Performance and Accuracy of Lattice-Boltzmann Kernels on Multi- and Manycore Architectures

Dirk Ribbrock, Markus Geveler, Dominik Göddeke, Stefan Turek

Institut für Angewandte Mathematik und Numerik (LS3), TU Dortmund,
Vogelpothsweg 87, D-44227 Dortmund, Germany

Abstract

We present different kernels based on Lattice-Boltzmann methods for the solution of the two-dimensional Shallow Water and Navier-Stokes equations on fully structured lattices. The functionality ranges from simple scenarios like open-channel flows with planar beds to simulations with complex scene geometries like solid obstacles and non-planar bed topography with dry-states and even interaction of the fluid with floating objects. The kernels are integrated into a hardware-oriented collection of libraries targeting multiple fundamentally different parallel hardware architectures like commodity multicore CPUs, the Cell BE, NVIDIA GPUs and clusters. We provide an algorithmic study which compares the different solvers in terms of performance and numerical accuracy in view of their capabilities and their specific implementation and optimisation on the different architectures. We show that an eightfold speedup over optimised multithreaded CPU code can be obtained with the GPU using basic methods and that even very complex flow phenomena can be simulated with significant speedups without loss of accuracy.

Keywords: High performance computing, Lattice-Boltzmann methods, shallow water equations, fluid-structure interaction, GPU computing, CUDA, Cell BE, multithreading
PACS: 02.70.-c, 07.05.Bx, 89.20.Ff, 47.11-j
2010 MSC: 35Q68, 74F10, 35Q35

1. Introduction and Motivation

The accurate and interactive simulation of fluid flow is very challenging, in particular in terms of computational resources. The (two-dimensional) *shallow water equations* (SWE) are a well known formulation to approximating a three-dimensional fluid by its surface using a depth-averaged quantity for the vertical velocity. They are used to reduce computational complexity, and are applicable for instance in dam-break- and open-channel flows. In the inhomogeneous case, source terms are used to represent external forces such as wind shear stress or slope and friction induced by the bed topography.

The Lattice-Boltzmann method (LBM) is a modern numerical technique that starts with a fully discrete model rather than discretising a set of partial differential equations and solving

Email address: dirk.ribbrock@math.tu-dortmund.de (Dirk Ribbrock)

them directly. One of the key features of the LBM with fully structured lattices is that an implementation in parallel is comparably easy, which makes it a promising method, especially in view of modern computational hardware, which evolves towards massive fine-grained parallelism:

During the past few years, computer architecture has reached a turning point. Together, the memory, power and ILP (*Instruction Level Parallelism*) wall form a 'brick wall' [1], and performance is no longer increased by frequency scaling, but by parallelisation and specialisation. Commodity CPUs have up to six cores, the Cell processor is heterogeneous, and throughput-oriented fine-grained parallel designs like GPUs are transitioning towards becoming viable general purpose compute resources. On the software side, programming models for fine-grained parallelism are subject to active discussion and are rapidly evolving. Programmers have to adapt to this inevitable trend, because compiler support is on the far horizon if at all, in particular for computations with low arithmetic intensity (ratio of arithmetic operations per memory transfer). Established parallelisation strategies for both shared and distributed memory architectures have to be revisited, and different strategies are necessary for different architectures. Various optimisation techniques may lead to numerical deviations and it is no longer granted that the same asymptotically very efficient algorithm is optimal in terms of runtime efficiency on all these architectures. In addition, recent publications show that any sophisticated extension like, for example *fluid-structure interaction* (FSI) often leads to a significant loss of performance on non-commodity hardware.

1.1. Related Work

Fan et al. [2] were the first to implement a Lattice-Boltzmann solver on a cluster of GPUs. Advanced Lattice-Boltzmann solvers on CPUs and GPUs have been implemented by Tölke and Krafczyk [3], Thürey [4] and Pohl [5]. Stürmer et al. [6] have implemented a Lattice-Boltzmann solver using the Cell BE simmulating a three dimensional blood flow. Closest in spirit are the publications by Williams et al. [7] and Peng et al. [8], who have developed and benchmarked the performance of Lattice-Boltzmann solver libraries supporting different emerging and leading multicore platforms. Hübner et al. [9] provide numerical studies for a sophisticated solver for the two-dimensional Navier-Stokes equations using a fully implicit approach to the Lattice-Boltzmann equation as a PDE.

