A Simple Methodology for Evolving Algorithms and Maximizing Hardware Performance

Hans Johansen (hjohansen@lbl.gov), *Applied Math & Computational Research Division, Lawrence Berkeley National Laboratory*

Introduction

Trends in ever-more heterogenous hardware platforms and systems have led to a gap in computational science software skillsets: few software developers can do optimization for specific hardware, AND understand which algorithms maximize performance.

We present a methodology for accomplishing this in a straight-forward way that can be applied to both computer and computational science contexts, from beginner to advanced levels, with good extension points for keeping most high performance computing (HPC) learners engaged.

The approach uses the "Roofline methodology" [W2009], which benchmarks hardware architectures in terms of *arithmetic intensity* (AI) of FLOPs to bytes of data transferred across memory cache hierarchies or network connections (Figure 1). This enables students to evaluate performance against baseline hardware benchmarks.

Family of Case Studies: Tridiagonal Matrices

Linear algebra concepts are good for demonstrating computational performance and algorithmic variation, with a variety of matrix size, operations, and parallel performance constraints:

- **Low AI**: Explore batch sparse matrix-vector multiplication (SPMv). Vectorization, bandwidth, and cache effects vs. "stream triad."
- **High AI**: For block-tridiagonal matrices, batch cyclic reduction increases FLOPs vs. matrix memory bandwidth. Using iterative approaches can exploit GPU matrix-multiply-accumulate (MMA).
- **Network AI**: For distributed batch tridiagonal solves, parallel algorithms contrast with network bandwidth/latency benchmarks.

Applying a methodology in phases combines performance models with students profiling and refactoring across iterations.

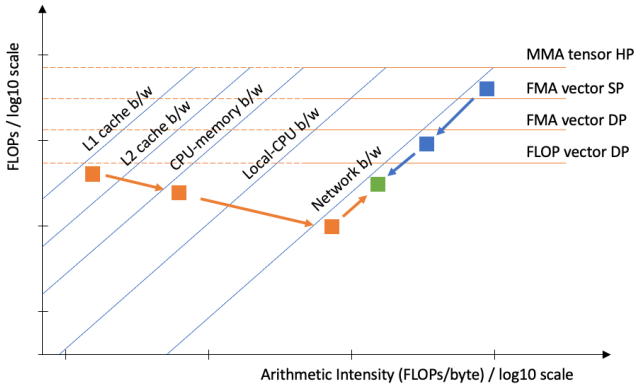

***Figure 1**: Example optimization path. Beginning with three kernels, match those to benchmarks: a FLOP-bound one (blue), a memory bandwidth one (orange), and a network bandwidth one (green). They apply the methodology to ``walk the roofline'' for each kernel, to evolve code to an optimized algorithm implementation.*

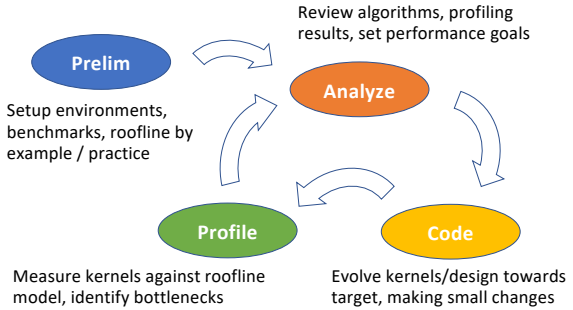

***Figure 2**: Methodology for taking students through an optimization effort iteratively from benchmark to optimized, correct code. Iterations are minor tweaks or full rewrites, with models as guide.*

Methodology & Workflow

In practice, this is treated as a phased, iterative approach (Figure 2) to performance engineering/optimization [S2001]:

- (Preliminary) setup, background, and roofline examples.
- **Analyze**: Review algorithms using visual models [L1997] and profiling vs. goals. Set new performance objectives.
- **Code**: Evolve kernels from baseline to optimized code in iterations. Use transformations with regression testing.
- **Profile**: Repeat roofline performance compared to theoretical performance models, discuss bottlenecks.

Iterations can be hackathons or weekly team assignments, emphasizing repeatability and discussing *tradeoffs*; not every algorithm is appropriate for every architecture.

Poorly performing baselines (counter-examples) are useful for demonstrating paths towards roofline optimization.

Conclusions & Future Work

This approach has been used informally for GPU hackathons and for teaching the roofline model to graduate students.

- Skills needed for refactoring [F1999] with performance *and* correctness are needed in computational science.
- "Fast but wrong doesn't count" is common lore, but in reality starting with a "correct" code that is very far from optimal makes performance optimization too difficult.
- Failure points have been the "grind" of consistent and repeatable collection of performance data ("hygiene"); solutions include automation and team support.
- Iterative evaluation and improvement of codes allows students to stay "close" to roofline and avoid frustration.

[W2009] S. Williams, et al, "Roofline: an insightful visual performance model for multicore architectures", https://doi.org/10.1145/1498765.1498785

[S2001] C. U. Smith and L. G. Williams, "Performance Solutions", ISBN-13: 978-0131489066.

[L1997] C. Larman, "Applying UML and patterns", ISBN-13: 978-0137488803.

[F1999] M. Fowler, et al, "Refactoring", ISBN-13: 978-0201485677.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration.