Computers and Mathematics with Applications 61 (2011) 3549–3563

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Free surface flow simulations on GPGPUs using the LBM

Christian Janßen *, Manfred Krafczyk

Institute for Computational Modeling in Civil Engineering, Technische Universität Braunschweig, Braunschweig, Germany

ARTICLE INFO

Keywords:
Free surface
Lattice Boltzmann method
D3Q19 model
Volume of fluid
Graphical processing unit
MRT
Smagorinsky LES

ABSTRACT

In this paper, we present the implementation of a volume-of-fluid-(VOF)-based algorithm for the simulation of free-surface flow problems on general purpose graphical processing units (GPGPUs). For the solution of the flow field and the additional advection equation for the VOF fill level, the lattice Boltzmann method on the basis of an MRT collision operator is used. A Smagorinsky LES model serves to capture the small-scale turbulent structures of the flow. We show that despite the additional non-local operations near the phase interface, we end up with an algorithm with good overall performance, which is suitable for the simulation of demanding real-world engineering applications.

1. Introduction

Free-surface flow problems occur in numerous fields of civil engineering. Characteristic free surface problems are breaking dams, the wave impact on offshore structures, hydraulic jumps, flood waves and tsunamis. These applications require fast, three-dimensional, turbulent and highly resolved simulations, so that nowadays the demand for powerful simulation frameworks is larger than ever before. Even hybrid models, which combine different fluid models to minimise simulation times, cannot solely solve this problem. We recently coupled a three-dimensional VOF solver to a two-dimensional potential flow code, in order to save computational time in regions of low interest far away from e.g. wave breaking. Nonetheless the run-times of the three-dimensional solver drastically slow down the coupled simulation [1]. Hence, in order to be able to compute large three-dimensional domains in a reasonable amount of time, the utilisation of parallel hardware is crucial. GPGPUs recently introduced the idea of supercomputing on the desktop, to run large-scale simulations locally on a desktop PC without the tedious access and data transfers to supercomputers. The new CUDA [2–4] technology gives computationally intensive applications access to the processing power of a GPGPU.

Apart from classical CFD, GPGPUs also play an important role in computational steering. The main idea of a computational steering environment is to provide a tool for engineers to design and check the quality of designs, desirably in real time. GPGPUs have rendered this possible, at least in conjunction with a well-chosen numerical method. Linxweiler lately presented a computational steering environment for CFD, which is based on an LBM implementation on GPGPU architecture [5,6]. It allows the user to interactively place and move buildings in a turbulent flow and to immediately see the influences on the flow patterns and vorticity. A competitive GPGPU implementation of a free surface algorithm could easily be included in this computational steering environment.

In this paper, we present the GPGPU implementation of a lattice Boltzmann (LB) based free surface algorithm. The free surface extension of the LB bulk scheme leads to additional non-local operations at the interface and a time-dependent computational domain. The resulting performance drop is evaluated and compared to the performance of the bulk scheme. The LB algorithm is verified and validated with a Poiseuille flow between plates. For the validation of the free-surface algorithm, we present simulations of a breaking-dam benchmark. Finally, two non-trivial examples show both the suitability and the performance of the LB implementation.

* Corresponding author. Tel.: +49 531 391 75 89; fax: +49 531 391 7599.
E-mail addresses: janssen@irmb.tu-bs.de (C. Janßen), kraft@irmb.tu-bs.de (M. Krafczyk).

0898-1221/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2011.03.016

2. The lattice Boltzmann method

The lattice Boltzmann method (LBM) has become an efficient approach for solving a variety of difficult CFD problems, including those in the field of multiphysics. LBM usually operates on a finite difference grid, is explicit in time and requires only next neighbour interaction. Hence, it is very suitable for the implementation on GPGPUs. Several authors accelerated their LBM computations on general-purpose graphics hardware, also concerning multiphysics, and even before the graphics vendors started to develop their software development kits (SDKs). The applications range from simulation of soap bubbles [7] to the simulation of miscible binary mixtures [8] and melting and flowing in a multiphase environment [9]. Recently, GPGPU clusters have been assembled for general-purpose computations [10] and LB simulations have been performed. With the development of SDKs, the programming style is not as close to the hardware as before. Toelke implemented two-dimensional and three-dimensional LB models on nVIDIA GPGPUs [11,12] and could show an efficiency gain up to two orders of magnitude compared to a single-core CPU code.

2.1. LBM basics

Opposite to classical CFD solvers which deal with the macroscopic Navier–Stokes equations, the LBM regards CFD problems on a microscopic scale. The primary variable of microscopic approaches is the particle distribution function $f(t, \boldsymbol{x}, \boldsymbol{\xi})$, which specifies the probability to encounter a particle at position $\boldsymbol{x}$ at time t with velocity $\boldsymbol{\xi}$. The evolution of these distribution functions f is described by the Boltzmann equation (Ludwig Boltzmann, 1872):

$$\frac{\mathrm{D}f}{\mathrm{D}t} = \frac{\partial f(t, \boldsymbol{x}, \boldsymbol{\xi})}{\partial t} + \boldsymbol{\xi} \cdot \frac{\partial f(t, \boldsymbol{x}, \boldsymbol{\xi})}{\partial \boldsymbol{x}} = \Omega. \tag{2.1}$$

The left-hand side of this equation is an advection-type expression, while the *collision operator* Ω describes the interactions of particles on the microscopic scale. In order to obtain a model with reduced computational costs, the Boltzmann equation is discretized in the velocity space ξ. In this work, the D3Q19 model [13] is used, which introduces 19 discretized microscopic particle velocities $\boldsymbol{e}_i$. The resulting set of discrete Boltzmann equations

$$\frac{\mathrm{D}f_i}{\mathrm{D}t} = \frac{\partial f_i(t, \boldsymbol{x})}{\partial t} + \boldsymbol{e}_i \cdot \frac{\partial f_i(t, \boldsymbol{x})}{\partial \boldsymbol{x}} = \Omega_i \tag{2.2}$$

merely has to be discretized in space and time. A standard finite difference discretisation on a grid with $c = \Delta x/\Delta t = 1$ (grid spacing Δx, time stepping Δt) leads to the *lattice* Boltzmann equation,

$$f_i(t + \Delta t, \boldsymbol{x} + \boldsymbol{e}_i \Delta t) - f_i(t, \boldsymbol{x}) = \Omega_i. \tag{2.3}$$

Finally, Eq. (2.3) may be split up into a non-linear *collision*, which drives the particle distribution functions to equilibrium locally and a non-local but linear *propagation* step, where the post-collision particle distribution functions are advected to the neighbour nodes. The solutions of the lattice Boltzmann equation (Eq. (2.3)) satisfy the incompressible Navier–Stokes equations up to errors of $\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\mathrm{Ma}^2)$ [14]. The well-known macroscopic values for density fluctuation ρ and momentum $\rho_0 \boldsymbol{u}$ are the first two hydrodynamic moments of the particle distribution functions:

$$\rho = \sum_{i=0}^{18} f_i \quad \text{and} \quad \rho_0 \boldsymbol{u} = \sum_{i=0}^{18} \boldsymbol{e}_i f_i. \tag{2.4}$$

2.2. Collision operators

For modelling the interaction between particles, different collision operators Ω_i may be used. In the single relaxation time (SRT) model [15], the particle distribution functions are driven to an equilibrium state with a single relaxation rate. In the more advanced MRT model [16], the particle distribution functions are transformed into moment space, where they are relaxed with several different relaxation rates. This increases the stability and at the same time enables the development of more accurate boundary conditions [17]. The collision operator for MRT is defined as

$$\boldsymbol{\Omega} = \mathsf{M}^{-1} \cdot \mathsf{S} \cdot \left(\mathsf{M} \cdot \mathbf{f} - \mathbf{m}^{\mathrm{eq}}\right) \tag{2.5}$$

M denotes the transformation matrix from distribution functions to the orthogonal moment space, and m_i^{eq} are the equilibrium moments. $\mathsf{S} = s_{i,i}$ is the diagonal collision matrix, which contains the relaxation parameters. The parameters

$$s_{9,9} = s_{11,11} = s_{13,13} = s_{14,14} = s_{15,15} = -\frac{\Delta t}{\tau} = s_\omega \tag{2.6}$$

are related to the kinematic viscosity ν via the relaxation time τ as follows:

$$\tau = 3\frac{\nu}{c^2} + \frac{1}{2}\Delta t. \tag{2.7}$$

The remaining relaxation parameters of non-conserved moments can be tuned to improve stability [18]. While the optimal values for these parameters depend on the specific system under consideration (geometry, initial and boundary conditions), reasonable values are given in [16]. We choose a value of -1.0.