1.2. Paper Contribution and Paper Overview

In section 2.1 and section 2.2 we briefly review the shallow water equations and their solution using the Lattice-Boltzmann method with support for internal boundaries. In section 2.3 we summarise some modifications to the LBM to incorporate more realistic simulation scenarios with nontrivial bed topologies, in particular the dynamic flooding and drying of areas. Furthermore, this section describes our approach to couple the simulation of fluids with moving solid objects that influence the behaviour of the fluid. Section 3 is dedicated to parallelisation and vectorisation techniques for the LBM-SWE solvers. We present efficient algorithms for all levels of parallelism encountered in modern computer architectures. In section 4 we measure the numerical accuracy and performance of our solvers for several prototypical benchmark problems. Performance is evaluated on a cluster of conventional CPUs communicating via MPI, on multi-socket multi-core systems, on a Cell blade, and on modern fully programmable GPUs. We are convinced that such algorithmic studies with respect to exploiting parallelism on various levels for a given application are necessary at this point, in particular in view of the challenges outlined in this section. We conclude with a summary and a discussion in section 5.

2. Mathematical Background

2.1. Shallow Water Equations

Using the Einstein summation convention (subscripts i and j are spatial indices) the two-dimensional shallow water equations in tensor form read

$$\frac{\partial h}{\partial t} + \frac{\partial (hu_j)}{\partial x_j} = 0 \quad \text{and} \quad \frac{\partial hu_i}{\partial t} + \frac{\partial (hu_i u_j)}{\partial x_j} + g\frac{\partial}{\partial x_i}(\frac{h^2}{2}) = -g\left(h\frac{\partial b}{\partial x_i} + n_b^2 h^{-\frac{1}{3}} u_i \sqrt{u_j u_j}\right), \quad (1)$$

where h is the fluid depth, $\mathbf{u} = (u_1, u_2)^T$ its velocity in x- and y-direction, and g denotes the gravitational acceleration. In addition, we apply a source term which internalises forces acting on the fluid due to the slope of the bed and material-dependent friction: The slope term is defined by the sum of the partial derivatives of the bed topography, weighted by gravitational acceleration and fluid depth (b denoting the bed elevation), and we define the friction term using the Manning equation, where n_b denotes a material-specific roughness coefficient.

2.2. Lattice Boltzmann Method

In order to solve problem (1) with some initial conditions $h(\mathbf{x}, t = 0)$, $\mathbf{u}(\mathbf{x}, t = 0)$ and a constant bed topography, $b(\mathbf{x})$, we apply the Lattice-Boltzmann method (LBM) with a suitable equilibrium distribution to recover the SWE. In the LBM, the fluid behaviour is determined by particle populations residing at the sites of a regular grid (the *lattice*). The particles' movement (*streaming*) is restricted to fixed trajectories $\mathbf{e}_\alpha$ (*lattice-velocities*) defined by a local neighbourhood on the lattice. We use the D2Q9 lattice, which defines the lattice-velocities in the direction of the eight spatial neighbours as $\mathbf{e}_0 = (0, 0)$, $\mathbf{e}_\alpha = e(\cos\frac{(\alpha-1)\pi}{4}, \sin\frac{(\alpha-1)\pi}{4})$ for $\alpha \in \{1, 3, 5, 7\}$ and $\mathbf{e}_\alpha = \sqrt{2}e(\cos\frac{(\alpha-1)\pi}{4}, \sin\frac{(\alpha-1)\pi}{4})$ for $\alpha \in \{2, 4, 6, 8\}$ with $e = \frac{\Delta x}{\Delta t}$ being the ratio of lattice spacing and timestep. Particle behaviour is defined by the Lattice-Boltzmann equation and a corresponding *collision* operator. Here, the Lattice-Bhatnagar-Gross-Krook (LBGK) collision operator [10] is used, which is a linearisation of the collision-integral around its equilibrium state with a single uniform relaxation time τ. Using this relaxation, the Lattice-Boltzmann equation can be written as

$$f_\alpha(\mathbf{x}+\mathbf{e}_\alpha\Delta t, t+\Delta t) = f_\alpha(\mathbf{x}, t) - \frac{1}{\tau}(f_\alpha - f_\alpha^{eq}) + \frac{\Delta t}{6e^2} e_{\alpha i}\left(-g\left(h\frac{\partial b}{\partial x_i} + n_b^2 h^{-\frac{1}{3}} u_i \sqrt{u_j u_j}\right)\right),\ \alpha = 0, \ldots, 8, \quad (2)$$

where f_α is the particle distribution corresponding with lattice-velocity $\mathbf{e}_\alpha$ and f_α^{eq} a local equilibrium distribution, which defines the actual equations that are solved. In order to recover the SWE, a suitable f_α^{eq} has to be defined for every lattice-velocity. Zhou [11] has shown that the equilibria can be written as

