Instruction-Based Energy Estimation Methodology for Asymmetric Manycore Processor Simulations

William J. Song
School of ECE
Georgia Institute of Technology
Atlanta, GA, 30332
wjhsong@gatech.edu

Sudhakar Yalamanchili
School of ECE
Georgia Institute of Technology
Atlanta, GA, 30332
sudha@ece.gatech.edu

Saibal Mukhopadhyay
School of ECE
Georgia Institute of Technology
Atlanta, GA, 30332
saibal@ece.gatech.edu

Arun F. Rodrigues
Sandia National Labs
Albuquerque, NM, 87185
afrodri@sandia.gov

ABSTRACT

Processor power is a complex function of device, packaging, microarchitecture, and application. Typical approaches to power simulation require detailed microarchitecture models to collect the statistical switching activity counts of processor components. In manycore simulations, the detailed core models are the main simulation speed bottleneck. In this paper, we propose an instruction-based energy estimation model for fast and scalable energy simulation. Importantly, in this approach the dynamic energy is modeled as a combination of three contributing factors: physical, microarchitectural, and workload properties. The model easily incorporates variations in physical parameters such as clock frequencies and supply voltages. When compared to commonly used cycle-level microarchitectural simulation approach with SPEC2006 benchmarks, the proposed instruction-based energy model incurred a 2.94% average error rate while achieving an average simulation time speedup of 74X for a 16-core asymmetric x86 ISA processor model with multiple clock domains operating at different frequencies.

1. INTRODUCTION

Advances in semiconductor technology and the evolution to manycore architectures have defined the challenges in energy or power management of processors. Increasing transistor counts can no longer be devoted to performance enhancements but rather must be used to improve performance per Joule or energy per instruction. These challenges in conjunction with the need to continue to effectively support serial components of an application have largely been responsible for the push towards asymmetric chip multiprocessors (ACMP) for sustainable and energy efficient performance growth [1, 2]. Thus, architecture-level energy and power modeling is central to the microarchitectural design space exploration for high core count future ACMPs.

The initial approaches to a single core power modeling were mostly measurement-based regression models. These models are known to be highly accurate but limited in applicability to explore evolutionary technologies and new architecture designs [3, 4]. In the past decade or so, power modeling approaches have been moving toward the counter-based model. This method decomposes the architecture into basic functional blocks (e.g., cache, execution units, decoders, etc.) and analyzes the power behavior at the block level by multiplying the access counts (or activity factor) by per-access unit energy. The gross power is calculated by adding contribution from each basic block. This approach is known to deviate from real measurements, depending on the accuracy and detail of the basic block model [5, 6, 7] but remains popular due to its applicability across new architectures and technologies. The predominant approach to using counter-based models is via cycle-level microarchitecture simulation. This is a time-consuming process where simulations can easily take days for small core count processors, and these simulations typically execute at tens of KIPs [8, 9]. This approach is not sustainable to high core count simulations.

This paper proposes an instruction-based energy estimation model that significantly speeds up energy simulation while maintaining accuracy close to that of detailed cycle-level simulation using the popular counter-based approach. Our model builds on the insights of counter-based energy calculation with highly detailed microarchitecture simulation to analyze how instructions exercise different execution paths through the core. The execution path information is distilled into an *architecture matrix* where the entries are the invocation counts of physical components in the datapath with respect to the workload metric. Thus, this matrix reflects the characteristics of microarchitecture and instruction set architecture (ISA). The per-access energy of physical components in the core is represented as *unit energy vector* which can be computed offline from circuit-level modeling tools [5, 6, 7]. The architecture matrix is used in conjunction with a *workload vector* that captures instruction-level properties of a thread, where the entries are the counts of key observation parameters such as instruction types (e.g., integer, floating-point, branch, or memory operation), number of operands, operand type, and so forth. Simple matrix-vector computations can replace the use of heavy microarchitecture models [8, 9] to increase the simulation speed while maintaining the energy estimation as accurate as the detailed models. This paper

presents our experiments with this model, its strengths and weaknesses, and areas of continuing work.

