Lattice-Boltzmann Simulation of the Shallow-Water Equations with Fluid-Structure Interaction on Multi- and Manycore Processors

Markus Geveler, Dirk Ribbrock, Dominik Göddeke, and Stefan Turek

Institut für Angewandte Mathematik, TU Dortmund, Germany,
markus.geveler@math.tu-dortmund.de

Abstract. We present an efficient method for the simulation of laminar fluid flows with free surfaces including their interaction with moving rigid bodies, based on the two-dimensional shallow water equations and the Lattice-Boltzmann method. Our implementation targets multiple fundamentally different architectures such as commodity multicore CPUs with SSE, GPUs, the Cell BE and clusters. We show that our code scales well on an MPI-based cluster; that an eightfold speedup can be achieved using modern GPUs in contrast to multithreaded CPU code and, finally, that it is possible to solve fluid-structure interaction scenarios with high resolution at interactive rates.

Keywords: High performance computing; Lattice-Boltzmann methods; shallow water equations; fluid-structure interaction; CUDA; Cell BE; multithreading

1 Introduction and Motivation

In many practical situations, the behaviour of a fluid can be modelled by the *shallow water equations (SWE)*, e.g., for tidal flows, open water waves (such as tsunamis), dam break flows and open channel flows (such as rivers). In such cases, vertical acceleration of the fluid is negligible because the flow is dominated by horizontal movement, with its wavelength being much larger than the corresponding height. In the SWE, vertical velocity is replaced by a depth-averaged quantity, which leads to a significant simplification of the general flow equations (like the Navier-Stokes equations which are derived from general conservation and continuity laws). In the inhomogeneous SWE, source terms are employed to internalise external forces, e.g., wind shear stress and, more importantly, forces resulting from the interaction between the fluid and the *bed topography*. Using such source terms, the two-dimensional SWE can be used for the simulation of a fluid given by its free surface, which significantly reduces the computational cost and makes them a popular method for instance in (ocean-, environmental- and hydraulic) engineering.

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 them directly. One of the key features of the

LBM 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 (see below).

Besides efficiency, a key feature of a method for advanced simulations involving a fluid is the capability of letting it interact with its environment. This interaction includes the internalisation of the 'world geometry' in terms of the surface the fluid is streaming over and interaction with rigid bodies that move through and are moved by the fluid (fluid-structure interaction, FSI). An algorithm that provides both reasonable performance on modern commodity based computer systems on the one hand and FSI functionality on the other hand is very attractive in engineering and for example in computer graphics, ranging from feature film to games and interactive environments.

During the past few years, computer architecture has reached a turning point. Together, the memory, power and instruction-level parallelism (ILP) 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.

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]. Many publications are concerned with interactive and (visually) accurate simulations of fluid flow [6,7,8,9].

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 present 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 FSI-LBM solver. We present efficient algorithms for all levels of parallelism encountered in modern computer architectures. In Section 4 we demonstrate the

applicability and performance of our approach for several prototypical benchmark problems. Performance is evaluated on a cluster of conventional CPUs communicating via MPI, on multi-socket multi-core SMP 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

$$\frac{\partial h}{\partial t} + \frac{\partial (hu_j)}{\partial x_j} = 0 \quad \text{and} \quad \frac{\partial hu_i}{\partial t} + \frac{\partial (hu_i u_j)}{\partial x_j} + g\frac{\partial}{\partial x_i}(\frac{h^2}{2}) = S_i^b, \tag{1}$$

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 S_i^b which internalises forces acting on the fluid due to the slope of the bed and material-dependent friction:

$$S_i^b = S_i^{\text{slope}} + S_i^{\text{friction}}. \tag{2}$$

The slope term is defined by the sum of the partial derivatives of the bed topography, weighted by gravitational acceleration and fluid depth (b denotes the bed elevation), and we define the friction term using the Manning equation (as suggested by Zhou [10]).

$$S_i^{\text{slope}} = -gh\frac{\partial b}{\partial x_i}, \qquad S_i^{\text{friction}} = -gn_b^2 h^{-\frac{1}{3}} u_i \sqrt{u_j u_j}, \tag{3}$$

n_b denotes a material-specific roughness coefficient. Using the inhomogeneous SWE with source term (2) enables the simulation of a fluid (bounded by its surface) with a two-dimensional method due to the coupling of fluid motion with the bed topography.

