D stacked memory allows for a steeper performance slope in the leftmost part of the performance graph, while the availability of 16 full-sized cores results in the highest peak performance across all architecture design points considered. Yet, tiles have to be kept small enough such that they fit in L2 cache; this architecture does not have an L3 cache so the cost of L2 misses, which become significant with tile sizes larger than 64^2, is much higher here than in the other architectures.

On the other hand, the *low-frequency* architecture, while being less high-performance than both the *8-core* and *3D* design points, reaches the highest energy efficiency. Because this application scales fairly good with core count, and the power envelope of the *low-frequency* chip is still modest, one could even consider another doubling of the number of cores which should almost double performance at very little additional cost in energy.

[1]According to the relevant component in the dynamic energy stacks computed by Sniper/McPAT, scaled by the number of FP operations or DRAM accesses throughout the benchmark, respectively.

Finally, the *dual-issue* architecture performs poorly for all software configurations. It was designed to be a middle-of-the-road out-of-order core, with a ROB size of 32 entries. This usually makes sense, as the complexity of a ROB (and hence area and energy cost) doubles superlinearly with its size, and it is not useful to discover more parallelism than the dual-issue execution stage is able to exploit. Yet, for this particular benchmark, performance is limited by dependencies, not the core's issue width. Significant amounts of instruction-level and memory-level parallelism do exist in the code; but since they are hidden behind long chains of dependencies, this parallelism would only be uncovered by a larger ROB.

The benefit from co-design. This case study clearly illustrates the necessity and huge potential of hardware/software co-design in order to achieve optimum performance and energy efficiency. If the algorithm would have been tuned for a specific architecture only, say the *8-core* design point, this would have yielded a maximum performance of 152 time steps per second or 1.99 time steps per Joule. However, by co-optimizing the hardware and the software, substantially better performance and energy efficiency can be achieved, up to 252 time steps per second if optimized for performance through the *3D* design point — an improvement by a factor 1.66× — or up to 2.48 time steps per Joule if optimized for energy through the *low-frequency* design point — an improvement by a factor 1.25×.

8. RELATED WORK

We now briefly describe related work in modeling and simulation, and architecture exploration.

8.1 Modeling and simulation

Architects in industry and academia heavily rely on detailed cycle-accurate simulation. The key benefit of cycle-accurate simulation obviously is accuracy, however, its slow speed is a significant limitation. Industry simulators typically run at a speed of 1 to 10 kHz. Academic simulators, such as gem5 [4] and PTLsim [34], are not truly cycle-accurate compared to real hardware, and therefore they are typically faster, with simulation speeds in the tens to hundreds of KIPS (kilo simulated instructions per second) range. Whereas cycle-accurate simulation is indispensable for detailed microarchitecture design studies inside a processor core, it does not scale to large multi-core systems.

Speeding up architectural simulation is an active field of research. Sampled simulation is likely the most widely used simulation speedup technique. The idea of sampled simulation is to simulate a limited, but representative number of simulation points [8, 25, 32]. Another approach that has gained interest recently is to accelerate simulation by mapping timing models on FPGAs [22, 29]. The timing models in FPGA-accelerated simulators are typically cycle-accurate, with the speedup coming from the fine-grained parallelism in the FPGA. Higher abstraction simulation, as employed in Sniper/McPAT, uses a different, and orthogonal, method for speeding up simulation: it models the processor at a higher level of abstraction. By doing so, higher abstraction models not only speed up simulation, they also reduce simulator complexity and development time compared to sampled and FPGA-accelerated simulation.

Existing power simulation methodologies typically rely on detailed activity factors. Wattch [5] uses SimpleScalar to obtained these activity factors at the cost of a 30% hit in simulation speed. PrEsto [26] incorporates power modeling in an FPGA-based simulation platform. Other work focuses on increasing the abstraction level of power modeling by using few high-level parameters and analytical scaling models [14, 18, 23]. Sniper/McPAT combines the high-abstraction levels for both performance (interval simulation) and power modeling (McPAT) into a fast yet accurate, hardware-validated simulation framework, that is still of low implementation complexity. All these characteristics are essential in allowing simulation to be used at early stages of design, and in achieving a short turn-around time for hardware/software co-design.

8.2 Design space exploration

The goal of architectural simulation is to drive design space exploration. A typical approach is to vary one architecture parameter at a time while holding the other architecture parameters constant in order to understand the sensitivity to that parameter. The number of simulation experiments that need to be performed quickly explodes with a large number of architecture parameters. Yi et al. [33] propose a method for identifying the most important architecture parameters to explore through a limited number of experiments. Empirical modeling is another popular approach to more quickly explore large design spaces, see for example [13, 16]. All of this prior work focuses on architecture exploration only, and is unlikely to be effective for hardware/software co-design, because these techniques rely on detailed cycle-accurate simulation to build the models. Employing these approaches for hardware/software co-design would require numerous detailed simulations for each version of the software; this would be prohibitively time-consuming.