$$f_\alpha^{\text{eq}} = \begin{cases} h(1 - \frac{5gh}{6e^2} - \frac{2}{3e^2}u_i u_i) & \alpha = 0 \\ h(\frac{gh}{6e^2} + \frac{e_{\alpha i}u_i}{3e^2} + \frac{e_{\alpha j}u_i u_j}{2e^4} - \frac{u_i u_i}{6e^2}) & \alpha = 1, 3, 5, 7 \\ h(\frac{gh}{24e^2} + \frac{e_{\alpha i}u_i}{12e^2} + \frac{e_{\alpha j}u_i u_j}{8e^4} - \frac{u_i u_i}{24e^2}) & \alpha = 2, 4, 6, 8 \end{cases} \quad (3)$$

and that the SWE can be recovered by applying Chapman-Enskog expansion on the LBGK approximation (2). Finally, macroscopic mass (fluid depth) and velocity are obtained by

$$h(\mathbf{x}, t) = \sum_\alpha f_\alpha(\mathbf{x}, t) \quad \text{and} \quad u_i(\mathbf{x}, t) = \frac{1}{h(\mathbf{x}, t)} \sum_\alpha e_{\alpha i} f_\alpha, \quad \alpha = 0, \ldots, 8 \quad (4)$$

respectively. We use the popular bounce-back rule as boundary conditions, where particles are reflected using opposite outgoing directions.

2.3. Extended methods

Using the LBM introduced above as a starting point for advanced solvers, we present two general extensions to the basic LBGK method.

The LBM for the shallow water equations presented above can interact with the bed surface and therefore is not restricted to simple scenarios (as it would be if an equilibrium distribution function corresponding to the two-dimensional Navier-Stokes equations had been used). However, it is restricted to subcritical flows, i. e., the fluid depth is significantly greater than zero. The first extension to the method aims at allowing so-called *dry-states*, since the dynamic drying and wetting of the bed topography is a feature desired by many applications. In our approach, we define a fluid depth lower than a specified small threshold parameter as *dry* and set the macroscopic velocity at dry sites to zero, to avoid the division by zero in the extraction phase of the original algorithm (the evaluation of equation (4)). Additionally, local oscillations caused by critical non-zero fluid depths are confined with an adaptive limiter approach.

In order to simulate rigid bodies moving within the fluid (fluid structure interaction, FSI), the method described so far is extended by three major algorithmic steps: In the first step, the forces acting on the fluid due to a moving boundary have to be determined. We use the so called BFL-rule [12], which interpolates the momentum values for the consistency with non-zero Dirichlet boundary conditions induced by a moving boundary. The interpolation is achieved by taking values in opposite direction of the solid movement similar to the bounce-back rule.

The second major step in performing FSI is the extrapolation of missing macroscopic quantities. Moving solids imply that the lattice is in general not invariant over time: Lattice sites that belong to a solid region at time t may become fluid sites at time $t + \Delta t$. In this case, the missing quantities have to be reconstructed. We use an indirect so-called *equilibrium refill* method proposed for example by Caiazzo [13], which uses a three point-backward approximation after calculating the opposite direction of the solid's movement in order to use one-dimensional extrapolation only.

Finally, the force acting on the solid due to fluid movement is determined by the Momentum-Exchange algorithm (MEA) [14].The MEA uses special distribution functions to compute the moments resulting from incoming particles and outgoing particles corresponding to a single lattice-velocity at a specific solid (boundary) point.

3. Implementation and Parallelisation

3.1. LBM Solvers

We have implemented a modular 'construction kit' for Lattice-Boltzmann based solvers for the SWE and concentrate our analysis on five configurations: Our basic solver ($\mathrm{LBM_{SWE}}$) is designed for simple simulations and does not support source terms. Hence, it is suitable for the uniform SWE (equations (1) with its right hand side set to zero) and for flow problems involving only a planar bed. To enable a numerical comparison with other CFD solvers, the equilibrium distribution functions (3) can be replaced by those for the Navier-Stokes equations (solver $\mathrm{LBM_{NAVSTO}}$), see for example Caiazzo for the exact equations [13]. Including the source term (right hand side of equation (1)) for the bed's slope and friction and the techniques described in section 2.3 to incorporate dry-states into our first solver leads to a method for the inhomogeneous SWE ($\mathrm{LBM_{SWE+}}$). Adding all functionality associated with moving self-propelled solids results in a FSI capable solver ($\mathrm{LBM_{FSI}}$). Finally, the same configuration as $\mathrm{LBM_{SWE+}}$ using the Navier-Stokes module instead of the SWE module is labelled $\mathrm{LBM_{NAVSTO+}}$. As reference we use

a FEM discretisation for the two-dimensional Navier-Stokes equations ($\text{FEM}_{\text{NAVSTO}}$) provided by the FEATFLOW project [15] for our numerical validation, see section 4.

