Optimizing Performance of the Lattice Boltzmann Method for Complex Structures on Cache-based Architectures

Stefan Donath[1], Thomas Zeiser, Georg Hager, Johannes Habich, Gerhard Wellein
Regionales Rechenzentrum Erlangen (RRZE), Martensstr. 1, 91058 Erlangen, Germany

Abstract

Delivering high sustained performance for memory-intensive applications in computational fluid dynamics on cache-based microprocessors is a long-standing challenge. In particular, non regular data access patterns, as arising from porous media flow within lattice Boltzmann codes, can lead to poor performance. To address this problem, we combine a 1-D list data representation with advanced code optimizations and are able to achieve a high performance level, which is mostly independent of geometry and obstacle/fluid ratio. The idea of traversing memory using space-filling curves is tested as well, but our results indicate that this approach alone can not compete with standard techniques, i.e. blocking and data layout optimization, which become architecture dependent if an indirect memory addressing scheme is being used.

1 Introduction

For high end applications in numerical simulation vector computers have long been the architecture of choice. In the past decade, owing to rapid advances in technology, commodity "off-the-shelf" (COTS) cache-based microprocessors have become an interesting alternative due to their unmatched price/peak performance ratio. These processors, however, show a large and rapidly growing gap between peak performance and memory bandwidth. To reduce this bottleneck, modern COTS architectures use sophisticated hierarchies of small but fast caches. Thus, performance optimization should aim both at the reduction of data transfer from/to main memory and at the increase of spatial and temporal locality, i.e. the increase the reuse of cached data.
A recent method for computational fluid dynamics (CFD) is the lattice Boltzmann method (LBM) [1, 2, 3], a promising alternative for numerical simulation of time-dependent incompressible flows. Compared to other methods the advantages of the LBM are computational speed, high accuracy and the capability to handle the flow through and around complex geometries efficiently. The algorithm allows for an easy implementation and thus comparison of different data layouts as well as optimization approaches as shown previously for simple geometries [4, 5, 6] is feasible.
Owing to the scientific and commercial interest, some of these aspects are now discussed with regard to complex geometries, i.e. domains with large solid parts and many fluid-solid interfaces, e.g. porous media. Due to the overhead of handling boundary conditions near

[1]E-mail address: Stefan.Donath@rrze.uni-erlangen.de

obstacles and the non consecutive memory accesses, performance per fluid cell is much lower than for simple systems, e.g. empty channel. Thus, it is worth to investigate sparse memory layouts that both save memory and provide good cache utilization.
We first give a brief introduction to the lattice Boltzmann method in Section 2. After describing the implementation of an LBM solver which processes the domain using a one dimensional data structure (Section 3) we study several possibilities of traversing the domain (Section 4) and discuss the results for different platforms in Section 5.

2 Basics of the Lattice Boltzmann Method

A widely used class of lattice Boltzmann models is based on the BGK approximation of the collision process [1, 2, 3] and the evolution equation

$$f_i(\vec{x} + \vec{e}_i \delta t,\, t + \delta t) = f_i(\vec{x},\, t) - \frac{1}{\tau}\left[f_i(\vec{x},\, t) - f_i^{eq}(\rho, \vec{u})\right] \qquad i = 1 \dots N. \tag{1}$$

Here, f_i is a particle distribution function which represents the fraction of particles located in a cell at position $\vec{x}$ at timestep t, moving with the microscopic velocity $\vec{e}_i$. The relaxation time τ determines the rate of approach to local equilibrium and is related to the viscosity. The equilibrium state f_i^{eq} depends only on the macroscopic values of the fluid density ρ and the flow velocity $\vec{u}$. Both can be easily obtained as the first moments of the particle distribution function.
A typical discretization scheme in 3-D is the D3Q19 model [7] which uses $N = 19$ discrete velocities $\vec{e}_i$. It results in a computational domain with equidistant Cartesian cells (voxels) as shown in Figure 1.

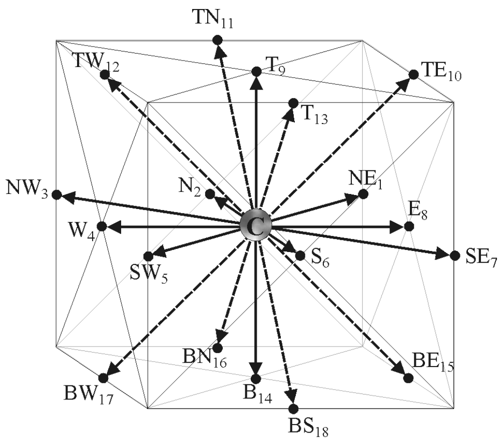

Figure 1: Lattice site and its discrete velocity vectors for the D3Q19 LBM model.

Each timestep $(t \rightarrow t + \delta t)$ of the LBM consists of the following steps which are repeated for all cells:

- Calculation of the local macroscopic flow quantities ρ and $\vec{u}$ from the distribution functions, $\rho = \sum_{i=1}^{N} f_i$ and $\vec{u} = \frac{1}{\rho} \sum_{i=1}^{N} f_i \vec{e}_i$.
- Calculation of the equilibrium distribution f_i^{eq} from the macroscopic flow quantities (see [7] for the equation and parameters) and execution of the "collision" (relaxation) process, $f_i^{*}(\vec{x}, t^{*}) = f_i(\vec{x}, t) - \frac{1}{\tau} [f_i(\vec{x}, t) - f_i^{eq}(\rho, \vec{u})]$, where the superscript * denotes the post-collision state.
- "Propagation" of the $i = 1 \ldots N$ post-collision states $f_i^{*}(\vec{x}, t^{*})$ to the appropriate neighboring cells according to the direction of $\vec{e}_i$, resulting in $f_i(\vec{x} + \vec{e}_i \delta t, t + \delta t)$, i.e. the values of the next timestep.

The first two steps are computationally intensive but involve only values of the local node while the third step is just a direction-dependent uniform shift of data in memory. At fluid-solid interfaces a fourth step, the so called "bounce back" rule [1, 2, 3], is performed which "reflects" the distribution functions and thus leads to an approximate no-slip boundary condition at walls.

3 Implementation of the LBM for complex structures

An efficient standard implementation for simple problems [4] uses a fully allocated grid for the domain, including memory locations representing obstacles. It performs the collision and propagation step in one loop, while handling of boundary conditions (bounce back) is done by an extra routine that traverses all obstacle surfaces. With a high number of obstacles inside the domain which is typical for porous media (see Figure 2) this approach wastes memory and requires a significant overhead due to handling the bounce back rule.

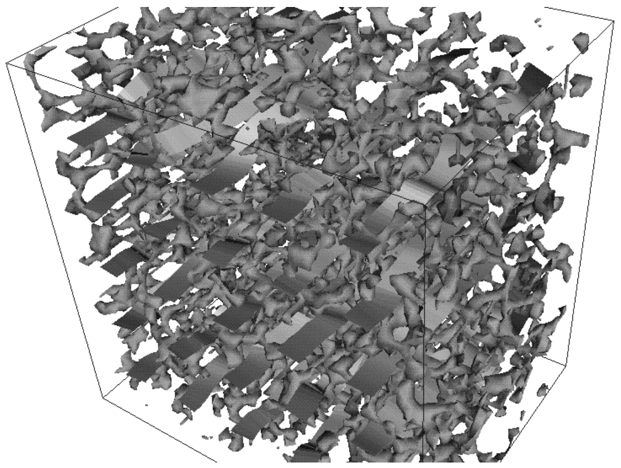

Figure 2: Complex structure of a SiC foam with large number of interior obstacles.

To evade these problems, the implementation discussed in this paper uses a 1-D representation of the domain, skipping the obstacles by storing only data of fluid cells and their

