Optimised Implementation of the Lattice Boltzmann Method on a Graphics Processing Unit Towards Real-Time Fluid Simulation

N. Delbosc[a], J.L. Summers[a], A. Khan[b], N. Kapur[a], C.J. Noakes[b]

[a] *School of Mechanical Engineering, University of Leeds, England*
[b] *School of Civil Engineering, University of Leeds, England*

Abstract

Real-time fluid simulation is an active field of research in computer graphics, but they usually focus on visual impact rather than physical accuracy. However, by combining a lattice Boltzmann model with the parallel computing power of a graphics processing unit, both real-time compute capability and satisfactory physical accuracy are now achievable. The implementation of an optimised 3D real-time thermal and turbulent fluid flow solver with a performance of half a billion lattice node updates per second is described in detail. The effects of the hardware error checking code and the competition between appropriate boundary conditions and performance capabilities are discussed.

Keywords: real-time, Lattice Boltzmann Method, Graphics Processing Unit, CUDA

1. Introduction

A variety of applications are in need of real-time compute capability for the simulation of indoor air-flow. A few examples are: the optimisation of designs, control and prediction of thermal loads in a data center [1], or the simulation of airborne pollutant transport in hospitals [2, 3]. Despite the growth in both speed and memory of computers during the last decades, real-time fluid simulation is still a challenge.

Computer graphics, such as video-games, achieve real-time fluid simulation by using simplistic models (usually wave-based or particle-based [4]), but they are focused on creating visually appealing animations rather than aiming for physical accuracy. The more physical models, such as semi-Lagrangian methods [5], are still rarely used.

Email addresses: mnnd@leeds.ac.uk (N. Delbosc), J.L.Summers@leeds.ac.uk (J.L. Summers), A.Khan@leeds.ac.uk (A. Khan), n.kapur@leeds.ac.uk (N. Kapur), c.j.noakes@leeds.ac.uk (C.J. Noakes)

Engineering applications commonly use a discrete version of the Navier-Stokes Equations (NSE) to simulate fluid flow. They are highly accurate but they require heavy computations, consuming a significant amount of memory and a high number of iterations to converge, thus they are still incapable of simulating flow in real-time with the current technology.

The implementation of early warning systems for data centers and hospitals require both real-time (or faster than real-time) compute capability, and a level of accuracy which enables the resolution of global flow structures. The small scale effects on the flow structures, however, are of minor importance in such systems, and therefore the goal of this work is not to develop a solver with a high level of accuracy. The efficiency of the Lattice Boltzmann Method (LBM) as a real-time flow solver was selected for investigation. The LBM originates from the lattice gas automata method and can be regarded as an explicit discretisation of the Boltzmann equation. The LBM has been used successfullly in the past to simulate indoor air flows [6, 7] and for interactive flow solver [8].

Fluid mechanics problem can be solved using the LBM or the NSE approaches. The former has advantages[9] that are useful for real-time implementation, it provides a simple algorithm both stable and accurate that can efficiently handle complex geometries, and its inherent data-parallelism makes it a good candidate for an implementation on massively parallel machines, like graphics processing units (GPU). But it also has drawbacks such as a high memory usage and the need to convert units between simulation and real-world applications.

This paper discusses several optimisation techniques in order to develop an efficient 3D LBM program on GPUs, that includes appropriate thermal and turbulence models.

2. The lattice Boltzmann method for three dimensional thermal and turbulent flow

The LBM can be derived from a special discretisation of the Boltzmann equation, the fluid is described by particle distribution functions residing at the sites of a regular grid (the lattice) and the macroscopic quantities of the fluid (like the density ρ or the velocity $\mathbf{v}$) can be recovered from moments of these distribution functions. The movement of the particle populations is restricted to a fixed set of directions $\mathbf{e}_\alpha$ defined on the links between neighbouring lattice sites.

The LBM is composed of two fundamental steps. First, the distribution functions of each node are streamed to their neighbouring nodes (the streaming-step). Then the distribution functions are relaxed toward a local equilibrium based on the new macroscopic quantities at each lattice site (the collision-step). While the streaming-step only depends on the lattice geometry, the collision-step embodies all the physics of the model and the chosen relaxation scheme specifies the stability and the accuracy of the method. Another important part of any LBM simulation is the implementation of the boundary conditions which takes place before or after the collision-step. Boundary conditions can be implemented

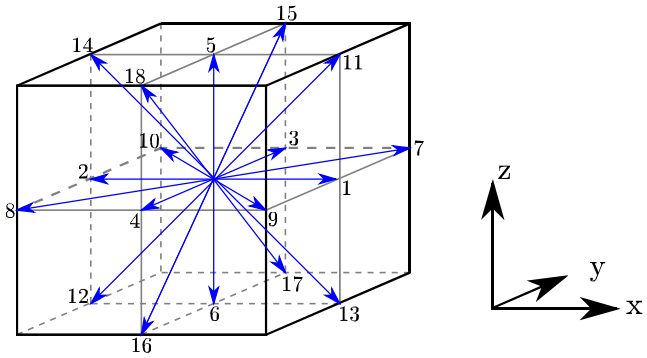


Figure 1: The D3Q19 node.

in various ways in the LBM, but in principle, they define the unknown distribution functions at the boundary (namely the distribution functions streamed from outside of the domain) in order to recover the desired macroscopic equations.

2.1. The D3Q19 model

A common labelling for lattices used in LBM is **DdQq**, where **d** is the spatial dimension and **q** the number of microscopic velocities. There are several possible 3D lattice constructions for hydrodynamics, such as D3Q13, D3Q15, D3Q19, D3Q27 and other higher order lattices that would require too many computations for real-time implementation. The D3Q19 model, illustrated in Figure 1, is applied here since it has minimum number of velocities while maintaining sufficient isotropy of the lattice.

The simulation of the velocity field is carried out on such a D3Q19 lattice, the complex collision operator is approximated by using the standard Bhatnagar-Gross-Krook (BGK) scheme[10] which states that the distribution functions $\mathbf{f} = \{f_i\}_{i\in\{0,\dots,18\}}$ is close to a local equilibrium $\mathbf{f}^{(eq)} = \{f_i\}_{i\in\{0,\dots,18\}}$ and relaxes towards this equilibrium over a characteristic time τ. The evolution of the distribution functions using the BGK collision approximation is described by the following lattice Boltzmann equation (referred to as LBGK):

$$f_i\left(\mathbf{x} + c\mathbf{e}_i\Delta t, t + \Delta t\right) = f_i\left(\mathbf{x}, t\right) - \frac{1}{\tau}\left(f_i\left(\mathbf{x}, t\right) - f_i^{(eq)}\left(\mathbf{x}, t\right)\right), \qquad (1)$$

where $c = \Delta x/\Delta t$ is the lattice speed, Δx and Δt are the lattice spacing and time increment, respectively.