(a) No-slip. (b) Slip.

Fig. 2.1. Wall boundary conditions.

2.3. Smagorinsky LES

Free surface flows usually occur at very high Reynolds numbers in the turbulent regime. In order to capture turbulent structures in the flow, a large eddy model [19] is used. A spatial filter is applied to the velocity field, which should be fine enough that the large turbulent structures of the flow do not get filtered out. Hence, only the effect of the small sub-grid eddies on the large-scale flow structures has to be modelled. This is included in the model through an additional turbulent viscosity ν_{T}. In a Smagorinsky model ν_{T} depends on the strain rate:

$$\nu_{\mathrm{T}} = (C_S \Delta x)^2 \|\mathbf{S}\| \tag{2.8}$$

with Smagorinsky constant C_S and strain rate tensor $S_{\alpha\beta}$, which can be computed from the moments as

$$S_{\alpha\beta} = \frac{s_{xx}}{2c_s^2\rho}\left(c_s^2\rho\delta_{\alpha\beta} + \rho u_i u_j - P_{\alpha\beta}\right) = \frac{s_{xx}}{2c_s^2\rho} Q_{\alpha\beta} \tag{2.9}$$

with speed of sound c_s, Dirac delta function δ, density ρ, velocity u, and the second-order moments of the distribution functions P, which can be locally computed from $m_{9,11,13,14,15}$. From Eq. (2.9) and

$$\tau_{\mathrm{total}} = 3\frac{\nu_{\mathrm{total}}}{c^2} + \frac{1}{2}\Delta t = 3\frac{(\nu_0 + \nu_{\mathrm{T}})}{c^2} + \frac{1}{2}\Delta t, \tag{2.10}$$

a quadratic equation is obtained, which yields

$$\tau_t = \frac{1}{2}\left(\sqrt{\tau_0^2 + 18C_S^2\Delta x^2 Q} - \tau_0\right) \tag{2.11}$$

and a modified relaxation rate s_{xx} for the second-order moments $m_{9,11,13,14,15}$

$$s_{xx} = \frac{\Delta t}{\tau_{\mathrm{total}}} = \frac{\Delta t}{\tau_0 + \tau_t}. \tag{2.12}$$

2.4. Boundary conditions and volume forces

For no-slip and velocity boundary conditions, a simple bounce back scheme is used (Fig. 2.1(a)). The incoming missing particle distribution function f_I is reconstructed as

$$f_I^{t+1}(\boldsymbol{x}) = f_i^t(\boldsymbol{x}) + 2\rho_0 w_i \frac{\boldsymbol{e}_i \bar{\boldsymbol{u}}}{c_s^2} \tag{2.13}$$

where i is the inverse direction to I and $\bar{\boldsymbol{u}}$ denotes the prescribed boundary velocity [20]. The weighting factors w_i for the D3Q19 model [21] are defined as

$$w_0 = \frac{1}{3}, \qquad w_{1\ldots6} = \frac{1}{18} \quad \text{and} \quad w_{7\ldots18} = \frac{1}{36}. \tag{2.14}$$

The subgrid wall distance is not taken into account in this model, so that the scheme is only second-order accurate for boundaries which are located in the middle of two lattice nodes. At slip boundaries, the bounce forward scheme (Fig. 2.1(b)) assures that the momentum in tangential direction is not modified at the wall. The missing particle distribution function f_I is reconstructed as

$$f_I^{t+1}(\boldsymbol{x} + \boldsymbol{e}_i\boldsymbol{t}) = f_i^t(\boldsymbol{x}) \tag{2.15}$$

where i is the mirrored direction to I with $\boldsymbol{e}_i\boldsymbol{t} = \boldsymbol{e}_I\boldsymbol{t}$ and $\boldsymbol{e}_i\boldsymbol{n} = -\boldsymbol{e}_I\boldsymbol{n}$ for wall normal vector $\boldsymbol{n}$ and tangential vector $\boldsymbol{t}$.

At the free surface boundary, the anti bounce back rule [22] enforces the equality in fluid pressure and surrounding pressure p_B:

$$f_I^{t+1} = -f_i^t + f_I^{\mathrm{eq}}(\rho_B, \boldsymbol{u}(t_B, \boldsymbol{x}_B)) + f_i^{\mathrm{eq}}(\rho_B, \boldsymbol{u}(t_B, \boldsymbol{x}_B)) \tag{2.16}$$

where $f_{i,I}^{\mathrm{eq}}(\rho_B, \boldsymbol{u}(t_B, \boldsymbol{x}_B))$ are Maxwellian equilibrium distribution functions and ρ_B is related to the surrounding pressure by $\rho_B = p_B c_s^{-2}$.

Zero-gradient boundary conditions are realised by copying the values of the penultimate node to the boundary node:

$$f_i^{t+1}(\boldsymbol{x}_b) = f_i^{t+1}(\boldsymbol{x}_{b-1}). \tag{2.17}$$

Gravity and other volume forces $\boldsymbol{F}$ are added directly to the distribution functions f_i in every time step [23]:

$$\Delta f_i = 3\omega_i \rho \boldsymbol{e}_i \cdot \boldsymbol{F}. \tag{2.18}$$

2.5. Force evaluation

The force $\boldsymbol{F}$ acting on an obstacle in the flow results from the momentum of the particles hitting the boundary. It can be computed by balancing the particle momentum before and after hitting the boundary:

$$\boldsymbol{F} = \sum_{i \in \Gamma} \boldsymbol{F}_i = -\frac{V}{\Delta t} \boldsymbol{e}_i \left(f_i(t + \Delta t, \boldsymbol{x}) + f_{\bar{l}}(t, \boldsymbol{x})\right) \tag{2.19}$$

for all links i that are cut by the obstacle [24].

3. Free surface model

Basically free-surface flows are two-phase flows where high viscosity ratios and high density ratios between the two phases are present. The flow behaviour is dominated by the denser phase and the interface is allowed to move freely. If capillary forces are neglected, the simulation of the denser and more viscous phase is sufficient and the influence of the second (less dense) phase on the flow dynamics can then be represented by appropriate boundary conditions at the interface.

3.1. State of the art

Numerically, the free surface represents a moving boundary, which is allowed to move freely, but at the same time has to be kept sharp. A couple of approaches have been developed to use the LBM for free surface flow simulations. Gunstensen proposed the immiscible lattice Boltzmann (ILB) model, a multiphase model combined with an additional anti-diffusion sweep (recolouring step) which prevents the mixture of the two phases [25]. Ginzburg and Steiner modify this approach and neglect the second fluid phase [26]. In contrast to the underlying multiphase model, the LB calculation steps occur only on the nodes of one phase. Lallemand combines an Eulerian LBM for the flow field and a Lagrangian front-tracking method for the advection step [27]. In this hybrid method, the interface is captured through marker particles, which are then advected on the basis of a valid pressure and velocity field. We recently presented a hybrid scheme that discretises the advection equation in a classical macroscopic way, on the basis of valid pressure and velocity fields provided by the LBM [28]. Apart from the aforementioned methods, Koerner and Thuerey [22,29] combine LBM with a VOF method and a flux-based advection scheme. Their algorithm initially was developed for the simulation of metal foams, but is capable of handling free-surface flow simulations as well. Opposite to common VOF methods, the flux terms are expressed directly in terms of LBM distribution functions. The straightforward surface reconstruction and the time-explicit advection made us choose this free surface capturing scheme as the basis of our GPGPU implementation.