Table 1: Part of the Table for Hilbert Curves Construction in 3D

Current Level	Next Level							
bne	enb, b	ben, n	ben, f	fse, e	fse, b	bws, s	bws, f	wnf
fws	swf, f	fsw, w	fsw, b	bes, s	bes, f	fne, e	fne, b	nwb
nwf	fwn, n	nfw, w	nfw, s	sef, f	sef, n	nbe, e	nbe, s	bws
enb	bne, e	ebn, n	ebn, w	wsb, b	wsb, e	efs, s	efs, w	fnw

connectivity (basis code by courtesy of [8]). A preprocessor generates metadata information which describes the data of the distribution functions f_i as well as the grid connections and cell positions. Like our previous implementation [4] the LBM algorithm uses the "push" scheme, meaning that the distribution functions of the *current* cell are read and after relaxation the updated values are written to the *adjacent* cells. In the case of the 1-D representation of the domain, the solver traverses the array during collision cell by cell. The updated values have to be propagated to the appropriate locations of the second part of the array (used due to data dependencies). The connectivity is stored in an extra buffer. Therefore, the bounce back rule can easily be defined implicitly by specifying either the adjacent or the same cell as target.
Within this approach, the number of memory accesses solely depends on the number of fluid cells in the domain. However, performance depends on the sequence of memory accesses, which is influenced by the number and location of obstacles. Consequently, different approaches for traversing memory were tested.

4 Traversing Schemes for Locality Enhancement

To optimize the run-time performance we compared two different approaches: (1) Changing the scheme of traversing memory to find an enhanced path through the obstacles, and (2) adapting the memory layout like in our previous work [4].
To ease the implementation of different memory traversing schemes the preprocessor routine was kept as modularized as possible. Thus, only one function had to be exchanged. This function influences the location of the lattice site's representation in memory. First, we introduced a standard blocking scheme which subdivides the domain into blocks that fit into cache. The second idea is based on a mesh reordering using space-filling curves (SFCs) on discrete grids. Hereby, the substitution of the primitive curves to next higher levels is only performed until the resolution fits to the grid size. Due to the recursive construction by dividing the unit interval by a factor of 2 for the Hilbert curve and 3 for the Peano curve (cf. [9]), only 3-D grid sizes of 2^{3n} and 3^{3n}, respectively, are supported at this time. For construction of the SFCs we use the same table-driven approach as described in [10]: In the left part of Figure 3 four of the overall 48 possible primitive curves for the Hilbert curve are shown. Table 1 lists their substitutions for constructing the next level. We are using the same naming convention as in [10], where the first and second letters represent the first and second move, respectively, and the third letter represents the fourth move, which is the

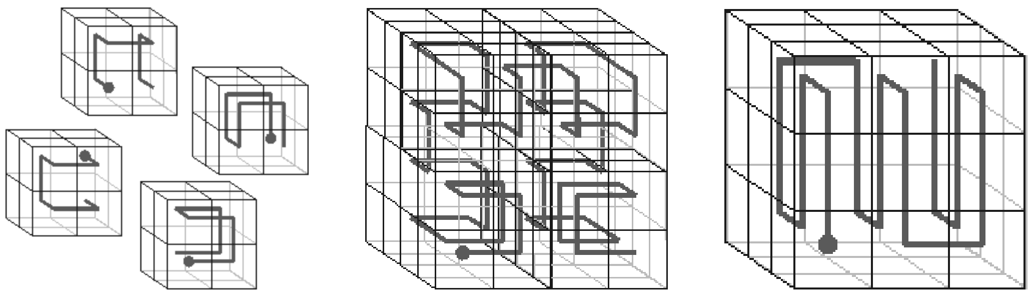

Figure 3: Hilbert primitive curves (left), Hilbert “bne” curve at level 2 (middle), Peano primitive curve (right).

move between the two planes. “e”, “w”, “n”, “s”, “f”, “b” stand for *east*, *west*, *north*, *south*, *front* and *back*, respectively. The middle part of Figure 3 shows the first recursion of the “bne” curve. Since there are always several possible combinations for replacing a primitive curve, we paid attention to use constructions which assure that cells already touched will be visited again as soon as possible, in order to improve data locality. When using the Peano curve in 3-D these problems do not occur. Here, we use only eight different primitive curves. The well-defined production rules use 27 primitive curves for replacing one of the previous level.

