2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

Modeling and Analysis for Performance and Power

Jee Whan Choi (4^{th} Year)
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, USA
Email: jee@gatech.edu

Richard W. Vuduc (Advisor)
Computational Science and Engineering
Georgia Institute of Technology
Atlanta, USA
Email: richie@cc.gatech.edu

***Abstract*—Accurately modeling application performance for specific architectures allows us to understand and analyze the impact of various architectural features on performance which will ultimately lead to improved performance and better architecture design choices for efficiency and scalability on future systems. Moreover, with the end of Dennard scaling, processors can no longer maintain constant power per unit area as before and consequently power and energy efficiencies has become arguably an even more important factor than performance for future systems.**

In this paper we propose to search for a method of modeling that, given an application, extract a unique set of performance-influencing features and automatically determine and match them to relevant architecture-specific features for the purpose of deriving accurate models for performance, energy and power. Different applications will display different sets of features which will allow for more accurate models and analysis. Ultimately these models will help us to better understand the impact of application and architectural features on performance, energy and power and help us design applications and systems to maximize their efficiencies.

***Keywords*-performance modeling, energy and power modeling, scientific computing, sparse matrix-vector multiply, lattice quantum chromodynamics, fast multipole method**

I. Introduction

Predicting the performance of an application accurately on a given architecture is an extremely difficult task due to hardware factors such as complex architectural features and software factors such as implementation choices and compiler optimizations. On one end of the spectrum, we have architecture-centric solutions such as cycle-accurate simulators (CAS) [1] are very accurate but require a full simulator that are difficult to write and often require an order of magnitude longer to run as compared to actual runtime. The problem becomes even worse for multi-core systems.

On the other end of the spectrum, there are algorithm-centric methods that captures the most important features such as flops and bytes and use those information in conjunction with system-specific parameters such as peak throughput and bandwidth to predict performance [2]. These methods may be much less accurate than CASs but are much easier to derive. However, these constant performance models (CPM) also have certain limitations and are sometimes too simple to be applied to applications that have complex DAGs and multiple phases with varying flops and bytes. Functional performance models [3] may be able to address a few of these issues but not all. There are also various other methods of modeling performance that have their unique sets of advantages and disadvantages [4], [5].

So the question is, is there a one-fits-all solution for predicting performance? The answer is no because each method addresses different issues and are useful in different circumstances. For example, when trying to improve performance by identifying bottlenecks and opprotunities for asynchronous execution, roofline model would be very useful. However, when trying to maximize instruction throughput or identify pipleline stalls, a CAS would be most useful.

For the purpose of analyzing algorithm and architecture in the context of performance, we believe that *all* algorithmic features that influence performance should be extracted in order to maximize accuracy of the model, and then these features should be, given a particular architecture, mapped to architectural specific parameters to create a cost model for execution time. Different algorithmic features will have different costs on different architectures, and the model must capture this relationship accurately while at the same time it must be general enough to apply to any application and its set of features. This approach differs from other methods in that we do not generalize all applications by certain identical features, but take each application and extract its own unique set of performance influencing features and map them to specific architectures to autmatically drive a cost function, and at the same time cover a wide enough range of features so that this approach can be applied to any application.

Once we apply these models to both CPU and GPU, we can also use them to model performance on CPU-GPU hybrid systems which can aid us in designing the optimal hybrid system as well as to schedule our application to minimize execution time on these systems.

In order to accomplish all these things, we first need a way of extracting and representing the features at algorithm level. Once we have these features we must be able to relate these features to architectural features. Once we have a mapping of the relationships, we can finally derive a cost function for the algorithm at hand. So far, we have studied several kernels and applications and derived accurate performance

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.304

models for these using features that were specific to each application at hand on both CPUs and GPUs. We believe that our experience in modeling these different kernels on various architctures will allow us to more readily define and formalize algorithmic features and map them to any architecture.

Lastly, we need to address the problem of energy and power. We believe that energy and power are costs that are similar to execution time, and can be modeled using a similar approach of extracting power influencing features from algorithms and map them to architectrual power influecing features to derive a cost function. Power has been modeled in a manner similar to performance in other research [6] and we believe that this is indicative of power and energy being similar to execution time as a cost.

In section II, we will cover some of our previous research in performance modeling to demonstrate how certain algorithmic features can be mapped to architectural features to derive a performance model. In section III, we will cover some of our recent studies on the relationship between energy/power and performance, and we will conclude our paper in section IV with what we have yet to accomplish to complete our story.

II. Case Studies

In this section we cover various kernels and their performance models. Since different kernels have different computational requirements and are therefore influenced by different architectural parmaeters, we try to capture only the relevant algorithmic and architectural features specific to the kernel and the hardware in our model when we try to predict the execution time. Due to page limitations, we cover only the basics in this section, but readers are encourged to look up the relevant papers for more details.

A. Sparse Matrix-Vector Multiply

In this study [7] we implemented a highly efficient GPU sparse matrix-vector multiply (SpMV) kernel and derived a performance model for the purpose of automatically tuning the kernel for optimal GPU specific parameters.

We abstracted the computation into number of *thread blocks*, blocking size $r \times c$ (for fill ratio) and N, the number of rows assigned to a thread block. In addition, a small number of off-line benchmarks were used to measure certain costs such as kernel startup and achievable performance of the target GPU for our SpMV implementation. Together, a cost model was created to predict the execution time for any input matrix. Our performance model predicted the execution time to within an average and median error of 3.74% and 2.73% respectively. A graph comparing the predicted and measured execution times for a Tesla C870 GPU is shown in Figure 1.

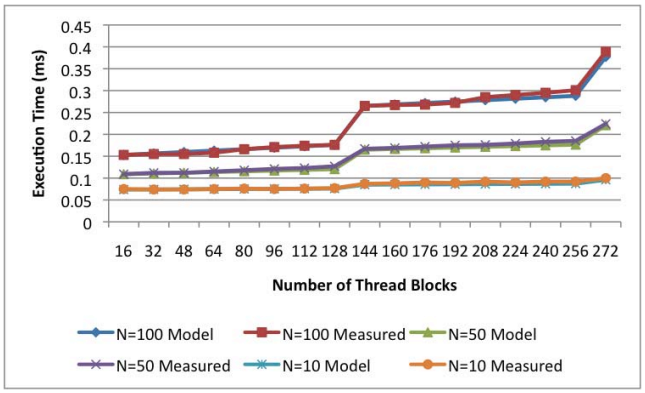

Figure 1. Comparison of model to real data on Tesla C870

B. Lattice Quantum Chromodynamics

Lattice-based quantum chromodynamics (LQCD) is a theory in particle physics about the strong force and is thus among the most fundamental areas of scientific inquiry in computational science. LQCD is a particular well-established algorithmic approach for performing QCD computations.

LQCD computes solutions to a sparse system of linear equations using Krylov subspace solvers, such as conjugate gradient (CG) and BiCGstab, for which a typical bottleneck is the matrix-vector multiply (matvec) step. The matrix for LQCD comes from an 8-point stencil discretization in a high-dimensional space. However, instead of explicitly constructing and storing the (sparse) matrix needed to perform the matvec, LQCD methods use matrix-free approaches in which matrix entries are evaluated on-the-fly. The advantage of this approach is to reduce total storage and increase the computational intensity of the matvec relative to the explicit sparse matrix approach, which would better utilize the massive computational resources available on GPUs. The matvec in LQCD consists of two so-called Dslash operations, one that uses even lattice sites as input to compute the output at odd lattice sites, and vice versa. The Dslash operator consumes majority of the cycles needed for the overall solver.

A conventional implementation of Dslash in single precision requires 1320 flops per 1440 bytes of main-memory data, which already constitutes a much higher computational intensity than classical matvec for stored matrices. In QUDA [11], Clark et. al use various techniques to further improve this intensity by a factor of 1.5. However, even at 960 bytes of memory traffic, the intensity of Dslash remains heavily bandwidth-bound, and nowhere near the theoretically smallest possible memory footprint of 448 bytes.

The latest development in cache blocking for stencil computations, known as 3.5-D blocking [12], has shown

improvements of 1.5× to 1.8× over previous state-of-the-art 7-point stencil implementations for CPU and GPU respectively. Therefore, intuitively one would assume that using this technique on Dslash should also show significant improvements on both CPU and modern GPUs that have caches, essentially by improving reuse at each lattice site.

Our back-of-the-envelope estimates suggest that 3.5-D blocking on an Intel Westmere-EP CPU, with 12 MB of last-level cache (LLC) and 22 GB/s of bandwidth, could yield up to 38 Gflop/s of performance. Our implementation, which we believe to be the best CPU implementation of Dslash currently available, comes within 90% of this target, thereby validating our approach.

