Performance Engineering: From Numbers to Insight

Georg Hager

Erlangen Regional Computing Center (RRZE)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr. 1, 91058 Erlangen, Germany
georg.hager@fau.de

The ultimate purpose of running simulation tasks on high performance computers is to solve numerical problems. The *performance* of an algorithm, or rather an implementation, is significant in several respects: Either a given problem should be solved in the least possible amount of time or a larger problem should be solved in an "acceptable" time; in both cases, the used resources must be utilized as efficiently as possible so that overall throughput and return on investment are maximized for all users of a system.

Reaching the latter goal implies that the user has some concept of what the "maximum possible performance" of their code is. More often than not, application programmers employ code optimizations without a clear concept of the expected gain. As a result, vast computational resources are wasted. Performance analysis, modeling, and engineering approaches are required to remedy this situation.

An analysis of performance properties naturally starts at the core and chip level, since this is where the actual numerical "work" is done. In-core execution delays, bottlenecks on the chip level, and communication overhead are typical factors that can lead to a deviation from "ideal" performance numbers. A *performance model*, which is typically constructed for every hot spot or loop in a program, predicts the maximum possible performance. The purpose of such models is not so much to predict but to *describe* performance behavior, often with respect to parameters such as problem size or machine properties. When the measured performance numbers deviate from the model, an opportunity arises to learn more about the hardware, the software, or the interaction of both.

The construction of a useful model could be a non-trivial task, however. One of the crucial inputs is a static analysis of the algorithm and/or the code of a hot spot, e.g., in terms of work (flops) performed, dependencies, data transfers, inter-process

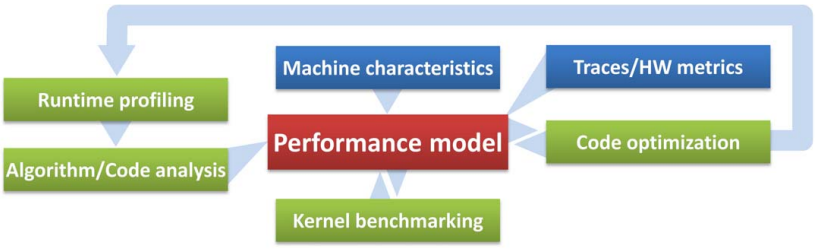

Fig. 1. The structured performance engineering process.

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 393–394, 2013.

communication, etc. A comparison with documented machine characteristics (maximum instruction throughput/latency, SIMD register width, theoretical latencies and bandwidths of data paths, etc.) can already lead to a first performance estimate. Sometimes features are undocumented, or real code execution cannot saturate a resource. In such cases, *microbenchmarks* can help establish a practical limit; a prominent example is the maximum main memory bandwidth of a chip, which can often not meet the theoretical limit given by the hardware, and should thus be measured by suitable benchmarks such as STREAM. The roofline model [1] and the ECM model [2,3] implement this modeling approach successfully for modern multi- and manycore processors. If power consumption and "energy to solution" is taken into account, a useful model will also estimate the sweet spot of minimum energy to solution with respect to clock frequency, resources used (cores, nodes), and single-thread performance [3].

When it comes to comparing the model with measurements, the pure performance number is just one of many possible data points. *Hardware performance metrics* or *event traces* can be used to further validate the model. For instance, the aggregated count of cache lines transferred between adjacent memory hierarchy levels can be checked against the model prediction. Finally, the insight that has been generated by the performance model and its validation often leads to a clear view on possible optimizations and their potential benefit. Since a code change might incur a substantial modification of the whole application's timing behavior, a new runtime profile may be in order and the whole workflow starts from the beginning.

The constructed performance model is thus embedded in a *structured performance engineering process* (see Fig. 1). Neither is it possible to automate this completely, nor can it be put into a single "catch-all" formula. One should also note that, although hardware metrics are a central component of performance analysis, they can only provide part of the necessary information. Manual code inspection and the subsequent construction of the performance model is perhaps the most time-consuming part of the process, which also requires substantial experience in some cases.

References

1. Williams, S.W., Waterman, A., Patterson, D.A.: Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Tech. Rep. UCB/EECS-2008-134, EECS Department, University of California, Berkeley (October 2008), http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html
2. Treibig, J., Hager, G.: Introducing a Performance Model for Bandwidth-Limited Loop Kernels. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part I. LNCS, vol. 6067, pp. 615–624. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14390-8_64
3. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multicore chips via simple machine models (submitted), http://arxiv.org/abs/1208.2908