3.2. Efficient Parallelisation and Vectorisation

In all solvers presented in the previous section, parallelism is trivially abundant: All work performed for each lattice site is independent of all other sites. However, this general observation does not lead in a straightforward manner to an efficient parallelisation and in particular vectorisation, especially for the more complex solvers. Our implementation supports coarse-grained parallelism for distributed memory systems, medium-grained parallelism on multicore shared memory machines, and fine-grained parallelism corresponding to the SIMD paradigm. The latter is important not only in the SSE units of conventional CPUs, but also on graphics processors. For instance, the SIMD width is 32 on current NVIDIA CUDA-capable GPUs. Only the actual implementation of the algorithms varies for different architectures, see section 3.3.

The SIMD processing implies that branches should be avoided in the innermost loops, because otherwise serialisation of the branches occurs. In the context of our FSI-LMB solver, sites can be fluid, solid, dry or moving boundary, and each type has to be treated differently. Furthermore, different computations are performed in the collision steps for the nine lattice velocities of the D2Q9 model. For an efficient vectorisation, we want to store all data contiguously in memory. A special packing algorithm is used to determine the largest connected areas in the given domain: For the basic solver without source terms and FSI, all obstacle sites can be eliminated a priori, as they contain no fluid throughout the entire calculation. In a second step, all remaining sites are classified with respect to their neighbours in all nine directions. For example, if the northern neighbours of two adjacent lattice-sites are also adjacent and have the same boundary conditions, the solver can process these sites in a vectorised manner without branches. However, for the advanced algorithms, this *lattice-compression* technique is not suitable since the lattice is dynamically altered by e.g. the movement of the solids. In this case, the packing algorithm is only run once in the preprocessing phase, packing only *stationary* obstacles as in the original algorithm. Dynamic lattice transformation in the actual simulation is achieved by tagging lattice sites either as *fluid*, *solid* or *fluid-boundary*, etc. In all cases, the packed data is stored in one-dimensional arrays that contain ranges of lattice-sites with related neighbours and similar boundary conditions. In addition to vectorisation, the approach also ensures good spatial and temporal locality of the computations.

Figure 1 exemplarily depicts the data layout stemming from the packing- and domain decomposition algorithms for a situation with one stationary obstacle (top) and one object moving to the right (bottom right). Hence, the vectors only contain data for fluid sites. The classification step generates index vectors containing intervals that guide the computation for each lattice velocity (illustrated by the differently colored arrows in figure 1): All fluid sites in an interval can be treated equally by the LBM algorithms. This approach is very similar to the one proposed by Krafczyk et al. [16]. Since moving objects alter the lattice in each timestep, the computationally intense preprocessing is avoided by using dynamic marking of the lattice sites geometrically enclosed by the moving solid's boundary (e.g. solid (gold), solid-boundary (tan) and sites that need to be initialised (red) in figure). Note, that the ratio of fluid and (stationary) solid sites is usually much greater than in our small example and the (integer) index vectors are only as long as needed for the specific lattice-velocity to keep the additional memory imprint as small as possible.

To be able to distribute the solver calculation across various cores and calculate the solution in parallel, the domain (the packed data vectors) is partitioned into different nearly independent parts. We pad local arrays with a few ghost entries (white sites in figure 1), allowing it to

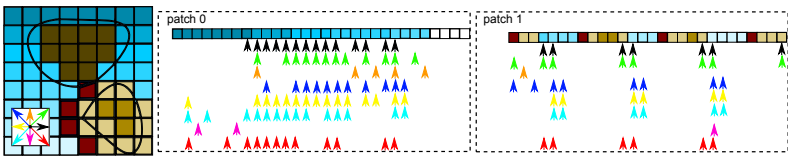


Figure 1: Data layout after preprocessing: Left: Full distribution function. Right: Data- and index vectors after partitioning. The colored arrows visualise index limits that define the intervals for homogeneous computations: The black arrows represent integer values associated with distribution function f_1, the green ones with f_2 and so on. For instance, in the first part of the domain (denoted with patch 0), the first black arrow indicates that streaming into direction $(1,0)^T$ is no longer valid, since the index reaches the right boundary. Hence, any streaming can be processed in a branch-free way. In addition, note that in patch 0, all non-fluid sites are not represented in the packed data and index vectors, because they represent a stationary solid obstacle whereas patch 1 contains the moving solid and therefore no packing applies.

