Interactive Simulation of Generalised Newtonian Fluids using GPUs

Somay Jain Nitish Tripathi P J Narayanan

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad, India

ABSTRACT

We present a method to interactively simulate and visualise Generalised Newtonian Fluids (GNF) using GPUs. GNFs include regular constant viscosity fluids as well as other fluids such as blood, which display variable viscosity due to variable shear rate. We use a statistical approach called Lattice Boltzmann Method (LBM) for the simulation. LBM is easy to understand and implement and does not include discretisation of differential equations. We exploit the inherent parallelism of LBM coupled with its memory access pattern to create a fast GPU implementation that gives scientifically accurate and fast results such as interactive real time simulations for reasonable domain size. MultiGPU implementations provide the potential to scale to larger problem sizes.

Keywords

Lattice boltzmann, computational fluid dynamics, GPU, CUDA

1. INTRODUCTION

Imitating the behaviour and characteristics of fluids with the help of a computer is called fluid simulation. Fluid simulation begins with the formulation of the Navier Stokes' equations originally developed in the 1840s on the basis of conservation laws and first order approximations.

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}.\nabla\mathbf{v} + \frac{1}{\rho}\nabla p = \mathbf{f}_{ext} + \nu\nabla.\nabla\mathbf{v} \quad (1)$$

$$\nabla.\mathbf{v} = 0. \quad (2)$$

Equation 1 is basically Newton's second law of motion, relating the rate of change of velocity field ($\mathbf{v}$) with the forces acting on the fluid. These are the forces due to the pressure (p) caused by the weight of the fluid, resistive force due to viscosity (ν) and the net force exerted externally ($\mathbf{f_{ext}}$). Eq 2 models incompressibility of the fluid. These equations, however, generalise the fluid behaviour seen around us. They describe a class of fluids called Newtonian fluids – fluids which have constant viscosity throughout. Water is an example of such a fluid. Fluids such as blood, mucus, multi-phase mixtures such as curry, emulsions, etc., fall in another class called non-Newtonian fluids. A majority of the fluids we see around us are non-Newtonian in nature.

ICVGIP '14, December 14-18, 2014, Bangalore, India

http://dx.doi.org/10.1145/2683483.2683562

Computational fluid dynamics models have existed for over five decades. Harlow et al. [14] were the pioneers in the field. The computer graphics community used Eulerian and Lagrangian viewpoints traditionally for simulation by discretising the Navier Stokes' equations post the era of hand-drawn animations. Lattice Boltzmann Method is a relatively new method, derived as a development over Lattice Gas Cellular Automata, it is a discretised model of the Boltzmann Equation of Kinetic Theory ([4]). It is a mesoscopic approach, with particles (logical in nature) colliding at grid centers, then progressing to their neighbours in fixed directions. Fluid properties are obtained from these particles through an aggregating method known as *coarse graining*. It is thus a statistical approach that eliminates the need to solve partial differential equations. Although traditional implementations of LBM require a few tweaks for higher Reynold's numbers for laminar flows, the method gives second order accuracy [4] in contrast to first order accuracy displayed by conventional Eulerian and Lagrangian methods [8].

LBM works on a Cartesian grid, with each cell functioning independently of others. This makes it highly suitable for parallel implementation. Transfer of data between the centers is ordered and can be utilised to make data access patterns conducive for implementing on a GPU. This enables us to simulate complex fluid behaviour in realtime or near-realtime. Such behaviour may include interactive simulations or implementing a single algorithm to tackle both Newtonian and non-Newtonian behaviour.

In this paper, we present a system for interactive simulation and visualisation of generalised Newtonian fluids using a parallel implementation of the LBM method on the GPUs building on our earlier work [25]. Our system provides fast and accurate simulation of a wide variety of fluids as well as different situations including free surface simulation. We show simulation of liquids with shear-thickening and shear-thinning properties and compare their behaviour with the analytical and real world expectations. We also show a multi-GPU implementation that can scale to larger grids and more general situations. We demonstrate simulation at 600 MLUPS using one NVIDIA K20c GPU (which translates to realtime performance on a 64^3 grid) and over 900 MLUPS using two K20c GPUs.

2. RELATED WORK

Before the 1990s, fluid animation was either hand drawn or used bump mapping tricks. CFD models were highly complex and had poor scalability. Foster and Metaxas did pioneering work on free surface flow [9] using the standard MAC grid. Stam [20] took the *Eulerian* method forward making it *semi-Lagrangian* in nature. To get the field value for a point at time $t + \Delta t$, he backtraces the point through the field over time Δt. These methods suffered from non-conservation of sub-grid mass. Enright et al. [7] solved the problem using *Particle Level Sets*. However, Eulerian simulations often have difficulty in producing small scale effects like sprays and foam which are essentially sub-grid in nature. Lagrangian methods were developed earlier to counter the shortcomings in Eulerian simulations. Desburn et al. [6] used Smoothed Particle Hydrodynamics as a means to simulate highly deformable bodies as particle systems. This was carried forward by Muller et al. [16]. Hybrid methods such as *FLIP* (Fluid Implicit Particle) are popular nowadays ([26]). Using the particle data, the Lagrangian moment equations are solved on (preferably) adaptive grids. [1] is another method evolved recently coupling Eulerian tetrahedral mesh discretisation with the FLIP method, leading to increased accuracy.