The fluid density ρ and velocity $\mathbf{u}$ are determined from the zero and first moments of the distribution functions,

$$\rho = \sum_{i=0}^{18} f_i, \qquad \rho\mathbf{u} = \sum_{i=0}^{18} c\mathbf{e}_i f_i,$$

where the discrete velocity set $\{\mathbf{e}_i\}$ is defined in accordance with the Figure 1 as,

$$\mathbf{e}_i = \begin{cases} (0,0,0) & i = 0, \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm 1) & i = 1-6, \\ (\pm 1,\pm 1,0) & i = 7-10, \\ (\pm 1,0,\pm 1) & i = 11-14, \\ (0,\pm 1,\pm 1) & i = 15-18. \end{cases}$$

The local equilibrium distribution functions are computed from the new density ρ and velocity $\mathbf{u}$ (obtained after the streaming step) by the following formula

$$f_i^{(eq)} = \rho w_i \left(1 + 3\frac{\mathbf{e}_i \cdot \mathbf{u}}{c} + \frac{9}{2}\frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{c^2} - \frac{3}{2}\frac{\mathbf{u}^2}{c^2}\right),$$

where w_i are weight coefficients that depend on the magnitude of $\mathbf{e}_i$, $w_0 = 1/3$, $w_{1,\ldots,6} = 1/18$, $w_{7,\ldots,18} = 1/36$.

It can be shown through a Chapman-Enskog expansion [11, 12] that the NSE can be recovered from the lattice BGK model as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \tag{2}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nu \left(\nabla^2 (\rho \mathbf{u}) + \nabla (\nabla \cdot (\rho \mathbf{u}))\right) \tag{3}$$

with an error proportional to $\mathcal{O}\left(Ma^3\right)$ in space[12] and proportional to $\mathcal{O}\left(Ma \cdot dt\right)$ in time[13], where $Ma = u/c_s$ is the Mach number of the system, $p = c_s^2 \rho$ is the pressure, $c_s = c/\sqrt{3}$ is the speed of sound and the kinematic viscosity ν is related to the relaxation time τ by

$$\nu = \frac{(2\tau - 1)}{6}\frac{\Delta x^2}{\Delta t}.$$

2.2. The coupled temperature model

In order to simulate the temperature, a coupled model[14] is used. In this model, the velocity and density are solved as usual using a D3Q19 lattice with a BGK collision operator and the temperature is solved on another, smaller, D3Q6 lattice, as depicted in Figure 2. The temperature can be seen as a scalar advected by the fluid, and the buoyancy effects are simulated by adding a forcing term in the NSE, relative to the temperature differences. The fluid buoyancy is accounted for by the application of the Boussinesq approximation.

It is worth noting that other models to simulate thermal flows within the LBM framework exist and have been implemented on GPU[15]. In these models, the velocity is solved using a standard LBM, and the temperature is solved with a finite difference scheme.

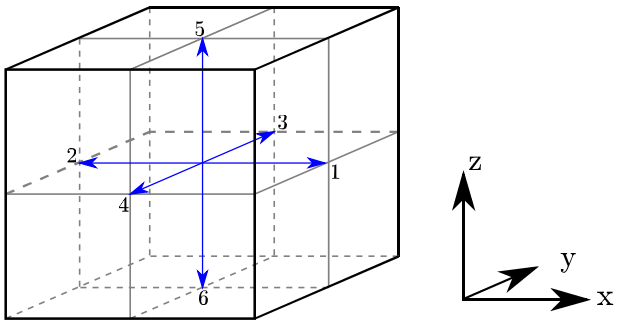


Figure 2: The D3Q6 node.

The six temperature distribution functions $\mathbf{T} = \{T_i\}_{i\in\{1,\dots,6\}}$ are streamed along the D3Q6 velocities and relaxed using the corresponding LBGK equation,

$$T_i\left(\mathbf{x} + c\mathbf{e}_i\Delta t, t + \Delta t\right) = T_i\left(\mathbf{x}, t\right) - \frac{1}{\tau_T}\left(T_i\left(\mathbf{x}, t\right) - T_i^{(eq)}\left(\mathbf{x}, t\right)\right),$$

where τ_T is the relaxation time for the temperature; T_i is the temperature distribution function corresponding to the direction $\mathbf{e}_i$; and $T_i^{(eq)}$ is the equilibrium distribution function given by

$$T_i^{(eq)} = \frac{T}{6}\left(1 + 2\frac{\mathbf{e}_i \cdot \mathbf{u}}{c}\right).$$

The fluid temperature is calculated from the zero moment of the temperature distribution functions:

$$T = \sum_{i=1}^{6} T_i.$$

The thermal diffusivity $\mathcal{D}$ of the fluid is linked to the temperature relaxation time by

$$\mathcal{D} = \frac{(2\tau_T - 1)}{6}\frac{\Delta x^2}{\Delta t}.$$

It can be shown[14] that the previous LBGK recovers the following temperature equation

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{u}T) = \mathcal{D}\nabla^2 T$$

with an error proportional to $\mathcal{O}\left(Ma^2\right)$ in space and proportional to $\mathcal{O}\left(Ma \cdot dt\right)$ in time.

In order to take account of the buoyancy effects the two lattice Boltzmann simulations are coupled via the Boussinesq approximation. With this approximation, it is assumed that all fluid properties (density, viscosity, thermal diffusivity) can be considered to be constant except in the body force term, where

the fluid density ρ is assumed to be a linear function of the temperature:

$$\rho = \rho_0 \left(1 - \beta \left(T - T_0\right)\right),$$

where ρ_0 and T_0 are respectively the average fluid density and temperature, β is the coefficient of thermal expansion. For details on the limitations of the Boussinesq approximation, the reader is referred to [16].

With the Boussinesq approximation, the gravity force may be rewritten as

$$\mathbf{G} = \rho_0 \mathbf{g} - \rho_0 \beta \left(T - T_0\right) \mathbf{g},$$

where $\mathbf{g}$ is the gravity acceleration vector. With ρ considered constant, equation (2) becomes simply,

$$\nabla \cdot \mathbf{u} = 0,$$

and after absorbing the first constant part of $\mathbf{G}$ into the pressure term, equation (3) becomes with the Boussinesq approximation:

$$\frac{\partial \mathbf{u}}{\partial t} + \left(\mathbf{u} \cdot \nabla\right) \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} - \mathbf{g} \beta \left(T - T_0\right).$$

In the LBM, the Boussinesq forcing term $\mathbf{F}_B = -\mathbf{g}\beta\left(T - T_0\right)$ is added to the right hand side of the LBGK equation (1),

$$f_i \left(\mathbf{x} + c\mathbf{e}_i \Delta t, t + \Delta t\right) = f_i \left(\mathbf{x}, t\right) - \frac{f_i - f_i^{(eq)}}{\tau} + F_i \, \Delta t, \tag{4}$$

where F_i is computed using the discretisation introduced in [17]

$$F_i = \left(1 - \frac{1}{2\tau}\right) \omega_i \left(\frac{\mathbf{e}_i - \mathbf{u}}{c} + \frac{\left(\mathbf{e}_i \cdot \mathbf{u}\right)^2}{c^2} \mathbf{e}_i\right) \cdot \mathbf{F}_B,$$

and the macroscopic fluid velocity $\mathbf{u}$ is redefined by the following:

$$\mathbf{u} = \frac{1}{\rho} \left(\sum_i \mathbf{e}_i f_i + \frac{\Delta t}{2} \mathbf{F}_B\right).$$

