# Entropic Lattice Boltzmann Simulation of Three Dimensional Binary Gas Mixture Flow in Packed Beds Using Graphics Processors

**Mohammad Amin Safi***
Institute of Applied Mathematics (LS III),
TU Dortmund,
Vogelpothsweg 87, D-44227, Dortmund, Germany
Email: amin.safi@math.tu-dortmund.de

*Corresponding author

**Mahmud Ashrafizaadeh**
Department of Mechanical Engineering,
Isfahan University of Technology,
Isfahan 84156-83111, Iran
Email: mahmud@cc.iut.ac.ir.

**Abstract**: The lattice Boltzmann method is employed for simulating the binary flow of Oxygen/Nitrogen mixture passing through a highly dense bed of spherical particles. Simulations are performed based on the latest proposed entropic lattice Boltzmann model for multi-component flows, using the D3Q27 lattice stencil. The curved solid boundary of the particles is accurately treated via a linear interpolation. To lower the total computational cost and time of the simulations, implementation on Graphics Processing Units (GPU) is also presented. Since the workload associated with each iteration is relatively higher than that of conventional 3D LBM simulations, special emphasis is paid in order to obtain the best computational performance on GPUs. Performance gains of one order of magnitude over optimized multi-core CPUs are achieved for the complex flow of interest on Fermi generation GPUs. Moreover, the numerical results for a three-dimensional benchmark flow show excellent agreements with the available analytical data.

**Keywords:** Lattice Boltzmann method; Entropic model; Binary mixture flow; Packed beds; Graphics processing unit; Optimized parallel algorithm.

**Biographical notes:** Mohammad Amin Safi is a PhD candidate at the Institute of Applied Mathematics, TU Dortmund. He received his Masters in mechanical engineering in 2010 from Isfahan University of Technology (IUT). His main research interest is lattice Boltzmann and finite element solutions of multi-phase and multi-component flows in porous media and high performance implementation of the solutions, particularly using the graphics processing units. He is also a former member and research assistant at the High Performance computing Center in IUT.

Mahmud Ashrafizaadeh is an Assistant Professor of Mechanical Engineering at the Isfahan University of Technology (IUT). He is also the founder and the former head of IUT's High Performance Computing Center. He received his PhD in mechanical engineering from the University of Waterloo in 1998. His

current research lies in the field of high performance computing for large scale turbulent and multi-phase flow simulations using lattice Boltzmann and finite volume methods.

## 1. Introduction

The complex process of binary mixture diffusion has several applications in fluidic systems such as chemically reacting flows, gas purification, pollutant dispersion and so forth. Diffusion of Oxygen and Nitrogen, discussed in the present work, is of a crucial importance since these two elements are the major constituents of atmospheric air and their diffusion plays a great role in combustion systems and Nitrogen/Oxygen purification processes. Of paramount importance for industry, is the flow of Oxygen and Nitrogen through highly packed beds of spherical particles, which is widely adapted for production of purified Oxygen or Nitrogen through an adsorption process. However, multi-component flows generally involve mass and momentum diffusion of two or more species dealing with complex geometries and/or boundary conditions such as adsorption and phase change. These conditions make the whole process difficult to be described and solved using the conventional continuum assumptions as they encompass many small scale transport phenomena (Nijemeisland and Dixon, 2004).

Recently, there has been a great interest in applying kinetic theory to capture the delicate diffusion processes in multi-component flows, and several models have been proposed to extend the lattice Boltzmann models to gas mixture problems. New consistent models for such flows (Arcidiacono et al., 2006a; 2006b; 2007; 2008) are based on the so-called entropic Lattice Boltzmann Method (LBM) and were successfully demonstrated to recover the Navier-Stokes and the Stephan-Maxwell diffusion equations. Despite the outstanding accuracy and flexibility of the new models, the entropic basis of the algorithm increases the computational cost of simulations to much more than twice for binary mixtures, as compared to single component Lattice Bhatnagar-Gross-Krook (LBGK) simulations. The situation aggravates from the computational point of view for the 3D flows as they require more spatial directions on each lattice point to recover the governing equations in the hydrodynamic limit.

Moreover, despite the well-known standard scheme of imposing the wall no-slip condition in conventional LBM, especial treatment for curved boundaries becomes necessary in order to maintain the accuracy of the solution when dealing with a multitude of arbitrary shaped solid particles. These modifications, in turn, increase the computational cost to at least one order of magnitude for geometries like highly packed beds.

On the other hand, due to its fully explicit nature of solution, lattice Boltzmann method is widely accepted to be a well suited candidate for massive parallelization. Besides using multi-core CPUs and compute clusters, many researchers have performed parallel LBM simulations on many-core Graphics Processing Units (GPU). Early implementations benefited from graphical languages such as OpenGL (Li et al., 2003; Zhu et al. 2008). By the advent of the modern programmable graphics cards and the new graphical programming language, CUDA, by nVIDIA, LBM-based computational fluid dynamics solvers have been extensively ported to graphics processors in order to facilitate the previously expensive, time-consuming flow simulations (Tölke and Krafczyk, 2008; Geveler et al., 2011). Fortunately, the new multi-component entropic model (Arcidiacono et al., 2006a; 2006b; 2007; 2008) still inherits its explicit nature and massive parallelization is viable through the new heterogeneous, general purpose GPU architectures.

### 1.1 Previous works

Considering the celebrated power of the Lattice Boltzmann method for complex geometries, the method has been also employed for the simulation of single and binary gas flows in packed beds. However, most of the implementations have assumed several simplifying approximations and were performed for packed beds of limited number and/or sizes of spheres. Maier et al. (1999) performed their simulations for the flow in a packed bed of spherical particles of two sizes. They prescribed a linear pressure drop in the axial direction and utilized the simple bounce-back scheme to satisfy the no-slip condition on the surface of the particles. Later on, Reynolds et al. (2000) studied the flow in a close-packed bed of spheres of the same size in a face-centered-cubic arrangement. To lower the total computational cost, they also avoided modifying the bounce-back condition on the spheres. The binary mixture flow through packed beds has also been investigated in the works by Manjhiet al. (2006a; 2006b) and Verma et al. (2007). In Manjhi and Verma (2006a; 2006b) the D3Q19 stencil is adapted to study 3D steady and unsteady velocity and concentration profiles in a tubular bed of spherical adsorbent particles. It is worth noting that all the mentioned studies have used simple regular packings with the diameter ratio of tube to particles $(d_t/d_p)$ being less than 10.

On the computational side, there have been several reports on porting conventional LBM solvers to graphics processors. Tölke (2010) was the pioneer in developing highly optimized LBM solutions on GPUs for 2D LBM using CUDA. Exploiting the fast shared memory space, he reported one order of magnitude performance gain compared to optimized multiple-CPU codes. Tölke and Krafczyk (2008) extended the implementations to 3D flows and reported efficiency gain of up to two orders of magnitude using the D3Q13 lattice stencil. Kuznik et al. (2010) employed a high-end platform equipped with nVIDIA GTX 280 to simulate lattice Boltzmann flows and even performed double precision calculations on their GPUs. Obrecht et al. (2011) exploited the D3Q19 stencil for their 3D simulations. They were able to extract more than %86 of the device throughput in their memory transactions without engaging the precious shared memory buffer. Safi et al. (2011) employed the entropic LBM scheme and developed a GPU code for simulating 1D binary diffusion of Oxygen and Nitrogen with zero bulk velocity. As a result, they could gain performance increases of more than one order of magnitude over single-core processors. Volkov (2010) has also described an efficient GPU implementation of the formidable D3Q27 stencil, with emphasis on following a reversed memory hierarchy to achieve high bandwidth usage and arithmetic intensity. As the same stencil has been employed in this work, we will briefly point to Volkov's strategies during the performance analysis part of section 5.

### 1.2 Paper Contribution and Overview

In this paper an efficient GPU based implementation of the Lattice Boltzmann method for steady state flow of Oxygen and Nitrogen through a packed bed of spherical particles is presented. In order to show the extent of improvement in the performance, our GPU implementation is further compared with an optimized multi-core CPU version based on a serial solver previously developed by Rastegari (2008). From the physical point of view, this work is distinguished from similar studies as it involves packed beds of $d_t / d_p$ being more than 30, which include more than 12000 particles in a variety of sizes. Moreover, no major physical simplification is introduced into simulations, and the no-slip boundary condition is accurately treated on the surface of the tube and on the particles, as well. As strictly required by the early entropic model of Arcidiacono et al. (2006a) we have successfully ported D3Q27 stencil to GPUs. On the computational side, conventional optimization techniques are considered along with a novel scaling scheme, so that calculations of both species could be performed in parallel on a single GPU.

In order to verify the accuracy of the presented solution, the 3D bulk flow of Oxygen and Nitrogen mixture in a tube has been selected as a benchmark and the results are compared to the available analytical data for this flow. The entropic lattice Boltzmann model for binary mixtures is described in section 2. A brief review on the GPU programming model is presented in section 3. Section 4 discusses the optimization strategies and the programming methodology applied to the present implementation. The results, including the benchmark and the main flow problem measurements as well as the performance metrics on different platforms are reported and analyzed in section 5. The paper conclusions are presented in section 6.

