Pieter Ghysels

Increasing the Arithmetic Intensity of Multigrid on Many-Core Chips

Middelheimcampus M G 320b
Middelheimlaan 1
2020 Antwerpen
pieter.ghysels@ua.ac.be
Wim I. Vanroose

The basic building blocks of a classic multigrid algorithm, which are essentially stencil computations, all have a low ratio of executed floating point operations per byte fetched from memory. This important ratio can be identified as the arithmetic intensity. According to the roofline model [1], applications with a low arithmetic intensity are typically bounded by memory traffic, whereas applications with higher arithmetic intensity are bounded by the floating point unit throughput. It is well known that many bandwidth limited algorithms typically perform at a small percentage of the theoretical peak performance of the underlying hardware.

In this work we look at changes to both the multigrid algorithm and its implementation in order to increase the arithmetic intensity.

We present a theoretical model for the cost of a multigrid cycle (V-cycle or full multigrid) that includes the cost of communication, based on the roofline model. Our multigrid model predicts performance benefits for code optimization strategies such as: fusion of operations, tiling, parallelization, vectorization (SIMDization) and cache prefetching. The multigrid convergence rate is taken into account in the model for different smoothers, different coarsening and interpolation strategies, different cycles and different coarsest grid solvers.

Our theoretical model is validated using numerical experiments on modern multi and many-core hardware. Instead of resorting to manual and tedious low level code optimizations, we piggyback on recent compiler improvements, such as stencil compilers, automatic parallelizers and automatic loop transformers for cache locality optimization [2,3]. An important observation is that the underlying hardware can only be used efficiently when the number of consecutive smoothing steps is large enough. In that case, the arithmetic intensity can be increased sufficiently such that SIMD vector units can be kept busy and the extra smoothing steps pay off in the overall performance of the algorithm. It is thus important to look for algorithmic changes that can put these extra smoothing steps to effective use.

[1] Williams S, Waterman A and Patterson DA. Roofline: an insightful Visual

Performance model for multicore architectures. Communications of the ACM 2009; 52(4):6576

[2] Christen M, Schenk O, and Burkhart H. Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, pages 676687. IEEE, 2011.

[3] Bondhugula U, Baskaran M, Krishnamoorthy S, Ramanujam J, Rountev A, and Sadayappan P. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. International Conference on Compiler Construction (ETAPS CC), Apr 2008, Budapest, Hungary.