A similar estimate for an NVIDIA GTX 480 GPU reveals that there would be no visible performance gain for Dslash over the current QUDA implementation. This is due in part to the GPU having much smaller last-level cache, and in part to lattice site data being much larger than those of a simple 7-point stencil. The redundantly loaded boundary of cache blocks would overshadow any savings.

However, there are alternative approaches better suited to a GPU architecture. One such method is to restrict Dslash to 2.5-D blocking, loading neighboring data in the temporal dimension on-the-fly. We estimate that although this would effectively raise the lattice site data from 448 bytes (3.5-D blocking) to 640 bytes, it would alleviate the redundant boundary problem by both allowing for larger 2-D blocks and minimizing the surface area of the blocks. This approach could yield a modest performance gain of 1.2 and we believe that a more thorough investigation will reveal that carefully blocking the data in the register file, the L1, and the L2 cache will increase the effective block sizes and increase performance significantly up to approximately 1.5

We can go further. Since all Dslash operation occurs in pairs, with the latter using the result of the former as input, fusing the two Dslash operations into one will allow us to effectively double the flops while keeping the data size constant. This will again be non-trivial, since several blocks of data need to be computed for the first Dslash before they can be used for the second. This will require careful use of barriers, as well as careful assignment of threads to workload in both Dslash operations. However, if done correctly, it will yield an additional 2 speedup on top of 2.5-D blocking, for a total of 3 in speed up over current implementation of QUDA.

More details on our back-of-the-envelope estimates can be found in [10], [12].

By extracting the relevant algorithmic features - in this case, the number of flops and bytes and data access pattern - and combining it with the relevant architectural feature - in this case, compute throughput, bandwidth, and cache size - we were able to come up with achievable performance by applying a set of simple blocking schemes.

C. Fast Multipole Method

Fast Multipole Method (FMM) is a $\mathcal{O}(n)$ work optimal algorithm with guaranteed approximation accuracy for a N-body particle simulation involving n interacting particles which naïvely requires $\mathcal{O}(n^2)$ computations. FMM is widely regarded as one of the most significant algorithms in scientific computing [13]. FMM consists of six phases: *Up*, *Down*, *VList*, *UList*, *Wlist*, and *Xlist*, each of which has different compute and memory characteristics. However, since the *VList* and *UList* phases constitute for over 90% of the overall execution time, we currently only model these two phases, leaving the analysis and modeling of the other phases for future work.

In this study, we present analytical performance models for *UList* and *VList* phases. The models have both algorithmic (number of target/source points, max points per leaf, desired accuracy) and architectural (last level cache size, memory bandwidth, etc,.) parameters making it practical and usable. We also validate our models using performance counter data.

III. Current Research

With the upcoming Exascale supercomputing clusters being limited to approximately $20MW$ of power and the energy cost of keeping the systems running out-pacing system build costs, power and energy efficiency has become one of the hottest research topics in the area of high performance computing.

As we have mentioned, we believe that power and energy are metrics similar to execution time in the sense that they are simply costs that have to be paid for running an application on a given system. Therefore, it should be possible to extract specific features that determine performance for a given application and determine energy and power costs corresponding to those features. For example, number of flops will determine how much energy is expended on the ALU, whereas the number of memory reads will correspond to energy usage by the DRAM and the bus, similar to how each of these features have an execution time costs determined by the capability of the architecture.

Some of current research in energy and power modeling involve dynamic voltage and frequency scaling (DVFS) [15], where the voltage, and subsequently the frequncy, is lowered to explore the possiblity of reducing energy usage without the loss of performance.

In other studies [14], [16], power and energy characterstics for various applications were profiled on single and multi-node systems and simply analyzed. These studies have proposed the possiblity of using power models to find energy bottlenecks energy optimizing schedules.

From our studies, we have found enough evidence that applications can be optimized to save energy and power and that an accurate energy and power models will be beneficial in analyzing the impact of application and architectural

features on their consumption and ultimately lead to better design choices for future applications and systems.

IV. Conclusions and Future Works

We have presented motivations for performance and energy/power modeling that extracts application features that influence performance and match them to corresponding architectural features to predict performance and energy/power. Performance models will allow us to analyze the algorithm and the architecture together to determine how these features influence each other and ultimately guide us into making better design decisions for future applications and systems. Similarly, we have presented evidence based on related research that has shown that 1) energy and power costs are dependent on application features that also determine performance, and 2) there is room for optimizing energy and power through DVFS and identifying energy bottlenecks. Theese conclusions tells us that having an accurate energy and power model is cruicial in reducing their consumption and help meet the energy requirements for Exascale.

Future works include coming up with a method of formalizing application features for different applications, perhaps even automatically by analyzing certain levels of algorithm (high level vs. real code). Then these application features must be matched to relevant architectural features and an *accurate* model must be derived. This process might be iterative or heuristic in terms of search space (e.g. finding the right blocking technique that yields the best performance). Then this process must be repeated and refined for determining a similar energy/power model.

References

[1] Peter Strazdins, Bill Clarke, and Andrew Over. 2007. Efficient cycle-accurate simulation of the UltraSPARC III CPU. In Proceedings of the thirtieth Australasian conference on Computer science - Volume 62 (ACSC '07), Gillian Dobbie (Ed.), Vol. 62. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 221-228.

[2] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 4 (April 2009), 65-76. DOI=10.1145/1498765.1498785 http://doi.acm.org/10.1145/1498765.1498785

[3] Alexey Lastovetsky and Ravi Reddy. 2007. Data Partitioning with a Functional Performance Model of Heterogeneous Processors. Int. J. High Perform. Comput. Appl. 21, 1 (February 2007), 76-90. DOI=10.1177/1094342006074864 http://dx.doi.org/10.1177/1094342006074864

[4] Sunpyo Hong, Hyesoon Kim, "An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness," Proceedings of the 36th International Symposium on Computer Architecture (ISCA) , Austin, TX, June 2009.

[5] Tikir, M., Carrington, L., Strohmaier, E., and Snavely, A. A genetic algorithms approach to modeling the performance of memory-bound computations. in Proceedings of the SC07 Conference (reno, nV, nov. 1016). acm Press, new york, 2007.

[6] R. Ge and K. W. Cameron, Power-Aware Speedup, in 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS), Long Beach, California, Mar. 2007.

[7] Jee W. Choi, Amik Singh, Richard W. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on GPUs. In Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP10, Bangalore, India, January, 2010.

[8] Sam Williams, Richard Vuduc, Leonid Oliker, John Shalf, Katherine Yelick, and James Demmel. Optimizing sparse matrix-vector multiply on emerging multicore platforms. Jour- nal of Parallel Computing, 35(3):178194, March 2009. http://dx.doi.org/10.1016/j.parco.2008.12.006.

[9] Chi-Keung Luk, Sunpyo Hong, Hyesoon Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping In MICRO 2009, December, 2009.

[10] Mikhail Smelyanskiy, Karthikeyan Vaidyanathan, Jee Choi, Balint Joo, Jatin Chhugani, Michael A. Clark, Pradeep Dubey. High-performance lattice QCD for multi-core based parallel systems using a cache-friendly hybrid threaded-MPI approach International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2011.

[11] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. In Computer Physics Communications, Volume 181, Issue 9, September 2010, Pages 1517-1528

[12] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey. 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs. In Proc. ACM/IEEE Conf. Supercomputing (SC), New Orleans, LA, USA, November 2010.

[13] J. Board and K. Schulten. The fast multipole algorithm. Computing in Science and Engi- neering, 2(1):7679, January/February 2000.

[14] X. Feng, R. Ge, K. Cameron. Power and energy profiling of scientific applications on distributed systems. Proc. 19th Int'l Parallel & Distributed Processing Symp. (IPDPS 05), Apr. 2005.

[15] Zhenwei Cao, Layne T. Watson, Kirk W. Cameron, and Rong Ge. 2009. A power aware study for VTDIRECT95 using DVFS. In Proceedings of the 2009 Spring Simulation Multiconference (SpringSim '09). Society for Computer Simulation International, San Diego, CA, USA, , Article 107 , 6 pages.

[16] Hatem Ltaief, Piotr Luszczek, Jack Dongarra. Profiling High Performance Dense Linear Algebra Algorithms on Multicore Architectures for Power and Energy Efficiency In Proceedings International Conference on Energy-Aware High Performance Computing, September 07-09, 2011, Hamburg, Germany (University of Tennessee Technical Report ut-cs-11-674, LAWN 251)