Application of statistical models to fluid simulation started in the 1970s. Lattice Gas Cellular Automata was the pioneering work in this direction [13]. LGCA did not evolve as a feasible method because of the aggregation of statistical noise. LBM emerged from LGCA, starting with Chen at al. [4]. Thuerey has been on the forefront of developing LBM to simulate free surface flows [23, 22, 21]. Recent times have seen parallel implementation of LBM come to the fore. Tölke [24] gave a 2D implementation of LBM using CUDA. Bailey et al. [2] gave a 3D parallel implementation of bulk LBM. Schreiber et al. [19] describe an OpenCL implementation for multicore architectures to obtain realtime simulations of free surfaces. Januszewski [15] present an LBM method on multiple GPUs for Newtonian fluids. They use diffuse interface models without explicit interface tracking and use additional lattices to represent multiple components and couple them using Shen-Chen or Free Energy models.

Over the last decade and a half, various problems concerning non-Newtonian fluids have been tackled. Goktekin et al. [12] dealt with viscoelastic fluids, i.e., the fluids exhibiting both viscous (characteristic of liquids) and elastic (characteristic of solids) properties. Clavet et al. [5] took a Lagrangian approach towards viscoelastic simulation. Application of second order accurate LBM in simulating non-Newtonian fluids was done by Boyd et al. [3]. Giroud et al. have presented a *multi-relaxation-time* LBM model for viscoelastic flows in [11] and [10]. Phillips et al. [18] produced a survey of the developments in coupling LBM with various non-Newtonian models. Pereira et al. [17] gave a parallel Navier-Stokes solver for generalised Newtonian fluids targeted to computational rheology applications.

Our method simulates generalised Newtonian fluids including those with shear thinning and shear thickening in a general situation with fluids, boundary, and free surfaces. Unlike Shen-Chen or Free Energy models, we have chosen the LBGK model for its simplicity and ease of implementation without compromising on accuracy. With the parallel-friendly LBM method for simulation and marching cubes for visualisation, we achieve interactive speeds using one or more GPUs.

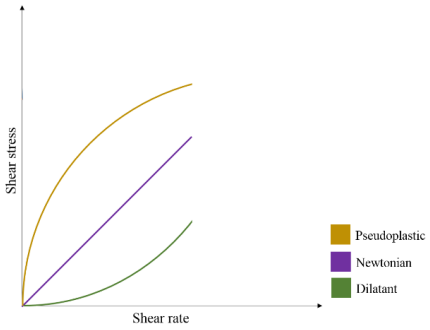


Figure 1: Flow curve for Generalised Newtonian Fluids

3. GENERALISED NEWTONIAN FLUIDS

For a Newtonian flow, the relation between the resultant shear stress τ and shear strain is given by Eq 3.

$$\frac{\mathbf{F}}{A} = \tau_{yx} = \mu\left(-\frac{dv_x}{dy}\right) = \mu\,\dot{\gamma}_{yx}. \tag{3}$$

Here, A is the area of cross section of the plates and $\mathbf{v}$ the velocity of the fluid. The minus sign implies a resistive force. μ is the Newtonian viscosity of the fluid which is a constant. It can be seen that Newtonian fluid follows a line through the origin (Fig 1). The deviatoric normal stress in a Newtonian flow in simple shear are identically zero.

The flow curves for non-Newtonian fluids are either non-linear or linear but not passing through the origin. In fact, Newtonian fluids fall into a subclass of a broader class called Generalised Newtonian Fluids (GNF). For a GNF, there is a one-to-one functional dependence of the rate of shear on the shear stress given by

$$\tau_{yx} = f\left(\dot{\gamma}_{yx}\right). \tag{4}$$

GNF can be of three categories:

1. Shear-thinning or pseudoplastic. These are fluids for which viscosity decreases with increasing shear rate.
2. Shear-thickening or dilatant. Viscosity increases with increasing shear rate for them.
3. Newtonian. Viscosity remains constant.

GNF behaviour is modelled by power law, also known as Ostwald-de Waele relationship between viscosity (ν) and rate of shear ($\dot{\gamma}$). It is given by the following equation.

$$\nu = m\dot{\gamma}^{n-1}, \tag{5}$$

where, $n < 1$ for shear-thinning fluids, $n = 1$ for Newtonian and $n > 1$ for shear-thickening fluids. The viscosity ν needs to be accounted explicitly to simulate departures from Navier Stokes' behaviour that GNFs entail.

The rate of shear is defined as,

$$\dot{\gamma} = \sqrt{2\mathbf{d} : \mathbf{d}}, \tag{6}$$

Where $\mathbf{d}$ is the strain rate tensor given by,

$$\mathbf{d} = \frac{1}{2}(\nabla\mathbf{v} + \nabla\mathbf{v}^t). \tag{7}$$

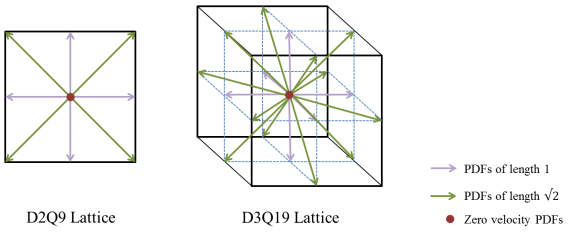


Figure 2: D2Q9 and D3Q19 Grids

Vector	Direction
$\mathbf{e_0}$	$(0,0,0)'$
$\mathbf{e_{1,2}}$	$(\pm1,0,0)'$
$\mathbf{e_{3,4}}$	$(0,\pm1,0)'$
$\mathbf{e_{5,6}}$	$(0,0,\pm1)'$
$\mathbf{e_{7\ldots10}}$	$(\pm1,\pm1,0)'$
$\mathbf{e_{11\ldots14}}$	$(0,\pm1,\pm1)'$
$\mathbf{e_{15\ldots18}}$	$(\pm1,0,\pm1)'$