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

$$\mathbf{e}_\alpha = \begin{cases} (0,0) & \alpha = 0 \\ e(\cos\frac{(\alpha-1)\pi}{4}, \sin\frac{(\alpha-1)\pi}{4}) & \alpha = 1,3,5,7 \\ \sqrt{2}e(\cos\frac{(\alpha-1)\pi}{4}, \sin\frac{(\alpha-1)\pi}{4}) & \alpha = 2,4,6,8, \end{cases} \tag{4}$$

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 [11] 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

$$f_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t) - \frac{1}{\tau}(f_\alpha - f_\alpha^{eq}) + \frac{\Delta t}{6e^2} e_{\alpha i} S_i^b \ , \alpha = 0, \ldots, 8, \tag{5}$$

where f_α is the particle distribution corresponding to the 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 [10] has shown that the equilibria can be written as

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

and that the SWE can be recovered by applying Chapman-Enskog expansion on the LBGK approximation (5). Finally, macroscopic mass (fluid depth) and velocity can be obtained by

$$h(\mathbf{x}, t) = \sum_\alpha f_\alpha(\mathbf{x}, t) \quad \text{and} \quad u_i(\mathbf{x}, t) = \frac{1}{h(\mathbf{x}, t)} \sum_\alpha e_{\alpha i} f_\alpha, \tag{7}$$

respectively. We use the popular bounce-back rule as boundary conditions, where particles are reflected using opposite outgoing directions and which therefore implements no-slip boundary conditions. In the following, this basic method is used as a starting point for our more sophisticated solver, capable of dealing with complex flow scenarios, including the interaction with moving solid obstacles.

2.3 Dry-States and Fluid Structure Interaction

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[1]

[1] Usually, the term *critical flow* is associated with a Froude number being smaller than one. Throughout this paper, we use it for flows over a bed topography with possibly very small (or even zero-valued) fluid depth.

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 below 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 (7)). Secondly, local oscillations caused by critical non-zero fluid-depths are confined with an adaptive limiter approach. Due to space constraints, we refer to Geveler [12] for details.

In order to simulate rigid bodies moving within the fluid, 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 [13], 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, see Figure 1. In the original

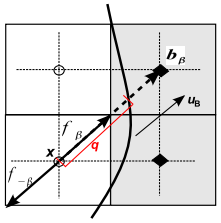


Fig. 1. Modified *bounce-back* scheme for moving boundaries.

BFL formulation, the interpolation needs four coefficients depending on the distance $q = \frac{|\mathbf{b}_\beta - \mathbf{x}|}{\Delta x}$ where $\mathbf{e}_\beta$ is the lattice-velocity approximating the direction of the solid movement, $\mathbf{b}_\beta$ the corresponding point on the solid boundary and $\mathbf{x}$ a location in the fluid (and on the lattice) in opposite direction. In our approach, we use a piecewise linear approximation of the boundary, and set $q = \frac{1}{2}$, which reduces three of the four coefficients to zero. We obtain a very simple formula[2] for the missing momentum, that still respects the moving boundary. Let $\mathbf{u}_B$ be the macroscopic velocity of the solid, our modified boundary condition then reads:

$$f_{-\beta}^{\text{temp}}(\mathbf{x}, t + \Delta t) = 6\Delta x \; w_{-\beta}(\mathbf{u}_B(\mathbf{b}_\beta) \cdot \mathbf{e}_{-\beta}). \tag{8}$$

The superscript temp indicates the distribution function after the collision step. The w_α are weights depending on the lattice, which can be set to $\frac{4}{9}$ for $\alpha = 0$ and $\frac{1}{9}$ for uneven α and $\frac{1}{36}$ in the even case for our D2Q9 lattice, see Caiazzo [14].

[2] It should be noted, that this simplification increases performance but reduces the spatial convergence order to one.