## 2. Lattice Boltzmann Model for Binary Mixtures

In the lattice Boltzmann model, the continuous spectrum of particle velocities is replaced by a set of discrete velocities of imaginary particles which are forced to move on a lattice structure only. A direct extension of the discretized velocity lattice Boltzmann equation for multi-component mixtures can be written as

$$\partial_t f_{ji} + c_{ji\alpha} \partial_\alpha f_{ji} = \Omega_{ji} \qquad (1)$$

Where $j = 1, ..., M$ and $M$ is the number of components in the mixture, $i = 0, ..., N$, with $N$ being the number of discrete lattice velocities $c_{ji\alpha}$, $\alpha = \{x, y, z\}$, and $\Omega_{ji}$ is the collision term. Equation (1) is further expanded based on fast-slow decomposition of motion in the vicinity of quasi-equilibrium state, as proposed by Gorban and Karlin (1994), which guarantees the positive entropy production of the process and hence satisfies the *H*-theorem. Therefore, the process is composed of a fast relaxation from the initial state $f$ to the quasi-equilibrium $f^*$, and then moving slowly from the quasi-equilibrium state $f^*$ towards the equilibrium $f^{eq}$. Expressing the two motions as BGK terms, the final collision term for the LB equation takes the following form for each species $j$ :

$$\Omega_{ji} = -\frac{1}{\tau_{1j}}(f_{ji} - f_{ji}^*) - \frac{1}{\tau_{2j}}(f_{ji}^* - f_{ji}^{eq}) \qquad (2)$$

where $\tau_{1j}$ and $\tau_{2j}$ are the two relaxation times corresponding to each of the relaxation stages, with the condition $\tau_{2j} > \tau_{1j}$ which ensures the stability of the model and puts a limit on the admissible Schmidt number $\nu / D_{AB}$, where $\nu$ is the kinematic viscosity and $D_{AB}$ is the binary diffusion coefficient. This restriction on the Schmidt number is further discussed at the end of this section. The moments of each component are defined as,

$$\rho_j = \sum_j f_{ji}^{eq} \quad , \quad J_{j\alpha} = \sum_i f_{ji} c_{ji\alpha}$$
$$P^{eq}_{j\alpha\beta} = \sum_i f^{eq}_{ji} c_{ji\alpha} c_{ji\beta} \quad , \quad Q^{eq}_{j\alpha\beta\gamma} = \sum_i f^{eq}_{ji} c_{ji\alpha} c_{ji\beta} c_{ji\gamma} \qquad (3)$$

where $\rho_j$, $J_{j\alpha}$, $P_{j\alpha\beta}$ and $Q_{j\alpha\beta\gamma}$ are the density, the momentum, the pressure tensor and the third order moment of component $j$, respectively. For the D3Q27 stencil (see Figure 1) considered in this paper, the discrete lattice velocities are

$$\vec{c}_{ij} = \begin{cases} (0,0,0) & i = 0 \\ (\pm 1,0,0)c_j,(0,\pm 1,0)c_j,(0,0,\pm 1)c_j & i = 1,\ldots,6 \\ (\pm 1,\pm 1,0)c_j,(0,\pm 1,\pm 1)c_j,(\pm 1,0,\pm 1)c_j & i = 7,\ldots,18 \\ (\pm 1,\pm 1,\pm 1)c_j & i = 19,\ldots,26 \end{cases} \qquad (4)$$

The lattice speed for each component is computed as $c_j = \sqrt{3k_B T_0 / m_j}$, where $k_b$ is the Boltzmann constant, $T_0$ is a reference temperature and $m_j$ is the molecular mass of component $j$. In the Entropic LB model proposed by Arcidiacono et al. (2006a) the equilibrium and quasi-equilibrium functions are obtained based on the minimization of the *H*-function defined by the following equation,

$$H = \sum_j \sum_i f_{ji} \ln \frac{f_{ji}}{W_i} \qquad (5)$$

which will be minimized under the constraints of conservation of density of each species, $\rho_j$ and the total mixture momentum $J_\alpha = J_A + J_B$. The cumbersome process of minimization in three dimensions which necessitates defining 27 spatial directions (resulting in the D3Q27 stencil (Rastegari, 2008)), leads to the following final equation for the equilibrium distribution functions,

$$f^{eq}_{ji} = \rho_j W_i \prod_{\alpha=1}^{d} \left( \frac{2c_j - \sqrt{c_j^2 + 3u_\alpha^2}}{c_j} \right) \left( \frac{2u_\alpha + \sqrt{c_j^2 + 3u_\alpha^2}}{c_j - u_\alpha} \right)^{\frac{c_{ji\alpha}}{c_j}} \qquad (6)$$

where $u_\alpha = J_\alpha / \rho$ is the mixture velocity in the $\alpha$ direction, and $\rho = \rho_A + \rho_B$. The corresponding weighting factors for each spatial direction are:

$$W_i = \begin{cases} \dfrac{8}{27} & i = 0 \\ \dfrac{2}{27} & i = 1,\ldots,6 \\ \dfrac{1}{54} & i = 7,\ldots,18 \\ \dfrac{1}{216} & i = 19,\ldots 26 \end{cases} \qquad (7)$$

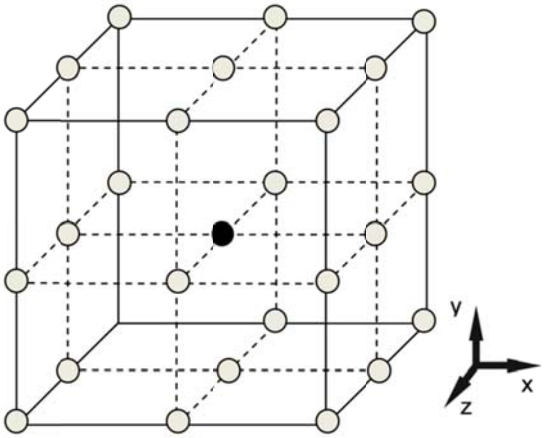


Figure 1. D3Q27 lattice stencil

The quasi-equilibrium function $f_{ij}^{*}$ can be obtained using two complementary approaches; taking either the difference between the momentum of the mixture and the momentum of each component i.e. $J_{\alpha} - J_{j\alpha}$, as an extra constraint for the minimization problem, or using the stress tensor, $P_{j\alpha\beta}$ as the extra quasi-equilibrium variable. The choice directly depends on the magnitudes of transport coefficients and the local concentration of the species. Using the difference between momentums as the extra constraint along with the previous conservation rules, one may derive the quasi-equilibrium function by merely substituting the velocity of the species $u_{j\alpha} = J_{j\alpha} / \rho_j$ instead of $U_{\alpha}$ in Equation (6)

$$f_{ji}^{*}(\rho_j, \vec{u}_j) = f_{ji}^{eq}(\rho_j, \vec{u}_j) \tag{8}$$

The relaxation times $\tau_{1j}$ and $\tau_{2j}$ for each component are related to the transport coefficients of the flow by using the Chapman-Enskog expansion in the hydrodynamic limit, adapting it to simplify the conservation equations for each component, and then comparing them to the standard Navier-Stokes and Stephan-Maxwell diffusion equations. The lengthy mathematical process is beyond the scope of this paper and the details can be found in Arcidiacono et al. (2006a; 2006b; 2007; 2008). The final set of equations for the relaxation times for each component are expressed as,

$$\begin{aligned} \tau_{1j} &= \frac{\mu_j}{n k_B T_0} \\ \tau_{2j} &= \frac{D_{AB} m_{AB}}{X_A X_B P} \end{aligned} \tag{9}$$

where $\mu_j$ is the dynamic viscosity of the component $j$, $X_A$ and $X_B$ are the mole fractions of components A and B, $n$ is the total number of moles in the mixture, $P = \sqrt{k_B T_0 n}$ is the mixture pressure, and $m_{AB} = \rho_A \rho_B / (\rho_A + \rho_B)$ is the reduced mass of the mixture. Considering these two equations, the aforementioned restriction on the Schmidt number can now be introduced. Rewriting the Schmidt number as,

$$Sc = \frac{\mu_j}{\rho_j D_{AB}} = \frac{\tau_1}{\tau_2} \frac{m_{AB}}{X_A X_B P} \tag{10}$$

The construction $\tau_{2j} > \tau_{1j}$ leads to the following inequality:

$$Sc_j = \frac{\mu_j}{\rho_j D_{AB}} \leq \frac{Y_A Y_B}{X_A X_B} \qquad (11)$$

where $Y_j$ is the mass fraction of the component $j$. Note that this restriction is a result of taking $J_\alpha - J_{j\alpha}$ as the extra quasi-equilibrium variable. The evaluation of the quasi-equilibrium function using the stress tensor $P_{j\alpha\beta}$ as the extra constraint leads to a complementary inequality,

$$Sc_j \geq \frac{Y_A Y_B}{X_A X_B} \qquad (12)$$

which results in covering any arbitrary Schmidt number. Deciding between the two models depends on which of the above inequalities is satisfied, considering the magnitudes of the species' transport coefficients (Arcidiacono et al., 2007).

### 2.1 Solid wall treatment

In LBM imposing the no-slip boundary condition on solid surfaces is mainly based on the bounce-back scheme on the solid nodes. In its simplest form, the bounce-back method needs to propagate the populations that are directed toward the solid nodes in the reverse directions. Despite the very straight forward procedure to impose this standard bounce-back, its accuracy for curved boundaries is questioned since it does not satisfy the no-slip condition on the desired location of a curved wall. Ladd (1994) showed that the standard bounce-back is of second order accuracy for straight walls when the wall is located exactly half-way between the solid and the fluid nodes, otherwise and in the case of a curved wall the accuracy of such a bounce-back will drop to first order (Yu et al., 2003).

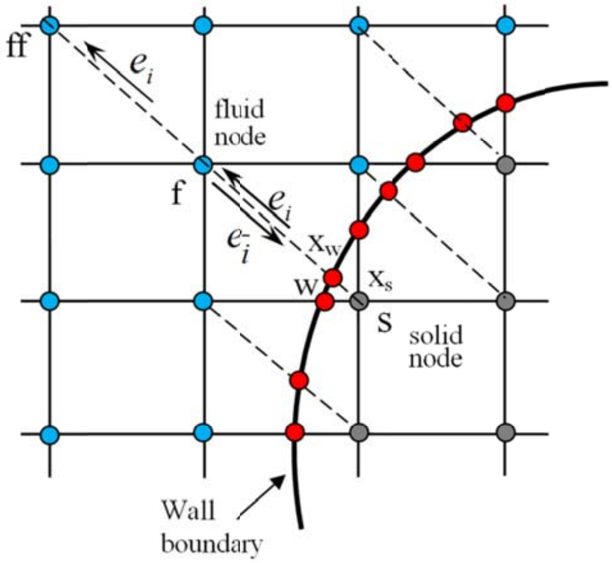


Figure 2. Schematic illustration of curved wall treatment in LBM discrete domain

To annihilate this deviation, several second-order accurate treatments have been suggested for curved entities (Filippova and Hänel, 1996; Mei et al., 1999; Bouzidi et al., 2001; Yu et al., 2002). One of the most accurate models, being relatively easy to implement, is the scheme proposed by Yu et al. (2002). As depicted in Figure 2, after the streaming step, $f_i(x_s)$ is known while the value of $f_{\bar{i}}(x_s)$ is still unknown. Having the exact location of the wall $x_w$, Yu et al. suggested the following linear extrapolation (or interpolation with regards to the internal nodes) to determine $f_{\bar{i}}(x_s)$ at $x_w$,

$$\begin{aligned} f_{\bar{i}}(x_f,t+\delta t)=\frac{1}{1+\Delta}\Big[&(1-\Delta).\,f_i(x_f,t+\delta t)+\Delta.\,f_i(x_s,t+\delta t) \\ &+\Delta.\,f_{\bar{i}}(x_{ff},t+\delta t)\Big] \end{aligned} \tag{14}$$

with $\Delta$ being defined as,

$$\Delta=\frac{\left|x_f-x_w\right|}{\left|x_f-x_s\right|} \tag{15}$$

Where $x_s$, $x_f$ and $x_{ff}$ are positions of $s$, $f$ and $ff$ as shown in Figure 2.The above treatment requires each fluid node to search in all spatial directions for any possible link to the neighboring solid nodes. Then, $\Delta$ has to be calculated for each of the links in order to perform the extrapolation in Equation (14) and determine the unknown populations. Considering the 27 spatial directions for $\alpha$, and the complex arrangement of the spherical particles in the packed bed, these calculations will account for a significant part of the total simulation time, as they ask for a large number of memory accesses to read data from the global memory of the graphics processor in a disordered pattern. The effect of the bounce-back step on the computational efficiency of GPU implementations will be further discussed later in section 5.

