Applying the Execution-Cache-Memory Model: Current State of Practice

Georg Hager
Erlangen Regional Computing Center
Erlangen, Germany
georg.hager@fau.de

Jan Eitzinger
Erlangen Regional Computing Center
Erlangen, Germany
jan.eitzinger@fau.de

Julian Hornich
Erlangen Regional Computing Center
Erlangen, Germany
julian.hornich@fau.de

Francesco Cremonesi
École Polytechnique Fédérale de Lausanne
Geneva, Switzerland
francesco.cremonesi@epfl.ch

Christie L. Alappat
Erlangen Regional Computing Center
Erlangen, Germany
christie.alappat@fau.de

Thomas Röhl
Erlangen Regional Computing Center
Erlangen, Germany
thomas.roehl@fau.de

Gerhard Wellein
Erlangen Regional Computing Center
Erlangen, Germany
gerhard.wellein@fau.de

ABSTRACT

The ECM (Execution-Cache-Memory) model is an analytic, resource-based performance model for steady-state loop code running on multicore processors. Starting from a machine model, which describes the interaction between the code and the hardware, and static code analysis it allows an accurate prediction of the runtime of sequential loop code. Together with a scaling assumption it also gives a performance scaling prediction. This poster summarizes the current state of practice in constructing and applying the ECM model, points out problems and open questions, and applies the model to three nontrivial use cases.

ACM Reference Format:
Georg Hager, Jan Eitzinger, Julian Hornich, Francesco Cremonesi, Christie L. Alappat, Thomas Röhl, and Gerhard Wellein. 2018. Applying the Execution-Cache-Memory Model: Current State of Practice. In *Proceedings of ACM (SC18).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Similar to the Roofline model [6], the ECM model [2, 5] can predict the runtime of loops on multicore CPUs. In contrast to Roofline, ECM works for sequential and parallel codes. It predicts relevant execution and data transfer bottlenecks and the performance saturation point with increasing number of cores. The premise of the ECM model is that the runtime of a program is determined by two basic resources: instruction execution and data transfer. Construction of the model for a sequential loop comprises four steps:

(1) Derive the traffic data volume V_i per loop iteration across all relevant data paths (i) in the CPU. Typically, a simplified model of the memory hierarchy is used, which includes only basic features (size, inclusive/exclusive).
(2) Calculate optimistic transfer times $T_i = V_i/b_i$ over all data paths. They are optimistic because latency effects are initially ignored and only asymptotic bandwidth b_i (measured or taken from hardware documentation) is considered. Latency penalties may be introduced as corrections.
(3) Calculate an "in-core" runtime prediction with all data coming from the innermost cache. Here one can either optimistically assume full throughput, i.e., all instructions are scheduled independently to the available execution units, or perform a critical path analysis for a worst-case prediction. The actual execution time typically lies between these limits. Part of the in-core execution time counts as "overlapping" (T_{OL}) while the rest is "non-overlapping" (T_{nOL}). How overlapping and non-overlapping parts enter the final prediction depends on the machine model (see below).
(4) All time contributions are put together to build a prediction T_{ECM} using a machine model. The machine model makes assumptions about which of the components overlap in time and which do not.

If the code is run in parallel, the model assumes that all time contributions that come from scalable resources (pipelines, core-private caches, shared but scalable caches) are divided by the number of cores, while data transfer times over a bottleneck (such as the main memory interface) stay constant. The runtime with n active cores in presence of a memory bottleneck is thus

$$T_{\mathrm{Mem}}^{\mathrm{ECM}}(n) = \max\left(\frac{T_{\mathrm{Mem}}^{\mathrm{ECM}}}{n}, T_{\mathrm{L3Mem}}\right), \tag{1}$$

where T_{L3Mem} is the data transfer time across the bottleneck. The number of cores needed to reach saturation is $n_s = T_{\mathrm{Mem}}^{\mathrm{ECM}}/T_{\mathrm{L3Mem}}$.

SC18, November 2018, Dallas, Texas USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The model can be validated by measuring the actual runtime of the code and the actual data transfer volumes using hardware performance counters. One particularly powerful way of validation is to observe data transfers change with some parameter, such as the problem size.

2 POSTER COMPONENTS

2.1 Model construction and notation

The ECM model makes runtime predictions. Model construction requires data transfer times through the memory hierarchy, an in-core model, and overlap assumptions (all covered on the poster). The notation we have introduced is convenient for reasoning about the model and its components; e.g., {8 ‖ 4 | 7 | 10 | 17} cy denotes that the designated number of iterations needs 8 cy for overlapping in-core execution, 4 cy for non-overlapping in-core execution (such as loads), and 7, 10, and 17 cy, respectively, for transferring the required data through the data paths between adjacent memory levels. The machine model puts these numbers together to form a prediction. E.g., {8 ⌉ 11 ⌉ 21 ⌉ 38} cy would denote runtime predictions for data residing in L1, L2, L3, and memory on a processor that shows a fully non-overlapping characteristic.

2.2 Case studies

A frequent criticism towards analytic performance modeling is that it can only handle "simple" cases, while real applications are allegedly too complicated for this approach. We chose the case studies specifically to counter this attitude. Two of them have rather complex loop bodies, while the third is a multi-loop algorithm. None of these results have been published before.