9. CONCLUSIONS

There are significant performance and energy benefits to be achieved by co-designing hardware and software for high-performance processors and scientific applications. We argued that making design decisions that involve both hardware and software changes should be done early in the design cycle, so that correct high-level design decisions are made and both hardware and software evolve in the right direction. In order to do so, we developed Sniper/McPAT, a fast simulation platform that predicts both performance and power. Sniper, which is a parallel, high-abstraction multi-core performance simulator, predicts activity statistics that serve as input to McPAT which then predicts power consumption. We validated Sniper/McPAT against real hardware and we report average performance and power errors of 22.1% and 8.3%, respectively. Collecting activity statistics for McPAT does not slow down performance simulation in Sniper, which achieves a simulation speed around 2 MIPS on an 8-core simulation host machine.

Sniper/McPAT is a useful tool for driving architecture explorations as well as hardware/software co-design studies. Through a case study in which we considered an important scientific kernel, namely stencil computation, we demonstrated that co-designing hardware and software can lead to substantial performance and energy-efficiency improvements that cannot be achieved by exploring the architecture space only. More specifically, for our case study, we reported 1.66×

and 1.25× improvements in performance and energy efficiency, respectively. Overall, this paper makes the case for hardware/software co-design as an important and promising design paradigm for future processor system design.

10. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. This work is supported by Intel and the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT). Experiments were run on computing infrastructure at the ExaScience Lab, Leuven, Belgium; the Intel HPC Lab, Swindon, UK; and the VSC Flemish Supercomputer Center. Additional support is provided by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement no. 259295.

11. REFERENCES

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. Lester, Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California at Berkeley, 2006.

[2] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady. SPEComp: A new benchmark suite for measuring parallel computer performance. In *OpenMP Shared Memory Parallel Programming, LNCS*, 2104:1–10, 2001.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, pages 72–81, 2008.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, 2000.

[6] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *SuperComputing*, 2011.

[7] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, pages 44–54, 2009.

[8] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *ICCD*, pages 468–477, 1996.

[9] M. Facchini, T. E. Carlson, A. Vignon, M. Palkovic, F. Catthoor, W. Dehaene, L. Benini, and P. Marchal. System-level power/performance evaluation of 3D stacked DRAMs for mobile applications. In *DATE*, pages 923–928, 2009.

[10] D. Genbrugge, S. Eyerman, and L. Eeckhout. Interval simulation: Raising the level of abstraction in architectural simulation. In *HPCA*, pages 307–318, 2010.

[11] P. Ghysels, P. Kłosiewicz, and W. Vanroose. Improving the arithmetic intensity of multigrid with the help of polynomial smoothers. In *Copper Mountain Conference on Multigrid Methods*, 2011.

[12] W. Heirman, T. E. Carlson, S. Che, K. Skadron, and L. Eeckhout. Using cycle stacks to understand scaling bottlenecks in multi-threaded workloads. In *IISWC*, 2011.

[13] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. In *ASPLOS*, 2006.

[14] H. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. Eickemeyer. Abstraction and microarchitecture scaling in early-stage power modeling. In *HPCA*, pages 394–405, 2011.

[15] D. Kanter. Inside Nehalem: Intel's future processor and system. `http://www.realworldtech.com`, 2008.

[16] B. Lee, J. Collins, H. Wang, and D. Brooks. CPR: Composable performance regression for scalable multiprocessor models. In *MICRO*, pages 270–281, 2008.

[17] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *IEEE Computer*, 36:39–48, 2003.

[18] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, MICRO 42, pages 469–480. 2009.

[19] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*, pages 190–200. 2005.

[20] A. M. G. Maynard, C. M. Donnelly, and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *ASPLOS*, pages 145–156, 1994.

[21] Micron. TN-41-01: Calculating memory system power for DDR3, 2007.

[22] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer. HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing. In *HPCA*, pages 406–417, 2011.

[23] M. Powell, A. Biswas, J. Emer, S. Mukherjee, B. Sheikh, and S. Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *HPCA*, pages 289–300, 2009.

[24] Racktivity RC0816 datasheet. Available from `http://www.racktivity.com`.

[25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, 2002.

[26] D. Sunwoo, G. Y. Wu, N. A. Patil, and D. Chiou. PrEsto: An FPGA-accelerated power estimation methodology for complex systems. In *FPL*, pages 310–317, 2010.

[27] S. Thoziyoor, J. Ahn, M. Monchiero, J. Brockman, and N. Jouppi. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *ISCA*, pages 51–62, 2008.

[28] V. Uzelac and A. Milenkovic. Experiment flows and microbenchmarks for reverse engineering of branch predictor structures. In *ISPASS*, pages 207–217, 2009.

[29] J. Wawrzynek, D. Patterson, M. Oskin, S.-L. Lu, C. Kozyrakis, J. C. Hoe, D. Chiou, and K. Asanovic. RAMP: Research accelerator for multiple processors. *IEEE Micro*, 27(2):46–57, 2007.

[30] S. Williams, A. Waterman, and D. A. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, Apr. 2009.

[31] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *ISCA*, pages 24–36, 1995.

[32] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA*, pages 84–95, 2003.

[33] J. Yi, D. Lilja, and D. Hawkins. A statistically rigorous approach for improving simulation methodology. In *HPCA*, pages 281–291, 2003.

[34] M. Yourst. PTLsim: A cycle accurate full system x86-64 microarchitectural simulator. In *ISPASS*, pages 23–34. 2007.