Leveraging the Performance of LBM-HPC for Large Sizes on GPUs using Ghost Cells

Pedro Valero-Lara[1],[2⋆]

[1] The University of Manchester, Manchester, United Kingdom
pedro.valero-lara@manchester.ac.uk
[2] Basque Center for Applied Mathematics, Bilbao, Spain

Abstract. Today, we are living a growing demand of larger and more efficient computational resources from the scientific community. On the other hand, the appearance of GPUs for general purpose computing supposed an important advance for covering such demand. These devices offer an impressive computational capacity at low cost and an efficient power consumption. However, the memory available in these devices is (sometimes) not enough, and so it is necessary computationally expensive memory transfers from (to) CPU to (from) GPU, causing a dramatic fall in performance. Recently, the Lattice-Boltzmann Method has positioned as an efficient methodology for fluid simulations. Although this method presents some interesting features particularly amenable to be efficiently exploited on parallel computers, it requires a considerable memory capacity, which can suppose an important drawback, in particular, on GPUs. In the present paper, it is proposed a new GPU-based implementation, which minimizes such requirements with respect to other state-of-the-art implementations. It allows us to execute almost 2× bigger problems without additional memory transfers, achieving faster executions when dealing with large problems.

Keywords: Computational Fluid Dynamics, Lattice-Boltzmann Method, GPU, CUDA

1 Introduction

The appearance of GPUs has been an important advance, emerging new challenges and opportunities for increasing performance in multiple scientific solvers. Many scientific applications and software packages have already been ported and redesigned to exploit GPUs. These developments have often involved major algorithm changes since some classical solvers may turned out to be inefficient or

⋆ This research has been supported by the Basque Excellence Research Center (BERC 2014-2017) program by the Basque Government, the Spanish Ministry of Economy and Competitiveness MINECO: BCAM Severo Ochoa accreditation SEV-2013-0323. The authors would like to thank the computing facilities of the Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), and NVIDIA GPU Research Center program for the provided resources.

difficult to tune [1, 2]. Fortunately, other solvers are particularly well suited for GPU acceleration and are able to achieve significant performance improvements. The Lattice Boltzmann method (LBM) [3] is one of those examples thanks to its inherently data-parallel nature. Certainly, the computing stages of LBM are amenable to fine grain parallelization in an almost straightforward way. This fundamental advantage of LBM has been consistently confirmed by many previous works [4, 5], for a large variety of problems and computing platforms. For instance, T. Pohl et al. [6] studied several memory access patterns to maximize the temporal locality, optimizing the cache performance over multicore architectures. Also P. R. Rinaldi et al. [5] proposed a different ordering of the LBM steps to reduce the number of memory accesses. LBM has been adapted to numerous parallel architectures, such as multicore processors [6], manycore accelerators [5],[7],[8] and distributed-memory clusters [9–11]. Given the growing popularity of LBM, multiple tools [9],[10],[12],[13] have recently arisen, consolidating this method in academia and industry. In particular, in this work we have considered the LBM-HPC framework [10] as our reference software tool.

Today, we are living a growing demand of higher computational resources from CFD community to be able to simulate and compute more and more complex scenarios. In particular, one of the most important challenges to deal with such scenarios is the excessive memory requirements. Despite LBM is an appropriate method for parallel systems, it requires a high memory capacity for its execution. For instance, to compute bi-dimensional problems using LBM, we need 21 elements (double precision) per mesh (macroscopic) point. These requirements are much bigger for tri-dimensional problems. Our motivation consists of developing a new approach, which minimizes such demand of memory for LBM implementations over CUDA compatible GPUs. We propose the use of *ghost cells* to minimize the memory requirements and to deal with race conditions. This idea forces us to develop a more complex strategy with a different memory mapping and one additional kernel (GPU code). However, this new approach allows us to execute bigger problems over the same platform, avoiding the impressive fall in performance (reducing the memory transfer between CPU and GPU) when other state-of-the-art approaches are considered.

The remainder of this paper is organized as follows. In Sec. 2 we introduce the general numerical and implementation framework for the LBM. After that, we describe the different optimizations and parallel strategies envisaged to achieve high-performance when dealing with large problems. Finally, we discuss the performance results of the proposed techniques in Sec. 3. We conclude in Sec. 4 with a summary of the main contributions of this work.