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 [14], 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. Again, the value $q = \frac{1}{2}$ is used and we obtain

$$\begin{aligned}\tilde{h}(\mathbf{x}, t + \Delta t) = 3h(\mathbf{x} + \mathbf{e}_{-\beta}\Delta t, t + \Delta t) - 3h(\mathbf{x} + 2(\mathbf{e}_{-\beta}\Delta t), t + \Delta t) \\ + h(\mathbf{x} + 3(\mathbf{e}_{-\beta}\Delta t), t + \Delta t)\end{aligned} \tag{9}$$

for the extrapolated fluid depth and

$$\begin{aligned}\tilde{\mathbf{u}}(\mathbf{x}, t + \Delta t) = 8/15\Delta x \mathbf{u}_B(\mathbf{b}_\beta, t + \Delta t) + 2/3\mathbf{u}(\mathbf{x} + \mathbf{e}_{-\beta}\Delta t, t + \Delta t) \\ - 2/5\mathbf{u}(\mathbf{x} + 2(\mathbf{e}_{-\beta}\Delta t), t + \Delta t)\end{aligned} \tag{10}$$

for the macroscopic velocities, respectively.

Finally, the force acting on the solid due to fluid movement is determined by the Momentum-Exchange algorithm (MEA) [15], in order to be able to couple the method with a solid mechanics (CSM) solver. The MEA uses special distribution functions to compute the moments resulting from incoming particles and outgoing particles corresponding with a single lattice-velocity $-\beta$ at a solid (boundary) point $\mathbf{b}$:

$$f_{-\beta}^{\mathrm{MEA}}(\mathbf{b}, t) = e_{\beta i}(f_{\beta}^{\mathrm{temp}}(\mathbf{x}, t) + f_{-\beta}^{\mathrm{temp}}(\mathbf{x}, t + \Delta t)). \tag{11}$$

The forces can be aggregated into the total force acting on $\mathbf{b}$:

$$F(\mathbf{b}, t) = \sum_{\alpha} f_{\alpha}^{\mathrm{MEA}}(\mathbf{b}, t). \tag{12}$$

3 Implementation and Parallelisation

3.1 Modular FSI-LBM Solver

The combination of all functionality presented in Section 2 results in the solver given by Algorithm 1. Note that the algorithm is designed in a modular way in order to be able to activate/disable certain functionality, for instance to disable the FSI components in scenarios without moving objects.

3.2 Efficient Parallelisation and Vectorisation

It can be seen in Algorithm 1 that parallelism is trivially abundant in the modified LBM solver: All work performed for each lattice site is independent of all other sites (in the basic algorithm). However, this general observation

Algorithm 1 LBM solver for SWE with FSI

perform preprocessing $\rightarrow h(\mathbf{x}, 0), \mathbf{u}(\mathbf{x}, 0)$
for all timesteps
  approximate extrapolation direction $\rightarrow -\beta$
  determine lattice sites to be initialised
  for all lattice sites to be initialised
    initialise fluid sites (equations (9) und (10))
  for all fluid sites
    for $\alpha = 0$ **to** 8:
      compute equilibrium distribution functions (equation (6))
      perform LBGK collision
      compute momentum exchange (equations (11) and (12))
      for all fluid sites adjacent to moving boundary
        apply modified BFL-rule (equation (8))
      compute and apply source-terms (equations (2) and (3))
      perform LBM streaming
      for all boundary fluid sites not adjacent to moving solid
        apply standard bounce-back scheme
      extract physical quantities (equation (7))

does not lead in a straightforward manner to an efficient parallelisation, and in particular vectorisation. 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. We apply the same techniques for coarse- and medium-grained parallelism on CPUs; and for fine-grained parallelism within CPU cores and GPU multiprocessors, respectively. Only the actual implementation of the algorithms varies for different architectures, see Section 3.4.

The SIMD paradigm 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, 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 in a similar way as it has been proposed by Krafczyk et al. [16]. 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 algorithm employing FSI, this *lattice-compression* technique is not

suitable since the lattice is dynamically altered by 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 areas of lattice-sites with related neighbours and the same boundary conditions. Despite vectorisation, the approach also ensures good spatial and temporal locality of the computations.

To be able to distribute the solver calculation across various cores and to calculate the solution in parallel, the domain (the packed data vectors) is partitioned into different nearly independent parts. We pad each local array with a few ghost entries, allowing it to synchronise with its predecessor and successor. As we use a one-dimensional data layout, each part has only two direct neighbour parts 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 Source Terms and FSI Implementation

The partial derivatives in the slope source term (3) are evaluated by means of the semi-implicit *centred* scheme proposed by Zhou [10], where the slope is computed at the midpoint between the lattice-site and one neighbouring lattice-site and therefore includes the interpolation of the fluid depth:

$$S_i^{\text{slope}} = S_i^{\text{slope}}(\mathbf{x} + \frac{1}{2}\mathbf{e}_\alpha \Delta t, t) \tag{13}$$

With this approach, we are able to achieve a horizontal steady-state even when nonplanar bed topographies are involved, see Section 4.1.