Table 1: Velocity vectors for D3Q19

Ostwald-de Waele relationship can be represented as a *power law*, which in truncated form is given below,

$$\nu = \begin{cases} k \times \dot{\gamma_0}^{n-1} & \dot{\gamma} < \dot{\gamma_0} \\ k \times \dot{\gamma}^{n-1} & \dot{\gamma_0} < \dot{\gamma} < \dot{\gamma_\infty} \\ k \times \dot{\gamma_\infty}^{n-1} & \dot{\gamma_\infty} < \dot{\gamma} \end{cases} \tag{8}$$

4. LATTICE BOLTZMANN METHOD

LBM depends on a Cartesian discretisation of the simulation domain into regular cells. Particles are constrained to travel in specific directions only. Some of the popular implementations allow particles to travel in 9 (two dimensional), and, 15, 19 and 27 (three dimensional) directions from a grid cell. On this basis, the grids are called D2Q9, D3Q15, D3Q19 and D3Q27 respectively. D3Q19 is the most popular among them as it is more precise than D3Q15 and involves lesser computations than D3Q27 without compromising on accuracy. D2Q9 and D3Q19 grids are shown in the Fig 2.

For ease in computation each cell is assumed to be unit sided and each particle unit massed. A cell keeps track of the number of its particles going in different directions using *particle distribution functions* (PDF). As the name suggests, this is not an actual count of the particles but it is a distribution function, and hence, is allowed to take fractional values. As, in a single time step, particles can only travel from a cell to its neighbour in one of the directions, each direction has a velocity vector associated with it.

For D3Q19, these ($\mathbf{e_i}$) are shown in Table 1. Density ρ for a cell is obtained by adding the PDFs as the particle is unit massed and the cell, unit sided, according to Eq 9. Here, df_i is the PDF in direction i.

$$\rho = \sum df_i \tag{9}$$

The velocity field value $\mathbf{u}$ for a cell is given by Eq 10.

$$\mathbf{u} = \sum df_i \cdot \mathbf{e_i} \tag{10}$$

Algorithm 1 Basic LBM for DXQY lattice

```
1: procedure STREAM(x, y, z)
2:     Update current DF with neighbours' DF
3: procedure COLLIDE(x, y, z)
4:     Calculate density(ρ) and velocity (u) using Eq 9, 10
5:     Calculate df^eq using Eq 11
6:     Update df using Eq 12
7: procedure LBM
8:     for all cells in parallel do
9:         stream(x, y, z)
10:        collide(x, y, z)
```

4.1 Basic LBM

Two steps, *streaming* and *collision* comprise the basic algorithm to simulate bulk of the fluid without a free surface. A cell of D3Q19 lattice at $\langle x, y, z\rangle$ maintains a vector of 19 PDF values, $\langle df_0, df_1, \ldots, df_{18}\rangle$.

4.1.1 Streaming

Streaming involves reading neighbours' distribution functions for corresponding directions and updating. Hence it involves 18 independent copy operations.

4.1.2 Collision

Velocity and density for each cell are calculated by *coarse graining* as given by Eq (10) and (9). Collision involves computation of equilibrium distribution functions $(df_0^{eq}, \ldots, df_{18}^{eq})'$ followed by a final update of DFs using BGK approximation [4].

$$df_i^{eq}(\rho, \mathbf{u}) = w_i \left(\rho - \frac{3}{2}\mathbf{u}^2 + 3\mathbf{e_i}\cdot\mathbf{u} + \frac{9}{2}(\mathbf{e_i}\cdot\mathbf{u})^2\right) \tag{11}$$

$$df_i = (1-\omega)df_i + \omega df_i^{eq} \tag{12}$$

The weights $(w_i)\prime$ are $\frac{1}{3}$ for the present cell, $\frac{1}{18}$ for neighbours at a Manhattan distance of one and $\frac{1}{36}$ for neighbours at a Manhattan distance of two. ω is the relaxation frequency.

Algorithm 1 gives an outline of the steps for Basic LBM.

4.2 Free Surface LBM

The above method outlined the two basic steps for simulating the bulk of fluid. To simulate free surfaces (the partition between the fluid and the environment) for a generalised Newtonian fluid, we need to expand the algorithm to account for the interaction of the fluid with the environment. We build upon the algorithm given by Thuerey et al [22].

The cells are differentiated on the basis of whether they contain fluid, gas (environment) or form the interface between the two. This interface is formed by cells partially filled with fluid. As the fluid progresses forward, the cells get relabelled after each iteration according the amount of fluid they hold. Atmospheric pressure, reference density and pressure of fluid are assumed to be unity for simplicity.

Since the label on a cell depends on how much fluid it holds, fluid fraction ϵ is calculated for each cell. It is defined as ratio of the cell mass m with its density ρ.

$$\epsilon = \frac{m}{\rho} \tag{13}$$

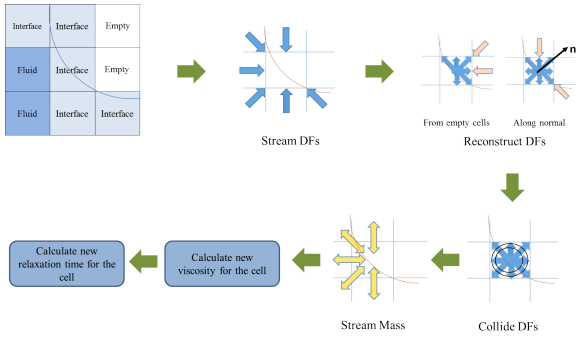


Figure 3: Overview of Free Surface LBM

4.2.1 Reconstruction of distribution functions

Streaming of distribution functions happens the same way as in the basic algorithm (Section 4.1.1), with empty cells not taking part in it. Since one side of the interface cells do not contain PDFs to stream we need to construct those. If for a cell at $\mathbf{x}$ there is an empty cell at $\mathbf{x} + \mathbf{e_i}$, then,

$$df_i' = df_i^{eq}(\rho_A, \mathbf{u}) + df_{\tilde{i}}^{eq}(\rho_A, \mathbf{u}) - df_{\tilde{i}}(\mathbf{x}, t) \qquad (14)$$

where df' is the updated distribution function and ρ_A, the density of gas (taken to be unity). $\tilde{i}$ is the direction opposite to i. The DFs coming from the direction of the interface normals are also reconstructed to counter the effect of asymmetrical streaming, using the same equation.

4.2.2 Mass Transfer

The collision step is the same as that of basic LBM (Section 4.1.2). Fluid cells are filled to their maximum capacity and mass exchange between them at any point of time is equal and opposite. Mass transfer from a fluid to an interface cell is given by,

$$\Delta m_i(\mathbf{x_i}, t + \Delta t) = df_{\tilde{i}}(\mathbf{x} + \mathbf{e_i}\Delta t, t) - df_i(\mathbf{x}, t) \qquad (15)$$

The mass exchange between interface cells depends on their mass densities.

$$\Delta m_i(\mathbf{x_i}, t + \Delta t) = s_e \frac{\epsilon(\mathbf{x} + \mathbf{e_i}\Delta t, t) + \epsilon(\mathbf{x}, t)}{2}, \qquad s_e = df_{\tilde{i}}(\mathbf{x} + \mathbf{e_i}\Delta t, t) - df_i(\mathbf{x}, t) \qquad (16)$$

4.2.3 Relabelling cells

Often, the amount of mass exchanged between cells makes their mass density go beyond the permissible range. This may happen when a cell empties or fills up completely in $t < \Delta t$. Hence we need to relabel the cells which emptied or filled up and their neighbourhood. Also, we need to distribute the excess or deficient mass.

If the current mass density exceeds a threshold value, it is labeled *filled*. Else, if it falls below the threshold, it is labeled *emptied*. The neighbourhood of the filled cells is checked and any empty cells are relabelled interface. Equilibrium DFs are awarded to them by allotting them average velocity and average density of their neighbourhood. We also remove the emptied interface cells from the emptied list, which will be used as boundary for the filled cell. The filled cells can now be labeled fluid. The process is repeated for emptied cells.

Algorithm 2 Free Surface LBM for DXQY lattice

```
1: procedure reconstructDF(x, y, z)
2:     Update df using Eq 14
3: procedure transferMass(x, y, z)
4:     Update mass using Eq 16
5:     if cell becomes completely filled then
6:         Mark as filled_interface_cell
7:     else if cell becomes completely empty
8:         Mark as emptied_interface_cell
9: procedure relabelCells(x, y, z)
10:    if filled_interface_cell then
11:        Convert empty neighbours into interface cells
12:        Make current cell a fluid cell
13:    else if emptied_interface_cell
14:        Convert fluid neighbours into interface cells
15:        Make current cell an empty cell
16: procedure distributeExcessMass(x, y, z)
17:    if filled_interface_cell or emptied_interface_cell then
18:        Distribute excess mass among neighbours
19: procedure calculateNewViscosity(x, y, z)
20:    Calculate viscosity using truncated power law
21: procedure Free Surface LBM
22:    for all cells in parallel do
23:        if fluid or interface cell then
24:            stream(x,y,z)                ▷ Same as Basic LBM
25:        if interface cell then
26:            reconstructDF(x, y, z)
27:        if fluid or interface cell then
28:            collide(x,y,z)               ▷ Same as Basic LBM
29:        if interface cell then
30:            transferMass(x, y, z)
31:            relabelCells(x, y, z)
32:            distributeExcessMass(x, y, z)
33:        if Non Newtonian fluid then
34:            calculateNewViscosity(x, y, z)
```

4.2.4 Excess Mass Distribution

Excess mass for an emptied or filled cell is given by m (negative) or $m - \rho$ respectively. Mass is distributed to the neighbours, weighted favourably for the cells lying along the direction of progression of the surface.

4.2.5 Calculating new viscosity

Generalised Newtonian Fluid (GNF) simulations employ localised omega values for each cell. Velocity field variation between cells give rise to variable strain. Using this we calculate the rate of shear. Applying *truncated power law* (Eq 8) we obtain localised viscosity using which we calculate relaxation time τ.

$$\tau = \frac{6\nu + 1}{2} \qquad (17)$$

The overview of the algorithm is shown in Figure 3.

5. PARALLEL IMPLEMENTATION WITH CUDA

We build upon the algorithm given by [22], with changes in the order of execution of the steps to make it conducive with the GPU architecture.

5.1 Data Requirement

The data requirement for each cell is given in Table 2. These are stored in the global memory, as described in the

Data	Size	Use
Previous DFs	19 floats	Previous iteration distribution function
Current DFs	19 floats	Current iteration distribution function
Previous State	1 int	Type of cell in previous iteration
Current State	1 int	Type of cell in current iteration
Epsilon	1 float	Intermediate, visualisation purposes
Velocity	3 floats	Intermediate, visualisation purposes

Table 2: Data Requirement for each cell

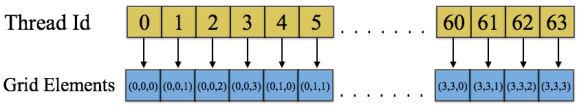


Figure 4: Thread Mapping with Grid Elements

following sections. We use double buffering for storing the state and the distribution function for the grid.

5.2 Thread Mapping

Since each cell reads its neighbour's previous data and writes only its own current data, the computation for each cell happens independent of the others. Thus, we assign one thread per cell for doing the computation.

We make a 1D grid of threads and map each thread to the grid elements in row major format as shown in Figure 4.

Because each warp consists of 32 threads, for grid sizes with x-dimension multiple of 32, each warp operates on cells which lie in the same row, thus leading to optimised access as explained in the following sections.

5.3 Data Layout

For efficiency, it is critical to store the data in a manner which allows maximum possible coalesced read and write operations. To achieve this, we employ a *SoA* (Structure of Arrays) data format to store the information required for each cell, wherein the data for the 3D grid is stored linearly in the memory as a 1D array in row major format.

The distribution function is stored the same way, with the values corresponding to a particular direction stored in contiguous memory blocks in row major format, as shown in Figure 5.

5.4 Memory Access Pattern

In `stream`, `reconstructDF`, `collide` and `transferMass` kernels given in Algorithm 2, all threads in a warp read/update the distribution function for a particular direction at the same time. These memory accesses are fully coalesced because adjacent threads map to horizontally adjacent cells of the grid. For instance, if a thread with thread index (tid) maps to the cell $\langle x, y, z \rangle$, then the thread $(tid + 1)$ will map to the cell $\langle x + 1, y, z \rangle$. Their k^{th} neighbour would be $\langle x + e_{ix}, y + e_{iy}, z + e_{iz} \rangle$ and $\langle x + 1 + e_{ix}, y + e_{iy}, z + e_{iz} \rangle$ respectively, where $\langle e_{ix}, e_{iy}, e_{iz} \rangle$ is the k^{th} direction vector. Hence, the k^{th} neighbour of adjacent cells are also adjacent. Because of the *SoA* data layout, the distribution function values of a particular direction for the k^{th} neighbour of adjacent cells are also adjacent in memory. This is shown in Figure 6. These kernels achieve 100% occupancy on the

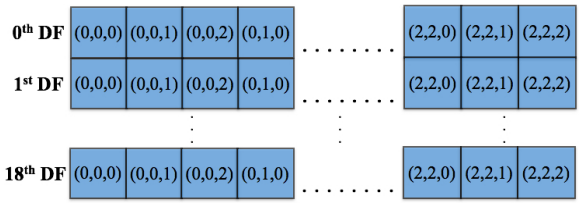


Figure 5: Distribution Function Layout for a 3^3 Grid, stored in row major format

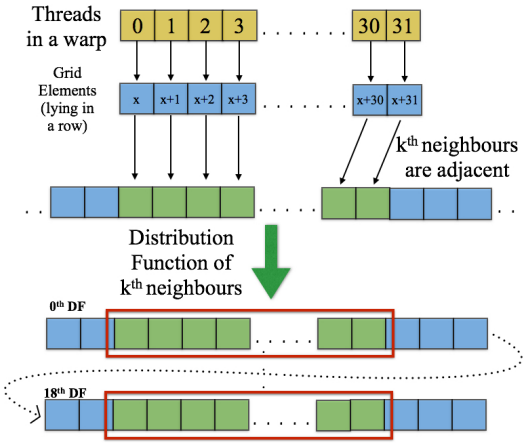


Figure 6: DFs for k^{th} neighbours of adjacent cells

GPU hardware.

The remaining steps, `relabelCells` and `distributeExcessMass`, read their neighbour's data and update their own. Since neighbours of adjacent cells are adjacent in memory, these too are coalesced accesses. These kernels only achieve 75% occupancy of the GPU due to the need for more registers to hold the variables used.

5.5 Thread Divergence

Since the steps for Free Surface LBM are performed only for the interface cells, and the kernels are called for all cells, it introduces thread divergence in the kernels. One solution to avoid it is to sort the cells according to their state. However, because of this, adjacent threads do not work on adjacent cells in memory, thus leading to uncoalesced memory accesses. To achieve coalesced memory access, data also needs to be moved, which worsens the situation, making the process much slower.

The interface cells form the boundary of the liquid and are much less in number. The threads corresponding to the non-interface cells simply return and there is thread divergence only for those warps which have both interface and non-interface cells. Thus, the overhead of thread divergence is much lower than the computational overhead of separating the interface cells and running it only for them.

5.6 Using Multiple GPUs

We use two GPUs on the same system to further scale the problem. We divide the data for each GPU by slicing the grid along the z-axis. We do not choose the y-axis because bulk of the fluid is present at the bottom of the grid, which would lead to uneven distribution of the filled and interface

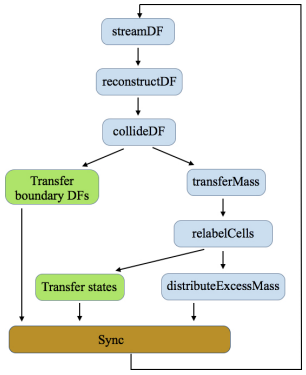


Figure 7: Overlap of data transfers with computation

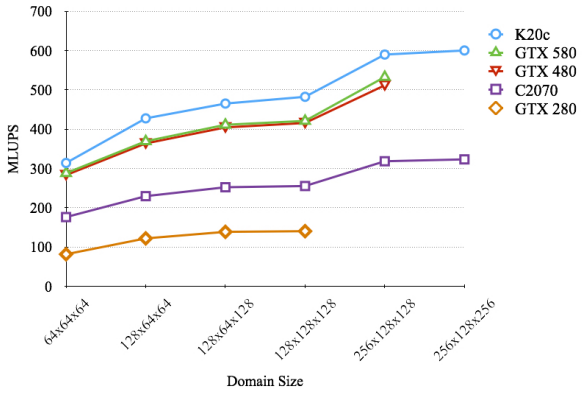


Figure 8: Performance of the Dam Break Experiment on various GPUs

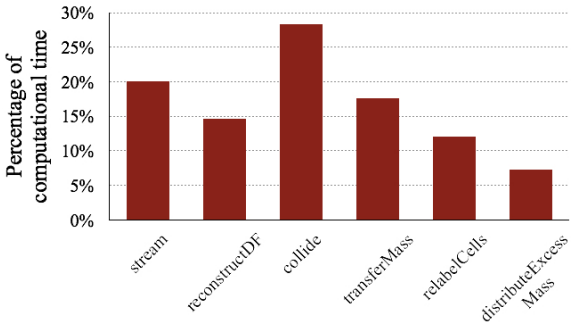


Figure 9: Relative Time Taken by each kernel on K20c for Dam Break on a 128^3 grid

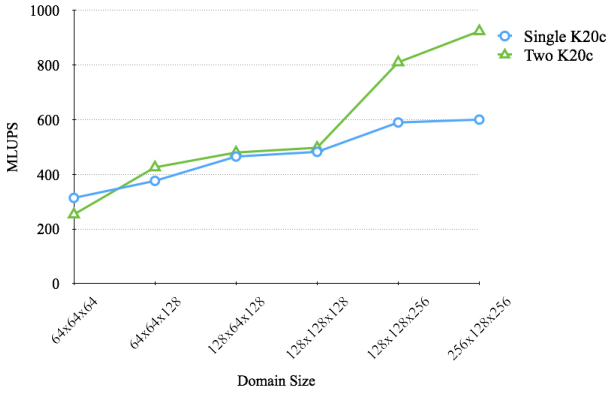


Figure 10: Performance of the Dam Break Experiment on single and multi-GPUs

cells among the two GPUs. The x-axis is not chosen to exploit the spatial locality along it.

The cells on the boundary of the dividing slice need the data from the neighbouring cells which reside on the other GPU. So, in each iteration, the slice of data on the boundary is transferred to the other GPU.

Each GPU needs to transfer the current DFs and state of the boundary cells to the other GPU. As evident from the pipeline shown in Figure 7, the DFs are available as soon as collision step is completed and are not required until the next iteration. So, we do an asynchronous transfer to the other GPU to overlap it with the computation. Similarly, the states are transferred as soon as they are reinitialised.

6. RESULTS

In this section, we show the results of our GPU implementation using dam break, falling drop, flow between two parallel plates, flow of a non-Newtonian fluid through a tube of varying cross section and flow of Newtonian and non-Newtonian fluids through a slit. The experiments discussed below are performed on the NVIDIA Tesla K20c, unless stated otherwise. We refer the reader to the supplementary videos for the simulations resulted out of the experiments.

6.1 Performance

The performance for the dam break experiment on various GPUs and grid sizes is given in the Figure 8. The performance is measured in *Million Lattice Updates Per Second* (MLUPS), which is the number of grid points processed per second. The optimal block size for all the GPUs is experimentally found to be 256, except for NVIDIA GeForce GTX 280, for which, it is 128. At these block sizes, the blocks fill up the GPU, giving close to 100% occupancy on most kernels.

Figure 9 shows relative percentage of time taken by each kernel for 1000 LBM iterations of the dam break experiment on a 128^3 grid. As expected, the `collide` step takes the most amount of time because it is run for both *filled* and *interface* cells and updates their DFs after computing $\mathbf{u}$ and ρ.

The performance of multi-GPU implementation is shown in Figure 10. Since on larger grid sizes, both GPUs are well occupied, it performs much better than a single GPU.

6.2 Visualisation

The visualisation of the fluid surface is done using marching cubes algorithm, with each frame rendered after 50 LBM iterations. We have taken 50 iterations per frame to maintain a significant visual difference between two frames.