2 Lattice-Boltzmann Method

2.1 Background

Most of the current methods for simulating the transport equations (heat, mass, and momentum) are based on the use of macroscopic partial differential equations [14]. On the other extreme, we can view the medium from a microscopic

viewpoint where small particles (molecule, atom) collide with each other (molecular dynamic) [15]. In this scale the inter-particle forces must be identified, which requires to know the location, velocity, and trajectory of every particle. However, there is no definition of viscosity, heat capacity, temperature, pressure, etc. These methods are extremely expensive computationally [15]. However, it is possible to use statistical mechanisms as a translator between the molecular world and the microscopic world, avoiding the management of every individual particle, while obtaining the important macroscopic effects by combining the advantages of both approaches, macroscopic and microscopic, with manageable computer resources. This is the main idea of the Boltzmann equation and the mesoscopic scale [15].

Multiple studies have compared the efficiency of LBM with respect to other "classic" methods based on Navier-Stokes (see [16, 17]). They show that LBM can achieve an equivalent numerical accuracy over a large number of applications. In particular, LBM has been used to simulate high Reynolds turbulent flows over Direct Numerical and Large Eddy simulations [18], aeroacoustics problems [19], bio-engineering applications [7], among others. Also, LBM has been efficiently integrated with other methods, such as the Immersed Boundary Method for Fluid-Solid Interaction problems [8],[20].

2.2 LBM Formulation

LBM combines some features developed to solve the Boltzmann equation over a finite number of microscopic speeds. LBM presents lattice-symmetry characteristics which allow to respect the conservation of the macroscopic moments [21]. The standard LBM [22] is an explicit solver for incompressible flows. It divides each temporal iteration into two steps, one for propagation-advection (streaming) and one for collision (inter-particle interactions), achieving a first order in time and second order in space scheme.

LBM describes the fluid behavior at mesoscopic level. At this level, the fluid is modeled by a distribution function of the microscopic particle (f). Similarly to the Boltzmann equation, LBM solves the particle speed distribution by discretizing the speed space over a discrete finite number of possible speeds. The distribution function evolves according to the following equation:

$$\frac{\partial f}{\partial t} + e\nabla f = \Omega \tag{1}$$

where f is the particle distribution function, e is the discrete space of speeds and Ω is the collision operator. By discretizing the distribution function f in space, in time, and in speed ($e = e_i$) we obtain $f_i(x,t)$, which describes the probability of finding a particle located at x at time t with speed e_i. $e\nabla f$ can be discretized as:

$$e\nabla f = e_i \nabla f_i = \frac{f_i(x + e_i \Delta t, t + \Delta t) - f_i(x, t + \Delta t)}{\Delta t} \tag{2}$$

In this way the particles can move only along the links of a regular *lattice* (Fig. 1) defined by the discrete speeds ($e_0 = c(0,0)$;$e_i = c(\pm 1, 0)$, $c(0, \pm 1)$, $i = 1, \cdots, 4$; $e_i = c(\pm 1, \pm 1)$, $i = 5, \cdots, 8$ with $c = \Delta x / \Delta t$) so that the synchronous particle displacements $\Delta x_i = e_i \Delta t$ never takes the fluid particles away from the *lattice*. In this work, we consider the standard two-dimensional 9-speed *lattice* *D2Q9* [21].

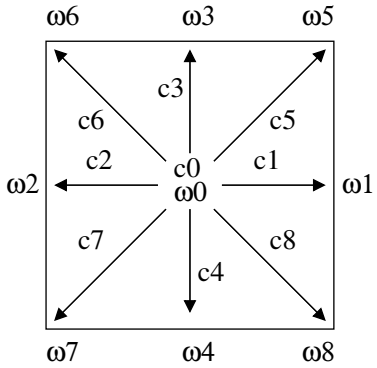

Fig. 1. The standard two-dimensional 9-speed lattice (*D2Q9*) [23].