In the following, we first review previous approaches to processor energy modeling and then present our analytical energy model based on instruction-level execution path analysis. The proposed method is compared with the results produced from a highly detailed cycle-level x86 architecture simulator integrated with the energy models [6, 8]. Comparisons are made using SPEC2006 benchmarks executed on a 16-core asymmetric manycore processor with out-of-order and in-order cores operating at different clock frequencies.

2. RELATED WORKS

In this section, we overview the major past approaches to processor power modeling and evaluate their applicability to ACMP power modeling.

2.1 Measurement-based Models

The measurement-based approach models the processor as a black box. The output of the processor (i.e., current, power) is measured in response to the workload input. A predictive model is constructed based on the observed data [3, 4]. This approach can be expressed as Equation 1.

$$P(t) = \alpha \frac{C_w \cdot w(t)}{T_{sampling}} + \beta \tag{1}$$

In this equation, $w(t)$ is a vector of workload metrics typically comprised of instruction types or key operational parameters such as cache miss ratio, pipeline stall cycles, and number of executed instructions. C_w is a weight vector that is calibrated for a target processor through measurement. $T_{sampling}$ is the sampling period, and α and β are the constants for the first-order regression model. The difficulty of using this approach is creating a correct C_w vector. In general, this cannot be predicted for an unknown architecture or technology.

2.2 Counter-based Model

The counter-based approach is the most popularly used for architecture-level energy explorations. This method starts by decomposing the microarchitecture into basic functional blocks (e.g., caches, decoders, execution units). These basic blocks are analyzed by circuit-level modeling tools to compute the per-access energy [5, 6, 7]. The basic block power is calculated as the product of the per-access energy and access counts, and the processor power is computed as the sum of basic block powers. The counter-based method can be expressed as follows.

$$P(t) = \frac{E_{unit} \cdot C_{activity} + E_{static}}{T_{sampling}} \tag{2}$$

E_{unit} is a vector of the per-access unit energy of basic blocks. $C_{activity}$ is a set of counters that represent the basic block invocation counts in the sampling period. E_{static} is static energy dissipated by all blocks during the sampling period, $T_{sampling}$. The difficulty of using this approach is obtaining the set of counters. The counters are statistical data that are typically collected through detailed microarchitecture simulations that are known to operate around or less than 50KIPS [8, 9] which is a major bottleneck for large-scale power simulations.

2.3 Instruction-based Model

The study in [10] and related papers describe a method of instruction-based energy modeling for an embedded processor where inter-instruction energy dissipation is modeled at each pipeline stage as summarized in the following equation.

$$E(W) = \sum_{p=1}^{P} [\sum_{n=1}^{N} E(w_n | w_{n-1})] + E_{static} \tag{3}$$

The workload is composed of N instructions, and $E(w_n|w_{n-1})$ means the switching energy at the pth pipeline stage between the $(n-1)$th and nth instructions. This approach is similar to our proposed method in that a priori instruction information is used to estimate energy. However, this approach applies to a very specific case of a pipelined processor and inter-instruction relations. Further, we demonstrate with our approach that such detailed interactions are not critical.

3. MODELING

Energy is a complex output that is a function of 1) device and packaging characteristics, 2) microarchitecture effects, and 3) the properties of executed workloads. In contrast to prior approaches, we propose an approach that captures these three factors through instruction-level datapath analysis.

3.1 Analytical Model

The proposed method is expressed through the following equation that includes the features of the aforementioned approaches (i.e., measurement-based, counter-based, and instruction-based methods).

$$P_{core}(t) = \frac{E_{unit} \cdot A_{datapath} \cdot w(t) + E_{static}}{T_{sampling}} \tag{4}$$

In this equation, the total energy is separated into static and dynamic portions. E_{static} is the static energy dissipated by all physical components and modeled as an activity-independent term that is proportional to execution time and depends on the operating conditions such as voltage level and temperature. Note that instruction latencies can vary in wide range as in Figure 1 example which shows the delay distributions of two different types of instructions. Therefore, previous works mostly focused on analyzing the static energy prediction [3, 11]. Instead, we point out that workload execution time can be estimated through performance predictors [12, 13] rather than simulating detailed core models. These time estimates can be used to compute static energy and power.