Figure 11 shows the dam break experiment for a Newtonian fluid on a 128^3 grid. The fluid has ω equal to 1.85. It initially runs at 5.5 frames per second, which drops to 4.3 frames per second when the fluid splashes around, giving an average of 5 frames per second. For a grid size of 64^3, the same experiment runs at an average of 27 frames per second.

The intermediate frames Figure 14 show the interactive simulation. Here, the user can add drops of fluid interac-

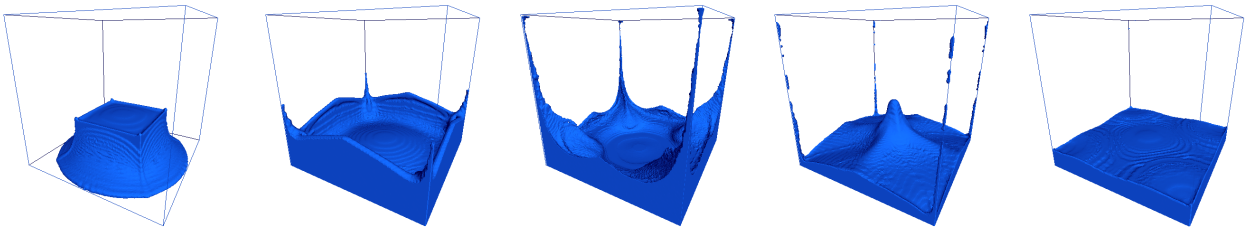
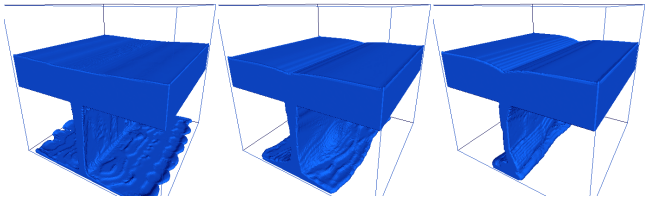


Figure 11: Intermediate frames for Dam Break Experiment for a Newtonian Fluid on a 128^3 grid, running at an average of 5 frames per second with 50 LBM iterations per frame



(a) Shear thinning (b) Newtonian (c) Shear thickening

Figure 12: Comparison between shear thinning, newtonian and shear thickening fluid

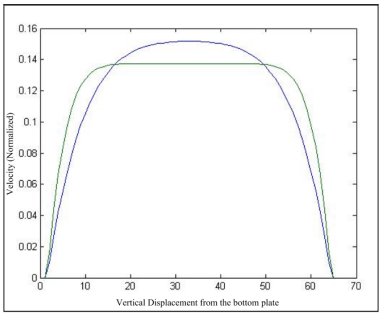


Figure 13: Comparison between flow curves of Newtonian (blue) and non-Newtonian (green) fluids

tively by clicking, while the simulation is going on. It runs on an average of 6.6 frames per second. The same simulation runs at an average of 30 frames per second for a 64^3 grid.

To test the visual accuracy of our model we simulate the flow of a shear-thinning fluid through a tube of varying cross section. Neumann boundary conditions were used to drive the flow, which is tracked by virtual dye. The dye particles change colour according to the change of viscosity of the fluid. As time progresses variable shear rate is experienced by the fluid due to the varying cross section of the tube. This leads to the formation of concentric regions in the tube with varying viscosity with the outermost region having the lowest. We display the simulation result in the supplementary video provided.

Figure 12 shows comparative snapshots of a shear-thinning, Newtonian and shear-thickening fluid respectively. *Truncated power law* gives the relation between shear stress (τ) and rate of shear ($\dot{\gamma}$) as $\tau = m\dot{\gamma}^{n-1}$. We compare three fluids with same values of m and different n being poured from a height on to a flat horizontal base. (a) is shear-thinning with $n < 1$, (b) is Newtonian with $n = 1$ and (c), shear-thickening with $n > 1$. As can be seen in the figure, (a) displays more *fluidity* (decrease in viscosity) upon impact with the ground whereas (c) displays folding on itself signifying greater resistance (increase in viscosity) on impact.

6.3 Correctness

To evaluate the correctness of the method, we look at the velocity profile for Newtonian and non-Newtonian fluids as they are made to pass between two parallel plates. It is assumed that the plates have a large area so that the fluid flows just between them and not around. A motion parallel to the two plates is induced in the fluid. The fluid lamina in contact with the two plates will not move on account of its viscosity. As we move further away from either of the two plates the velocity of each fluid lamina increases, until we reach the center, where, due to symmetry the velocity is expected to be the maximum.

Analytical calculations augur a parabolic velocity profile for a Newtonian fluid. For a non-Newtonian fluid, the profile will be more complex since shear between laminae will give rise to changes in viscosity. These changes in viscosity correspondingly would affect the velocity of the laminae. Indeed, for a pseudo-plastic fluid, it has been shown by [3] that a parabolic curve which is plateaued (flattened) in the center is to be expected. Figure 13 shows the normalised velocity profiles obtained from our experiment. It can be seen that whereas the Newtonian fluid curve follows a parabolic path the non-Newtonian fluid curve flattens on approaching the center of the channel. The experimental results therefore, conform to the analytical expectation.

7. CONCLUSIONS & FUTURE WORK

We presented a system to simulate and visualise Generalised Newtonian Fluids accurately and quickly in this paper. Using boundary conditions such as *no-slip* reduces the accuracy from its inherent second order, although it is still as good as other conventional methods. Also, size of the channel matters. As the tube becomes thinner and thinner particular nature of the flow becomes prominent. These factors need to be studied to make the algorithm more comprehensive. We have dealt with laminar fluids in this work. A study of turbulent fluids using LBM is an interesting area for further development.

