High Performance Lattice Boltzmann Applications on Supercomputers

Thomas Pohl[1], Markus Kowarschik[1], Jens Wilke, Klaus Iglberger, and Ulrich Rüde[1]
[1] Lehrstuhl für Systemsimulation (Informatik 10)
University of Erlangen-Nuremberg
thomas.pohl@cs.fau.de

The Lattice Boltzmann Method (LBM) is rapidly gaining popularity among scientists and engineers because of its comparable simple implementation, its robustness, its wide field of applications, and its flexibility when combined with other models.

For moderate domain sizes up to a few million lattice sites most users of LBM are satisfied with the performance of their simulations. When dealing with larger domain sizes with up to one billion lattice sites, however, deficiencies in the LBM kernel and the chosen data layout can degrade the application's performance to an unusable level. Two potential bottlenecks can be identified: the CPU's floating point performance and the memory interface. While the maximum floating point performance can only be improved by migrating to a more powerful CPU, adapting the data access patterns in the code to the underlying architecture has the potential to significantly increase the performance especially for large-scale applications.

One has to distinguish between two common application scenarios. The first scenario is an LBM code running on a single-CPU commodity PC. In this case programming techniques such as loop unrolling and loop blocking can be applied. These techniques aim at improving the spatial and temporal locality of the implementation to exploit the advantages of today's large CPU caches, which can deliver the requested data much faster than main memory. Although these techniques are well-known and have already been successfully used in other areas of scientific computing, their application to LBM codes is still uncommon. By combining these cache-based optimizations with a sophisticated data layout, which reduces the required memory by almost 50%, we were able to improve the performance on certain architectures up to a factor of three.

The second typical scenario is the application of LBM on supercomputers. Many current supercomputers are either based on vector CPUs or at least inherit the vector processing paradigms, which rely on simple and predictable data flows instead of a hierarchy of cache levels. Therefore, completely different optimization strategies are necessary. An additional difficulty arises for distributed memory architectures since the computational domain has to be decomposed and data at the interfaces of adjacent domains must be interchanged after each time step. We investigate different techniques to preserve single-CPU performance and present scale-up results for up to 512 CPUs.

A comparison of the performance on different architectures using various optimization techniques concludes the presentation.