Dynamic energy dissipation is incurred due to the switching activities of instruction executions. In Equation 4, E_{unit} is an $1 \times m$ vector of the per-access unit energy of m microarchitecture components. This can be computed offline using circuit-level modeling tools [5, 6, 7]. $A_{datapath}$ is an $m \times n$ architecture-level datapath matrix. Each column represents the microarchitecture components that an element of the workload vector will access during execution. $w(t)$ is a workload vector composed of n metrics which drive different sets of architecture components (correspoding to the column of $A_{datapath}$). Note that the product $E_{unit} \cdot A_{datapath}$ corresponds to the weight factor vector, C_w in Equation 1, and $A_{datapath} \cdot w(t)$ corresponds to $C_{activity}$ in

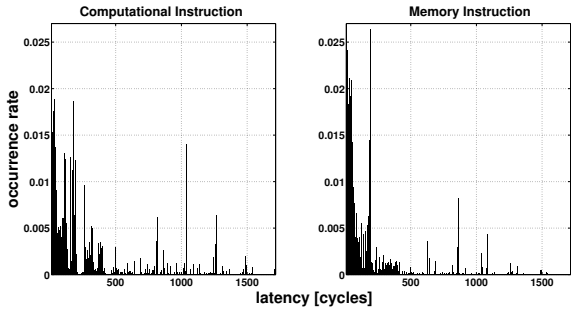


Figure 1: Delay distributions of x86 computation (add) and memory (mov) instructions.

Equation 2. Both are computable in our approach without the disadvantages previously described.

Table 1: Test Models for Energy Profiling

Configuration	In-order core	Out-of-order core
Feature size	16nm technology	
ISA	x86 IA32	
Clock freq.	1.85GHz	3.75GHz
V_{DD}	1.0V	1.0V
Pipeline depth	16	24
Decode width	2 insts/cycle	4 insts/cycle
Exec width	3 ports	6 ports
L1 cache	32KB	32KB
L2 cache	256KB	256KB

3.2 Unit Energy Calculation of Basic Blocks

The unit energy vector, E_{unit}, is computed offline for each core using circuit-level modeling tools. In this paper, we used the McPAT models for block-level energy computation [6]. Table 1 summarizes the configuration of the asymmetric processor analyzed in this paper. The processor comprises two different types of core execution models operating at different clock frequencies. The in-order execution model sequentially processes instructions in the order they are fetched. In the out-of-order model, a core can issue instructions out of order to avoid stalling the pipeline, and thus the instructions may complete not in order. The latter core has higher performance at the expense of higher energy per instruction. Consequently, each core type has different values of $A_{datapath}$ and E_{unit}.

3.3 Workload Representation

The simplest workload representation is a vector that lists the count of each instruction type occurring during the workload execution. It is simple to acquire either from a trace or the output of multicore emulators [13]. However, its usage is limited because of the vector size especially in the tested x86 ISA environment. Alternatively, a few key observations can be used to construct more compact forms of workload vector. First, consider that every instruction regardless of its type uses a common set of blocks such as the instruction fetch units and decoders. In this case, a raw instruction count can be used to estimate the baseline energy of all instructions. Second, some instruction types use a specific sequence of hardware blocks. For example, branch instructions invoke branch prediction components, but other computational instructions do not. Thus, the instruction classes can be compressed to a greater degree without loss of accuracy with respect to the energy estimation.

In this paper, we represented the workload vector, $w(t)$, as a set of 16 observed metrics that can be obtained from an instruction stream. The key metrics are the number of instructions (including macro and micro operations for x86 instructions), operand type and length, and opcode. The opcode does not list all of the operations but rather enumerates the types that exercise a distinct set of processor components such load, store, integer, floating-point, system function, jump, call, etc. Thus, the workload is modeled as the collection of activities that place demands on various parts of the core.