Besides the source terms, two additional solver modules are needed to provide FSI functionality. To increase efficiency, Algorithm 1 can be reformulated, resulting in a fused kernel that performs LBGK collision and LBM streaming and all steps between these two. The corresponding module also automatically corrects the streaming of particles that are influenced by a moving boundary, i. e., a boundary that is not treated as a solid obstacle by our packed lattice data structure. In addition, the BFL rule is applied and the MEA distribution functions are computed. The second FSI module performs the initialisation of fluid sites with all necessary extrapolations.

Keeping track of the lattice flags is achieved by boolean vectors directly corresponding to the compressed data vectors needed by the basic solver and calculations concerning these flags, e.g., determining fluid sites that need to be initialised, are performed on the fly.

3.4 Hardware-Oriented Implementation

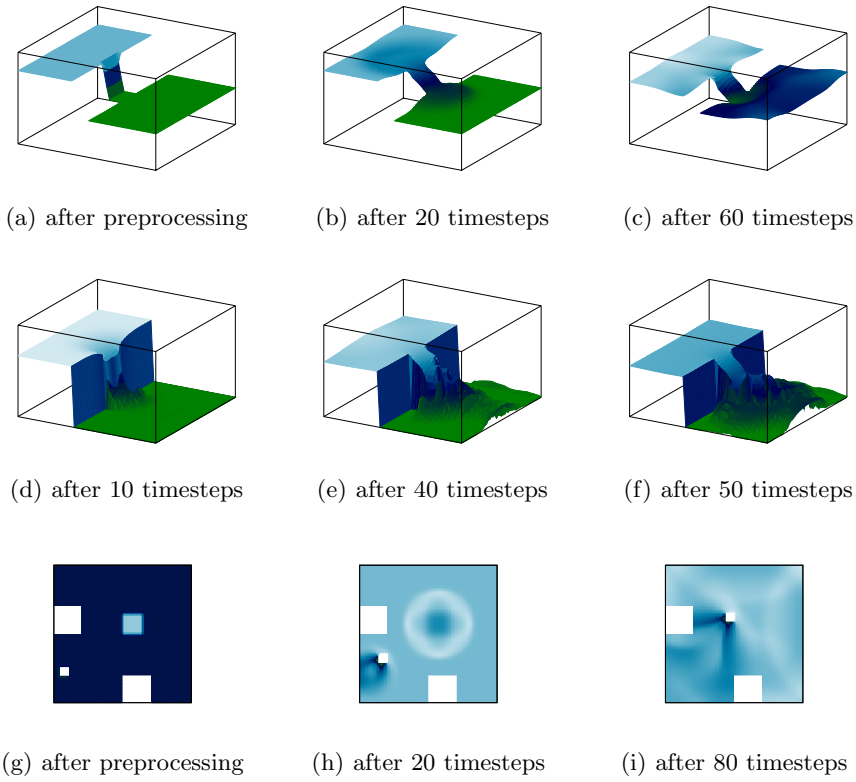
The solver is built on top of the HONEI libraries [17] to be able to use the different target hardware plattforms efficiently. HONEI provides a wide range of software backends to access different hardware via a unified interface. Its generic backend approach enables the programmer to develop code without 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

Figure 2 demonstrates the applicability of our solver for various dam break scenarios including the flooding of dry areas and self-propelled objects moving through the resulting fluid surface. We abstain from giving detailed numerical test results here, and refer to the theses of the first two authors [12,18]. There, it is shown that all solver configurations provide good accuracy in terms of mass conservation, smoothness of quantity-fields and stability, as well as a comparison in terms of accuracy with a solver using a finite element discretisation of the Navier-Stokes equations. The treatment of dry-states by the experimental limiter methods combined with the cutoff of particle populations at the fluid-solid interface may lead to a small loss of mass. Nonetheless, the FSI-LMB solver always computes a stable and visually accurate solution.