synchronise with its predecessor and successor. As we use a one-dimensional data layout, each part has only two direct neighbour subdomains to interact with. After each time step every part sends its own results corresponding to subdomain boundaries to the ghost sites of its direct neighbours. As soon as every part has finished these independent, non-blocking transfers, the next time step calculation can begin. On shared memory architectures, the synchronisation phase does not involve message passing, but can be realised via locks on shared data, and thus the procedure is conceptually the same. Consequently, the communication between the different parts is very efficient, because it involves no serialisation.

3.3. Hardware-Oriented Implementation

The solver is built on top of the HONEI libraries [17] to be able to use the different target hardware plattforms efficiently. HONEI provides a wide range of software backends to access different hardware via a unified interface. Its generic backend approach enables the programmer to develop code without having to care about specific hardware details, and applications built on top of the libraries are written only once and can directly benefit from hardware acceleration by simply designating them with a hardware tag. Furthermore, the backend specific infrastructure eases the development of application-specific functionality, because hardware specific optimisation has not to be done from scratch: The CPU backend is built around SSE intrinsics. The multicore backend uses PThreads to provide an abstract thread type and tools to execute and synchronise these threads. The GPU backend is based on NVIDIA CUDA and provides simplified access to any CUDA enabled GPU. In addition, all memory transfers between main memory and the GPU device memory are done automatically and executed only if necessary. The Cell backend enables support for the IBM Cell BE and grants a comfortable way to create custom SPE programs on top of the IBM SPE libraries. Finally, the MPI backend encapsulates the common message passing interface.

4. Results

4.1. Validation

Since all backend-specific implementations compute the same numerical results (within a very small limit) as the CPU-implementation, the following numerical tests concentrate on a comparison with an external CFD code as well as with comparisons between the different solvers. We first set up two test scenarios for each LBM-SWE solver described in section 3.1 and compare

their ability to conserve macroscopic mass (i. e., fluid depth in the SWE case). The basic LBM solver $\mathrm{LBM}_{\mathrm{SWE}}$ is tested with a full dam-break simulation with a cuboidal fluid body collapsing over the initially planar surface (scenario FCuDB). In addition, we introduce some stationary obstacles to simulate a partial dam-break situation (scenario PCuDB). For the solver configuration that makes use of the source terms, we modify the first scenario by employing a non-planar bed topography given by $b_\gamma(x, y) = \gamma(x^2 + y^2)$ for subcritical values of γ and label this scenario with $\mathrm{FCuDB}_{\mathrm{slope}}$. Critical fluid depths are simulated using the partial dam-break situation with an initially dry lower basin (scenario $\mathrm{PCuDB}_{\mathrm{dry}}$). Finally, the solver with full functionality is set up to simulate a single box moving through the fluid in non-cartesian direction with planar bed topography (scenario BoxFSI) as well as with the bottom defined by the function b_γ (scenario $\mathrm{BoxFSI}_{\mathrm{slope}}$).

We measure the relative error in volume (which is the error in macroscopic mass in the SWE case) as $E_{\mathrm{vol}} = \left|\frac{V_{\mathrm{ana}} - V_{\mathrm{res}}}{V_{\mathrm{ana}}}\right|$, where V_{ana} is the analytical volume. Table 1 displays the resulting relative errors after convergence for the three LBM-SWE solvers and the corresponding scenarios for different levels of discretisation (level d implies a lattice resolution of $(50 \cdot 2^{d-1})^2$). We emphasise, that these numbers do not vary significantly when using different architectures. The results presented are produced by using the CUDA backend with a NVIDIA GeForce GTX 285 in single precision.

	$\mathrm{LBM}_{\mathrm{SWE}}$		$\mathrm{LBM}_{\mathrm{SWE+}}$		$\mathrm{LBM}_{\mathrm{FSI}}$	
d	FCuDB	PCuDB	$\mathrm{FCuDB}_{\mathrm{slope}}$	$\mathrm{PCuDB}_{\mathrm{dry}}$	BoxFSI	$\mathrm{BoxFSI}_{\mathrm{slope}}$
1	1.83E-04	9.33E-04	2.75E-02	1.11E-01	1.74E-02	1.66E-02
2	6.37E-05	5.56E-04	2.74E-02	5.05E-02	3.41E-03	1.94E-02
3	7.96E-06	4.45E-06	2.74E-02	5.70E-03	9.98E-04	1.50E-02
4	1.11E-06	1.48E-06	1.01E-02	5.01E-03	2.58E-06	3.00E-03
5	7.97E-07	7.40E-06	8.96E-03	5.00E-03	4.00E-07	2.60E-03
6	7.96E-07	0.96 E-06	4.70E-04	4.98E-03	2.01E-07	1.01E-04

Table 1: Relative error in volume E_{vol} for different solvers and simulations.