For LBM, data in memory can be represented in different ways. Most common is a multi-dimensional array. For this case, Wellein et al. [4] pointed out that the structure-of-arrays layout, the so-called *propagation optimized layout*, is to be preferred because it results in higher performance than the array-of-structures layout (so-called *collision optimized layout*) on most architectures. However, for the 1-D array implementation with indirect addressing additional optimization is in order: A block preload technique (described in detail for the vector triad in [11]) was applied that effectively separates loading data to cache from arithmetic operations. To avoid register spill and congesting the write combine buffers of IA32 processors when storing the 19 updated values, relaxation was split into several loops (3 to 5). While the first loop reads the current cell’s values and calculates the macroscopic quantities like density and velocity, the following loops calculate the relaxation and write the values to the adjacent cells for only four directions, for example.

5 Comparison of Performance Results

The benchmarks were performed on three cache based architectures: Intel Itanium 2 (IA64), Intel Nocona/Irwindale (IA32 compatible) and AMD Opteron (IA32 compatible). On all systems, blocking and SFCs improve the spatial locality for both memory layouts. We benchmarked systems of different sizes for both empty channels and porous media. Due to the sensibility of the propagation optimized data layout to cache thrashing at sizes of powers of two [4], the empty channels contain four obstacles to shorten the array length and thus

Table 2: Comparison of Different Optimization Techniques

Cache	Arch.	Best	Prop. Opt.	Coll. Opt.	SFC influence
$\leq$ 1 MB	IA32	Prop. Opt.	MBP Loop Splitting	Hilbert	improves Coll. Opt. only
$>$ 1 MB	IA32	Coll. Opt.	MBP Loop Splitting	Blocking	improves Prop. Opt. only
$>$ 1 MB	IA64	Coll. Opt.	Hilbert, w/o MBP, w/o Loop Splitting	Blocking	improves Prop. Opt. only

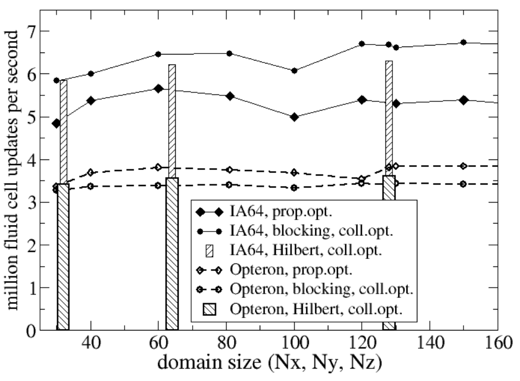

Figure 4: Comparison of implementations on IA64 and Opteron using MC geometry

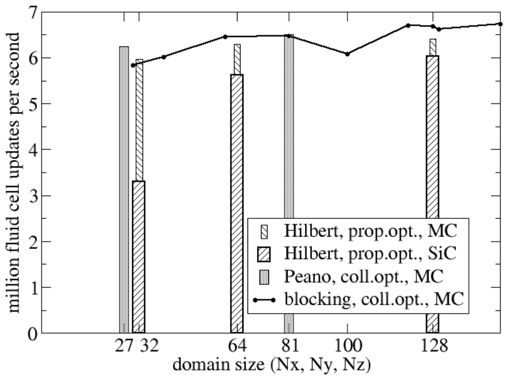

Figure 5: Comparison of SFC-schemes with best implementation on IA64

the stride of successive accesses. The porous media test case contains a huge number of obstacles such that compression ratios of 30 to 60 percent (depending on system size) are achieved.

Figure 4 and Table 2 clearly show, that the results depend on cache size and architecture: For small caches (AMD Opteron and Intel Nocona with 1 MB L2 cache each) the propagation optimized layout combined with manual block preload (MBP) and split loops outperforms all other implementations. While SFCs cannot improve the propagation optimized layout, traversion by a Hilbert curve accelerates the collision optimized layout stronger than the implicit blocking technique.

Using larger caches (tested on Itanium2, 1.5 MB L3 cache, and Irwindale, 2 MB L2 cache), the cache line saving property of the propagation optimized layout does not countervail against the blocked collision optimized layout any more. Due to the eliminated loop overhead of implicit data blocking, the collision optimized layout now yields the best results. On IA64 the MBP technique must not be used as it prevents the compiler from more effective prefetching.