The first scenario we present (top row in the figure) is a partial dam break simulation, a standard benchmark problem for shallow water solvers. For this test case, the modules treating source terms, dry-states and FSI are deactivated in our modular solver. The middle row in the figure depicts the same partial dam break simulation, but with supercritical initial fluid depth (dry-states). The results show that such configurations can be stabilised by our solver with no significant loss of mass. The third scenario (bottom row) contains two stationary obstacles and one moving solid coupled with a full cuboidal dam break simulation. Furthermore, this configuration includes the nontrivial bed topography given by $f_\gamma(x, y) = \gamma(x^2 + y^2)$ and is therefore, even though no dry-states



(a) after preprocessing (b) after 20 timesteps (c) after 60 timesteps

(d) after 10 timesteps (e) after 40 timesteps (f) after 50 timesteps

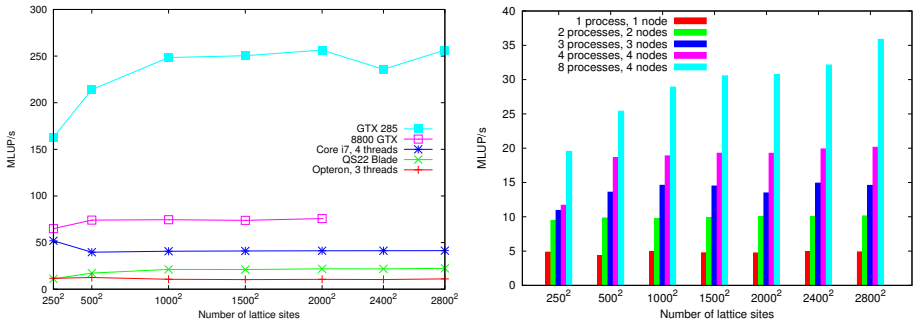
(g) after preprocessing (h) after 20 timesteps (i) after 80 timesteps

Fig. 2. Partial dam break simulations. Top: without source terms and FSI (the lower basin is filled with water initially); middle: with dry-states (the lower basin is empty initially), without FSI; bottom: full FSI simulation (bird's eye view) with cuboidal dam break.

are present, a test case for the full functionality of our solver, because all solver modules including FSI and the treatment of source terms are activated. In all simulations with an uneven bed geometry present, the steady-state solution always cancels out the non-vanishing forces and converges to a horizontal plane.

4.2 Performance Benchmarks

We first evaluate the performance on different multi- and manycore architectures of the basic solver without its source term and FSI modules. Figure 3 (left) shows the mega lattice updates per second (MLUP/s) for increasing lattice size, simulating a partial dam break scenario like the one depicted in Figure 2 (top row). The CPU SSE/multicore backend is evaluated on a dual-socket dual-core AMD Opteron 2214 system and an Intel Core i7 quadcore workstation. The Cell backend is tested with an IBM QS22 blade with two Cell BE processors. The



(a) Performance on different architectures (b) Strong scalability with MPI

Fig. 3. Partial dam break benchmark on different architectures.

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 GTX 8800 outperforms all CPU systems but is restricted to small lattice sizes due to its comparatively small amount of device 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 87 GB/s and 160 GB/s for the two GPUs. Detailed measurements 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 [18]. Figure 3 (right) shows almost perfect strong scaling of the MPI backend on a small cluster of Opteron 2214 nodes.

Our last experiment compares the performance of increasingly complex solver configurations on the GeForce GTX 285 by simulating the three test cases described above. 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 in Figure 4 demonstrate that all solver configurations can cope with high resolutions in reasonable time. For example, even the full algorithm 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.

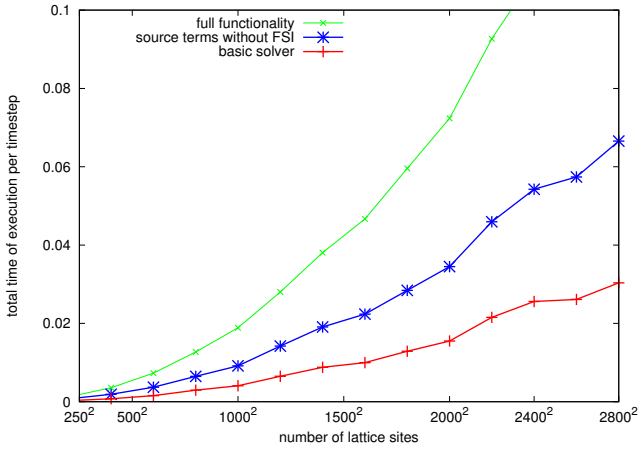


Fig. 4. Time per timestep (in seconds) for the three scenarios with corresponding solver configurations (basic solver, with source terms, full functionality with FSI) on the GeForce GTX 285.