3.2. VOF interface capturing

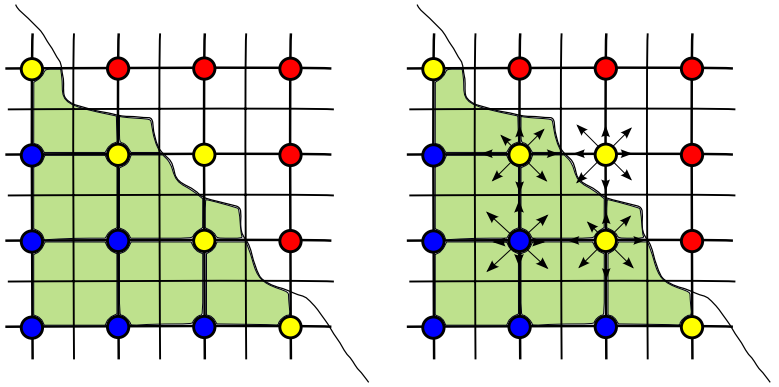
In a VOF interface capturing approach, the interface is captured via the *fill level* of a cell, which qualifies the amount of a cell which is filled with fluid:

$$\varepsilon = \frac{V_{\mathrm{fluid}}}{V_{\mathrm{cell}}}. \tag{3.1}$$

A fill level of 0.0 marks an empty cell in the inactive gas domain, a fill level of 1.0 corresponds to a filled cell inside the fluid domain. Fluid and gas cells are separated by a closed interface layer (Fig. 3.1(a)) with a fill level between 0.0 and 1.0. If the VOF control volume Ω_{cell} is assigned to one lattice node, yielding a cell volume $V_{\mathrm{cell}} = 1.0$, a simplified definition of the fill level is obtained:

$$\varepsilon = \frac{V_{\mathrm{fluid}}}{V_{\mathrm{cell}}} = \frac{m_{\mathrm{fluid}}}{\rho_{\mathrm{fluid}}} \tag{3.2}$$

where m_{fluid} and ρ_{fluid} refer to the mass content of the control volume Ω_{cell} and the fluid density, respectively. In order to calculate the evolution of the free surface in time, an additional advection equation has to be solved. In a weakly compressible



(a) Fluid, interface and gas nodes. (b) Flux between interface nodes.

Fig. 3.1. Fluid, interface and gas nodes.

approach, as the LBM, the VOF fill level ε is not conserved, so that a recourse to the continuity equation and the principle of conservation of mass is used to derive the advection algorithm:

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} = \frac{\partial \rho}{\partial t} + \nabla \left(\boldsymbol{v} \cdot \rho\right) = 0. \tag{3.3}$$

This equation is discretized with a classical finite volume method by integrating the equation over the control volume Ω_{cell} and applying the divergence theorem to the convective term. For the resulting surface integral, the flux term Φ_i is introduced, denoting the flux on the i-th face of the control volume. Finally, Eq. (3.3) yields

$$\frac{\partial m}{\partial t} + \sum_i \Phi_i = 0. \tag{3.4}$$

A further discretisation in time with an explicit Euler finite difference scheme leads to

$$m^{t+1} = m^t - \sum_i \Phi_i \cdot \Delta t \tag{3.5}$$

representing the evolution equation for the mass m in a VOF cell. So far, this derivation does not contain LB specifics and is the basis of nearly all VOF methods. However, the following Lattice Boltzmann advection scheme can be considered as a specialised, geometry-based VOF method using a mesoscopic advection model, in which the flux terms Φ_i between neighbouring cells are expressed in terms of particle distribution functions:

$$\Phi_i = \left[f_{\bar{l}}\left(\boldsymbol{x}, t\right) - f_l\left(\boldsymbol{x}, t\right)\right] \cdot A_i \tag{3.6}$$

with the two antiparallel particle distribution functions $f_{i,l}$ entering or leaving the corresponding cell. A_i denotes the wet area between two cells and is calculated on the basis of a simplified surface reconstruction. It can be estimated e.g. as the arithmetic mean of the fill level of two neighbouring cells:

$$A_i = \begin{cases} 1.0 & : \text{neighbour FLUID cell} \\ \dfrac{\varepsilon(\boldsymbol{x}, t) + \varepsilon(\boldsymbol{x} + \boldsymbol{e}_i, t)}{2} & : \text{neighbour INTERFACE cell} \\ 0.0 & : \text{neighbour GAS or SOLID cell.} \end{cases} \tag{3.7}$$

Opposite to higher-order schemes, the normal vector information is not considered. Hence, this approach does not reproduce all line (2D) or plane (3D) segments exactly and is of the first order in space. Once the flux terms have been evaluated, the new fill level of a cell can be calculated via

$$\varepsilon^{n+1} = \frac{m^{n+1}}{\rho^{n+1}} = \frac{\rho^n \varepsilon^n + \sum_i \Delta m_i}{\rho^{n+1}} \tag{3.8}$$

where $\rho^{n/n+1}$ is the fluid density at time step n resp. $n+1$ (Eq. (2.4)) and ε^n is the fill level at time step n [22,29].

3.3. Resulting algorithm for the node update

After the fill levels ε have been updated in all interface cells, the consistency of node state and new fill level has to be assured. In the advection step, cells with a fill level larger than 1.0 or lower than 0.0 can appear, as the interface evolves

in time. Consequently, cells that run empty change their state from interface to gas, and neighbouring fluid nodes have to become interface nodes. Analogous, the cells which have been filled up change their state from interface to fluid. The neighbour gas nodes, which were inactive before, switch their state to interface. These new interface nodes have to be initialised, as they do not contain any valid distribution functions. Therefor the macroscopic values of density and velocity (Eq. (2.4)) from neighbouring existing fluid nodes are interpolated:

$$\bar{\rho}(\boldsymbol{x}) = \sum_i w_i \rho(\boldsymbol{x} + \boldsymbol{e}_i) \quad \text{and} \quad \bar{\boldsymbol{v}}(\boldsymbol{x}) = \sum_i w_i \boldsymbol{v}(\boldsymbol{x} + \boldsymbol{e}_i). \tag{3.9}$$

Based on this information the particle distribution functions f are initialised with Maxwellian equilibrium distribution functions Eq. (3.11):

$$f_i = f_i^{\mathrm{eq}}(\bar{\rho}, \bar{\boldsymbol{v}}). \tag{3.10}$$

The equilibrium distribution functions tuned for incompressible flows are

$$f_i^{\mathrm{eq}} = w_i \left[\rho + \rho_0 \left(3\frac{\boldsymbol{e}_i \cdot \boldsymbol{u}}{c^2} + \frac{9}{2}\frac{(\boldsymbol{e}_i \cdot \boldsymbol{u})^2}{c^4} - \frac{3}{2}\frac{u^2}{c^2} \right) \right] \tag{3.11}$$

where ρ_0 is the reference density and w_i are weighting factors according to Eq. (2.14). In the future, a local, LB-specific, Poisson-type iteration [30] might be used for the improvement of the non-equilibrium part of the distribution functions. The resulting overall algorithm is given in Alg. 1.

Algorithm 1 Update interface algorithm

```
collide, add forcing, propagate
apply boundary conditions
—
{update the interface}
if cell type == interface then
  determine wet area for all lattice directions (Equation 3.7)
  calculate mass flux and evaluate new fill level (Equation 3.8)
  —
  if new fill level ε_i^{t+1} < 0.0 then
    convert cell to gas cell
    check and – if necessary – convert neighbour lattice nodes
  end if
  if new fill level ε_i^{t+1} > 1.0 then
    convert cell to fluid cell
    check and – if necessary – convert neighbour lattice nodes
  end if
end if
initialise new interface nodes (Equation 3.10)
```

4. Implementation

For the implementation of the algorithm on GPGPU hardware, we use the nVIDIA CUDA Toolkit, which is an entire software development solution for programming CUDA-enabled nVIDIA GPGPUs. First the mapping of the lattice nodes to the parallel architecture is shown. After that, details on boundary conditions and the free surface part are given.