2.3. The Smagorinsky turbulence model

Although most current simulation software to study the airflow in data centers and hospitals are limited to steady flows using the RANS[1] method, these flows are in fact very dynamic as the load or the ventilation changes. In order to simulate dynamic and turbulent flows with the lattice Boltzmann method, the basic algorithm needs to be extended, as it is limited to low Reynolds number and it becomes unstable as the relaxation time τ in equation (1) approaches 1/2 (i.e., the viscosity tends toward 0). Resolving all the scales of a simulation,

[1] Reynolds Average Navier-Stokes

including the smallest ones, would require very fine lattices and very long computational times, thus making impossible the simulation in real-time. Instead a sub-grid model, like the Smagorinsky model[18], can be used to model the physical effects that the unresolved sub-grid motion has on the resolved fluid motion. This model uses a positive turbulent eddy viscosity, ν_t, to represent small scale energy damping. This viscosity ν_t is computed from the local stress tensor, $\bar{S}_{\alpha\beta}$, as follows,

$$\nu_t = C\Delta^2 \left|\bar{S}\right|, \tag{5}$$

where $C > 0$ is the Smagorinsky constant[2], Δ is the filter width[3], and $\left|\bar{S}\right| = \sqrt{2\bar{S}_{\alpha\beta}\bar{S}_{\alpha\beta}}$ is the magnitude of the local stress tensor

$$\bar{S}_{\alpha\beta} = \frac{1}{2}\left(\frac{\partial \bar{v}_\alpha}{\partial x_\beta} + \frac{\partial \bar{v}_\beta}{\partial x_\alpha}\right).$$

The total viscosity of the fluid equals the sum of the physical viscosity and the eddy viscosity

$$\nu_{total} = \nu + \nu_t.$$

Turbulent thermal flows affected by Boussinesq forces require another term to describe the effect of stratification on the subgrid kinetic energy. Following [19], this term is added to the stress tensor in equation (5),

$$\nu_t = C\Delta^2 \left(\left|\bar{S}\right|^2 + \frac{Pr}{Pr_t}\nabla T \cdot \frac{\mathbf{g}}{|\mathbf{g}|}\right)^{\frac{1}{2}} \tag{6}$$

This term can also be simulated in the LBM[20], but it requires to store the macroscopic temperature at each node of the lattice in order to compute ∇T with finite differences, and the terms in the square root need to be capped to remain positive. These added computations slow down the program while having a minor impact on the considered flows. Thus, the second term in equation (6) is neglected in this study.

In the LBM, the effect of the eddy viscosity is incorporated into a local relaxation time τ_S given by[21]

$$\tau_S = 3\nu_{total} + \frac{1}{2} = 3\left(\nu + C\Delta^2\left|\bar{S}\right|\right) + \frac{1}{2}.$$

This modified relaxation time is then used in the relaxation process of the LBGK equations; so each node of the lattice relaxes at different rates.

[2]The Smagorinsky constant C can take values in the interval 0.01, 0.05, which was shown to yield good modelling of the sub-grid vortices, but its actual value depends on the geometry of the system.

[3]The filter width is the scale at witch the sub-grid model is used, it corresponds to the lattice spacing (often equal to 1 in the lattice Boltzmann method).

The local stress tensor is relatively easy to compute within the LBM, compared to traditional schemes (requiring finite difference computations) and can be computed locally from the non-equilibrium stress tensor,

$$\bar{\Pi}_{\alpha\beta} = \sum_{i=1}^{18} \mathbf{e}_{i\alpha}\mathbf{e}_{i\beta}\left(f_i - f_i^{(eq)}\right),$$

where α and β run over the three spatial dimensions.

The intensity of the local stress tensor $\bar{S}_{\alpha\beta}$ is then computed as

$$\left|\bar{S}\right| = \frac{1}{6C\Delta^2}\left(\sqrt{\nu^2 + 18C^2\Delta^2\sqrt{\Pi_{\alpha\beta}\Pi_{\alpha\beta}}} - \nu\right).$$

The relaxation time for the temperature distribution functions can be obtained with the following formula[20],

$$\tau_T = 3\left(\mathcal{D} + \mathcal{D}_t\right) + \frac{1}{2} = 3\left(\mathcal{D} + C\Delta^2\left|\bar{S}\right|/Pr_t\right) + \frac{1}{2}.$$

Where Pr_t is the turbulent Prandtl number. It is a non-dimensional term defined as the ratio between the eddy viscosity and the heat transfer eddy diffusivity.

3. Implementation and optimisations on GPU

This section describes the numerical implementation of a real-time 3D LBM on a GPU using the Compute Unified Device Architecture (CUDA), in addition to optimisation strategies to achieve the highest computational efficiency.

Initiated in 2007 by the company NVIDIA, CUDA is a programming language designed around the concept of Single Instruction Multiple Data (SIMD) programming model and allows the use of NVIDIA GPUs for scientific computing. Although the algorithms and techniques presented in this section were implemented in CUDA and tested on a NVIDIA GPU, they should be amenable to other SIMD programming languages (e.g. OpenCL) as they do not rely on any functionality specific to NVIDIA cards.

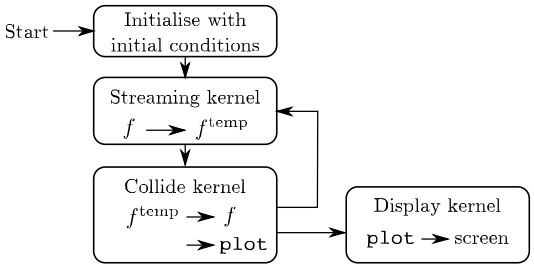


Figure 3: Main structure of the LBM program.

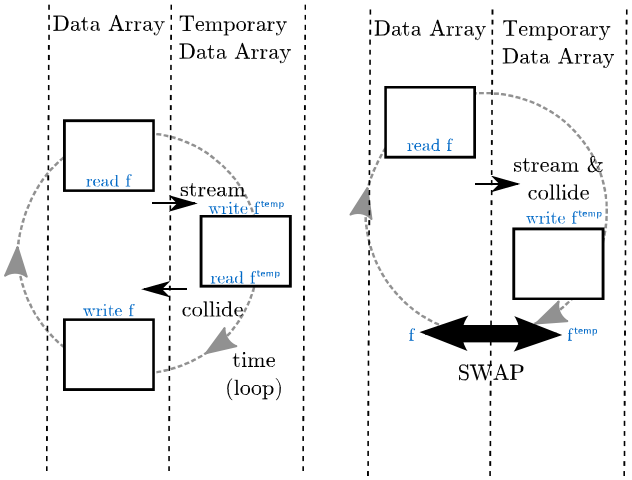


Figure 4: Blending the streaming et collision kernels together reduces the number of memory accesses by a factor of two.

The structure of the LBM software is represented in Figure 3. At the start of the program, two sets of 19 arrays, called $\mathbf{f}$ and $\mathbf{f}^{\text{temp}}$, are allocated and initialised on the CPU before being transferred to the GPU. The second set of distribution functions $\mathbf{f}^{\text{temp}}$ is used during the streaming-step to store the streamed distribution functions without overwriting the previous ones. Another array, called `plot`, is also allocated on the GPU in order to save the macroscopic quantity that the user wants to visualize in real-time. The program enters its main loop and calls recursively the streaming, collide, and display kernels. The streaming-kernel propagates the distribution functions stored in $\mathbf{f}$ into $\mathbf{f}^{\text{temp}}$. The collide kernel computes the equilibrium distributions from $\mathbf{f}^{\text{temp}}$ and saves the result of the collision operator back into $\mathbf{f}$, as well as the quantity for the visual display into the `plot` array. The `plot` array can later be displayed on the user's screen by the display kernel.