5 Conclusions and Future Work

The combination of the shallow water equations and a suitable Lattice-Boltzmann approach with methods to stabilise dry-states as well as for fluid-structure interaction has been used for a novel approach to simulate 'real-world' free surface fluids. With the presented algorithm, the problem of lacking computational resources for the direct solution of flow equations can be overcome if quantitative accuracy is less important than, for example, visual appearance as it is the common case in computer graphics or entertainment.

In addition to this numerical modeling, we implemented our method on a wide range of parallel architectures, addressing coarse, medium and fine-grained parallelism. 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.

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. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., Yelick, K.A.: The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006)
2. Fan, Z., Qiu, F., Kaufman, A., Yoakum-Stover, S.: GPU cluster for high performance computing. In: SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing. (2004) 47
3. Tölke, J., Krafczyk, M.: TeraFLOP computing on a desktop PC with GPUs for 3D CFD. International Journal of Computational Fluid Dynamics **22**(7) (2008) 443–456
4. Thürey, N., Iglberger, K., Rüde, U.: Free Surface Flows with Moving and Deforming Objects for LBM. Proceedings of Vision, Modeling and Visualization 2006 (2006) 193–200
5. Pohl, T.: High Performance Simulation of Free Surface Flows Using the Lattice Boltzmann Method. PhD thesis, Universität Erlangen-Nürnberg (2008)
6. Molemaker, M.J., Cohen, J.M., Patel, S., Noh, J.: Low viscosity flow simulations for animations. In Gross, M., James, D., eds.: Eurographics / ACM SIGGRAPH Symposium on Computer Animation. (2008)
7. van der Laan, W.J., Green, S., Sainz, M.: Screen space fluid rendering with curvature flow. In: I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games, New York, NY, USA, ACM (2009) 91–98
8. Baboud, L., Décoret, X.: Realistic water volumes in real-time (2006)
9. Krüger, J.: A GPU Framework for Interactive Simulation and Rendering of Fluid Effects. PhD thesis, Technische Universität München (2006)
10. Zhou, J.G.: Lattice Boltzmann methods for shallow water flows. Springer (2004)
11. Higuera, F.J., Jimenez, J.: Boltzmann approach to lattice gas simulations. EPL (Europhysics Letters) **9**(7) (1989) 663–668
12. Geveler, M.: Echtzeitfähige Interaktion von Festkörpern mit 2D Lattice–Boltzmann Flachwasserströmungen in 3D Virtual–Reality Anwendungen. Diploma thesis, Technische Universität Dortmund (2009)
13. Bouzidi, M., Firdaouss, M., Lallemand, P.: Momentum transfer of a Boltzmann-lattice fluid with boundaries. Physics of Fluids **13**(11) (2001) 3452–3459
14. Caiazzo, A.: Asymptotic Analysis of lattice Boltzmann method for Fluid-Structure interaction problems. PhD thesis, Technische Universität Kaiserslautern, Scuola Normale Superiore Pisa (2007)
15. Caiazzo, A., Junk, M.: Boundary forces in lattice Boltzmann: Analysis of momentum exchange algorithm. Computaters & Mathematics with Applications **55**(7) (2008) 1415–1423
16. Krafczyk, M., Lehmann, P., Philippova, O., Hänel, D., Lantermann, U.: Lattice Boltzmann Simulations of complex Multi-Phase Flows. Springer (2000)
17. van Dyk, D., Geveler, M., Mallach, S., Ribbrock, D., Göddeke, D., Gutwenger, C.: HONEI: A collection of libraries for numerical computations targeting multiple processor architectures. Computer Physics Communications **180**(12) (2009) 2534–2543
18. Ribbrock, D.: Entwurf einer Softwarebibliothek zur Entwicklung portabler, hardwareorientierter HPC Anwendungen am Beispiel von Strömungssimulationen mit der Lattice Boltzmann Methode. Diploma thesis, Technische Universität Dortmund (2009)