4.1. Topology

The GPGPU is a computing device with a tremendous amount of cores, 240 on the nVIDIA Tesla c1060, which execute a number of threads. To arrange these threads, the CUDA toolkit offers a two-level parallelism. First, all threads are grouped in one *thread block*. In a thread block, extremely fast shared memory is available among the threads and the threads can be synchronised. Each thread is identified by its three-dimensional thread index, which is the position in the thread block. To exploit the hardware efficiently, the total number of threads per block should be in the range of 64–512. This number can be smaller due to the amount of local and shared memory which is available on the particular GPGPU. Secondly, the thread blocks are bundled in the *grid*. Opposite to threads in one and the same thread block, threads in different thread blocks can only communicate via the device memory and a synchronisation is not possible. Blocks are identified by their two-dimensional block index, namely the position in the grid. Further details on the thread processing, the grouping in warps and the distribution among the GPGPU multiprocessors can be found in [31].

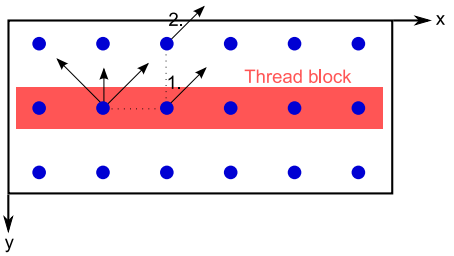


Fig. 4.1. Grid mapping and propagation via shared memory.

4.2. Grid mapping

The main design element in the GPGPU implementation of a numerical method is the mapping of the numerical grid to the computational hardware, i.e. in our case the mapping of lattice nodes to the grid, blocks and threads. Some restrictions for the memory access pattern have to be taken into account in order to achieve maximum performance. In particular, the thread k must access the k-th word in a memory segment aligned to 16 times `sizeof(float)`. When these access requirements are met, global memory accesses are coalesced by the device in only one single transaction. However, if this pattern is violated, the memory accesses cannot be coalesced and the performance drops substantially. Although the recently released computing cards with a compute capability larger than 1.1 offer a higher flexibility, we decided to keep the below-mentioned optimised memory access pattern, for the sake of backward compatibility of the code.

In our grid mapping we assign one single lattice node to one CUDA thread. The memory is allocated as a one-dimensional array, and the memory index is calculated via `k = nx*(ny*z + y) + x` for a node at position (x, y, z) and a total of $nx \times ny \times nz$ nodes. Consequently, the PDFs propagating in x-direction, are copied to a memory position with a `sizeof(float) = 4 Byte` shift. Due to the above-mentioned restrictions, this access to neighbouring memory positions is very slow. To solve this problem, Toelke proposed to propagate those particle distribution functions via the shared memory which is available in a one-thread block and does not suffer the restrictions for the memory access pattern [12]. Consequently, all nodes along one line in x-direction have to be gathered in one single thread block. The remaining y-and z-dimensions are mapped to the grid. The coordinates of a node can be determined via `x = threadId.x`, `y = blockIdx.x` and `z = blockIdx.y` The shared memory also is used for the propagation step in the diagonal lattice directions, as shown in Fig. 4.1. The PDFs are first advected to an intermediate grid location via the shared memory inside a thread block (1.0), before the advection to the final grid location takes place (2.0). If nx is a multiple of 16, the latter memory access now is in accordance with the strict memory access limitations of the GPU.

The total amount of shared memory is currently limited to 16 kB. In the D3Q19 model, ten PDFs propagate in the positive or negative x direction, so that 40 Bytes of shared memory per LB node are needed, which corresponds to a maximum number of 400 nodes in one thread block. If the x dimension of the computational domain exceeds this maximum number of threads, the domain also has to be partitioned in the x direction. The CUDA *grid*, which groups the thread blocks, is limited to two dimensions and cannot deal with two consecutively aligned thread blocks in a third direction. Hence, this feature has to be included in the kernels explicitly. The block index in the x direction is passed to the kernels and the new x coordinate is calculated manually as `x = threadId.x + blockIndexX*nodes_per_threadBlock`.

The dimension of the grid and the number of threads is passed to the kernel, and CUDA manages the exact distribution of tasks among the multiprocessors and cores. The collision and propagation kernel bundles the collision and propagation steps (Eq. (2.3)) in order to save additional memory accesses. The distribution functions are fetched from the main device memory, the collision takes place, and the post-collision distribution functions are written to the neighbouring memory positions. For the distribution functions propagating in the x direction, an intermediate memory transaction in the shared memory is used.

4.3. Boundary conditions

From the algorithmic point of view, boundary conditions disturb the original homogeneous bulk algorithm, as they require additional operations on a subset of nodes. In general, a unique LB kernel for all lattice nodes is preferable on a data- and thread-parallel system for an optimal load balancing.

4.3.1. No-Slip, velocity (included in collision and propagation)

The no-slip and velocity boundary conditions can be incorporated in the LB collision and propagation kernel. Instead of the collision step, the boundary nodes simply reflect the incoming particle distribution function and – if necessary – add the contribution of the boundary velocity $\bar{\boldsymbol{u}}$. After that, the unmodified propagation step takes place, which advects the reflected distribution functions to the corresponding neighbour, as shown in Fig. 4.2. Ghost layers surround the whole computational domain, so that all 18 particle distribution functions can safely be advected to the neighbouring nodes, even at the domain boundary.

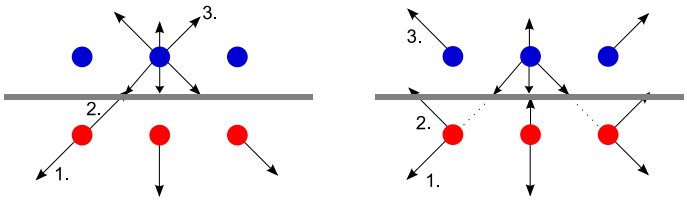


Fig. 4.2. Implementation of boundary conditions.

4.3.2. Slip, free surface, zero-gradient

Opposite to that, the bounce-forward scheme for slip boundary conditions violates the common propagation pattern. This time, the distribution functions are not reflected to the donating lattice node but forwarded to the neighbouring lattice node. Consequently, the BC does not fit into the common propagation pattern and requires an additional kernel for the advection. The solid nodes at the boundary receive PDFs from neighbouring fluid nodes during propagation. After collision and propagation, a second kernel is executed and shifts the PDFs to the correct location, see Fig. 4.2

The same holds for the extrapolation boundary conditions. A separate kernel performs a copy operation, where the complete set of distribution functions of the next to last fluid node is copied to the last one. Opposite to the previous boundary conditions, the free surface boundary condition on interface nodes additionally needs to read the node state at neighbouring memory locations. If – and only if – the neighbouring node is a gas state, the pressure boundary condition has to be applied.

4.4. Force evaluation

The evaluation of the forces on obstacles in the flow (Eq. (2.19)) basically consists of a loop over all lattice nodes, summing up the nodal contribution to the force vector. For performance reasons, and since a continuous force evaluation in every time step is preferable, the force is evaluated directly on the GPGPU. Unfortunately, these all-reduce operations in thread-parallel systems always ask for a careful treatment to avoid race conditions among the threads. A first kernel computes the contribution of one thread block to the total force. Each single node evaluates Eq. (2.19), and the first thread in the whole thread block is responsible for the summation: $\tilde{F}_B = \sum_{nx} F_i$. A thread–global synchronisation point guarantees that all threads finish their force calculation before the summation. The resulting force of one thread block is stored in the global device memory, resulting in a matrix of $ny \times nz$ entries. In the following, these sub-forces are accumulated by two additional loops: $\boldsymbol{F} = \sum_{ny} \sum_{nz} \tilde{F}_B$.