3.4 Architecture Matrix

The architecture matrix is defined in conjunction with the workload vector. Each element (i, j) of the matrix corresponds to the number of accesses to the processor component i invoked by the workload metric j. This design assumes that the workload vector always exercises the same set of microarchitectural components and consequent energy dissipation, which is not always the case. In order to understand the variances in the dynamic energy dissipation of instructions, we empirically analyzed instruction behaviors using a detailed x86 architecture simulator integrated with the energy models [6, 8]. The execution of every instruction was tracked through the pipeline, and the diversity of energy behaviors was analyzed.

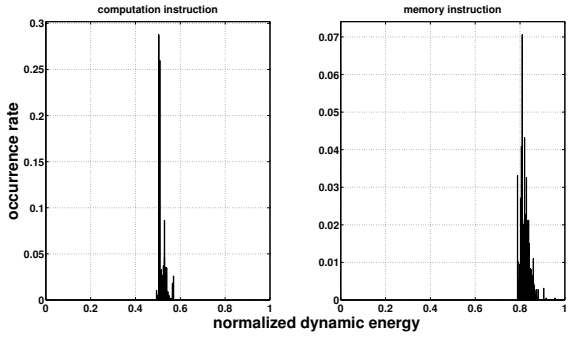


Figure 2: Dynamic energy distributions of x86 computation (add) and memory (mov) instructions.

Figure 2 shows an example of dynamic energy distributions for compute and memory instructions. The instruction energy is normalized to the maximum possible energy per instruction. As exemplified in the figure, we found that the energy distribution of different executions of an instruction is concentrated around the median value despite the variations in physical execution paths and microarchitectural state across each execution instance. For instance, if an instruction has an operand dependency on one of the preceding instructions, it drives the components for scheduling, tag broadcasting, and data bypass, etc. If there are no dependencies, the instruction then accesses the register files for data fetch. The two cases are driving

different sets of components, but the energy variation is only the energy difference between two alternatives, which we found to be much smaller than the overall energy dissipation. Thus, we found that disregarding inter-instruction dependency does not significantly degrades the accuracy, and the use of the most common execution path results in good accuracy relative to the results from detailed microarchitecture simulations. In addition, assuming the worst-case execution path for each instruction provides an upper bound of energy dissipation for the workload, which is often of more interest.

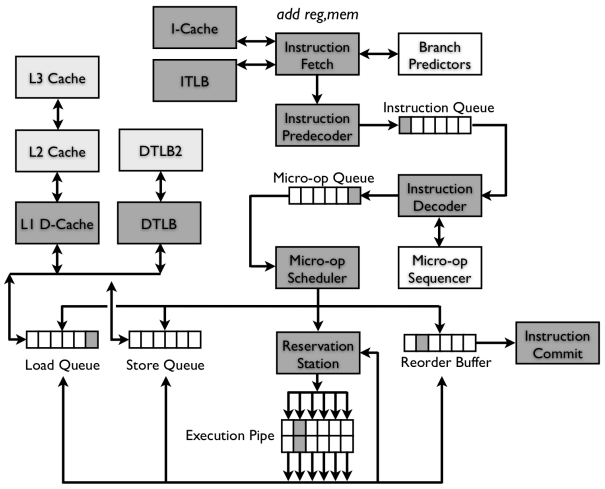


Figure 3: Execution path example of *add <reg,mem>* instruction in the out-of-order core execution.

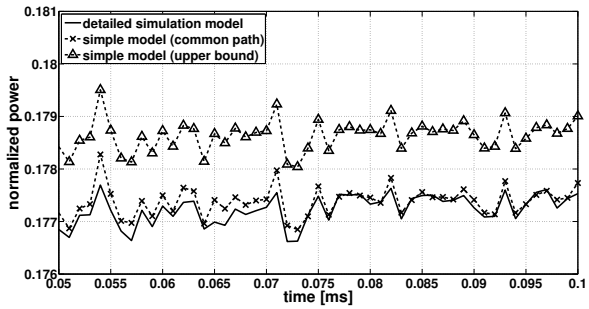
Such observations are used to construct the architecture matrix in Equation 4. The alternative execution paths of an instruction are rather small and can be analyzed offline for different architectures, for example as in Figure 3. In this figure, highlighted functional blocks are in the execution path of an example instruction *add <reg,mem>*. This instruction follows the common execution path in the fetch and decode stages. This operand has a register input and output and thus reads and writes the register file, while the other operand is fetched from the data cache through scheduling in the load queue. This instruction drives the ALUs for arithmetic and memory address calculations. Such execution path information can be recorded in the architecture matrix in Equation 4 by using four metrics in this case: decoded micro-op counts, operation type, operand counts, and operand types. However, we find that tracing the energy of components in the low-level of the architectural hierarchy (e.g., last-level cache, network) is relatively harder using this approach. In microarchitecture simulations, low-level components are not critical simulation bottlenecks as cores, and thus a conventional counter-based approach can still be applied to those components.