## 3. Efficient Implementation on GPU

CUDA technology is a computational architecture introduced by NVIDIA Corporation and includes computing innovations at both software and hardware levels. The nVIDIA's C for CUDA programming language is an extension to the conventional C language and allows the programmers to define new class of functions, called *kernels* which will be launched on GPU. By calling each kernel, *N* different CUDA threads will be distributed and executed in parallel on the hardware's large number of streaming processors as shown in Figure 3.

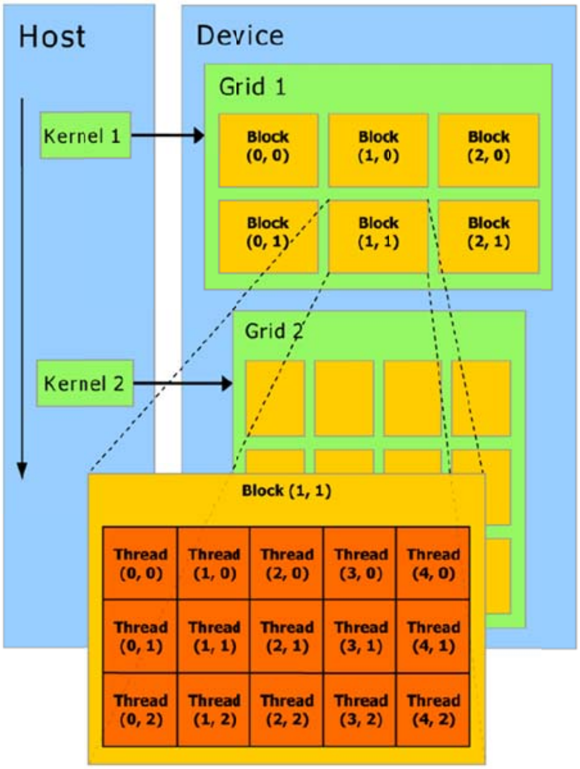


Figure 3. Grids of blocks and block of threads in the GPU programming model

Yet, a critical point in developing efficient CUDA applications is to appropriately manage the memory accesses between the device's global memory and the multiprocessors (a group of 32 process cores, e. g. in Fermi GPUs). Most importantly, the programmers should pay special attention to attain coalescing during memory accesses as further discussed below.

### 3.1 Memory access management

In order to efficiently exploit the relatively high effective bandwidth of the GPU devices, the hardware is designed to combine a number of memory calls requested by a half-warp (in Tesla GPUs) or one warp (in Fermi and Kepler GPUs) of threads (each warp equals 32 threads) into a single memory transaction. This behavior is often referred to as *coalescing*. However, coalescing requires special access patterns by the threads which is different for devices with and without caches;

In Tesla generation GPUs, the programmer had to consider alignment in his memory calls (nVIDIA, 2010a, 2010b). Memory alignment requires the pointer address value to be a multiple of the size of the type of the variable. In this pattern, each thread in a half-warp (a group of 16 threads), points to an aligned address in the global memory space which leads to a single 64-byte transaction, if the block sizes are chosen as multiples of 16. In contrast, a misaligned pattern of access results in 16 separate transactions executed in serial which drastically degrades the performance of the simulation and the memory bandwidth will drop to almost 10% of its maximum value (Tölke, 2010).

In the more recent Fermi and Kepler generation GPUs, the difficulty with the alignment is alleviated, thanks to the concept of cache-lines. Here the rule in data transfer is that the number of transactions is equal to the number of cache lines necessary to service an entire warp. In the simplest case, if memory request corresponding to one warp fits into one cache line, there would be one transaction for that warp meaning that the coalescing efficiency is 100% and we are close to the peak bandwidth of the device. However, if the requested memory space falls into *n* different cache lines, *n* transactions would be issued in serial, and thus the efficiency would drop to (100/*n*)%. Therefore, misaligned accesses are tolerated if the memory requests of a certain warp do not leak into a second cache line. The situation could become more complex for various access patterns and we refer to nVIDIA's documentations for more in-depth elaborations (nVIDIA, 2012a, 2012b).

### 3.2 Hardware Occupancy

Since each multiprocessor has a limited amount of registers and shared memory, it will have a certain capacity to accept threads. By definition, *occupancy* is the ratio of the number of active threads per multiprocessor to the maximum number of possible active threads (nVIDIA, 2012b). One can check for the amount of allocated registers and shared memory in his code by setting the `--ptxas-options=-v`, before compiling the code, and then put an appropriate bound on the maximum permissible register count using the `–maxrregcount` compiling option. It should be noted that decreasing the allocated registers for each thread to a minimum count (in order to maximize the occupancy) might degrade the computational performance of kernel instructions.

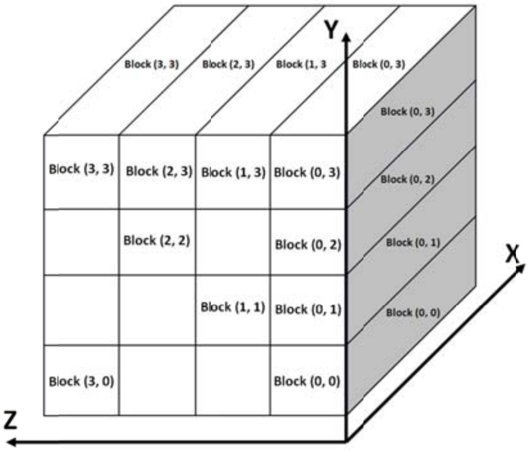


Figure 4. Configuration of the grid and its blocks in a typical 3D computational domain.

## 4. Implementation Outlines and Performance Optimizations

The first step in implementing LBM applications on GPUs is to map the computational domain into a grid of thread blocks. Here, the 3D scheme suggested by Tölke and Krafczyk (2008) is adapted. This scheme proposes a 2D grid of blocks, with each block being one dimensional, thus covering a three dimensional space. This requires that each row in the *X* direction represents a one dimensional thread block of the size of the domain in this direction, meaning that the size of the *X* dimension is equal to the block size. Eventually, a group of thread blocks will fit into the *Z*-*Y* plane as illustrated in Figure 4.

The next step is to allocate the distribution functions in the global memory. In order to satisfy the requirements mentioned in the previous section, most of the previous works (Tölke and Krafczyk, 2008; Tölke, 2010; Kuznik et al., 2010; Obrecht et al. 2011) have advocated the use of a Structure of Arrays (SOA), in which, each array is defined for one spatial direction. The SOA scheme guarantees that each thread would access the required memory address in an aligned pattern during the collision step and thus the memory requested by a warp fits into one cache line which is crucial for high bandwidth utilization in Fermi devices.

Finally, in order to fully satisfy efficient memory transaction guidelines on GPU, one has to also take care of the memory accesses during the streaming step. In this step, transfer of data for the rest particles and for $\pm Y$ , $\pm Z$ directions (a total of 9 directions in the *Z*-*Y* plane), automatically satisfies the memory coalescing requirements. For the other misaligned 18 directions; however, the fast shared memory is used as proposed by Tölke (2010) to prevent additional cache loading.

In addition to the above conventional optimizations, we have applied a new scaling scheme for the particular problem of interest in this paper. Here, we have two sets of distribution functions; one for Oxygen and the other for Nitrogen. Since updating the distributions for each species in a certain grid point in the computational domain is entirely independent from other species, they can be solved simultaneously. Therefore, one can set the device to also perform a job parallelization task on a single GPU and conduct the calculation of different components in parallel. To this end, two different approaches are viable:

- Launch and execute separate kernels for each component in parallel, using CUDA streams.
- Map the distribution functions of different components in a single grid of blocks and hence a single kernel.

The first approach does not guarantee a fully concurrent execution of the two kernels and the concurrency level is restricted by the computational resources of the device and the workload of each kernel. The second approach, however, ports a single grid of blocks on the device and hence the blocks corresponding to different components are launched simultaneously. For this purpose, here we have combined the two set of grids into a hybrid grid of twice the size of a regular 2D grid of blocks. In order to prevent warp divergence to select the right arrays for a specific component, we have modified the original SOA data structure such that we launch two arrays as arguments to the kernel; the first array contains the 27 distribution functions in SOA format (27 arrays combined into one) for the first component, followed by the distributions of the second component. By selecting the domain size to be a multiple of the warp size, this arrangement implies that the threads could automatically access their desired array elements while the nice coalescing properties are still preserved. Figure 5 shows how the separate grids of blocks will constitute a single grid in the proposed manner.