4.5. Free surface algorithm

The free surface part of the algorithm is hard to optimise for GPGPU hardware, precisely because it is mainly non-local, the computational domain varies in time and the advection steps apply to the interface nodes only. The latter is considered via a supplementary flag field, which marks the interface nodes and is queried in the following interface-update kernels. Opposite to the collision-propagation routines, no thread synchronisation is needed, because all kernels update local variables only and do not introduce non-local memory write accesses.

The calculation of new fill levels is straightforward, although the evaluation of Eq. (3.8) requires neighbour information about the node state and the fill level. If the computational domain evolves in time, nodes change their state, e.g. from interface to fluid. In order to ensure a closed interface layer, all surrounding nodes have to be checked and – if necessary – changed from gas to interface state. This non-local *write* operation has to be modified, as one cannot guarantee exclusive memory access for one thread or thread block. Hence, the change of nodal states is transferred from a non-local write to a non-local read operation: all gas nodes check if they are in the vicinity of an interface node, and – in case of a positive outcome – if this interface node changed its state to fluid. In analogy, fluid nodes near emptying interface nodes execute a similar algorithm.

The initialisation of new fluid nodes has to be split up into two parts. First, a valid set of particle distribution functions is set for all new fluid nodes, according to Eq. (3.10). For the interpolation of macroscopic values only existing neighbouring fluid nodes may be used. Thus, a second kernel switches the state of the new fluid nodes from gas to fluid once the initialisation procedure is finished. Moreover, as the interface evolves and new node states are set, lost interface cells might occur in underresolved areas. These interface cells without any fluid neighbours have to be detected and condensed.

The resulting algorithm (Alg. 2) consists of two independent parts: the flow field is calculated, then the advection equation is solved.

5. Results

The following simulations are carried out on nVidia hardware, namely on the nVIDIA GTX 275 and Tesla C1060 (Table 5.1). For our enriched D3Q19 model, 165 Bytes per lattice node have to be allocated, which yields maximum numbers of 6.5 resp. 26 E6 lattice nodes.

Algorithm 2 Time loop

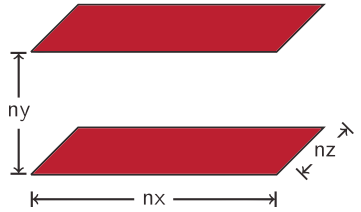
```
initialise
copy data to device memory
for t = 0 to number of calc steps do
  {LB part}
  collision, propagation
  slip boundary condition
  free surface boundary condition
  periodic boundary condition
  calculate force on obstacles

  {Free surface part}
  calculate new fill level (Equation 3.8)
  set consistent node states
  initialise new fluid nodes (Equation 3.10)
  condense lost interface cells

  swap pointers for f, ftemp and fill, filltemp
end for
```

Table 5.1
nVIDIA Hardware.

Card	Device memory (GB)	Cores	Clock (GHz)	Nodes (million)
nVIDIA GTX 275	1	240	1.4	6.5
nVIDIA Tesla c1060	4	240	1.3	26

Parameter	Value
Grid	see Tab. 5.2
Re	100
Ma	0.017
BC x	periodic
BC y	periodic
BC z	no-slip

(a) Parameters.



(b) Geometry.

Fig. 5.1. Poiseuille flow between plates.

In order to obtain comparable results, the dimensionless parameters of real world experiments or application and numerical simulation have to match. The most important dimensionless quantity in fluid dynamics is the *Reynolds number*. It indicates the relation of inertia terms to viscous terms and is defined as $\mathrm{Re} = \frac{v_\infty \cdot L_0}{\nu}$, with velocity v_∞, characteristic length L_0 and kinematic viscosity ν. For free surface flows, the Froude number Fr is the second fundamental dimensionless parameter. It is defined as the ratio of inertial to gravitational forces, namely $\mathrm{Fr} = \frac{v}{\sqrt{gh}}$ with velocity v, characteristic length h and gravity g. It characterises the nature of the flow to be super- or subcritical. In subcritical flows ($\mathrm{Fr} < 1$), the flow velocity is lower than the wave velocity $c = \sqrt{gh}$, in supercritical flows ($\mathrm{Fr} > 1$) it is higher. The equivalent in gas dynamics is the Mach number.

5.1. Verification and performance

In this section verification test cases for the flow solver and the free surface part are shown. Initially, the validity and performance of the D3Q19 bulk scheme is analysed in order to have a reference for the following free surface performance benchmarks.

5.1.1. D3Q19 bulk scheme

The performance and accuracy of the fluid solver without free surface extension is demonstrated with a Poiseuille flow between plates on several different grid configurations. In this straightforward problem, the fluid is moving laterally between two plates with infinite length and width. As the grid is refined, the viscosity and the body force are adjusted to match the fixed Reynolds and Mach numbers given in Fig. 5.1(a).

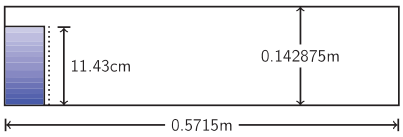
The analytical solution for the velocity and pressure profiles is known and can be used to check the accuracy of the solver. The flow is driven by a pressure gradient and retarded by viscous drag along both plates. Demanding balance of these forces

Table 5.2
Performance for the Poiseuille flow problem (MNUPS).

ny, *nz*	*nx*				Rel. error ε_{rel}
	32	64	128	256	
32 × 32	288	337	249	346	7.9321E−04
64 × 64	289	329	346	353	2.2706E−04
96 × 96	293	358	328	351	1.5168E−04
128 × 128	320	276	325	330	1.1534E−04
192 × 192	260	358	288	327	9.0002E−05
256 × 256	318	358	346	356	9.6668E−06

Param.	Value
Re	103483
Fr	2.418
u_{max}	1.71 [-]
Domain	0.5715m x $(0.142875\text{m})^2$
	(22.5in x 5.625in^2)
Water column	5.715cm x 11.43cm
	(2.25in x 4.5in)
Lattice	128x32x32
	256x64x64
	384x96x96
	512x128x128

(a) Parameters.



(b) Geometry.

Fig. 5.2. Dam break setup (2D cut).

leads to the following solution for the flow velocity u_x:

$$u_x(z) = \frac{z^2 - L_z^2}{2\nu} \frac{\mathrm{d}p}{\mathrm{d}x}. \tag{5.1}$$

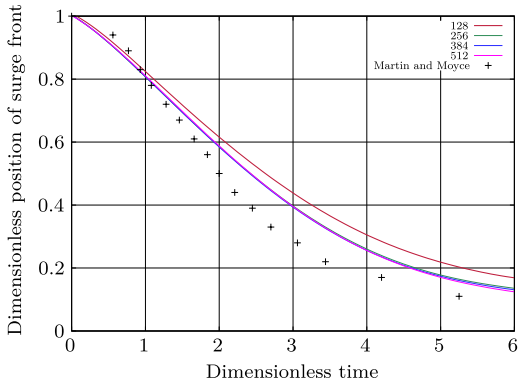
The resulting performance on a Tesla C1060 and the relative error $\varepsilon_{\text{rel}} = \frac{|u_{\text{actual}} - u_{\text{target}}|}{u_{\text{target}}}$ for the maximum velocity u_{max} in the channel centre at $z = 0.5L_z$ are given in Table 5.2. *MNUPS* corresponds to Million Node Updates Per Second. We can see a performance maximum of 358 MNUPS for a grid resolution of 64 × 96 × 96 nodes. The average performance yields approximately 320 MNUPS. Concerning the error norm, convergence can clearly be observed.