4. TRACING THE RUNTIME ENERGY

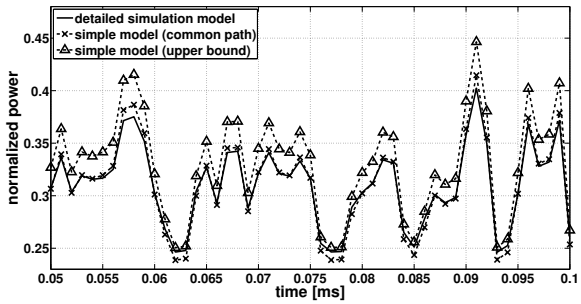
The proposed method was implemented for the x86 ISA and compared with results from a detailed cycle-level timing simulator. In both cases, the McPAT models were used to model the block-level energy computation, E_{unit} [6]. The instruction-based energy estimator was constructed for both in-order and out-of-order core models. For each core type, the datapath was analyzed for different workload metrics $w(t)$ and represented in $A_{datapath}$. This energy estimator was connected to the simulator front-end where the instructions were decoded and emulated. It replaced the cycle-level back-end timing model which was the major bottleneck for simulation time. Comparisons were made at this back-end stage to illustrate the accuracy and speedup of the instruction-based energy model. The workload metrics were collected from the executions of SPEC2006 benchmarks. The first 10 billion instructions were skipped before starting the simulation, and the simulator was warmed up by fast-forwarding the next 100 million clock cycles before comparison experiments were begun.

4.1 Point-to-Point Energy Comparison

The proposed method was first tested with a single core at a fine granularity by sampling the energy at every $100\mu s$ period. The computed energy divided by the sampling period is shown as the point-to-point power trace in Figure 4. This figure shows three cases: the detailed simulation model, common-path (the proposed instruction-based model), and upper bound using the worst case energy path for each workload metric. The upper bound case occurs when the instructions invoke the most energy-hungry components through the execution paths. The estimation error occurs primarily due to 1) machine dependencies as explained in Section 3.4 and 2) deviation of energy prediction in lower level caches (i.e., L2 cache).



(a) In-order core test with SPEC2006 gamess-cytosine



(b) Out-of-order core test with SPEC2006 bzip2-combined

Figure 4: Runtime power comparison between detailed simulation and instruction-based (simple) model with common-path and upper-bound architecture matrices.

4.2 ACMP Energy Comparison

The proposed method was extended to a larger scale. Energy estimation was performed for an asymmetric manycore processor model comprised of two different types of cores operating at different clock frequencies as presented in Table 1. Figure 5 shows the layout of a 16-core asymmetric processor model used in this experiment with 4 out-of-order cores and 12 in-order cores. Each core executed different SPEC2006 benchmarks. In this experiment, the energy dissipation was computed for the entire simulation time instead of point-to-point estimation.

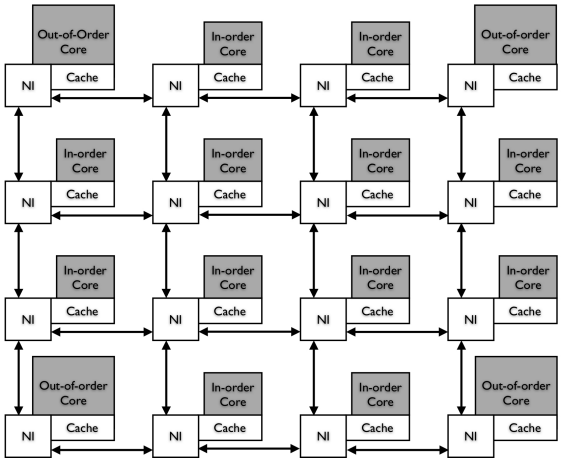


Figure 5: 16-core asymmetric processor simulation layout.

