Data Locality Optimizations to Improve the Efficiency of Multigrid Methods

Christian Weiß, Hermann Hellwagner
Institut für Informatik, Technische Universität München, Germany
Linda Stals
School of Mathematical Sciences, University of Bath, United Kingdom
Ulrich Rüde
Institut für Mathematik, Technische Universität Augsburg, Germany

Abstract

Multigrid methods have been proven to be among the most attractive algorithms for the solution of PDEs. The program *mgd9v* for example can compute a solution of Poisson's equation with one million unknowns in about 90 seconds on a current RISC computer[1]. This compares favorably to other methods like the conjugate gradient method with hierarchical preconditioners. Presently, however, these methods can only exploit less than 10 per cent of the performance of current RISC computers because they disregard the architectural properties of these machines.

Current superscalar microprocessors operate with more than 500 MHz clock frequency and are able to process at least two floating point operations per cycle. However, the current main memory technology does not provide the data needed fast enough to keep the CPU busy. To minimize idle times of the CPU, a small expensive high speed memory - called *cache* - is used. The cache contains copies of main memory blocks to speed up accesses to frequently used data. Because of the limited size, caches can only hold copies of recently used data. To exploit caches, the software must be aware of the caches and reuse data in the cache before it is replaced. Unfortunately, all conventional multigrid codes are not cache-aware and hence fail to reach the peak performance on cache based machines by a large margin.

Our studies with linear PDEs with constant coefficients show that it is possible to speed up the execution of our multigrid method by a large factor and hence solve a Poisson's equation with one million unknowns in less than three seconds. The optimized reuse of data in the cache allows us to exploit 30 per cent of the peak performance of the CPU, in contrast to *mgd9v* which achieves less than 5 per cent on the same machine. To achieve this, we used several techniques like *loop unrolling* and *loop fusion* to better exploit the memory hierarchy and the superscalar CPU and studied the effects of these techniques on the runtime performance in detail. We also studied several tools which guide the optimizations and help to restructure the code.

In the presentation we will also outline the generalization of our techniques to more complex problems and algorithms.

[1]We used a SGI machine based on a 75 MHz clocked R8000 CPU for the computations.