Performance Modeling and Engineering Using Kerncraft

(Poster Summary)

Julian Hammer
Friedrich-Alexander-Universitt Erlangen-Nuremberg
Email: julian.hammer@fau.de

Georg Hager (advisor)
Friedrich-Alexander-Universitt Erlangen-Nuremberg
Email: georg.hager@fau.de

Gerhard Wellein (advisor)
Friedrich-Alexander-Universitt Erlangen-Nuremberg
Email: georg.hager@fau.de

***Abstract*—Achieving optimal program performance requires deep insight into the interaction of hardware and software. For software developers without an in-depth background in computer architecture, understanding and fully utilizing modern architectures is close to impossible. Therefore we must support them with easy to understand models and tools.**

We present two simple to apply approaches and a tool for regular stencil and streaming codes: Execution-Cache-Memory (ECM) [1] and layer condition (LC) [2] modeling with Kerncraft [3]. The ECM performance model gives precise overall performance predictions of computational kernels, and the LC analysis gives analytically derived hints for well-suited spatial blocking factors. By utilizing Kerncraft, we are able to take the pain out of the process using automation. We show its applicability to stencils and the insights gained through analysis, performance improvements achieved by following Kerncraft's suggestions derived from generalized LCs, and multi-core saturation point predictions gained through ECM modeling.

I. Introduction

We present performance modeling and engineering approaches, supported by our modeling toolkit Kerncraft, our Execution-Cache-Memory performance model and our generalized layer conditions model. These approaches allow us to precisely predict and validate performance and make design decisions, such as scaling point and block size selections.

II. Execution-Cache-Memory (ECM) Model

On the left we present the ECM model [1], [4], [5], where a unit of work corresponds to one cache line length (64 bytes). The hierarchy from execution to memory represents the time contributions that go into the model: $\{T_{OL}||T_{nOL}|T_{L1-L2}|T_{L2-L3}|T_{L3-MEM}\}$. Each contribution is given in cycles per work-unit. The model is applied by implementing assumptions about how the time contributions overlap. On Intel x86 processors, each stage adds an additional latency to the overall execution time. The in-core execution portion, overlaps with the rest of the data transfers, which are non-overlapping. The overall prediction is $\max(T_{OL}, T_{nOL} + T_{L1-L2} + T_{L2-L3} + T_{L3-MEM})$, as shown in the stacked bar graph at the bottom. For other architectures, such as Power8, data transfer times may overlap.

III. Kerncraft

On the right we present the inputs that Kerncraft requires (top), the automation Kerncraft applies for the modeling (center), and results given by the models and their validation(bottom).

A. User Input

Kerncraft requires the kernel code and a machine configuration. The kernel code is given in a standard-compliant C99 subset, which does neither allow pointer arithmetic nor branches within the loops. "Constants" (`N` and `M` in the example code) may be used for parameter studies.

On the right side, we see a (shortened) semi-automatically generated machine configuration file. It contains information about the architecture (CPU type, cores per socket, etc.), the description of the first-level cache (configuration and relation to other caches), and benchmark results from streaming kernels.

B. Automatic Modeling

The ECM and Roofline [6] models both require the same information about data transfers between memory hierarchy levels and in-core execution. The latter is gained using the Intel Architecture Core Analyzer (IACA), which provides out-of-order execution predictions. Loop assembly code is extracted from the compiler output, instrumented, compiled, and analyzed with IACA to get runtime predictions.

Data transfer information is gained through pycachesim [7], our cache simulator. By passing on memory access locations of accessed data, it predicts hits and misses. Note that the code is not executed but statically analyzed and accesse patterns are fed into the simulator.

Generalized layer conditions are the latest addition. They provide a fast analytic estimate of blocking factors for least-recently-used (LRU) cache hierarchies. The required cache size for minimum code balance (inverse computational intensity), and maximum block size at a given cache size can be estimated. We also provide an interactive online LC calculator for stencils [8].

Benchmark mode validates performance by executing an instrumented [9], [10] executable compiled from the kernel code.

C. Results and Validation

In this section we show example usage of Kerncraft along with expected outputs and graphs. First, a basic single-core ECM analysis is presented for the 3D long-range stencil [11] as seen in "User Input".

A parameter study is then done by scanning the inner two dimensions. The orange and blue stacked predictions make up the ECM prediction and the black line refers to the Roofline prediction. The symbols mark measurements of the same code. Below, we see which layer conditions are fulfilled, providing an explanation for the above changes.

When running in parallel the question arises if and when scalability is limited by a common bottleneck. The ECM model predicts the saturation point, which is at four cores in this example (`N=1000`).

In the last column we present the layer condition analysis. The predicted LCs match up with those seen in the second plot. Guided by the model predictions we scan the performance with a block size that fulfills the 3D LC in the L3 cache. As expected, the steep jumps in execution time are eliminated. The remaining small optimization space may be covered by an autotuning scan.

IV. Future Work

Kerncraft is currently limited by IACA, which is a closed-source component that only supports Intel processors, and for which further development has ceased. We will replace it with a simple but flexible out-of-order simulator of our own design.

By introducing the layer condition model into LLVM-Polly, developers will automatically profit form our research and tools. We will also utilize LLVM-Polly for exporting kernels form complex code, eliminating the need for manual extraction.

At the moment, only stencil and pure streaming loops can be modeled, but by encompassing queuing theory and statistical models, it may be possible to extend Kerncraft towards graph algorithms in the near future.

V. Conclusion

We have developed a comprehensive, automatic approach to performance modeling and engineering, with wide applicability. Streaming and stencil loop patterns may be analyzed, and spatial block sizes predicted.

In addition we provide a generalized model of layer conditions, which can also be evaluated by hand to gain a quick insight into expected performance.

Our software is freely available and open source, see https://github.com/rrze-hpc/.

References

[1] J. Treibig and G. Hager, *Introducing a Performance Model for Bandwidth-Limited Loop Kernels*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 615–624. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14390-8_64

[2] G. Rivera and C.-W. Tseng, "Tiling optimizations for 3d scientific computations," in *Supercomputing, ACM/IEEE 2000 Conference*, Nov 2000, pp. 32–32.

[3] J. Hammer, G. Hager, J. Eitzinger, and G. Wellein, "Automatic loop kernel analysis and performance modeling with kerncraft," in *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*, ser. PMBS '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:11. [Online]. Available: http://doi.acm.org/10.1145/2832087.2832092

[4] H. Stengel, J. Treibig, G. Hager, and G. Wellein, "Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 207–216.

[5] J. Hofmann, D. Fey, J. Eitzinger, G. Hager, and G. Wellein, "Analysis of intel's haswell microarchitecture using the ecm model and microbenchmarks," in *International Conference on Architecture of Computing Systems*. Springer, 2016, pp. 210–222.

[6] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[7] J. Hammer, "pycachesim – Python Cache Hierarchy Simulator." [Online]. Available: https://github.com/RRZE-HPC/pycachesim

[8] ——, "Layer Condition – Calculator." [Online]. Available: https://rrze-hpc.github.io/layer-condition/#calculator

[9] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 2010, pp. 207–216.

[10] T. Roehl and J. Eitzinger, "LIKWID – Lightweight Performance Tools." [Online]. Available: http://tiny.cc/LIKWID

[11] T. Malas, G. Hager, H. Ltaief, H. Stengel, G. Wellein, and D. Keyes, "Multicore-optimized wavefront diamond blocking for optimizing stencil updates," *SIAM Journal on Scientific Computing*, vol. 37, no. 4, pp. C439–C464, 2015. [Online]. Available: http://dx.doi.org/10.1137/140991133