As it is common with the LBGK approximation for the SWE, we verify, that single versus double precsion is not an issue here as well and double precision results are usually not significantly better than single precision ones. Secondly, all error values decrease as expected at higher lattice-resolution. At the largest problem sizes, the relative errors range from 0.000005 % using the basic solver, to 0.05 %with a more complex bed topography. The same holds true for our $\mathrm{LBM}_{\mathrm{FSI}}$ solver which implies that the added FSI functionality does not lead to a significant mass loss. With dry-states present, our experimental approach reaches an error of approximately 0.5 % at higher lattice sizes.

	E_{FEM}	E_{FEM}		#iters	#iters
$(\Delta x, \Delta t, \tau)$	$\mathrm{LBM}_{\mathrm{NAVSTO}}$	$\mathrm{LBM}_{\mathrm{NAVSTO+}}$	reduction	$\mathrm{LBM}_{\mathrm{NAVSTO}}$	$\mathrm{LBM}_{\mathrm{NAVSTO+}}$
(1, 1, 1)	2.00E-03	1.55E-03	22.5%	26250	26120
(1, 1, 1.5)	2.12E00	8.90E-01	58%	17369	17119
(1, 1, 0.6)	1.57E00	1.02E00	35%	74665	64434
(1, 1, 0.9)	5.0E-01	8.50E-02	83%	30031	29257
(1, 1, 1.1)	2.52E-01	1.08E-01	57%	23654	23580
(1, 0.9, 1)	1.95E-03	1.55E-03	20.5%	26700	26347
(1, 1.1, 1)	1.95E-03	1.55E-03	20.5%	26025	25894
(0.9, 1, 1)	1.95E-03	1.55E-03	20.5%	26024	25669

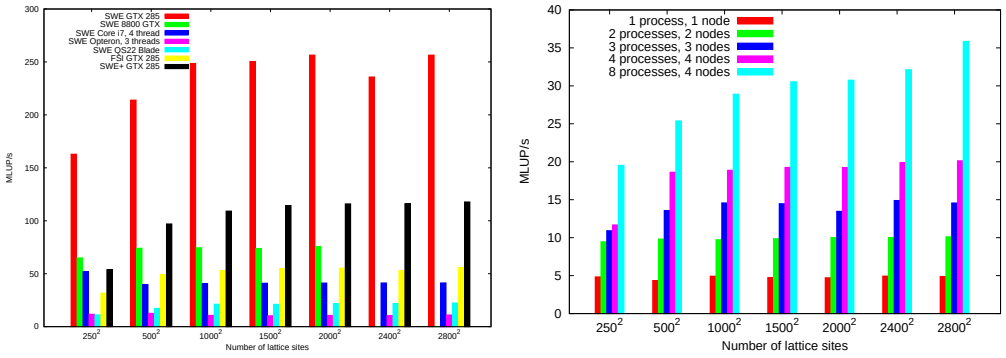
Table 2: Relative error E_{FEM} and numbers of iterations until steady-state for the driven-cavity test.

In a second test series, we compare our Navier-Stokes based solver configurations to the results computed by the corresponding FD solver provided by Hübner [9] as well as with the corresponding solver within the FEATFLOW project [15]. Besides the comparison with external codes, this test aims at showing that our most sophisticated solver, $\mathrm{LBM}_{\mathrm{FSI}}$, may produce

