Integrated, Application-Level, Performance-Energy Modeling for Heterogeneous Architectures

[2]Hyesoon. Kim, [1]E. Anger, [2]P. Gera, [3]Jeremiah J. Wilke, [4]Patrick S McCormick, [1]S. Yalamanchili

[1]School of ECE and [2]School of CS, Georgia Institute of Technology, Email:hyesoon.kim@gatech.edu
[3]Sandia National Labs, Livermore, CA 94550, Email: jjwilke@sandia.gov
[4]Los Alamos Laboratory, Email: pat@lanl.gov

1. The Need for Application Level Energy Modeling

Energy consumption is heavily dependent on hardware implementations and most energy models are built on low-level hardware dependent metrics. However, energy consumption is strongly affected by application characteristics such as choice of algorithms, data structures, and program inputs. For example, Figure 1 shows the energy consumption of a BFS algorithm on NVIDIA GPUs for 3 different inputs. The results show 4 different implementations of BFS (HIPC, LS, SHOC1, SHOC2) and the energy is normalized to the minimum energy consumption version for each input. The figure shows that depending on an input to the program, both the best implementation version and the energy consumption profile vary significantly. For example, the SHOC1 implementation shows the minimum energy consumption with the eu-2005 input, but it consumes 29 times more energy than the best version with the italy input. Unfortunately, there is little understanding how these application-level attributes affect energy consumption on a specific hardware platform. The increasingly divergent computational and memory reference characteristics of emergent applications, e.g., graph analytics, adaptive mesh refinement, etc., exacerbate this problem.

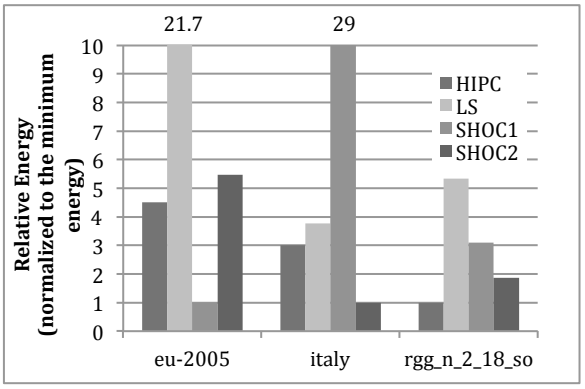

Figure 1 Energy consumption behavior for BFS algorithm on GPUs

The challenge is in formulating *application-level* energy and performance models, which inform energy and performance consequences of application implementations on different hardware platforms. Lessons can be learned from application-level performance modeling efforts such as SST [6] and Byfl [2][3], e.g., algorithm skeletons for abstracting communication behavior in SST (particularly the macroscale components [7]) or estimates of memory demand from Byfl. We argue that similar techniques are necessary for abstracting energy behavior. Specifically we see a need for decoupling architecture dependent parameters (e.g., memory access behavior) from hardware *implementation* characteristics to isolate application-dependent energy characteristics. Towards this end we advocate a system approach built on the use of statistical techniques[1] for constructing application level energy models and using these in conjunction with performance models to inform application developers.

2. Constructing Extensible, Scalable Models

Figure 2 shows the proposed system. The system consists of 3 steps: application profiling, architectural model construction, and energy model construction. First, an application is profiled to generate architecture *independent* metrics by instrumenting the application using LLVM and Byfl [2][3], e.g., intermediate instruction statistics. In the second step, we collect architecture *dependent* application characteristics (e.g., #misses, #branch mispredictions) using PAPI[4] and energy values using RAPL[5]. This is used to construct an architecture model (ISA translation

model and memory hierarchy model). Finally, an energy model is constructed using the statistical modeling tool Eiger [1], which takes input from Byfl, architecture specifications (cache sizes, DRAM frequencies etc.) and ISA information and produces energy values. By constructing the ISA translation model and the memory hierarchy models, we can separate architecture dependent metrics from the energy model. Note that PAPI and RAPL measurements are only used to construct models and are not direct input metrics to the final energy model.