Interactivity in realtime as shown in our simulations on 64^3 domain can be utilised in areas ranging from education to games on mobile platforms. We are working towards enhancing the visual quality of our simulations by ray-tracing them. As with any GPU based method, we were limited by

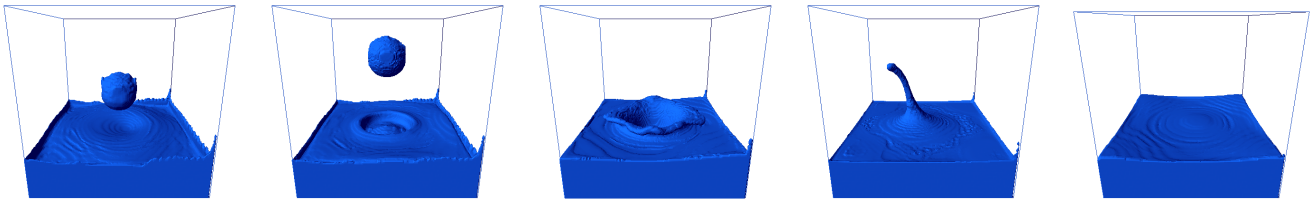


Figure 14: Intermediate frames for interactive simulation of a Newtonian Fluid on a 128^3 grid, running at an average of 6.6 frames per second with 50 LBM iterations per frame. The user can add fluid drops while simulation is running.

memory constraints. Out of core grids (size 512^3 and above) require data transfer between host and device which slows the simulation down considerably. We want to enhance our method further to result in real time or near-realtime simulation over larger (and consequently) more detailed grids.

8. REFERENCES

[1] R. Ando, N. Thuerey, and C. Wojtan. Highly Adaptive Liquid Simulations on Tetrahedral Meshes. *ACM Transactions on Graphics (SIGGRAPH)*, 32 (4), August 2013.

[2] P. Bailey, J. Myre, S. Walsh, D. Lilja, and M. Saar. Accelerating lattice boltzmann fluid flow simulations using graphics processors. In *Parallel Processing, 2009. ICPP '09. International Conference on*, 2009.

[3] J. Boyd, J. Buick, and S. Green. A second-order accurate lattice boltzmann non-newtonian flow model. *Journal of Physics A: Mathematical and General*, 39(46).

[4] S. Chen and G. D. Doolen. Lattice boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, 30(1), 1998.

[5] S. Clavet, P. Beaudoin, and P. Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005.

[6] M. Desbrun and M.-P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, 1996.

[7] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1), 2002.

[8] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Comput. Struct.*, 83(6-7), 2005.

[9] N. Foster and D. Metaxas. Realistic animation of liquids. *Graph. Models Image Process.*, 58(5), 1996.

[10] L. Giraud, D. d'HumiÃĺres, and P. Lallemand. A lattice boltzmann model for jeffreys viscoelastic fluid. *EPL (Europhysics Letters)*, 42(6).

[11] L. Giraud, D. d'HumiÃĺRes, and P. Lallemand. A lattice-boltzmann model for visco-elasticity. *International Journal of Modern Physics C*, 08(04).

[12] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien. A method for animating viscoelastic fluids. In *ACM SIGGRAPH 2004 Papers*, 2004.

[13] J. Hardy, Y. Pomeau, and O. de Pazzis. Time evolution of a two-dimensional model system. i. invariant states and time correlation functions. *Journal of Mathematical Physics*, 14(12), 1973.

[14] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12), 1965.

[15] M. Januszewski and M. Kostur. Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method. *Computer Physics Communications*, 185, 2014.

[16] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[17] S. P. Pereira, K. Vuik, F. T. Pinho, and J. M. NÃşbrega. On the performance of a 2d unstructured computational rheology code on a gpu. *AIP Conference Proceedings*, 1526(1), 2013.

[18] T. N. Phillips and G. W. Roberts. Lattice boltzmann models for non-newtonian flows. *IMA Journal of Applied Mathematics*, 76(5), 2011.

[19] M. Schreiber, P. Neumann, S. Zimmer, and H.-J. Bungartz. Free-surface lattice-boltzmann simulation on many-core architectures. *Procedia Computer Science*, 4(0), 2011.

[20] J. Stam. Stable fluids. SIGGRAPH '99, 1999.

[21] N. Thuerey, K. Iglberger, and U. Ruede. Free Surface Flows with Moving and Deforming Objects for LBM. *Proceedings of Vision, Modeling and Visualization*, 2006.

[22] N. Thuerey and U. Ruede. Free Surface Lattice-Boltzmann fluid simulations with and without level sets. *Proc. of Vision, Modelling, and Visualization VMV*, 2004.

[23] N. Thuerey and U. Ruede. Optimized Free Surface Fluids on Adaptive Grids with the Lattice Boltzmann Method. *Poster, SIGGRAPH '05*, 2005.

[24] J. Tölke. Implementation of a lattice boltzmann kernel using the compute unified device architecture developed by nvidia. *Computing and Visualization in Science*, 13(1), 2010.

[25] N. Tripathi and P. Narayanan. Generalized newtonian fluid simulations. In *Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*, 2013.

[26] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3), 2005.