quantitatively more accurate results than the basic solver. In order to do so, we use a lid-driven cavity scenario with the Reynolds number set to 100 and represent the moving wall by a moving solid in order to apply the microscopic moving-boundary conditions instead of resetting only macroscopic velocity in each timestep. Table 2 shows the error $E_{\mathrm{FEM}} = \|u_{\mathrm{FEM}} - u_{\mathrm{res}}\|$ and the total numbers of iterations until a steady state is reached, where u_{FEM} is the x-component of the macroscopic velocity at a vertical testline through the center of the domain computed by the solver $\mathrm{FEM}_{\mathrm{NAVSTO}}$ and u_{res} the corresponding result using $\mathrm{LBM}_{\mathrm{NAVSTO}}$ or $\mathrm{LBM}_{\mathrm{NAVSTO+}}$ respectively. The results are shown for several different solver parameterisations labelled with the corresponding values of $(\Delta x, \Delta t, \tau)$. It can be seen that the LBM solvers can be parameterised so that they are competitive to the FEM solvers: Error norms around 0.2 % imply component-wise differences in the macroscopic velocity at the fourth or fifth digit in average, although, as expected, they can of course not compete against them when more accuracy is required. Furthermore, our moving boundary-capable solver enhances numerical accuracy by 20 to 80 % compared to the basic solver and slightly reduces the needed numbers of iterations until the steady-state is reached. Again, these results are independent of the evaluated hardware architectures.

4.2. Performance Benchmarks

We first assess the performance of the basic solver without its source term and FSI modules. Figure 2 shows the mega lattice updates per second (MLUP/s) for increasing lattice size, simulating a partial dam-break scenario.



(a) Performance benchmark

(b) Strong scalability with MPI

Figure 2: Left: Partial dam-break benchmark on different architectures (solver $\mathrm{LBM}_{\mathrm{SWE}}$); with non-planar bed (solver $\mathrm{LBM}_{\mathrm{SWE}*}$); with moving solids (solver $\mathrm{LBM}_{\mathrm{FSI}}$). Right: Strong scaling with MPI on an Opteron cluster with infiniband interconnect.

The CPU SSE/multicore backend is evaluated on a dual-socket dual-core AMD Opteron 2214 system and an Intel Core i7 quad-core workstation. The Cell backend is tested with an IBM QS22 Cell blade with two Cell BE processors. The GPU-based solver is executed on a NVIDIA GeForce 8800 GTX and a GeForce GTX 285. The QS22 blade executes twice as fast as the Opteron system, but is outperformed by a factor of two by the Core i7 system. Even the older 8800 GTX outperforms all CPU systems but is restricted to small lattice sizes due to its comparatively small amount of on-chip memory. Finally, the GTX 285 reaches eight times the performance of the fastest CPU system. These speedup factors are proportional to the bandwidth

to off-chip memory of the various architectures, ranging from 6.4 GB/s per socket (Opteron), 25 GB/s (Cell) and 33 GB/s (i7) to 89 GB/s and 160 GB/s for the two GPUs. We emphasise that the speedup of the GPU over other architectures is in line with the measurements for the basic LBM solver, even with full FSI functionality. The timing measurements for the more complex solvers demonstrate that all solver configurations can cope with high resolutions in reasonable time. For example, even the full algorithm running at 50 MLUP/s can compute a single timestep on a lattice with approximately 1.7 million lattice-sites in less than 0.04 seconds, corresponding to 30 timesteps per second. The more advanced solvers do not benefit from the static lattice compaction (cf. section 3.2) to the same extent as the basic solver, because the domain changes in the course of the simulation. Besides the additional computational effort, the loss in performance compared to the basic solver is therefore certainly due to the increase in conditional branches in the code.

	`EquilibriumDistribution`	`CollideStream`	`Extraction`	`Boundaries`
Opteron 3 threads	61	24	15	0
Core i7 4 threads	48	35	17	0
QS22 blade	7	72	16	5
GeForce 8800 GTX	19	52	19	10
GeForce GTX 285	32	37	24	7

Table 3: Breakdown of the different kernels in one solver step (solver LBM_{SWE}): Fraction of total runtime for the kernel computing equations (3) (`EquilibriumDistribution`), a fused collisions and streaming kernel (`CollideStream`) and a single kernel for determining the macroscopic quantities (equations (4), labeled `Extraction`) and finally, a module responsible for applying boundary conditions (`Boundaries`).

Detailed measurements of the LBM_{SWE} solver reveal, that only the kernel responsible for computing the equilibrium distribution functions is compute-bound, all other operations are limited in performance by the memory bandwidth. Table 3 shows that the equilibrium distribution kernel consumes only a small percentage of execution time on the devices with a high floating point throughput. In contrast these architectures spend much more relative time in the collide and stream kernels, which requires many additional memory transfers.

Figure 2(b) finally shows almost perfect strong scaling of the MPI backend using the LBM_{SWE} solver on a small cluster of Opteron 2214 nodes.

5. Conclusions and Future Work

A hardware oriented approach to two-dimensional flow problems based on different governing equations has been presented. Performance critical kernels have been implemented as 'building blocks' for Lattice-Boltzmann based solvers. Besides a wide applicability, the approach aims at targeting multiple floating point hardware architectures under a unified interface. It has been shown that our approach can cope with complex flow problems with reasonable performance and accuracy on all architectures under consideration.