Figure 6 compares the total energy dissipation between the detailed simulation model and proposed method for the ACMP with each core in Figure 5 executing different SPEC2006 benchmark suites (i.e., multi-programmed mode). Most of the benchmarks match closely with an average error of 2.94%, but cactusADM and bwaves deviate up to 14.03%.

Figure 7 shows the simulation speedup of the dynamic energy computation time of the proposed method over the detailed simulation model. In this experiment, the speedup was achieved by replacing the detailed but compute-heavy back-end timing model with the instruction-based energy estimator. The simulation speedup shown in the figure does not include the common simulation overheads such as the front-end instruction emulation and low-level architecture components (e.g., memory controller, network, and last-level cache). Consequently, the proposed method accelerated the energy simulation speed by an average of 74.39X over detailed microarchitectural simulations.

4.3 Summary

In this paper, we presented an instruction-based dynamic energy estimation methodology. The proposed approach is based on the empirical observations regarding the energy behavior of instructions in cores; the variance in energy dissipation is rather small even in an complex out-of-order core. Consequently, using the principle of common execution paths, we can construct a matrix-vector model of dynamic energy dissipation at the instruction level with little compromise in accuracy over detailed simulations. The major features of the proposed approach include the following.

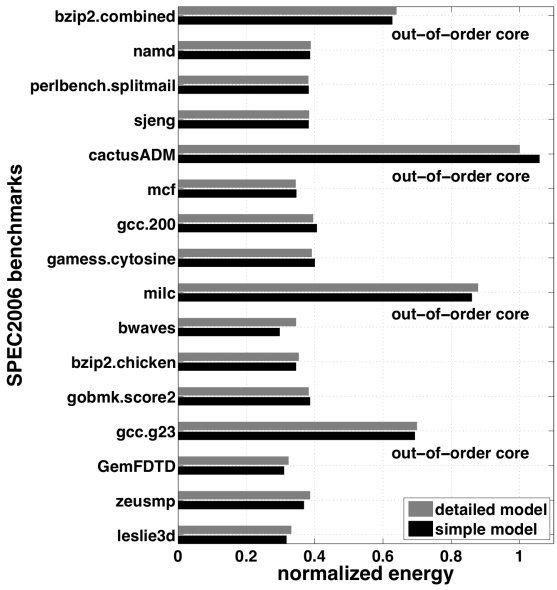


Figure 6: Total energy comparisons between the proposed method (simple model) and detailed model.

- The decomposition of power into three contributing factors (i.e., physical properties, microarchitectural effects, and workload characterization) enables the configurable implementation and exploration for dynamic energy dissipation across various microarchitecture designs and ISAs.
- One to two orders of magnitude speedup in the energy calculation of programs with little sacrifice in accuracy enables the fast exploration of energy behavior for high core count processors.
- The approach is easily integrated into manycore processor emulators and instruction-level simulators
- The workload information is characterized as in Equation 4, and thus the energy or power estimate fully reflects the properties of executed workloads.

Collectively, these features enable the fast design space exploration of energy behaviors in high core count architectures.