5.1.2. Breaking dam

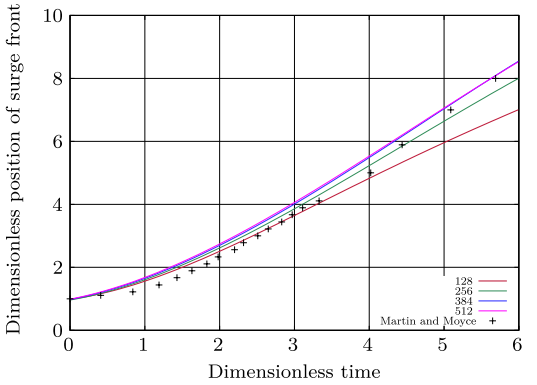
The classic breaking dam benchmark [32] is used to demonstrate that the model is able to cope with real-world fluid simulations. A water column in a channel is constrained by a waxed paper diaphragm. Once the waxed paper is freed, the water column collapses. The main setup is shown in Fig. 5.2. We use no-slip boundary conditions for all six walls. Martin and Moyce [32] determined a maximum dimensionless velocity of $U = 1.71$, which corresponds to Re $\approx$ 103483 and Fr $\approx$ 2.418. Lattice viscosity ν and forcing g are adjusted to match the given dimensionless numbers. The calculations are stopped when the surge front reaches the back wall of the container.

During the simulation, the position of the surge front and the height of the collapsing water column are observed. In Fig. 5.3(a) and (b) the numerical results for four different grid resolutions are compared to the experimental reference data from [32]. Good agreement and a convergent behaviour can be observed for both the surge front and the collapsing column. Our numerical surge front (Fig. 5.3(b)) evolves slightly faster than the one in the experiment. This might be due to the fact that the delay owing to the triggering (a thin diaphragm which is released by an electric current) is not modelled in our numerical wave tank. This effect also was observed by various other groups, e.g. [33–35]. Concerning the height of the collapsing water column, the initial high discrepancy decreases for higher resolutions (Fig. 5.3(a)). The remaining difference might be due to boundary conditions, as the water height is evaluated near the back wall of the channel.

Apart from the numerical quality of the free surface model, the performance is of great interest. In Fig. 5.4(a), the performance is plotted over the dimensionless time. At first glance, it can be seen that the overall node update rate varies with time. At the beginning of the simulation, a maximum performance of approximately 160 MNUPS for the finest grid can be measured, which increases up to 200 MNUPS as the water column collapses. The similarities of Fig. 5.4(a) and Fig. 5.3(b) are not a coincidence, when thinking of the way the GPGPU deals with threads and blocks and warps. Basically, if a kernel which is executed on the GPGPU contains branching (as in if-statements), the whole kernel is executed multiple times, once for each possible branch. Moreover, threads are executed in warps of 32 threads each, which are aligned along the *x*-axis in our grid mapping. Hence, if one warp of 32 threads contains a nonzero number of interface nodes, the whole warp executes the kernel twice: once for the normal fluid nodes (and the classical collision and propagation routines) and once for the interface nodes and the additional update interface algorithms. As a consequence, the warps only contain a small number of interface nodes due to the steep dam/wave front during the initial stages of the breaking dam simulation. As the water

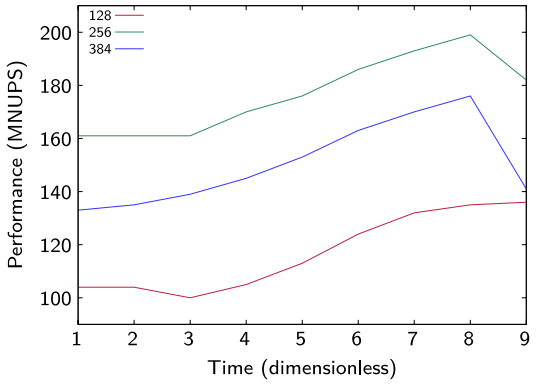


(a) Position of the water column top.

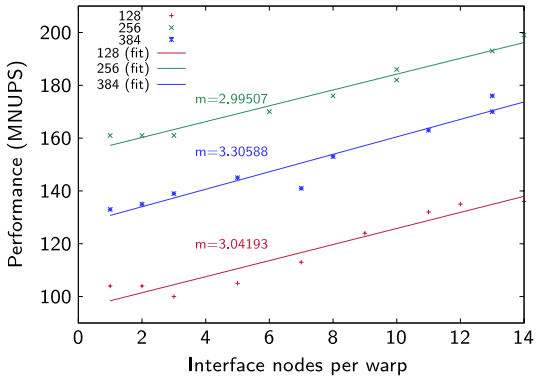


(b) Position of the surge front.

Fig. 5.3. Breaking dam, comparison of numerical and experimental results.



(a) Performance over time.



(b) Performance over number of interface nodes per warp.

Fig. 5.4. Breaking dam, performance.

column collapses, the fluid is spread out, trying to establish a flat water surface, which would be in perfect alignment with the numerical grid mapping. The warps would be fully occupied with interface nodes. In practise, the optimal number of interface nodes (32) will not be reached due to local fluctuations of the water height. Fig. 5.4(b) depicts the relation of the number of interface nodes per warp to the overall performance of the algorithm, and a nearly linear dependency can be seen. Correspondingly, as the surge front impacts the right wall of the container at time step $t = 10$ in Fig. 5.4(a), the performance drops immediately, as the average number of interface nodes per warp drops.

5.2. Applications

After this basic validation we apply the free surface algorithm to two generic applications in the field of civil and structural engineering.

5.2.1. Hydraulic jump

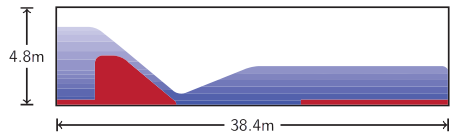
As a first application, the flow past a weir is examined (Fig. 5.5). We apply a velocity boundary condition on the left and a zero-gradient boundary condition on the right wall. No slip BCs are used at the bottom, slip BCs at the front and back wall of the domain. The parameters for our simulation are chosen in such a way that the subcritical inflow switches to a supercritical state while or shortly after passing the weir and leaves the domain again in a subcritical state.

Such a properly designed hydraulic jump is used in civil engineering as a mean to dissipate energy and reduce erosion effects in the river bed. The corresponding water heights h_{1-3} (see Fig. 5.7(c)) to generate such a scenario can be estimated with the help of reduced one-dimensional inviscid equations. For the water height h_2 on the back of the weir, the continuity equation and Bernoulli's equation yield a water level of

$$h_2 = \frac{1}{4} h_1 \mathrm{Fr}_1^2 \left(\sqrt{1 + 8\mathrm{Fr}_1^{-2}} \right). \tag{5.2}$$

Parameter	Value
Domain	38.40m × 6.40m × 4.80m
q	2.8 m^2/s
v_1	0.70 m/s
g	9.81 m/s^2
ν	1 · 10E-6 m^2/s
Lattice	192x32x24 ($\Delta x = 0.2m$)
	384x64x48 ($\Delta x = 0.1m$)
	576x96x72 ($\Delta x = 0.067m$)
	768x128x96 ($\Delta x = 0.05m$)

(a) Parameters.



(b) Geometry (2D cut).

Fig. 5.5. Flow past a weir.

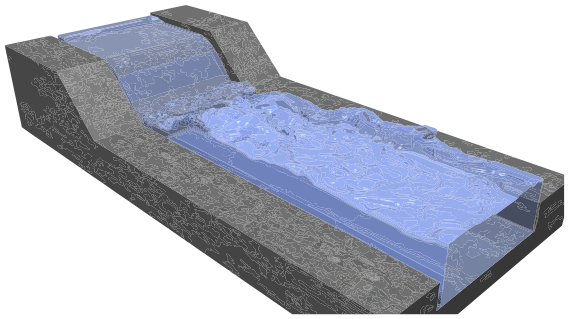


Fig. 5.6. Hydraulic jump, $t = 60$ s.