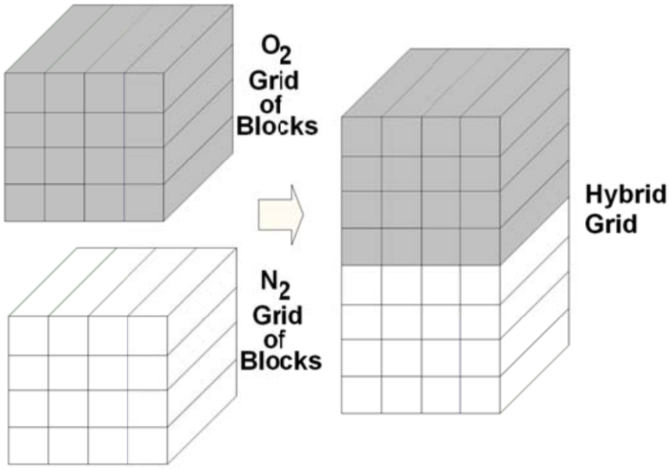


Figure 5. The grids corresponding to the two distinct computational domains, combined to form a single one.

The host code eventually launches the following kernels successively:

```
__global__ void collide_stream_knl(float *f_in, float *f_out, int *s);

__global__ void bounce_back_knl(float *f_out, int* s, float *d_sphere);

__global__ void BC_knl(float *f_out, int *s);
```

where `f_in` and `f_out` are the pointers to the incoming and outgoing distribution function arrays respectively, and `s` simply indicates whether the boundary state of the node is solid or fluid. The collision and streaming steps are conducted within a single kernel to avoid redundant memory transactions. However, the large number of instructions associated with the inlet and outlet boundary conditions are managed via a separate kernel (`BC_knl`), this way keeping the register and shared memory consumption of the kernels within a reasonable bound. In order to handle the linear extrapolation scheme and the multiple reads and writes of the modified bounce-back stage, a separate kernel is also considered for the bounce-back stage, named bounce_back_knl. Here `d_sphere` is a pointer to a structure containing geometry data (including center coordinates and diameter) of the spheres in the packed bed.

For the present study, we have analyzed several versions of our code, using the CUDA Visual Profiler provided by nVIDIA (2012c), to check whether the optimum memory throughput is achieved. The resulting memory throughput quality will be discussed in the next section.

## 5. Results and Discussion

### 5.1 Hydrodynamics

Using the binary lattice Boltzmann model described in section 2, simulations are first performed for the benchmark flow of Oxygen and Nitrogen mixture in an empty circular tube. Pressure boundary conditions are imposed at the inlet and outlet of the tube using the so-called Zou-He correlations modified for the multi-component D3Q27 lattice Boltzmann model (Rastegari, 2008). A slight pressure difference of $P_{out} / P_{in} = 0.99$ is imposed and the following uniform molar distribution is considered for the components at the inlet,

$$X_{O_2} = 47\% \quad , \; X_{N_2} = 53\% \qquad (16)$$

The computational domain includes 64 nodes in *X* and *Y* directions and 128 nodes along the *Z* axis. Both the simple and the modified bounce-back schemes are imposed on the interior wall of the tube. Figure 6 compares the bulk flow velocity profiles with the analytical parabolic velocity profile along the minor dimension of the tube, using single precision computations. While the profiles are generally in excellent agreement with the analytical parabolic answer, it can be seen that using the second order bounce-back clearly improves the results in the vicinity of the wall.

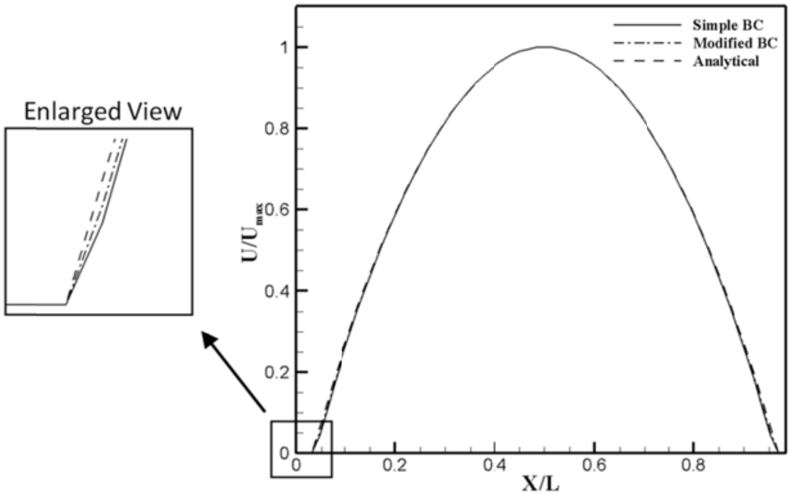


Figure 6. Comparison between the analytical and numerical velocity profiles.

The geometry of the main packed-bed problem consists of a circular tube within a box of size $128^2 \times 256$ in lattice units. The tube is filled with spherical particles of the size in the range $4 \le d_p \le 8$, resulting in a tube to particle diameter ratio $(d_t/d_p)$ of more than 30. The final bed configuration consists of 12753 particles of almost random sizes within the above mentioned range. The average porosity of the bed is almost 48% which gives a highly dense packed-bed. In order to let the flow develop and become uniform at inlet and outlet, open sections of 32 lattice units in length are considered at both ends.

The physical total inlet density is selected to be equal to the atmospheric air density at $1.2\ Kg/m^3$ and the molar fractions of $O_2$ and $N_2$ gases are,

$$X_{O_2} = 21\% \quad , \; X_{N_2} = 79\% \qquad (17)$$

In LBM, the pressure is related to density as,

$$P_{j\alpha\beta} = \rho_j \frac{c^2}{3} \qquad (18)$$

consequently, variations in density directly point to the pressure changes along the tube. The outlet total pressure is set to $P_{out} = 0.958 P_{in}$, which is a somewhat greater pressure difference as compared to that in the previous problem, thus giving the flow a boost to overcome the particle obstruction. The no-slip boundary condition is accurately treated on the solid surfaces using the second order bounce-back scheme of Yu et al. (2002) described in section 2. Simulations are performed until the following criterion for the residuals is met,

$$\left| \frac{c_t}{c_{t-1}} - 1 \right| \le 10^{-7} \quad (19)$$

where $c_t$ is determined at time $t$ by,

$$c_t = \left( \frac{\sum_j \sum_{i=1}^{N} \sum_\alpha J_{j\alpha}^2(i)}{N} \right)^{\frac{1}{2}} \qquad (20)$$

And $N$ is the total number of fluid nodes in the domain.

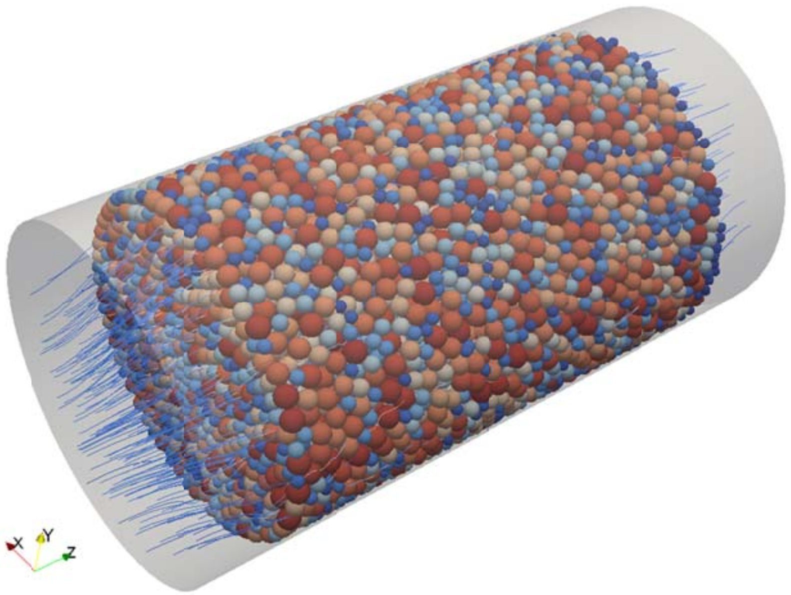


Figure 7. Illustration of the packed bed and the bulk flow streamlines.

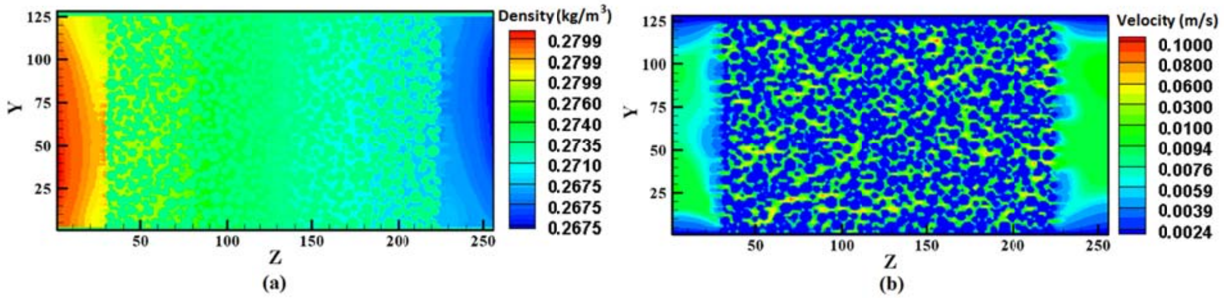
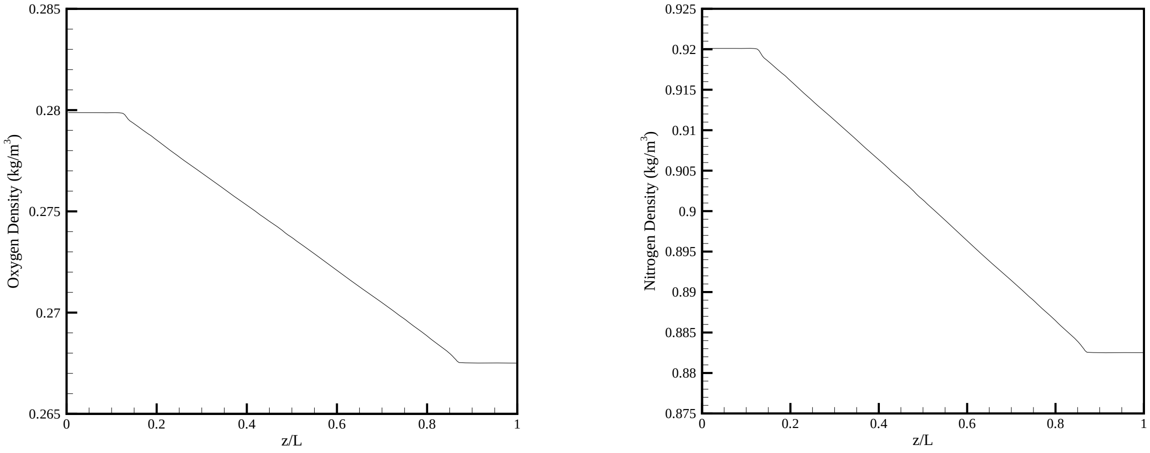


Figure 8. Density (pressure) and velocity contours for Oxygen in the cross section of the flow through the packed bed. (a) density contour, (b) velocity contour and the streamlines.