3.1. Minimise memory accesses

The main drawback that rapidly appears when implementing the LBM on GPU is the memory bandwidth, i.e., accessing data in the GPU memory takes longer than the actual computations on these data. Thus, one should focus on increasing the memory bandwidth in order to implement an optimised LBM program. An easy and efficient way to increase the memory throughput of a program is to reduce the number of accesses to global memory by avoiding redundant accesses to data.

In an LBM program, the number of global memory accesses can be reduced by a factor of two by blending the two streaming and collide kernels into a single big kernel, as presented in Figure 4. Indeed, when using two kernels, the program performs two read-accesses and two write-accesses (for each distribution function

f_i), while using a single stream&collide kernel only requires one read-access to f_i and one write-access to f_i^{temp}, the two sets of arrays are then swapped (this is a very fast operation that doesn't require any memory transfer).

It is worth noting that the stream&collide kernel should save intermediary results (like for instance the equilibrium distribution functions), as saving them in global memory would impair the optimisation. The registers are a set of very-fast, low-latency memories, only available in limited quantity. Any variable defined locally in a kernel will be stored in a register.

3.2. Increase data coalescence

Another efficient way to increase the memory throughput of a GPU program is to ensure memory accesses are *coalesced*. A memory access pattern is called *coalesced* when all the threads in a block access consecutive memory locations. In this case, the accesses are combined into one single request by the hardware. This is a limitation that the CPU does not suffer from, because it makes use of cache memory and has more sophisticated control units than the GPU.

An example of an uncoalesced memory access pattern is the so-called Array of Structures (AoS), described in Listing 1. The 19 distribution functions are grouped together in a structure called `Node`, then all the nodes are aligned in a one-dimensional array.

Listing 1: Array of Structures

```
//regroup 19 float variables together
struct Node { float f0, f1, f2, ..., f18; }
//create an array of Nx*Ny*Nz nodes
Node Lattice[Nx*Ny*Nz];
//access the fifth distribution on the node (i,j,k)
float value = Lattice[i+j*Nx+k*Nx*Ny].f5
```

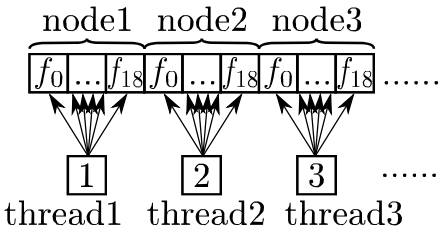


Figure 5: Memory access pattern for an Array of Structures

With the AoS pattern, each thread accesses a local area of the memory, as depicted in Figure 5. This access pattern is not coalesced, as threads are not accessing consecutive memory locations. For example, the accesses to the distribution f_0 by each thread are separated in memory space by $18 \cdot 4 = 72$ bytes (the 18 distributions f_1 to f_{18}) and cannot be grouped into a single big memory access. They need to be serialized into a lot of small accesses, which

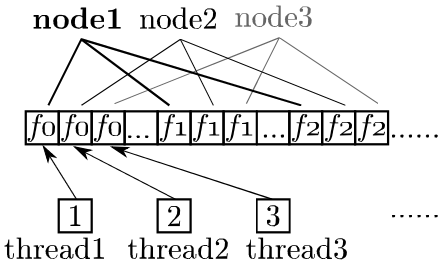


Figure 6: Access pattern for a Structure of Arrays.

significantly slows down the execution of the kernel. One should notice that this pattern would be perfect for a CPU implementation because the main thread would access the nodes one by one, and distribution functions would be likely to be stored in fast cache memory using this pattern.

A more useful example is the Structure of Arrays pattern (SoA), described in listing 2, which allows coalesced access to the memory. In this pattern, all the distribution functions are grouped together into a set of 19 arrays, based on their corresponding direction $\mathbf{e}_i$.

Listing 2: Structure of Arrays

```
//regroup 19 float arrays of size Nx*Ny*Nz together
struct Lattice
{
  float f0[Nx*Ny*Nz];
  float f1[Nx*Ny*Nz];
  ...
  float f18[Nx*Ny*Nz];
}
//access the fifth distribution on the node (i,j,k)
float value = Lattice.f5[i+j*Nx+k*Nx*Ny];
```

As shown in Figure 6, with an SoA pattern the set of 18 distribution functions defining a node are not grouped together, instead they are spread throughout the GPU memory. Memory accesses are now coalesced, and consecutive threads access consecutive memory addresses, therefore they can be grouped into one single big access (actually, one per distribution function).

In order to ensure coalesced memory access in an LBM program, it is recommended to match the layout of the memory and the layout of the threads. The three-dimensional arrays containing the distribution functions are physically stored as one-dimensional arrays. The memory alignment follows the standard alignment is C programming: first aligned in the x-direction, then in the y-direction and finally in the z-direction. The following code snippet shows how to access the element (x, y, z) of the three-dimensional array "`array`" using this alignment:

```
//compute the 1D index corresponding to (x,y,z)
int index = x + y * Nx + z * Nx*Ny;
//access the element (x,y,z) in the array
array[index] = ...
```

The threads are organised in a similar way to the memory, as shown in Figure 7: threads are aligned along a one-dimensional block of size N_x and the blocks of threads are aligned on a two-dimensional grid of size (N_y, N_z). Threads could also be organised on 2D blocks of size (N_x, N_y) aligned on a 1D grid of size N_z, but the number of threads in a block is limited[4] so this is only possible for low resolution simulations.

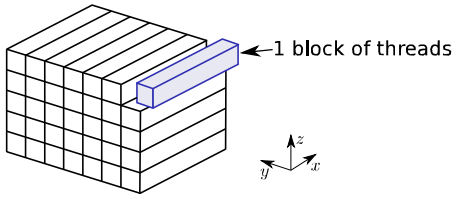


Figure 7: 3-dimensional threads layout.

The following code snippet summarizes how to compute the 3D position of a thread and the corresponding memory index in the CUDA programming language.

Listing 3: Thread organisation in CUDA.

```
__global__ void ExampleKernel(real* array, int Nx, int Ny, int Nz)
{
  // compute the 3D position of the thread
  int x = threadIdx.x;
  int y = blockIdx.x;
  int z = blockIdx.y;
  // compute the corresponding 1D index
  int index = x + y * Nx + z * Nx*Ny;
  //access the element (x,y,z) in the array
  array[index] = ...
  ...
}

int main(void)
{
  ...
  // define grid and block sizes
  dim3 block_size(Nx, 1, 1);
```

[4]There is a maximum of 1024 threads per block on the Tesla C2070.

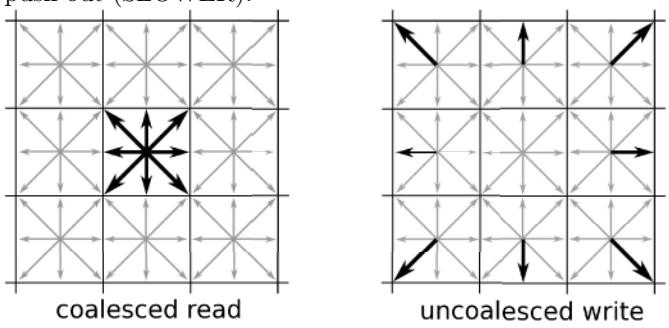
```
  dim3 grid_size(Ny, Nz, 1);
  // launch the kernel
  ExampleKernel <<<grid_size, block_size>>> (array, Nx, Ny, Nz);
  ...
}
```

3.3. The streaming issue

Although using coalesced memory enables an important gain in memory bandwidth, with a corresponding increase in program speed, the LBM algorithm cannot be fully coalesced because of the streaming step, which requires access to the distribution functions of the neighbouring nodes, hence breaking the coalesced access pattern. Therefore, the streaming step is the most critical step in terms of computational time and limits the performances of the stream&collide kernel.

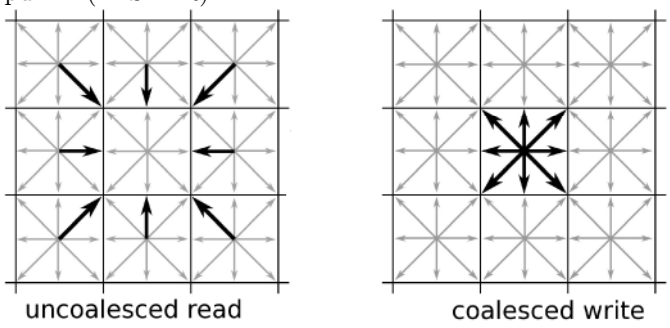
But after observing that uncoalesced reads are faster than uncoalesced writes, the uncoalesced memory accesses performed in the streaming-step can be dealt with more efficiently via a different interpretation of the streaming. Indeed, the streaming of the distribution functions can achieved in two different ways:

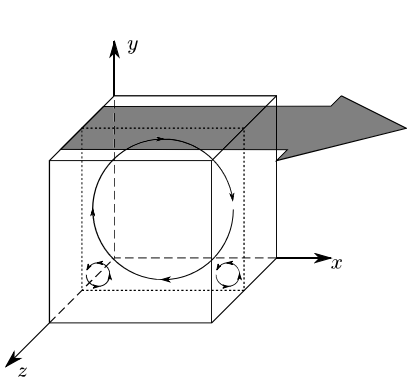
- push-out (SLOWER):



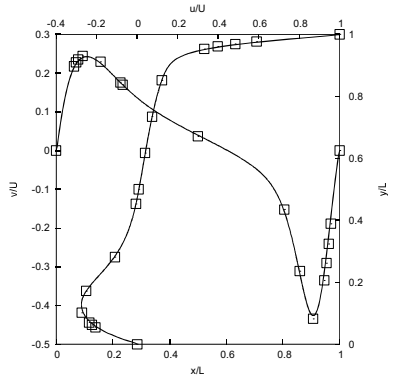
  The distribution functions are pushed from the centre node to the adjacent nodes. This involves local reads (so coalesced) of distribution functions and non-local writes (so uncoalesced).

- pull-in (FASTER):





(a) Geometry of the cavity.



(b) Velocity profiles through cavity geometric centre at $Re = 1000$.

Figure 8: 3D lid-driven cavity

The distribution functions are pulled from the neighbouring nodes to the centre node. This involves non-local uncoalesced reads and local coalesced writes.

Because uncoalesced reads are faster than uncoalesced writes, the pull-in streaming method is faster than the push-out method. Numerical experiments using a Tesla C2070 card showed that the pull-in method is about 6% faster than the push-out method. The speed-up on this hardware is rather small because the Fermi architecture from NVIDIA uses cached global memory access which tends to hide uncoalesced accesses. Thus, this speed-up is likely to vary depending on the hardware and should be more important for older GPU architecture.

4. Results and discussion

4.1. 3D lid-driven cavity

In order to first validate the results of the LBM program at low Reynolds number, the standard lid-driven 3D cavity benchmark was simulated. In this benchmark, a 3D cavity contains an incompressible viscous fluid and the flow is driven by the constant translation of the top lid. The boundary condition on the top-lid is: $\mathbf{u}\,(y = L) = U_{\text{lid}}\mathbf{e}_x$ and the no-slip boundary conditions on the other walls are: $\mathbf{u} = \mathbf{0}$, the geometry is sketched in Figure 8a. Its popularity comes from its ability to generate rich flow structures while keeping a simple geometry and boundary conditions. Flow fields in the lid-driven cavity have been studied extensively both experimentally[22, 23] and numerically[24, 25, 26].

As shown in Figure 8b the results of the 3D LBM simulation running on a GPU are in good agreement with the benchmark results of [24], obtained through spectral analysis method.

4.2. Test chamber

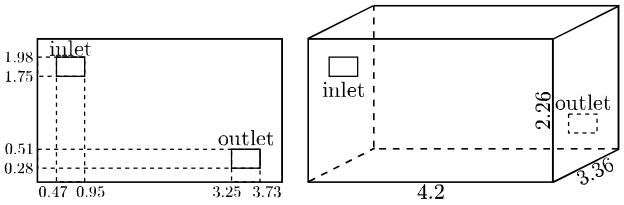


Figure 9: Geometry of the environment test chamber, left figure shows a front view, right figure shows a 3D representation.

In order to study the applicability of the previously described LBM program for real-world applications in real-time, the simulation of a $32\,m^3$ room filled with air (kinematic viscosity $\nu = 1.46\,10^{-5}\,kg.m^{-1}.s^{-1}$) is conducted. The simulated room is based on a real Class II bioaerosol chamber built in the School of Civil Engineering at the University of Leeds [27]. The results are compared with a Smagorinsky LES (Large Eddy Simulation) finite volume simulations using the ANSYS Fluent software[28]. The geometry of the room is shown in Figure 9, there is an inlet on one side pushing hot air into the room at a constant speed $U = 0.48\,m.s^{-1}$and a constant temperature $T_{in} = 22°C$. On the other side of the room, there is a free outlet (Neumann boundary). The walls of the room use a no-slip boundary condition and they are maintained at a temperature of $T_{wall} = 15°C$. The Reynolds number of the room computed from the hydraulic diameter of the inlet $L = 2\frac{0.23\times0.48}{0.23+0.48}\,m$ is $Re = 10200$.

The LBM simulation is performed on a regular mesh of size $160 \times 86 \times 127$ nodes to respect the room aspect ratios, this is a total of 1.7 million nodes. The inlet surface is composed of only 171 nodes, this could be improved in the future by using a refined mesh method [29, 30]. The speed at the inlet is $U_{LBM} = 0.1$ in lattice units, and the viscosity is computed to recover the proper Reynolds number in the room.

The ANSYS Fluent simulation is performed on a mesh composed of 534000 hexahedral cells, the mesh is refined such that the inlet surface contains 1100 nodes. The flow is simulated for a total of 560 physical seconds and the results are considered to be converged when the residuals of all the equations are less than 10^{-5}. The simulation is performed on a server with 16 Intel Xeon CPU and each physical second of simulation requires 7 minutes of computations.

Figure 10 shows a comparison of the average velocity profile in a cross section of the room that is aligned with the centerline of the inlet, as well as the average temperature profile. The fields are averaged from $t = 460\,s$ to $t = 560\,s$. The LBM simulation shows important fluctuations near the inlet but they are benign, as they do not propagate to the rest of the fluid and the main flow structures are still recovered. These fluctuations form a checkerboard pattern and are often seen as instabilities of the BGK collision operator for high Reynolds number flow. Using Figure 10, the profiles from the LBM simulation at 1.7 million nodes

can be compared with two other LBM simulations using respectively a lower and a higher resolutions (with respectively 0.1 million nodes and 7.2 million nodes). While the lower resolution is too coarse for a satisfactory solving of the flow structures, the higher resolution captures more details in the velocity profile and does not present instabilities; although without being significantly different from the results at $1.7M$ resolution. This show that the $1.7M$ resolution achieves a good trade-off between computational speed and accuracy. It is the authors belief that the current accuracy would be sufficient for use as an early warning system, and such accuracy would be improved with the use of a multiple relaxation-time approach [31, 32].

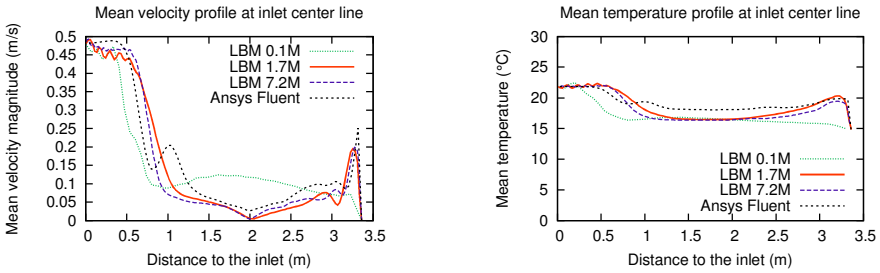


Figure 10: Comparison of the mean velocity and the mean temperature profiles averaged from $t = 460\,s$ to $t = 560\,s$ across the room, computed from LBM simulation and ANSYS Fluent simulation.

4.3. Performance

4.3.1. Observed performance

This section reports the observed performances of the 3D LBM program running on a GPU NVIDIA Tesla C2070 with 448 cores working at 1.15 GHz using the CUDA programming language version 4.1. A common method for comparing the performances of LBM programs is to measure the number of nodes updated per second. This number can be expressed in million of lattice-node updated per second, i.e., MLups. It depends on the type of lattice used, as the number of microscopic velocities of a node directly impacts the time required to update a node, so it should be noted that a D3Q19 lattice with 19 velocities is used for all the simulation tests. Moreover, on a GPU, the number of MLups also depends on the lattice resolution because the GPU performs better when it has more data to work on (as shown is Figure 11). Unless stated otherwise, the following performance comparison where obtained with a cubic lattice of size 128^3. The computations are always performed in single-precision, if computed in double precision, performances would be divided by a factor of two.

The poor performance on a 32^3 lattice is a result of the thread layout described in 3.2, using one-dimensional blocks of 32 threads is not enough for a proper use of GPU resources. For such small lattice sizes, two-dimensional

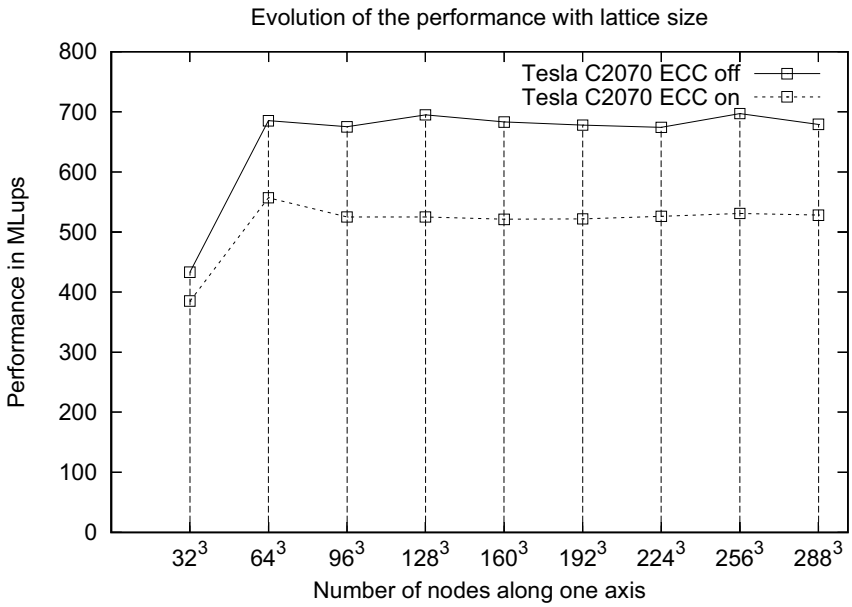


Figure 11: Performance fluctuations with lattice size

blocks of threads should be used in order to increase the number of threads per block, and improve the occupancy of the GPU.

The GPU was observed to achieve a sustained 530 MLups performance for the 3D isothermal LBM simulation. The 3D thermal simulation performs at 380 MLups, as it requires 6 additional distribution functions, and the program is bandwidth-limited. The turbulence model, however, does not add any new memory accesses but only increases the amount of computations required to evaluate the equilibrium distributions, thus it only slows down the program by 10 MLups, for both the thermal and isothermal models.

Ultimately, the performances for each model are summarised in Table 1, and compared with the performances with the ECC disabled (c.f. Section 4.3.4). The LBM program reaches a computational speed better or similar to other recent highly optimised GPU implementation published in the literature (even though in our implementation, some time is spent on the visualisation). For example J.Habitch et. al. [33] achieved 380 MLups (and 650 MLups with ECC disabled) for an isothermal D3Q19 LBGK model running on the same Tesla C2070 card model.

4.3.2. Discussion on real-time performance

This section discusses the feasibility of real-time indoor air flow simulations using the developed LBM program based on the test chamber discussed in Section 4.2. A fluid is simulated in real-time (or faster than real-time) if the physical

time between two simulation-steps is equal to (or bigger than) the time taken by the computer to simulate one time-step.

To compute the physical time corresponding to one time-step of the simulation, the "lattice-units" used by the program need to be converted to "real-world units". Each physical quantity is rescaled into a dimensionless quantity through a conversion factor, $Q_{\text{phys}} = C_Q Q_{\text{LBM}}$ where Q_{phys} is the physical quantity (with dimensions), C_Q is the conversion factor (with dimensions) and Q_{LBM} is the dimensionless quantity used in the LBM.

For length, time, and speed the following scaling can be written: $L_{\text{phys}} = C_L L_{\text{LBM}}$, $t_{\text{phys}} = C_t t_{\text{LBM}}$ and $V_{\text{phys}} = C_V V_{\text{LBM}}$. However, only two of these factors are independent thus the conversion factor for the time can be expressed as $C_t = C_L/C_V$.

Based on the velocity at the inlet, the speed conversion factor can be computed as $C_V = V_{\text{phys}}/V_{\text{LBM}} = 0.48\,\text{m.s}^{-1}/0.1 = 4.8\,\text{m.s}^{-1}$.

Based on the size of the cavity in meters in the direction orthogonal to the inlet (3.36 m) and the number of nodes along this direction (i.e.127), the conversion factor for lengths can be computed: $C_L = 3.36\,\text{m}/127 = 0.026\,\text{m}$.

Finally, the physical time Δt between two time-steps is given by $\Delta t = C_t = C_L/C_V = 5.5 \times 10^{-3}$ s.

To compute the speed-up, the physical time needs to be compared to the computational time. The flow in the test chamber can be simulated at up to 460 MLups (see Table 1), this means that the room made of 1.7 million nodes is updated 270 times per second, so each time-step is computed in 3.7×10^{-3} s. Computing the ratio to the physical time shows that a speed-up of 1.5 is obtained. Thus, the air-flow in this room can be simulated 1.5 times faster than the real flow.

4.3.3. Maximum theoretical performance

Observed performances can be compared with the theoretical performance. A LBM program running on a GPU is limited by its bandwidth, hence the theoretical performance can be obtained from the theoretical bandwidth divided by the number of bytes accessed in the global memory per iteration, which varies depending on the model. According to the specification, the GPU Tesla C2070 has a maximum theoretical bandwidth of $B_{\text{th}} = 144$ GB/s, this value is obtained by multiplying the number of memory interfaces (also known as buses) by the frequency at which they transfer data. In practice, the observed bandwidth is less than (and guaranteed not to exceed) the advertised bandwidth. Therefore, in order to obtain the effective bandwidth the "bandwidth-test" program, available in the CUDA Software Development Kit, was used. This program carries out simple memory transfers (read, write, copy) and computes an average bandwidth. The observed bandwidth for a GPU Tesla C2070 with this program is $B_{\text{SDK}} = 97$ GB/s, this is 32% less than the theoretical bandwidth B_{th}.

The maximum performance for a given model can be computed based on the observed bandwidth B_{SDK}. For example the D3Q19 isothermal model reads 19 distributions in $\mathbf{f}$ and writes 19 distributions in $\mathbf{f}^{\text{temp}}$ as well as 1 write in the `plot` array. Each of this access in single precision uses 4 Bytes, so

the D3Q19 isothermal model accesses a total of $4 \cdot (19 \cdot 2 + 1) = 156$ Bytes of memory per node. Based on this number and the observed bandwidth, the maximum performance for the D3Q19 isothermal model is 97 GB.s^{-1}/ (156 B) = 621 MLups. This number corresponds to the speed of the LBM program if all memory accesses are fully optimised and no computations are made. It is only 17% more than the actual performance of this model (530 MLups), this indicates that the program is well optimised. The maximum performance for the D3Q19 thermal model can be computed using the same technique, 97 GB.s^{-1}/ $(4 \cdot (2 \cdot (19 + 6) + 1)$ B) = 476 MLups, this is 25% more than the actual performance of this model (380 MLups). The thermal model, is relatively less efficient than the isothermal model because more computations are required comparatively (for instance to apply the Boussinesq forcing term).

4.3.4. Effect of the error-correcting code (ECC)

The observed memory bandwidth can be improved by disabling the Error-Correcting Code on the GPU. The ECC is a memory error checking system available only on high-end NVIDIA GPU, like the Tesla cards. When this mode is enabled, the GPU reserves 12% of the total memory and performs more operations in order to check that no error occurs during each memory access. Disabling this mode can obviously provide a performance boost but it may introduce errors in the simulation. The decision of whether or not to activate the ECC mode depends if the boost in performance is worth the loss in accuracy and stability in a simulation.

Experiments conducted on the LBM program showed no noticeable difference in the velocity and pressure field when disabling the ECC mode, the errors introduced by disabling ECC are smaller than discretisation errors. It is probable that the collision operator in the LBM has a "smoothing-effect" on the erroneous distribution functions by diluting them into the other 18 distribution functions of the node

Moreover, disabling the ECC significantly improves the performance by increasing the maximum memory bandwidth. Indeed, the bandwidth-test program achieves $B_{\mathrm{SDK2}} = 113$ GB/s (a 16% improvement) after disabling ECC. And the LBM isothermal model reaches 680 MLups (a 28% improvement). This LBM performance is only 6% less than the maximum performance computed from B_{SDK2} (724 MLups). The LBM program demonstrates a huge speed-up by disabling ECC, and it is also getting closer to the maximum memory bandwidth.

According to these results, it is advised to disable the ECC when running an LBM program on a GPU. The computational speed obtained with the ECC disabled are summarized in Table 1.

4.3.5. The issue of branch divergence

The use of flow control instructions (like an "if" condition) inside a kernel can cause branch divergences within each block of threads running this kernel and drastically slow down the program. This is because the GPU uses a SIMD programming model (where every thread executes the same instructions on a section of the data) and has limited control units. Implementing different actions

3D Model	Performances on GPU (MLups)	
	with ECC	without ECC
isothermal	530	680
isothermal and turbulent	520	670
thermal	380	470
thermal and turbulent	370	460

Table 1: Comparison of the performances of each model.

for each thread goes against the SIMD principle. However, some kind of flow control is always required to implement boundary conditions.

In practice, branch divergence will slow-down the LBM program when using complex boundary conditions instead of periodic boundary conditions (effectively no boundary). The actual value of the slow-down, depends on how early in the kernel the divergence happens, but it can be up to two times slower. However, this slow-down can be avoided if one direction can be kept as periodic. Indeed, branch divergence only happens within the same block of thread (actually it is within a warp, a group of 32 threads inside a block), so as long as the block is aligned with the boundary, there will be no divergence, as all the threads will execute the same boundary code. On the contrary, if the block of threads is normal to the boundary, the two threads on the edges execute the boundary code while all the other threads execute the bulk code, this creates branch divergence and slows down the computations.

In practice, using the thread organisation described in section 3.2, boundaries along the x-axis do not create branch divergences, execution is fast. Whereas boundaries perpendicular to the x-axis suffer from branch-divergence and slow down the execution of the program. This is represented in Figure 12.

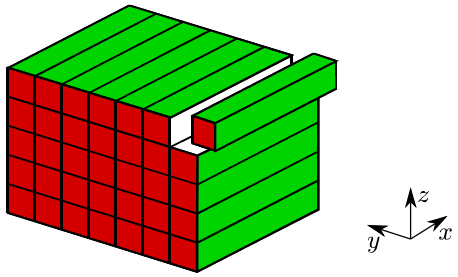


Figure 12: Effect of branch divergence on the boundaries. Green boundaries are fast, red boundaries are slow.

Periodic boundaries do not occur in most engineering applications and objects can be placed inside the domain, so thread divergences are likely to arrise. However, the resulting performance degradation can be limited by aligning complex boundaries (i.e. those requiring numerous computations and logical operations) with blocks of threads.

5. Conclusion

This study presents several optimisation principles, leading to an efficient 3D LBM kernel for an NVIDIA GPU using CUDA. The proposed program reaches up to 680 MLups for an isothermal model (and 460 MLups for a thermal and turbulent model), and achieves 94% of the effective maximal memory bandwidth in single precision, which is the main limiting factor of the current generation of GPUs.

This near-optimal performance is made possible by a carefully chosen memory access pattern in order to improve data-coalescence. It was also demonstrated that the performance degradation caused by the branch divergences can be avoided if one direction can be setup as periodic.