Figure 5 points out that SFCs generally yield similar performance as blocking techniques with practically no difference between Hilbert and Peano curves. With the collision optimized layout, performance is very robust with respect to geometry, while the propagation

optimized layout shows instabilities depending on the number and surface of obstacles. (In Figure 5 propagation optimized layout shows worse performance for the fine-grained SiC-foam with many small obstacles (see Figure 2) than for coarse-grained foams with few and large structures (MC).)
In general, the indirect addressing of the 1-D array yields stable performance behavior that is independent of obstacle geometries for all described implementations. Only the propagation optimized layout shows slight weaknesses in the case of few adjacent fluid cells. Such geometries can lead to an inefficient cache line use and nullify the effect of automatic hardware prefetchers (or compiler-generated prefetching on IA64) for this layout.

6 Conclusions and Outlook

We demonstrated optimization techniques for an LBM implementation which are able to achieve high performance on cache-based microprocessors both on complex geometries and for high obstacles ratio. The use of indirect addressing through 1-D array data representation calls for architecture dependent optimization. While space-filling curves are an interesting approach which is able to improve performance, there are other techniques that lead to better results. For large caches, the *collision optimized layout* combined with data blocking shows best performance, while the *propagation optimized layout* suffers from too many obstacles (or too few adjacent fluid cells, respectively) and compiler optimization problems due to array addressing. On systems with smaller caches manual block preload and splitting of the collision into several loops can significantly improve the performance of the *propagation optimized layout* such that it supersedes the *collision optimized layout.*
In our test cases the use of space-filling curves did not show any benefit compared to standard blocking techniques.
To maintain the idea of using space-filling curves, future work could include staggered Hilbert curves for the different directions of the particle distribution function, such that data loaded during the update of cells that are neighboring to lattice sites which are visited much later does not lead to waste of underused cache lines. As a second possibility, a Galerkin-discretization of LBM could enable a stack technique in combination with space-filling curves as was demonstrated in [12] for Navier-Stokes solvers. However, before space-filling curves can be applied in real-world problems, their construction on non-cubic grids has to be investigated.

7 Acknowledgements

This work is financially supported by the Bavarian Graduate School for Computational Engineering. Support from the Lattice Boltzmann Development Consortium, in particular Jörg Bernsdorf (NEC CCRLE) who provided the initial implementation of the 1-D LBM code, is gratefully acknowledged.

References

[1] D. A. Wolf-Gladrow, Lattice-Gas Cellular Automata and Lattice Boltzmann Models, Vol. 1725 of Lecture Notes in Mathematics, Springer, Berlin, 2000.

[2] S. Succi, The Lattice Boltzmann Equation – For Fluid Dynamics and Beyond, Clarendon Press, 2001.

[3] S. Chen, G. D. Doolen, Lattice Boltzmann method for fluid flows, Annu. Rev. Fluid Mech. 30 (1998) 329–364.

[4] G. Wellein, T. Zeiser, G. Hager, S. Donath, On the single processor performance of simple lattice Boltzmann kernels, accepted by Computers & Fluids, 2005 .

[5] T. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers, G. Wellein, T. Zeiser, Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures, in: Proceedings of Supercomputing Conference SC2004, Pittsburgh, 204, CD-ROM, 2004.

[6] S. Donath, On optimized implementations of the lattice Boltzmann method on contemporary high performance architectures, Bachelor's thesis, Chair of System Simulation, University of Erlangen-Nuremberg, Germany (2004).

[7] Y. H. Qian, D. d'Humières, P. Lallemand, Lattice BGK models for Navier-Stokes equation, Europhys. Lett. 17 (6) (1992) 479–484.

[8] J. Bernsdorf, unpublished (2004).

[9] H. Sagan, Space-Filling Curves, Springer-Verlag, 1994.

[10] G. Jin, J. Mellor-Crummey, SFCGens: A framework for efficient generation of multi-dimensional space-filling curves by recursion, ACM Transactions on Mathematical Software Volume 31 (1) (2005) pp. 120 – 148.

[11] G. Hager, T. Zeiser, J. Treibig, G. Wellein, Optimizing performance on modern HPC systems: Learning from simple kernel benchmarks, in: Computational Science and High Performance Computing. Proceedings of the 2nd Russian-German Advanced Research Workshop, 2005.

[12] F. Günther, Eine cache-optimale Implementierung der Finite-Elemente-Methode, Ph.D. thesis, Fakultät für Informatik der Technischen Universität München (2004).