A 3D image of the bed geometry with the bulk flow streamlines is presented in Figure 7. Figure 8 shows the density and velocity contours for Oxygen. Figure 9 and Figure 10 show the X-Y plane averaged density and velocity variations along the tube, respectively. The resulted linear density (pressure) drop in Figure 9 agrees with the anticipated linear trend in homogeneous packed beds expressed by the so-called Ergun correlation (Freund et al., 2003)

$$\frac{\Delta P}{L} = f(a,b,c,...) \qquad (21)$$

where $f(a,b,c,...)$ is a function of density, velocity, fluid viscosity, size of the spheres and porosity of the packed bed and $L$ is the length of the bed. This agreement further validates the correctness of the present results. As anticipated, it is evident from Figure 10 that the lighter gas (Nitrogen) can attain higher velocities under the same driving force. Since no adsorption is applied on the surface of the spheres, molar distributions will remain unchanged throughout the tube.



(a) (b)

Figure 9. Variations in density along the packed bed for (a) Oxygen, and (b) Nitrogen.

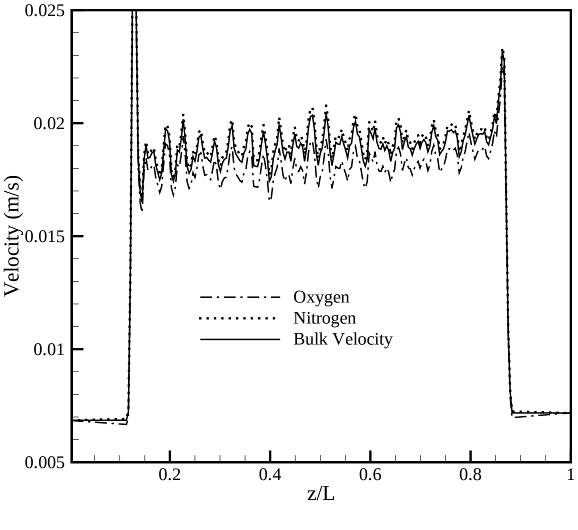


Figure 10. Average z-velocity fluctuations for both species and the bulk flow along the packed bed.

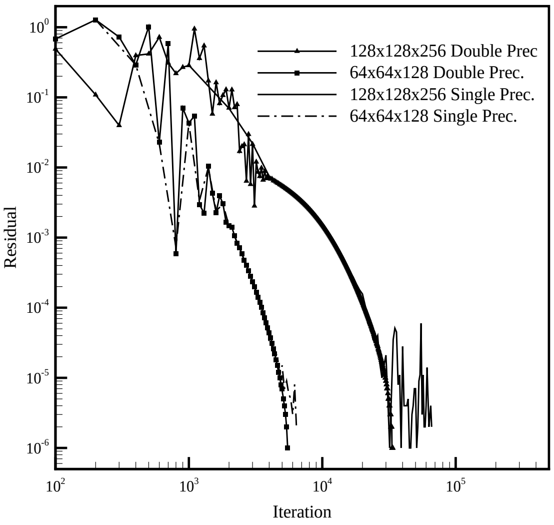


Figure 11. Convergence trend of the simulations for two different grid levels.

The convergence trend of the simulations is presented in Figure 11, for two different grid levels. The residual decay behavior depicts that the condition number of the problem becomes worse as the problem size increases. The quality of the single and double precision results is also compared in Figure 12 for the average axial bulk velocity along the tube on two grid levels. It can be seen that there is a slight difference in the results for each level. The relative error for the coarser level varies between the maximum of 4% at the inlet, to less than 0.05% in the bed zone, while these values would be slightly higher at 6% and 1% respectively for the finer level due to smaller time and length scales and hence less accuracy in the single precision mode. Yet, one can realize that for such complex flows in LBM, single precision computations can deliver relatively accurate results which suffice our numerical investigations.

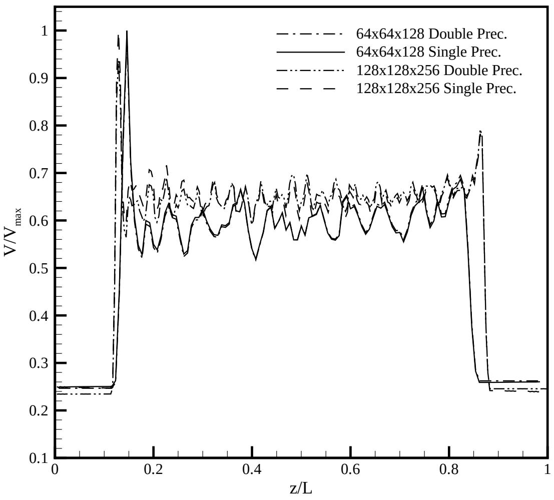


Figure 12. Averaged bulk velocity fluctuations along the packed bed using single and double precision computations for two different grid levels.

### 5.2 Computational Performance

We have run our implementations on three different configurations. A 6-core (2 threads each) Intel Xeon X5680 CPU processor at a clock rate of 3.33 GHz and peak memory bandwidth of 32 GB/s and 8 GB of memory space is selected for multi-core CPU implementations. On the GPU side two nVIDIA Fermi GPUs including a GTX-480 and a Tesla C2070 are employed. The latter is a high-end computing GPU with no graphics output. Table 1 presents a summary of the major specifications of the GPUs in use. Note that although the Tesla GPU is superior with regards to the available memory and peak double precision power (by factor of 4 and 3 respectively), its single precision support and the maximum bandwidth are still inferior to those for the GTX 480 by factors of 1.3 and 1.2 respectively.

Table 1: Major specifications of different GPU devices used in the present work. DP=Double Precision, SP=Single Precision, GFLOPS=Giga Floating point Operations Per Second, MTB=Maximum Theoretical Bandwidth, MEB=Maximum Effective Bandwidth

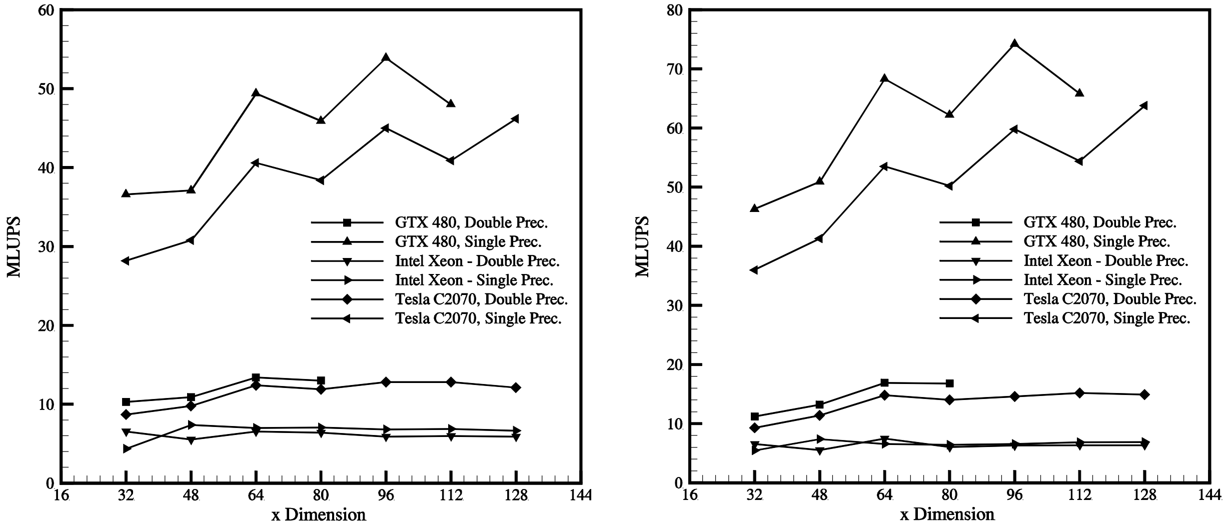
| GPU Type | No. of Thread Processors | Processor Clock(MHz) | Memory (GB) | Max DP GFLOPS | Max SP GFLOPS | MTB (GB/sec) | MEB (GB/sec) |
|---|---|---|---|---|---|---|---|
| GTX 480 | 15×32 | 70 | 1.5 | 168 | 1350 | 177.4 | 138 |
| Tesla C2070 | 14×32 | 57.5 | 6 | 515 | 1003 | 144.0 | 106 |

The above devices are employed via a 64-bit Ubuntu 12.04 operating system with Intel C compiler and CUDA 5.0 installed. The CPU code is thus compiled under *icpc* compiler with SSE 4.2 option activated and uses OpenMp directives for a multi-core implementation. The GPU code is compiled using *nvcc* command and the third level optimization flag (-*O3*) is turned on for both CPU and GPU codes.

The computational performance for different problem sizes on different Fermi GPUs as well as a multi-core CPU for the two problems discussed in the previous section (empty tube and packed bed) is presented in Figure 13. Note that the maximum problem size on each device is restricted by the amount of the available memory. The performance metrics for both problems are calculated in terms of Millions of

Lattice Updates Per Second (MLUPS). One could notice that an order of magnitude speedup is gained over an optimized 6-core CPU implementation using both GTX and Tesla GPUs in single precision mode. In double precision, the performances on both GPUs are around twice that for a multi-core CPU.