Enabling ECC on the hardware significantly slows down the execution (by about 30%), and when disabled, no artefacts could be seen in the computed simulation results. Therefore, the implementation of the LBM presented in this paper does not require the use of ECC.

The high computational speed, combined with efficient visualisation techniques, allows for real-time fluid simulation with interactive computational steering.

The benchmark problem of a fluid in a cubed lid-driven cavity demonstrates excellent agreement with proven solutions. The simulation of the test chamber showed an acceptable agreement with the finite volume simulation via ANSYS Fluent, although fluctuations of the flow in the vicinity of the inlet were detected.

More accurate collision models (such as the multiple relaxation time model) are required to remove these fluctuations and improve the program stability. As highlighted in 4.3, the performance impact of using such models should be minimal [34], as they usually require more computations and not necessarily additional memory accesses.

References

[1] B. Sammakia, S. Bhopte, M. Ibrahim, Numerical modeling of data center clusters, in: Y. Joshi, P. Kumar (Eds.), Energy Efficient Thermal Management of Data Centers, 2012, pp. 335–382.

[2] Y. Li, G. M. Leung, J. W. Tang, X. Yang, C. Y. H. Chao, J. Z. Lin, J. W. Lu, P. V. Nielsen, J. Niu, H. Qian, A. C. Sleigh, H.-J. J. Su, J. Sundell, T. W. Wong, P. L. Yuen, Role of ventilation in airborne transmission of infectious agents in the built environment - a multidisciplinary systematic review, Indoor Air 17 (1) (2007) 2–18.

[3] C. Noakes, P. Sleigh, A. Escombe, C. Beggs, Use of CFD analysis in modifying a TB ward in Lima, Peru, Indoor and Built Environment 15 (41).

[4] M. Müller, D. Charypar, M. Gross, Particle-based fluid simulation for interactive applications, in: Proceedings of the 2003 ACM SIGGRAPH, 2003, pp. 154–159.

[5] J. Stam, Stable fluids, in: Proceedings of SIGGRAPH 99, 1999, pp. 121–128.

[6] B. Crouse, M. Krafczyk, S. Kühner, E. Rank, C. van Treeck, Indoor air flow analysis based on lattice Boltzmann methods, Energy and Buildings 34 (9) (2002) 941–949.

[7] C. van Treeck, E. Rank, M. Krafczyk, J. Tölke, B. Nachtwey, Extension of a hybrid thermal LBE scheme for large-eddy simulations of turbulent convective flows, Computers and Fluids 35 (8-9) (2006) 863–871.

[8] J. Linxweiler, M. Krafczyk, J. Tölke, Highly interactive computational steering for coupled 3D flow problems utilizing multiple GPUs, Computing and Visualization in Science 13 (7) (2010) 299–314.

[9] S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Numerical mathematics and scientific computation, Oxford University Press, USA, 2001.

[10] 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, Phys. Rev. 94 (1954) 511–525.

[11] S. Chapman, T. Cowling, The mathematical theory of non-uniform gases, 3rd Edition, Cambridge : Cambridge University Press, 1970.

[12] X. He, L. Luo, Lattice Boltzmann model for the incompressible Navier-Stokes equation, Journal of Statistical Physics 88 (3-4) (1997) 927–944.

[13] J. Wu, C. Shu, A solution-adaptive lattice Boltzmann method for two-dimensional incompressible viscous flows, Journal of Computational Physics 230 (6) (2011) 2246–2269.

[14] Z. Guo, B. Shi, C. Zheng, A coupled lattice BGK model for the Boussinesq equations, International Journal for Numerical Methods in Fluids 39 (4) (2002) 325–342.

[15] C. Obrecht, F. Kuznik, B. Tourancheau, J. Roux, The TheLMA project: A thermal lattice Boltzmann solver for the GPU, Computers and Fluids 54 (2012) 118–126.

[16] D. Tritton, Physical Fluid Dynamics (Oxford Science Publications), 2nd Edition, Oxford University Press, USA, 1988.

[17] Z. Guo, C. Zheng, B. Shi, Discrete lattice effects on the forcing term in the lattice Boltzmann method, Phys. Rev. E 65 (2002) 046308.

[18] J. Smagorinsky, General circulation experiments with the primitive equations, Monthly Weather Review 91 (3) (1963) 99–164.

[19] P. Sagaut, Large Eddy Simulation for Incompressible Flows, Springer, New York, 2006.

[20] H. Liu, C. Zou, B. Shi, Z. Tian, L. Zhang, C. Zheng, Thermal lattice-BGK model based on large-eddy simulation of turbulent natural convection due to internal heat generation, International Journal of Heat and Mass Transfer 49 (23-24) (2006) 4672–4680.

[21] S. Hou, J. Sterling, S. Chen, G. Doolen, A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows, Vol. 6 of Fields Institute Communications, AMS, Providence, 1996, pp. 151–166.

[22] N. G. T. C. K. Aidun, J. D. Benson, Global stability of a lid-driven cavity with throughflow: Flow visualization studies, Physics of Fluids 3.

[23] J. R. Koseff, R. L. Street, Visualization studies of a shear driven three-dimensional recirculating flow, J. Fluids Eng.-Transactions ASME 106 (1984) 21–29.

[24] S. Albensoeder, H. C. Kuhlmann, Accurate three-dimensional lid-driven cavity flow, J. Comput. Phys. 206 (2) (2005) 536–558.

[25] U. Ghia, K. Ghia, C. Shin, High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, Journal of Computational Physics 48 (3) (1982) 387–411.

[26] H. Ku, R. Hirsh, T. Taylor, A pseudospectral method for solution of the three-dimensional incompressible Navier-Stokes equations, Journal of Computational Physics 70 (2) (1987) 439–462.

[27] M. King, C. Noakes, P. Sleigh, M. Camargo-Valero, Bioaerosol deposition in single and two-bed hospital rooms: A numerical and experimental study, Building and Environment 59 (2013) 436–447.

[28] ANSYS Academic Research, Release 13, Help System, FLUENT User's Guide, ANSYS, Inc. Southpointe, 275 Technology Drive, Canonsburg, PA 15317 .

[29] O. Filippova, D. Hänel, Grid refinement for lattice-BGK models, Journal of Computational Physics 147 (1) (1998) 219–228.

[30] M. Schönherr, K. Kucher, M. Geier, M. Stiebler, S. Freudiger, M. Krafczyk, Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs, Computers and Mathematics with Applications 61 (12) (2011) 3730–3743.

[31] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, L. Luo, Multiple-relaxation-time lattice Boltzmann models in three dimensions, Phil. Trans. R. Soc. A 360 (2002) 437–451.

[32] L. Luo, M. Krafczyk, J. Tölke, Large-eddy simulations with a multiple-relaxation-time LBE model, International Journal of Modern Physics B 17 (2003) 33–39.

[33] J. Habich, C. Feichtinger, H. Köstler, G. Hager, G. Wellein, Performance engineering for the lattice Boltzmann method on GPGPUs: Architectural requirements and performance results, Computers and Fluids 80 (2013) 276–282.

[34] C. Obrecht, F. Kuznik, B. Tourancheau, J. Roux, A new approach to the lattice Boltzmann method for graphics processing units, Computers and Mathematics with Applications 61 (12) (2011) 3628–3638.