2.2.1 Complex arithmetic stencil. This stencil update loop nest is taken from a Time Harmonic Inverse Iteration Method (THIIM) code using finite-difference frequency domain (FDFD) discretization for simulating thin-film solar cells [4]. All variables are of double precision complex type, which makes SIMD vectorization by compiler a challenge. Hence, AVX2 vectorization was done manually via C intrinsics. This is a stencil algorithm with a 3D layer condition in z direction. The condition is broken if two successive layers of the grid do not fit into the cache any more; on the Intel Xeon Haswell CPU used here, this happens at the L3 cache beyond a problem size of about 350^3 lattice sites. The results show that the model can predict the runtime with an error of 5% or less on either side of the layer condition. The measured data traffic between adjacent cache levels is also very well in line with the model. The main insight from the model is that spatial blocking has only a limited benefit, and that temporal blocking should be employed for better performance and scalability.

2.2.2 Kernels from brain cell simulations. The two loop kernels are taken from mini-apps developed in the Blue Brain Project [1]. Case 1 (synaptic current) has mainly streaming data accesses, an exponential function call, and some divides. The two indirect accesses are actually sequential, with consecutive blocks of identical indices; they can be accommodated by an adjusted data volume. Due to the function call and the many concurrent data streams, some integer register spill occurs and there is a rather strong intra-iteration dependency chain. On the Ivy Bridge CPU, this leads to the critical path latency being a very good estimate of the runtime per iteration. On the Haswell CPU, consecutive iterations can overlap better and the data transfers become the bottleneck. Although the two architectures expose different behavior, the performance is expected to saturate with all cores of the ccNUMA domain.

Case 2 (sodium ion channel) has different characteristics. It comprises multiple `exp()` calls and divides, but no strong dependency chain and a moderate data volume to memory. Even when assuming full throughput of all instructions (and the exponential function, whose throughput was determined experimentally), the overlapping part of the in-core runtime dominates by far. In such a case the runtime is largely set by the out-of-order hardware being able to overlap successive iterations. On Ivy Bridge, the measured runtime (190 cy per iteration) is 36% longer than the optimistic estimate, which indicates that this overlap is less than perfect.

2.2.3 A Conjugate-Gradient solver. We solve a Poisson problem on a quadratic plate with Dirichlet boundary conditions and a source term. Due to the finite-difference discretization the coefficient matrix does not have to be stored. The ECM model for each of the six loops is set up for a problem size of 40000×1000 grid points. All loops are dominated by data transfers. Their sum (152 cy for 8 iterations) is the runtime prediction for the entire solver, which is very close to the measurement of 159 cy on a Haswell core. The saturated runtime (with seven cores in a ccNUMA domain) is just the sum of the memory-L3 transfer times in this case; it is within 0.5% of the measurement. The scaling graph shows performance vs. number of cores together with the model predictions.

2.3 Additional information

2.3.1 Overlap assumptions. The current working hypotheses for overlap assumptions on different platforms are as follows. Intel server CPUs up to Broadwell EP show no overlapping between any data transfer contributions (including loads from the L1 cache), while all the remaining code execution fully overlaps with data transfers. Experiments indicate that the non-overlapping feature is retained on Skylake SP, but the data flow analysis must be modified because of the L3 victim cache. On AMD Epyc (Zen architecture), initial findings suggest that the memory hierarchy is fully overlapping. Finally, on the IBM Power8 the best assumption is full overlap at the L1 cache and non-overlap of all other data transfers.

2.3.2 Applicability of the model. The ECM model is built upon the *steady-state assumption*, which says that loop startup and winddown effects, and all latency contributions connected with them, are ignored. Hence, short loops cannot be accurately modeled. On some architectures, latency penalties can be introduced to reduce prediction error. Non-streaming data access can be accommodated via a worst/best-case analysis.

2.3.3 Improved saturation prediction. The saturation assumption of the plain ECM model is too optimistic near the saturation point. This can be corrected by a dynamic latency penalty that depends on the utilization of the memory interface [3]. Although this correction makes the model extremely accurate, it contains a fit parameter that is not code-independent. This problem needs to be investigated.

REFERENCES

[1] Timothée Ewart, Judit Planas, Francesco Cremonesi, Kai Langen, Felix Schürmann, and Fabien Delalondre. 2017. Neuromapp: A Mini-application Framework to Improve Neural Simulators. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes (Eds.). Springer International Publishing, Cham, 181–198.

[2] Georg Hager, Jan Treibig, Johannes Habich, and Gerhard Wellein. 2013. Exploring performance and power properties of modern multicore chips via simple machine models. *Concurrency Computat.: Pract. Exper.* (2013). DOI: 10.1002/cpe.3180.

[3] Johannes Hofmann, Georg Hager, and Dietmar Fey. 2018. On the Accuracy and Usefulness of Analytic Energy Models for Contemporary Multicore Processors. In *High Performance Computing*, Rio Yokota, Michèle Weiland, David Keyes, and Carsten Trinitis (Eds.). Springer International Publishing, Cham, 22–43.

[4] T. M. Malas, J. Hornich, G. Hager, H. Ltaief, C. Pflaum, and D. E. Keyes. 2016. Optimization of an Electromagnetics Code with Multicore Wavefront Diamond Blocking and Multi-dimensional Intra-Tile Parallelization. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 142–151. https://doi.org/10.1109/IPDPS.2016.87

[5] Holger Stengel, Jan Treibig, Georg Hager, and Gerhard Wellein. 2015. Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model. In *Proceedings of the 29th ACM International Conference on Supercomputing (ICS '15)*. ACM, New York, NY, USA, 10. https://doi.org/10.1145/2751205.2751240

[6] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76. https://doi.org/10.1145/1498765.1498785