Although the 6 GB memory space of Tesla C2070 allows for implementing large problem sizes, yet, its double precision performance could not take over that of GTX 480. This seems to be against our expectations since the double precision arithmetic power of Tesla C2070 is 3.06 times higher than that for the GTX 480. This is mainly due to the fact that the current problem is memory bound rather than being compute bound, and thus, instead of the floating point power, the memory bandwidth is the key factor in limiting the performance. Therefore, the GTX 480 with 1.2 time higher bandwidth exhibits a slightly better performance (a factor of 1.08 for both problems).



(a) (b)

Figure 13. Computational performance on different platforms and different problem sizes. (a) flow through a packed bed, (b) flow in an empty tube.

A closer look at Figure 13 reveals that the CPU performance drop from the empty tube to the packed bed problem is not substantial, while on the GPU side the performance of the packed bed problem is significantly lower than that for the empty tube problem by a factor of ≈1.3. This performance loss is partly due to the warp divergence during the collision step to find the fluid nodes leading to warp serialization, but to some greater extent because of the disordered memory accesses and hence poor bandwidth usage in the bounce-back kernel. In a CPU implementation, however, the large available caches annihilate such effects and the performance difference is negligible.

Table 2. Single precision performance metrics for flow in a packed bed for different kernels on Tesla C2070. BW= Percentage of the achieved bandwidth to the device's maximum effective bandwidth. PTGT= Percentage of the Total GPU Time

| | 32x64 | 64x128 | 96x192 | 128x256 |
|---|---|---|---|---|
| Performance (MLUPS) | 28.2 | 40.6 | 45.0 | 46.2 |
| collide_stream_knl Max BW(%) | 62.2 | 90.5 | 88.7 | 91.3 |
| bounce_back_knl Max BW(%) | 15.0 | 28.0 | 32.1 | 33.6 |
| collide_stream_knl Occupancy (%) | 16.6 | 32.9 | 30.8 | 32,7 |
| bounce_back_knl Occupancy (%) | 16.6 | 31.3 | 44.0 | 56,6 |
| collide_stream_knl PTGT | 47.0 | 53.5 | 57 | 57.5 |
| bounce_back_knl PTGT | 50.7 | 45.1 | 42 | 41.5 |

In order to provide a better insight into the packed bed problem, various performance metrics as well as the kernel contribution to the total GPU time in the flow through packed bed are listed in Table 2 for different problem sizes on Tesla C2070. The data for the two most expensive kernels have been presented in this table. The best computational performance is achieved for the largest problem size of $128^2$×256 to around 46.2 MLUPS which best exploits the computing power of the GPU. A maximum of 91.3% of the effective memory bandwidth is attained in the collide_stream kernel which is comparable to the values reported for single component D3Q19 cavity flow by Obrecht et al. (2011) and also is close to the measurements in the D3Q27 stencil study by Volkov (2010). However, the highly erratic memory access pattern in the bounce_back kernel results in poor bandwidth efficiency for a packed bed problem. As the bounce-back kernel contributes to almost 40-50% of the total GPU time, optimizing this kernel with regards to memory accesses would lead to significant improvement in the overall performance.

For the collide_stream kernel the occupancy of the device is maintained in the range of 16% to 32%. The large number of memory accesses and instructions in D3Q27 LBM significantly increases the kernel’s register demand (53 registers in single and 63 in double precision), which restricts the maximum number of threads managed simultaneously by the multiprocessor, and lowers the multiprocessors occupancy. On the other hand, the relatively high performances achieved, admits the fact that launching a large number of blocks (y-dim×z-dim) of small sizes (up to 128 here) pays off for heavy stencils, e. g. D3Q27 by allowing each thread to compute more outputs and take advantage of the large on-chip register space available on Fermi devices. The idea was first introduced by Volkov (2010) where he showed that using more registers than shared memory and thus following an inverse memory hierarchy for heavy stencils, results in higher arithmetic intensity and bandwidth usage at a rather low occupancy (Volkov, 2010)

### 6. Summary and Conclusions

A 3D, 27 stencil lattice Boltzmann flow solver for the binary flow of Oxygen and Nitrogen mixture has been developed and implemented on GPUs. The no-slip boundary condition is treated accurately on the solid surface of the spherical particles which further increases the computational cost of the simulations in a dense packed bed. As such, and to exploit the maximum computational power of GPUs we used an optimized algorithm and scaled the work flow in a way that the transport equations for both species can be managed in parallel. It is shown that using modern many-core graphics processors, it is possible to obtain one order of magnitude faster simulations over our optimized multi-core CPU implementations for such subtle simulations. Yet, one could further enhance the computational efficiency by improving the memory access layout particularly in the expensive bounce-back kernel for packed bed problems. The single precision computations show to be very promising in providing relatively accurate results.

Although the problem size of the present simulations is limited by the available GPU memory resources, porting the problem to large GPU-based clusters alleviates such restrictions.

Another challenge for future work would be to simulate unsteady effects via adding adsorption properties to the spherical particles which requires modifying the surface boundary conditions. This, along with scaling the problem size to real dimensions, is certainly of high interest for the industry. Early results are very promising and are subject to our future publications.

## References

Arcidiacono, S., Ansumali, S., Karlin, I.V., Mantzaras, J. and Boulouchos, K.B. (2006) 'Entropic lattice Boltzmann method for simulation of binary mixtures', *Mathematics and Computers in Simulation*, vol. 72, no. 2-6, pp. 79-83.

Arcidiacono, S., Karlin, I.V., Mantzaras, J. and Frouzakis, C.E. (2007) 'Lattice Boltzmann model for the simulation of multi-component mixtures', *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, no. 4.

Arcidiacono, S., Mantzaras, J., Ansumali, S., Karlin, I.V., Frouzakis, C. and Boulouchos, K.B. (2006) 'Simulation of binary mixtures with the lattice Boltzmann method', *Physical Review E-Statistical, Nonlinear, and Soft Matter Physics*, vol. 74, no. 5.

Arcidiacono, S., Mantzaras, J. and Karlin, I.V. (2008) 'Lattice Boltzmann simulation of catalytic reactions', *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 78, no. 4.

Bouzidi, M., Firdaouss, M. and Lallemand, P. (2001) 'Momentum transfer of a Boltzmann-lattice fluid with boundaries', *Physics of Fluids*, vol. 13, no. 11, pp. 3452-3459.

Filippova, O. and Hänel, D. (1998) 'Grid refinement for lattice-BGK Models', *Journal of Computational Physics*, vol. 147, no. 1, pp. 219-228.

Freund, H., Zeiser, T., Huber, F., Klemm, E., Brenner, G., Durst, F. and Emig, G. (2003) 'Numerical simulations of single phase reacting flows in randomly packed fixed-bed reactors and experimental validation', *Chemical Engineering Science*, vol. 58, no. 3-6, pp. 903-910.

Geveler, M., Ribbrock, D., Mallach, S., Göddeke, D. (2011) 'A simulation suite for lattice-Boltzmann based real-time CFD applications exploiting multi-level parallelism on modern multi- and many-core architectures', *Journal of Computational Science*, vol. 2, no. 2, pp. 113-123.

Gorban, A. N. and Karlin, I.V. (1994) 'General approach to constructing models of the Boltzmann equation', *Physica A: Statistical Mechanics and its Applications*, vol. 206, no. 3-4, pp. 401-420.

Kuznik, F., Obrecht, C., Rusaouen, G. and Roux, J.-. (2010) 'LBM based flow simulation using GPU computing processor', *Computers and Mathematics with Applications*, vol. 59, no. 7, pp. 2380-2392.

Li, W., Wei, X. and Kaufman, A. (2003) 'Implementing lattice Boltzmann computation on graphics hardware', *Visual Computer*, vol. 19, no. 7-8, pp. 444-456.

Ladd, A.J.C. (1994) ‘Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 2. Numerical results’, *Journal of Fluid Mechanics*, vol. 271, pp. 311-339.

Maier, R.S., Kroll, D.M., Davis, H.T. and Bernard, R.S. (1999) ‘Simulation of flow in bidisperse sphere packings’, *Journal of colloid and interface science*, vol. 217, no. 2, pp. 341-347.

Manjhi, N., Verma, N., Salem, K. and Mewes, D.(2006) ‘Lattice Boltzmann modeling of unsteady-state 2D concentration profiles in adsorption bed’, *Chemical Engineering Science*, vol. 61, no. 8, pp. 2510-2521.

Manjhi, N., Verma, N., Salem, K. and Mewes, D. (2006) ‘Simulation of 3D velocity and concentration profiles in a packed bed adsorber by lattice Boltzmann methods’, *Chemical Engineering Science*, vol. 61, no. 23, pp. 7754-7765.

Mei, R., Luo, L.-. and Shyy, W. (1999) ‘An accurate curved boundary treatment in the lattice Boltzmann method’, *Journal of Computational Physics*, vol. 155, no. 2, pp. 307-330.

Nijemeisland, M. and Dixon, A.G. (2004) ‘CFD study of fluid Flow and wall heat transfer in a fixed bed of spheres’, *AIChE Journal*, vol. 50, no. 5, pp. 906-921.

nVIDIA (2010) *CUDA Programming Guide,V3.0.*

nVIDIA (2010) *CUDA Best Practice Guide, V3.0.*

nVIDIA (2012) *CUDA Programming Guide,V5.0.*

nVIDIA (2012) *CUDA Best Practice Guide, V5.0.*

nVIDIA (2012) *CUDA Compute Profiler,V5.0.0.*

Obrecht, C., Kuznik, F., Tourancheau, B. and Roux, J.-. (2011) ‘A new approach to the lattice Boltzmann method for graphics processing units’, *Computers and Mathematics with Applications*, vol. 61, no. 12, pp. 3628-3638.

Rastegari, A. (2008) ‘Simulation of gaseous mixtures in a packed bed of unfixed particles’, M.Sc. thesis, Department of Mechanical Engineering, Isfahan University of Technology, Isfahan, Iran.

Reynolds, A.M., Reavell, S.V. and Harral, B.B. (2000) ‘Flow and dispersion through a close-packed fixed bed of spheres’, *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, vol. 62, no. 3 A, pp. 3632-3639.

Safi, M.A., Ashrafizaadeh, M. and Ashrafizaadeh, A.A. (2011) ‘Lattice Boltzmann simulation of binary mixture diffusion using modern graphics processors’, *World Academy of Science, Engineering and Technology*, vol. 73, pp. 875-882.