The water height h_3 in the outflow region can be calculated demanding conservation of mass and momentum, yielding

$$h_3 = \frac{1}{2} h_2 \left(\sqrt{8\mathrm{Fr}_1^2 + 1} - 1 \right). \tag{5.3}$$

Energy is not conserved between Sections 2 and 3 due to the high turbulent dissipation. These expressions are widely used in the context of hydraulic jumps over plane beds, and they are also valid along streamlines in more complex flows, so that they still serve as a good estimation for the target water heights in the flow past a weir. The corresponding values of Froude number and velocity in the Sections 1–3 of the weir are given in Fig. 5.7(b). The Froude number Fr_2 behind the weir also determines the stability of the hydraulic jump, see e.g. [36]. For Froude numbers around 4.5 we expect an oscillating and wavy hydraulic jump. In order to guarantee a minimum flux past the weir, the weir dimensions have to be set correctly. Following Poleni, the average flux q (per unit width) past a weir can be calculated via

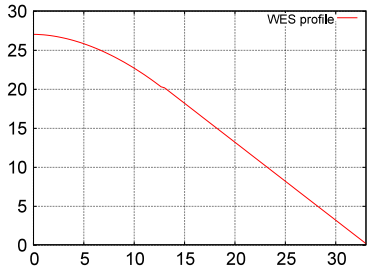
$$q = \frac{2}{3} \mu \sqrt{2g} h_w^{1.5} \tag{5.4}$$

with gravity g, water height h_w above the top of the weir and a shape parameter μ which accounts for the weir shape. The weir geometry has to be chosen carefully, to guarantee the required performance and to avoid negative pressures and the risk of cavitation. A good hydraulic weir profile with a shape parameter $\mu_{\mathrm{WES}} \approx 0.7$ is the *WES profile*, which is described by the following relation:

$$z_{\mathrm{weir}} = \begin{cases} x^{1.85} / (2h_d^{0.85}) & : x < x_t \\ \tan(\alpha)\, x & : x > x_t, \end{cases} \tag{5.5}$$

with $x_t = h_w \cdot 1.0961 \cdot \tan(\alpha)^{1.1765}$ and the weir inclination α. With the given incoming flux $q = 2.8$ m^2/s, Eq. (5.4) yields a water level above the weir top of $h_w \approx 1.20$ m (Fig. 5.7(a)) and hence a maximum weir height $h_{\mathrm{weir}} = h_1 - h_w = 2.80$ m. A step is installed in the rear part of the domain prevents the hydraulic jump from moving downstream and leaving the domain. Similar steps are used in practise to stabilise hydraulic jumps.

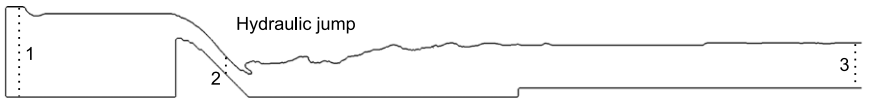
A snapshot of the simulation after $t = 60$ s is shown in Fig. 5.6. The gauge heights and flow velocities in Sections 2 and 3 match the estimated values, see Fig. 5.7(b). The gauge height on the back of the weir is higher than predicted. Note that the reference data is obtained from Bernoulli's equation, which is only valid along streamlines in inviscid flows and does, in addition, not consider three- dimensional turbulent effects. Hence, the reference data only can serve as an approximate value. Apart from that, the over-prediction of the water height h_2 might be due to the no-slip boundary condition on the weir surface and to the low resolution in this high-speed area of the flow. Grid refinement slightly improves the value of water height probe 2. In Fig. 5.7(d), the x-component of the flow velocity in the hydraulic jump is shown. The maximum flow velocity does match the predicted value of $v_2 = 8.51$ m/s, and the expected flow behaviour in the hydraulic jump



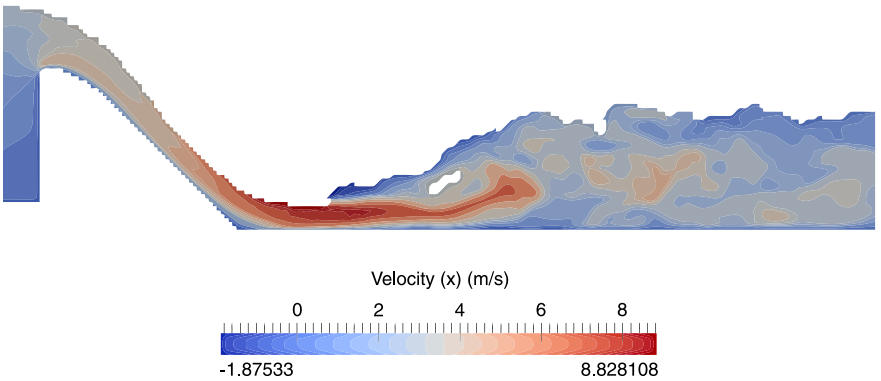
(a) Shape of WES profile.

Section	1	2	3
Fr	0.11	4.74	0.31
v	0.70	8.51	1.37
h_{theory}	4.0	0.33	2.05
h_{192}	4.0	0.9	2.00
h_{384}	4.0	0.78	2.00
h_{576}	4.0	0.78	1.93
h_{768}	4.0	0.69	1.90

(b) Parameters in sec. 1–3.



(c) Hydraulic jump (slice) with location of the sections 1–3.



(d) Velocity profile (slice, $nx = 576$).

Fig. 5.7. Results for the weir test case.

can be observed, including turbulent structures and local backflow. Moreover, the importance of the wall effects and the low-velocity region at the transition between the weir and the river bed can be seen. Finally, the Smagorinsky LES model tends to overestimate the turbulent effects in the near-wall region, which should be tackled by the use of dynamic Smagorinsky models or an LES WALE approach in future work.

For this simulation, we obtain an average performance of 55 MNUPS on a GTX 275 for the coarse grids ($\Delta x = 0.1$ m and a time step of $\Delta t = 1.4 \cdot 10^{-3}$ s). For the high-resolution simulation with a grid spacing of $\Delta x = 0.05$ m and a total number of 9.5 million nodes, a modern C1060 card is used. The average performance for the 60s-simulation yields 130 MNUPS.

5.2.2. Wave impact on cylinder

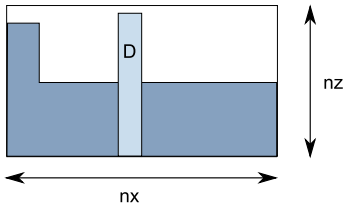
Finally, we apply the algorithm to a classical benchmark in the field of civil engineering and examine the impact of a breaking wave on a cylinder in the flow. Several experiments have been carried out to estimate the resulting slamming force on lean structures [37]. The test setup is shown in Fig. 5.8, including the initial shape of the water column to generate a wave-similar shape. The resulting force is given in Fig. 5.9. The force peak can clearly be observed and qualitatively corresponds to experimental data by [37]. Further examinations are mandatory, especially with a more sophisticated wave generator. Realistic wave profiles can either be generated by a piston wave maker or by a hybrid method, where the initial flow field is initialised with the results of a potential flow calculation [1].

6. Conclusions and outlook

We presented the GPGPU implementation of an existing LBM free surface scheme. The non-local operations of the free surface extension lead to a performance drop, compared to the stand-alone LB scheme. Nevertheless, the numerical wave tank ends up with approximately 30%–40% of the performance of the bulk scheme, which is still at least one order of magnitude faster than comparable CPU implementations. For the simulation of the flow past a weir with a total of 1.2 million lattice nodes and a corresponding time step of $1 \cdot 1.4\,\mathrm{E} - 3$ s, we achieved a performance of 55 MNUPS. Consequently, the calculation of 1 s of real world fluid behaviour lasts only 10 s, locally on one single GPGPU in an ordinary workstation.

Parameter	Value
Domain	64m x 8m x 8m
g	9.81 m/s^2
ν	$1 \cdot$ 10E-6 m^2/s
D_{water}	4.25m
Lattice	512x64x64

(a) Parameters.



(b) Geometry (2D cut).

Fig. 5.8. Wave tank setup.