In future work, we will explore heterogeneous systems, e.g., the simultaneous use of the CPU cores and GPUs in cluster nodes, to maximise the computational efficiency. In addition, performance comparison with different state of the art approaches to solving the SWE are in progress.

Acknowledgements

We would like to thank Danny van Dyk, Sven Mallach and all contributors to HONEI. This work has been supported by Deutsche Forschungsgemeinschaft (DFG) under the grant TU

102/22-2, and by BMBF (call: HPC Software für skalierbare Parallelrechner) in the SKALB project (01IH08003D / SKALB). Thanks to NVIDIA for generous hardware donations, and to IBM Germany for access to QS22 blades.

References

[1] M. V. Wilkes, The memory gap (keynote), in: Workshop on Solving the Memory Wall Problem, ISCA-2000, 2000, http://www.ece.neu.edu/conf/wall2k/wilkes1.pdf.

[2] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover, GPU cluster for high performance computing, in: SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, 2004, p. 47. doi:10.1109/SC.2004.26.

[3] J. Tölke, M. Krafczyk, TeraFLOP computing on a desktop PC with GPUs for 3D CFD, International Journal of Computational Fluid Dynamics 22 (7) (2008) 443–456. doi:10.1080/10618560802238275.

[4] N. Thürey, K. Iglberger, U. Rüde, Free surface flows with moving and deforming objects for LBM, Proceedings of Vision, Modeling and Visualization 2006 (2006) 193–200.

[5] T. Pohl, High Performance Simulation of Free Surface Flows Using the Lattice Boltzmann Method, Ph.D. thesis, Universität Erlangen-Nürnberg (2008).

[6] M. Stürmer, J. Götz, G. Richter, U. Rüde, Blood flow simulation on the Cell Broadband Engine using the Lattice Boltzmann Method, Tech. Rep. 07-9, Lehrstuhl für Systemsimulation, Universität Erlangen-Nürnberg (2007).

[7] S. Williams, J. Carter, L. Oliker, J. Shalf, K. A. Yelick, Lattice Boltzmann simulation optimization on leading multicore platforms, in: IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008, pp. 1–14. doi:10.1109/IPDPS.2008.4536295.

[8] L. Peng, K. Nomura, T. Oyakawa, R. Kalia, A. Nakano, P. Vashishta, Parallel Lattice Boltzmann Flow Simulation on Emerging Multi-core Platforms, Springer-Verlag, Berlin, Heidelberg, 2008.

[9] T. Hübner, S. Turek, Efficient monolithic simulation techniques for the stationary Lattice Boltzmann equation on general meshes, Computing and Visualization in Science 13 (3) (2010) 129–143. doi:1007/s00791-009-0132-6.

[10] F. J. Higuera, J. Jimenez, Boltzmann approach to lattice gas simulations, EPL (Europhysics Letters) 9 (7) (1989) 663–668. doi:10.1209/0295-5075/9/7/009.

[11] J. G. Zhou, Lattice Boltzmann methods for shallow water flows, Springer, 2004.

[12] M. Bouzidi, M. Firdaouss, P. Lallemand, Momentum transfer of a Boltzmann-lattice fluid with boundaries, Physics of Fluids 13 (11) (2001) 3452–3459. doi:10.1063/1.1399290.

[13] A. Caiazzo, Asymptotic Analysis of lattice Boltzmann method for Fluid-Structure interaction problems, Ph.D. thesis, Technische Universität Kaiserslautern, Scuola Normale Superiore Pisa (Feb. 2007).

[14] A. Caiazzo, M. Junk, Boundary forces in lattice Boltzmann: Analysis of momentum exchange algorithm, Computers & Mathematics with Applications 55 (7) (2008) 1415–1423.

[15] C. Becker, S. Turek, Featflow – finite element software for the incompressible Navier-Stokes equations, User manual, Universität Dortmund (1999).

[16] M. Krafczyk, P. Lehmann, O. Philippova, D. Hänel, U. Lantermann, Lattice Boltzmann Simulations of complex Multi-Phase Flows, Springer, 2000.

[17] D. van Dyk, M. Geveler, S. Mallach, D. Ribbrock, D. Göddeke, C. Gutwenger, HONEI: A collection of libraries for numerical computations targeting multiple processor architectures, Computer Physics Communications 180 (12) (2009) 2534–2543. doi:10.1016/j.cpc.2009.04.018.