Tölke, J. (2010) ‘Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA’, *Computing and Visualization in Science*, vol. 13, no. 1, pp. 29-39.

Tölke, J. and Krafczyk, M. (2008) ‘TeraFLOP computing on a desktop PC with GPUs for 3D CFD’, *International Journal of Computational Fluid Dynamics*, vol. 22, no. 7, pp. 443-456.

Verma, N., Salem, K. and Mewes, D. (2007) 'Simulation of micro- and macro-transport in a packed bed of porous adsorbents by lattice Boltzmann methods', *Chemical Engineering Science*, vol. 62, no. 14, pp. 3685-3698.

Volkov, V. (2010) 'Programming inverse memory hierarchy: case of stencils on GPUs', *International Conference on Parallel Computational Fluid Dynamics 2010 (ParCFD 2010) List of Presentations, Computers & Fluids,* vol. 45, no. 1, pp. 283-295

Yu, D., Mei, R., Luo, L. S. and Shyy, W. (2003) 'Viscous flow computations with the method of lattice Boltzmann equation', *Progress in Aerospace Sciences*, vol. 39, no. 5, pp. 329-367.

Yu, D., Mei, R. and Shyy, W. (2002) 'A unified boundary treatment in lattice Boltzmann method', *American Physical Society, Division of Fluid Dynamics 55th Annual Meeting*, abstract #JA.011.

Zhu, H., Liu, X., Liu, Y. and Wu, E. (2006) 'Simulation of miscible binary mixtures based on lattice Boltzmann method', *Computer Animation and Virtual Worlds*, vol. 17, no. 3-4, pp. 403-410.

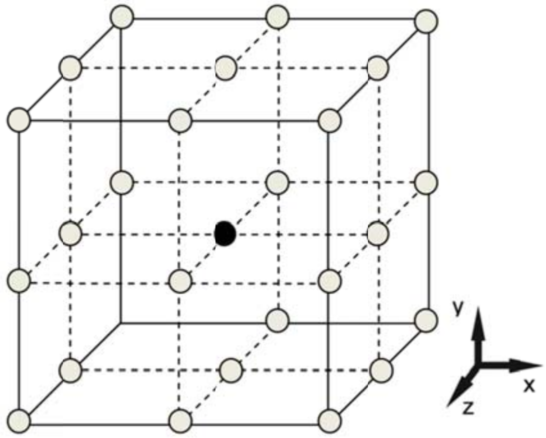


Figure 1. D3Q27 lattice stencil

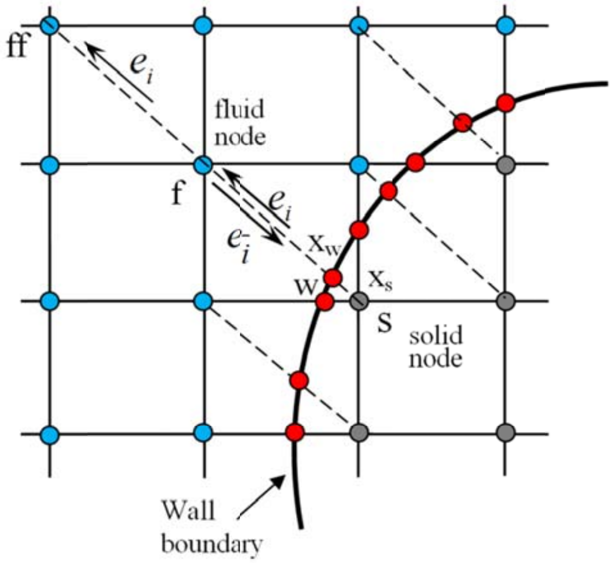


Figure 2. Schematic illustration of curved wall treatment in LBM discrete domain

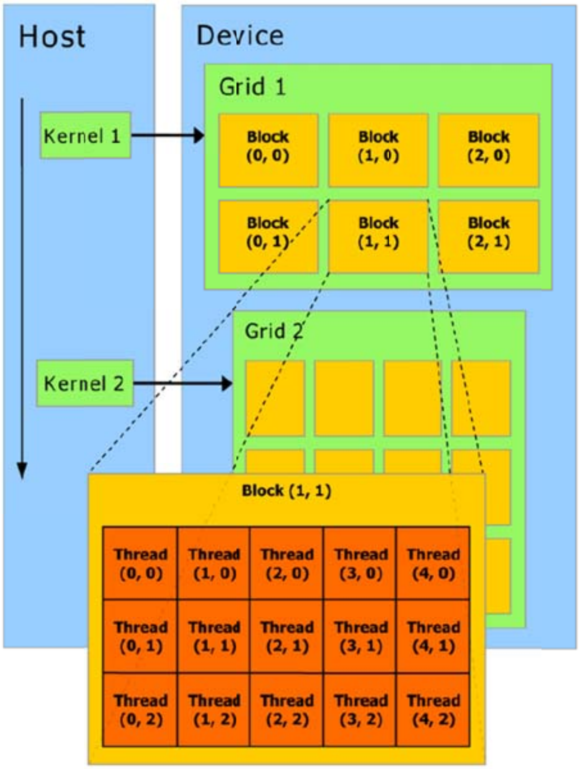


Figure 3. Grids of blocks and block of threads in the GPU programming model

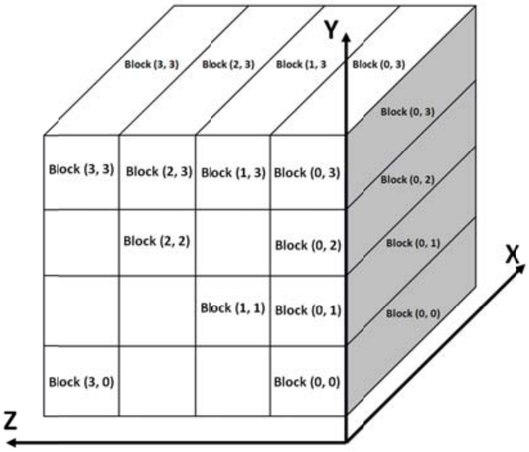


Figure 4. Configuration of the grid and its blocks in a typical 3D computational domain.

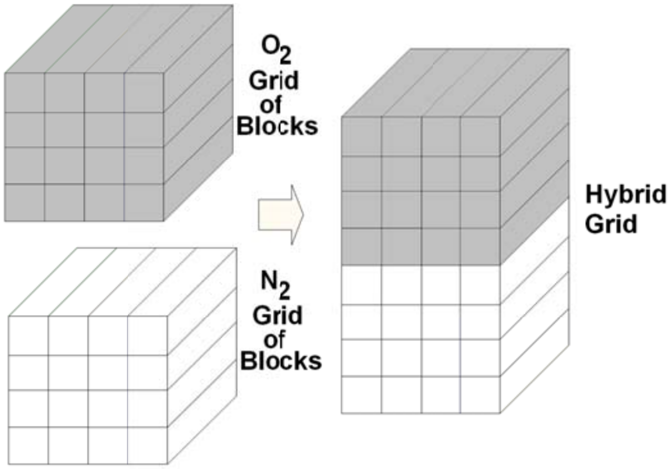


Figure 5. The grids corresponding to the two distinct computational domains, combined to form a single one.

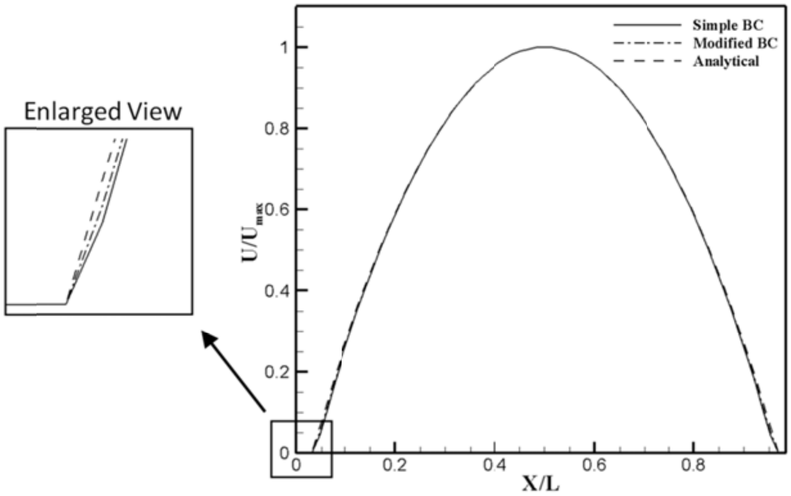


Figure 6. Comparison between the analytical and numerical velocity profiles.

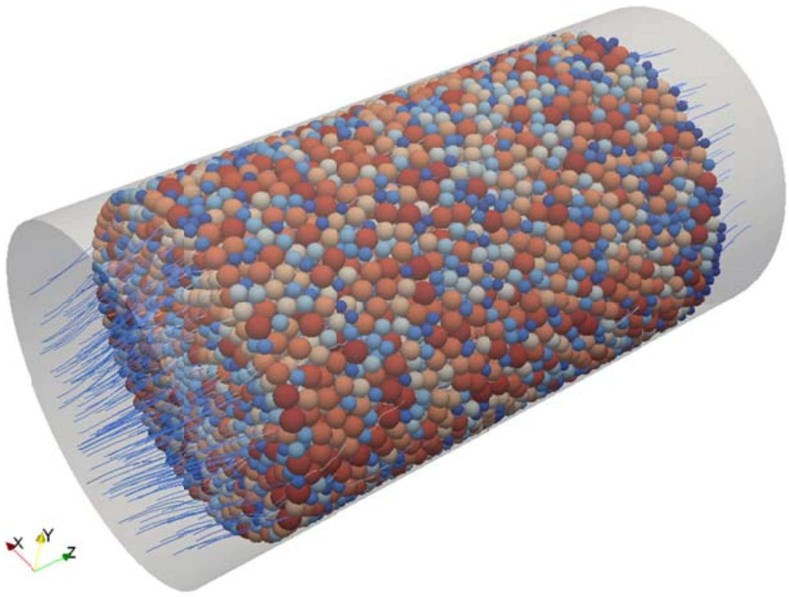


Figure 7. Illustration of the packed bed and the bulk flow streamlines.