The final energy model produces energy consumption values for application developers by displaying estimated energy consumption values for each function call (or even for source line). At the same time, this energy model is exported for use in an SST macroscale component for energy consumption simulation for different hardware implementations

3. Discussions

Challenges addressed: We target two goals. (a) Scalable energy model construction: Future compute nodes may compromise several hundred to several thousand cores with large memory footprints - traditional modeling and simulation techniques are impractical. Our approach is to use statistical methods to develop energy models applicable across different architectures and applications. (b) Energy related metrics that are associated with higher level program behaviors such as algorithms, data structures, and program input characteristics.

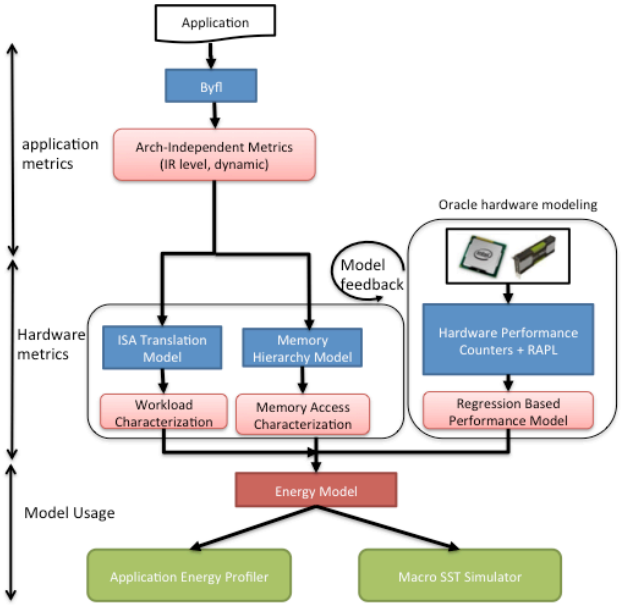

Figure 2 Proposed modeling system

Uniqueness: We provide an integrated model that yields an understanding of application behaviors as well as implications for architecture design. Furthermore, the proposed energy model is applied across a range of architecture types, e.g., both CPUs and GPUs, targeting heterogeneous systems.

Maturity: Our preliminarily energy model evaluated with hardware performance counters shows very good accuracy. We find the energy consumption is often dependent on relatively few application metrics, implying hardware models can be constructed with reasonable error bounds.

Novelty: Despite having become nearly as important as performance profilers and performance debugging tools, energy profiling tools are relatively rare. For example, to our knowledge there are no tools that can identify program segments where most of the energy is spent.

Applicability: The proposed system is an integrated simulator and profiler providing feedback to developers on the energy consequences and energy behavior of different algorithms, data structures, application inputs or architecture parameters. Based on the model's outcomes, both application developers and hardware designers can choose optimal design points.

Effort: First pass versions of most of individual modeling components/frames in Figure 2 have been implemented. The memory hierarchy model and extensions to other architecture platforms and to multithreaded applications are the main remaining efforts.

References

[1] Andrew Kerr, Eric Anger, Gilber Hendry, and Sudhakar Yalamanchili. “Eiger: A framework for the automated synthesis of statistical performance models”, In High Performance Computing, 2012.

[2] Byfl, https://github.com/losalamos/Byfl

[3] Scott Pakin, Patrick McCormick, “Hardware-independent application characterization,” IISWC 2013

[4] Dongarra, J., London, K., Moore, S., Mucci, P., Terpstra, D. "Using PAPI for Hardware Performance Monitoring on Linux Systems," Conference on Linux Clusters: The HPC Revolution, Linux Clusters Institute, Urbana, Illinois, June 25-27, 2001.

[5] RAPL, https://01.org/blogs/tlcounts/2014/running-average-power-limit-%E2%80%93-rapl

[6] SST: Structural Simulation Toolkit, https://sst-simulator.org, Sandia National Laboratories

[7] SST Macroscale Components, http://sst.sandia.gov/about_sstmacro.html, Sandia National Laboratories