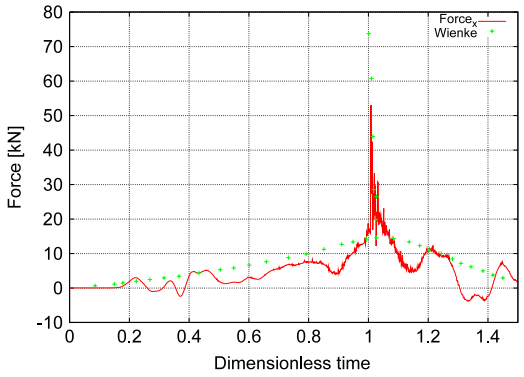


Fig. 5.9. Force on cylinder.

Apart from the high performance of the kernel, we have shown that the free surface implementation is in general suitable for the simulation of typical free surface flow problems in civil engineering. Both the flow past a weir and the wave impact on a cylinder were successfully simulated.

The main limitations of the implementation are the strict requirements for the test case dimensions that have to be fulfilled in order to gain high performance. The memory access pattern requires the domain length (i.e. the number of threads per thread block) to be a multiple of 16. Moreover, the total number of thread blocks should be a multiple of the number of multiprocessors which are available on the GPU. In addition, the maximum number of nodes is currently limited by device memory (1 resp. 4 GB only). Hence, in the future, a multi-GPGPU implementation has to be addressed, and the algorithm will be embedded in a computational steering environment.

References

[1] C. Janßen, S.T. Grilli, M. Krafczyk, Modeling of wave breaking and wave-structure interactions by coupling of fully nonlinear potential flow and lattice-Boltzmann models, in: Proceedings of the 20th Offshore and Polar Engineering Conference (ISOPE10, Beijing, China, June 20–25, 2010), pp. 686–693. Intl. Society of Offshore and Polar Engng., 2010.
[2] NVIDIA. URL: http://www.nvidia.com/object/cuda_home_new.html.
[3] D. Kirk, W.-M.W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, Morgan Kaufmann Publishers Inc., 2010.
[4] J. Sanders, E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 2010.
[5] J. Linxweiler, J. Toelke, M. Krafczyk, Applying modern soft–and hardware technologies for computational steering approaches in computational fluid dynamics, in: International Conference on Cyberworlds, 2007.
[6] J. Linxweiler, M. Krafczyk, J. Tölke, Highly interactive computational steering for coupled 3D flow problems utilizing multiple GPUs, Computing & Visualization in Science 13 (7) (2011) 299–314.
[7] X. Wei, Y. Zhao, Z. Fan, W. Li, F. Qiu, S. Yoakum-Stover, A. Kaufman, Lattice-based flow field modeling, IEEE Transactions on Visualization and Computer Graphics 10 (6) (2004) 719–729.
[8] H. Zhu, X. Liu, Y. Lui, E. Wu, Simulation of miscible binary mixtures based on lattice Boltzmann method, Computer Animation and Virtual Worlds 17 (2006) 403–410.
[9] Y. Zhao, et al., Melting and flowing in multiphase environments, Computers & Graphics 30 (4) (2006) 519–528.
[10] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover, GPU cluster for high performance computing, in: Proceedings of ACM/IEEE Supercomputing Conference, 2004, pp. 47–59.
[11] J. Tölke, M. Krafczyk, Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by nVidia, Computing and Visualization in Science 1 (2008) 29–39.
[12] J. Tölke, M. Krafczyk, Teraflop computing on a desktop PC with GPUs for 3D CFD, International Journal of Computational Fluid Dynamics 22 (2008) 443–456.
[13] Y.H. Quian, D. d'Humieres, P. Lallemand, Lattice BGK models for Navier Stokes equations, Europhysics Letters 17 (1992) 479–484.
[14] M. Junk, Z. Yang, Pressure boundary condition for the lattice Boltzmann method, Computers & Mathematics with Applications 58 (5) (2009) 922–929.
[15] P.L. Bhatnagar, E.P. Gross, M. Krook, A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems, Physical Review 94 (3) (1954) 511–525.
[16] D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, L.-S. Luo, Multiple relaxation-time lattice Boltzmann models in three-dimensions, Royal Society of London Philosophical Transactions Series A 360 (2002) 437–451.
[17] I. Ginzburg, D. D'Humieres, Multireflection boundary conditions for lattice Boltzmann models, Physical Review E 68 (6) (2003) 066614.1–066614.30.

[18] P. Lallemand, L.-S. Luo, Theory of the lattice Boltzmann method: dispersion, dissipation, isotropy, Galilean invariance, and stability, Physical Review E 61 (2000) 6546–6562.
[19] M. Krafczyk, J. Tölke, L.-S. Luo, Large-eddy simulations with a multiple-relaxation-time LBE model, International Journal of Modern Physics B 17 (2003) 33–39.
[20] M. Bouzidi, M. Firdaouss, P. Lallemand, Momentum transfer of a lattice-Boltzmann fluid with boundaries, Physics of Fluids 13 (2001) 3452–3459.
[21] X. He, L.-S. Luo, Lattice Boltzmann model for the incompressible Navier–Stokes equation, Journal of Statistical Physics 88 (1997) 927–944.
[22] C. Körner, M. Thies, T. Hofmann, N. Thürey, U. Rüde, Lattice Boltzmann model for free surface flow for modeling foaming, Journal of Statistical Physics 121 (1–2) (2005) 179–196. 18.
[23] Z. Guo, C. Zheng, B. Shi, Discrete lattice effects on the forcing term in the lattice Boltzmann method, Physical Review E 65 (4) (2002) 046308.1–046308.6.
[24] N. Nguyen, A. Ladd, Sedimentation of hard-sphere suspensions at low Reynolds number, Journal of Fluid Mechanics 535 (2004) 73–104.
[25] A.K. Gunstensen, D.H. Rothman, S. Zaleski, G. Zanetti, Lattice Boltzmann model of immiscible fluids, Physical Review A 43 (8) (1991) 4320–4327.
[26] I. Ginzburg, K. Steiner, Lattice Boltzmann model for free-surface flow and its application to filling process in casting, Journal of Computational Physics 185 (1) (2003) 61–99.
[27] P. Lallemand, L.-S. Luo, Y. Peng, A lattice Boltzmann front-tracking method for interface dynamics with surface tension in two dimensions, Journal of Computational Physics 226 (2) (2007) 1367–1384.
[28] C. Janßen, M. Krafczyk, A lattice Boltzmann approach for free-surface-flow simulations on non-uniform block-structured grids, Computers and Mathematics with Applications (2009).
[29] N. Thürey, U. Rüde, Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids, Computing and Visualization in Science (2008).
[30] R. Mei, L.-S. Luo, P. Lallemand, D. d'Humieres, Consistent initial conditions for lattice Boltzmann simulations, Computers & Fluids 35 (8–9) (2006) 855–862.
[31] nVIDIA, NVIDIA CUDA Programming Guide, 2010. http://www.nvidia.com/object/cuda_develop.html.
[32] J.C. Martin, W.J. Moyce, Part IV. An experimental study of the collapse of liquid columns on a rigid horizontal plane, Royal Society of London Philosophical Transactions Series A 244 (1952) 312–324.
[33] J. Sauer, Instationär kavitierende strömungen - ein neues modell, basierend auf front capturing (VOF) und blasendynamik, Ph.D. Thesis, Universität Karlsruhe, 2000.
[34] A. Kölke, Modellierung und diskretisierung bewegter diskontinuitäten in randgekoppelten mehrfeldsystemen, Ph.D. Thesis, Technische Universität Braunschweig, 2005.
[35] A. Salih, S.G. Moulic, A level set formulation for the numerical simulation of impact of surge fronts, in: SADHANA—Academy Proceedings in Engineering Sciences, vol. 31, 2006, pp. 697–707.
[36] V. te Chow, Open-Channel Hydraulics, MC Graw-Hill, 1959.
[37] J. Wienke, Druckschlagbelastung auf schlanke zylindrische Bauwerke durch brechende Wellen, Ph.D. Thesis, TU Braunschweig, 2001.