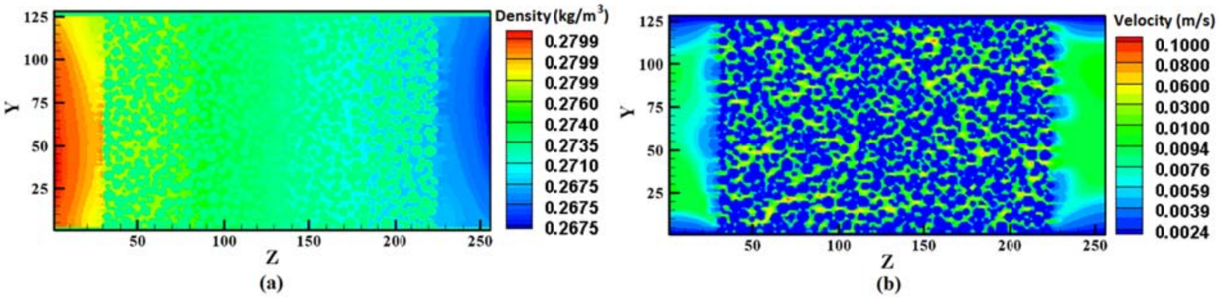
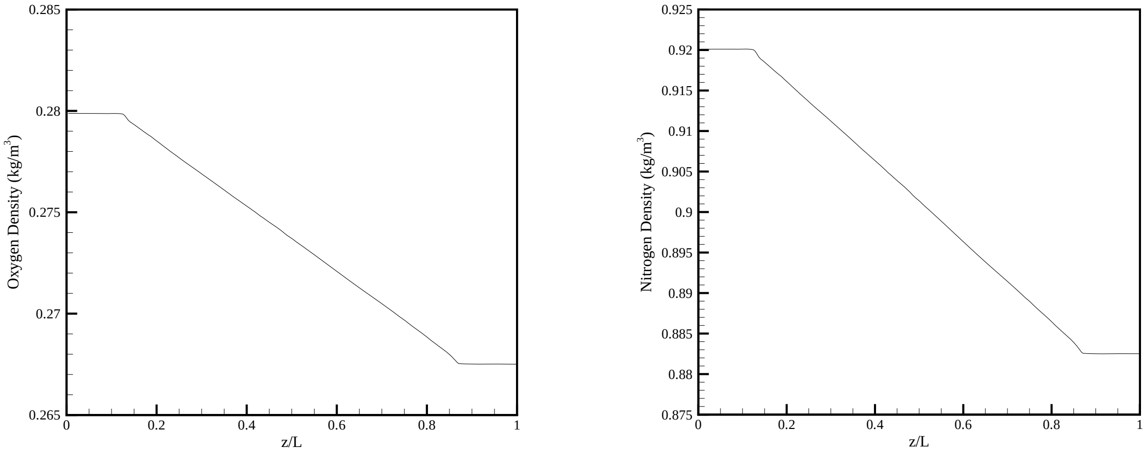


Figure 8. Density (pressure) and velocity contours for Oxygen in the cross section of the flow through the packed bed. (a) density contour, (b) velocity contour and the streamlines.



(a) (b)

Figure 9. Variations in density along the packed bed for (a) Oxygen, and (b) Nitrogen.

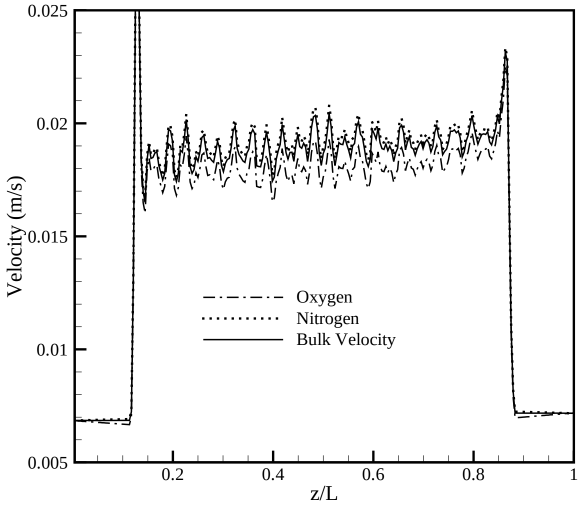


Figure 10. Average z-velocity fluctuations for both species and the bulk flow along the packed bed.

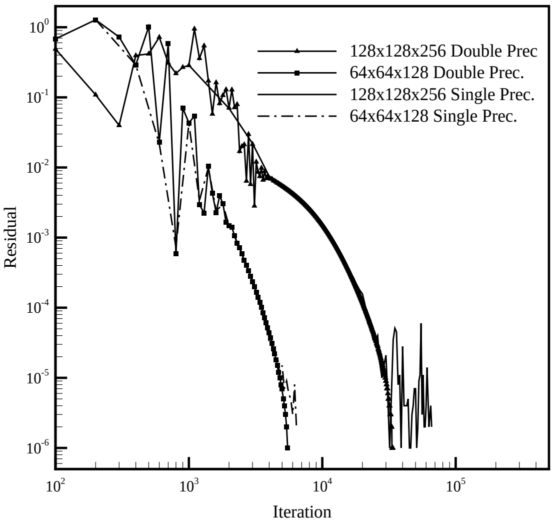


Figure 11. Convergence trend of the simulations for two different grid levels.

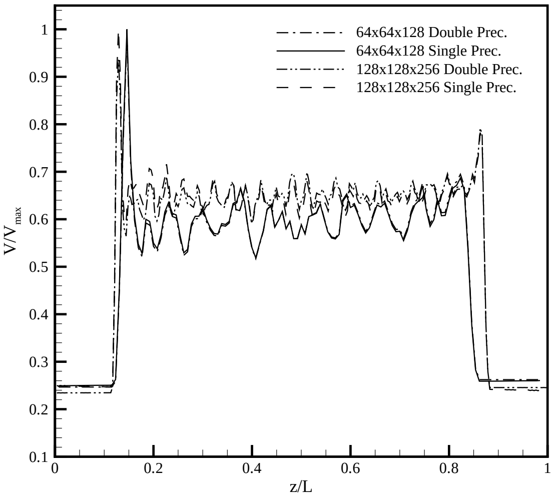
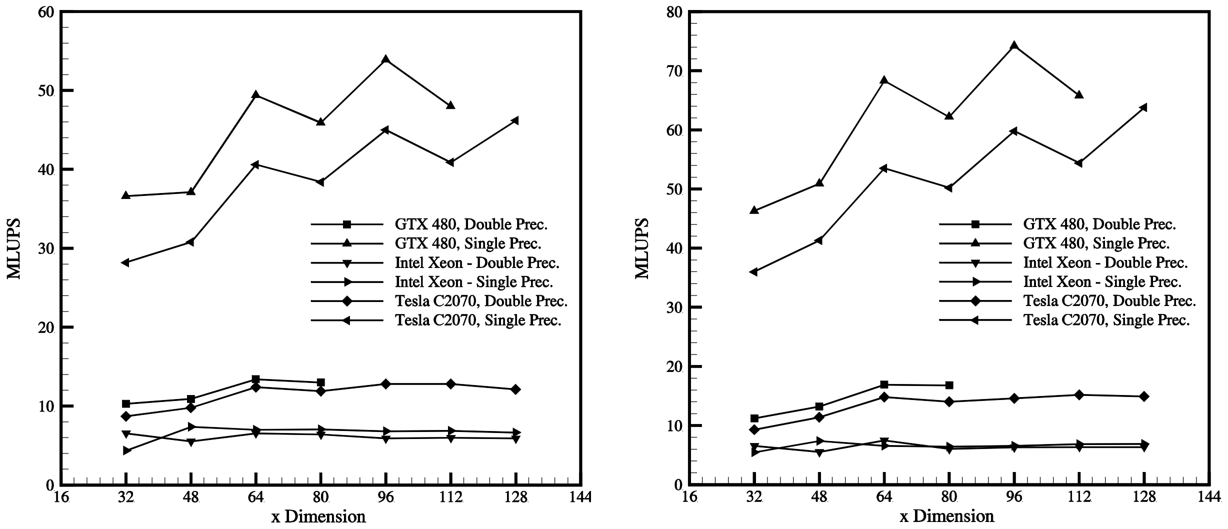


Figure 12. Averaged bulk velocity fluctuations along the packed bed using single and double precision computations for two different grid levels.



(a) (b)

Figure 13. Computational performance on different platforms and different problem sizes. (a) flow through a packed bed, (b) flow in an empty tube.

Table 1: Major specifications of different GPU devices used in the present work. DP=Double Precision, SP=Single Precision, GFLOPS=Giga Floating point Operations Per Second, MTB=Maximum Theoretical Bandwidth, MEB=Maximum Effective Bandwidth

| GPU TYPE | No. of Thread Processors | Processor Clock(MHz) | Memory (GB) | Max DP GFLOPS | Max SP GFLOPS | MTB (GB/sec) | MEB (GB/sec) |
|---|---|---|---|---|---|---|---|
| GTX 480 | 15×32 | 70 | 1.5 | 168 | 1350 | 177.4 | 138 |
| Tesla C2070 | 14×32 | 57.5 | 6 | 515 | 1003 | 144.0 | 106 |

Table 2. Single precision performance metrics for flow in a packed bed for different kernels on Tesla C2070. BW= Percentage of the achieved bandwidth to the device's maximum effective bandwidth. PTGT= Percentage of the Total GPU Time

| | 32x64 | 64x128 | 96x192 | 128x256 |
|---|---|---|---|---|
| Performance (MLUPS) | 28.2 | 40.6 | 45.0 | 46.2 |
| collide_stream_knl Max BW (%) | 62.2 | 90.5 | 88.7 | 91.3 |
| bounce_back_knl Max BW (%) | 15.0 | 28.0 | 32.1 | 33.6 |
| collide_stream_knl Occupancy (%) | 16.6 | 32.9 | 30.8 | 32,7 |
| bounce_back_knl Occupancy (%) | 16.6 | 31.3 | 44.0 | 56,6 |
| collide_stream_knl PTGT | 47.0 | 53.5 | 57 | 57.5 |
| bounce_back_knl PTGT | 50.7 | 45.1 | 42 | 41.5 |