5. CONCLUSION

The manycore era has opened up a new dimension of processor and system design that requires the rapid and comprehensive explorations of design options. Such explorations are limited by simulation speed and accuracy. They must also include energy and power as the first class entities to be modeled, and cores are the primary source of energy and hence power dissipation in high performance processors. Current techniques for modeling energy and power rely on cycle-level microarchitectural models for accuracy at the expense of long simulation times. This approach is not sustainable to high core counts. In this

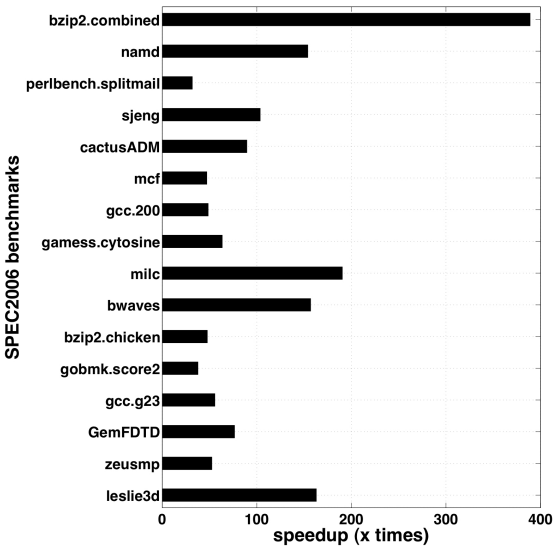


Figure 7: Speedup of dynamic energy estimation time for the ACMP executing 16 SPEC2006 benchmarks.

paper, we proposed an instruction-based energy estimation model and methodology that speeds up energy estimation by 1-2 orders of magnitude with marginal sacrifices in accuracy over detailed simulation models.

The proposed instruction-based energy profiling method decomposes the power into three contributing factors: device and packaging characteristics, microarchitecture, and workload properties. The strength of this approach is that the implementation is configurable across various types of microarchitecture designs and ISAs. Further the model easily incorporates physical variations such as different core types operating at different frequencies. The analytical composition of the microarchitecture behaviors in response to the workload enables the use of simple core models in the manycore simulations to increase the simulation speed while maintaing the accuracy of power estimation as close as possible to that of detailed simulation models.

6. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of this work by the Semiconductor Research Corporation (Task ID# 2084.001), National Science Foundation under grant NSF CNS-855110, and Sandia National Laboratories.

7. REFERENCES

[1] T. Y. Morad, U. Weiser, A. Kolodny, M. Valero, and E. Ayguade, "Performance, Power Efficiency and Scalability of Asymmetric Cluster Chip Multiprocessors," in *Computer Architecture Letters*, June 2006.

[2] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-core Architecture for Multithreaded Workload Performance," *Int'l Symp. on Computer Architecture*, June 2004.

[3] A. Sinha and A. P. Chandrakasan, "JouleTrack - A Web-based Tool for Software Energy Profiling," *Design Automation Conf.*, May 2001.

[4] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Skeikh, and S. Yardi, "CAMP: A Technique to Estimate Per-Structure Power at Run-time Using A Few Simple Parameters," *Int'l Symp. on High Performance Computer Architecture*, Feb. 2009.

[5] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architecture-level Power Analysis and Optimizations," *Int'l Symp. on Computer Architecture*, June 2000.

[6] 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," *Int'l Symp. on Microarchitecture*, Dec. 2009.

[7] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to The Design and Analysis of Future Memory Hierarchies," *Int'l Symp. on Computer Architecture*, June 2008.

[8] G. H. Loh, S. Subramaniam, and Y. Xie, "Zesto: A Cycle-level Simulator for Highly Detailed Microarchitecture Exploration," *Int'l Symp. on Performance Analysis of System and Software*, Apr. 2009.

[9] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, and A. G. Saidi, "The M5 Simulator: Modeling Networked Systems," *Int'l Conf. on Parallel Processing*, Sept. 2008.

[10] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "An Instruction-level Energy Model for Embedded VLIW Architecture," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Sept. 2002.

[11] A. Bhattacharjee and M. Martonosi, "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors," *Int'l Symp. on Computer Architecture*, June 2009.

[12] S. Cho, S. Demetriades, S. Evans, L. Jin, H. Lee, K. Lee, and M. Meong, "TPTS: A Novel Framework for Very Fast Manycore Processor Architecture Simulation," *IEEE Micro*, Aug. 2006.

[13] C. Kersey, A. F. Rodrigues, and S. Yalamanchili, "A Universal Parallel Front End for Execution Driven Microarchitecture Simulation," *Workshop on Rapid Simulation and Performance Evaluation*, Jan. 2012.