The operator Ω describes the changes suffered by the collision of the microscopic particles, which affect the distribution function (f). To calculate the collision operator we consider the *BGK* (Bhatnagar-Gross-Krook) formulation [23] which relies upon a unique relaxation time, τ, toward the equilibrium distribution f_i^{eq}:

$$\Omega = -\frac{1}{\tau}\left(f_i\left(x,t\right) - f_i^{eq}\left(x,t\right)\right) \tag{3}$$

The equilibrium function $f^{eq}\left(x,t\right)$ can be obtained by *Taylor* series expansion of the *Maxwell-Boltzmann* equilibrium distribution [22]:

$$f_i^{eq} = \rho\omega_i\left[1 + \frac{e_i \cdot u}{c_s^2} + \frac{\left(e_i \cdot u\right)^2}{2c_s^4} - \frac{u^2}{2c_s^2}\right] \tag{4}$$

where c_s is the speed of sound ($c_s = 1/\sqrt{3}$), u is the vertical or horizontal component (see Algorithm 1) of the macroscopic velocity in the given position, and the weight coefficients ω_i are $\omega_0 = 4/9$, $\omega_i = 1/9,\ i = 1, \ldots, 4$ and $\omega_i = 1/36,\ i = 5, \ldots, 8$ based on the current normalization. Through the use of the collision operator and substituting the term $\frac{\partial f_i}{\partial t}$ with a first order temporal discretization, the discrete Boltzmann equation can be written as:

$$\frac{f_i(x,t+\Delta t)-f_i(x,t)}{\Delta t}+\frac{f_i(x+e_i\Delta t,t+\Delta t)-f_i(x,t+\Delta t)}{\Delta t} = -\frac{1}{\tau}\left(f_i\left(x,t\right)-f_i^{eq}\left(x,t\right)\right) \tag{5}$$

which can be compactly written as:

$$f_i\left(x+e_i\Delta t,t+\Delta t\right)-f_i\left(x,t\right)=-\frac{\Delta t}{\tau}\left(f\left(x,t\right)-f_i^{eq}\left(x,t\right)\right) \tag{6}$$

The macroscopic velocity u in equation 4 must satisfy a Mach number requirement $\mid u \mid /c_s \approx M << 1$. This stands as the equivalent of the CFL number[3] for classical Navier Stokes solvers.

As mentioned above, the equation 6 is typically advanced in time in two stages, the collision and the streaming stages.
Given $f_i(x,t)$ compute:

$\rho = \sum f_i(x,t)$ and
$\rho u = \sum e_i f_i(x,t)$

Collision stage:

$f_i^*\left(x,t+\Delta t\right) = f_i\left(x,t\right) - \frac{\Delta t}{\tau}\left(f\left(x,t\right)-f_i^{eq}\left(x,t\right)\right)$

Streaming stage:

$f_i\left(x+e_i\Delta t,t+\Delta t\right) = f_i^*\left(x,t+\Delta t\right)$

2.3 LBM Solvers & Implementation

LBM exhibits a high degree of parallelism and is amenable to fine granularity (one thread per lattice node), since the solving of every *lattice* point is totally independent with respect to the others. To compute streaming in parallel, we need two different distribution functions (f_1 and f_2 in Algorithm 1).

In the present work, we have opted to work with the *pull* approach (introduced by [24]), which has been recently considered in [5],[8],[20]. This is an efficient approach based on a single-loop strategy, in which each *lattice* node can be independently computed by performing one complete time step of LBM. A schematic sketch of this LBM implementation is given in Algorithm 1. Basically, the *pull* approach fuses in a single loop (that iterates over the entire domain), the application of both operations, collision and streaming (see Sec. 2.2) to improve temporal locality. Furthermore it does not need any synchronization among these operations. Also, it eases pressure on memory with respect to other approaches, as the macroscopic level can be completely computed on high levels of memory hierarchy (registers/L1 cache).

[3] Courant Friedrichs Lewy (CFL) number arises in those schemes based on explicit time computer simulations. As a consequence, this number must be less than a certain time to achieve coherent results.

Algorithm 1 LBM *pull*.

1: **for** $ind = 1 \rightarrow Nx \cdot Ny$ **do**
2: **Streaming**
3: **for** $i = 1 \rightarrow 9$ **do**
4: $x_{stream} = x - c_x[i]$
5: $y_{stream} = y - c_y[i]$
6: $ind_{stream} = y_{stream} \cdot Nx + x_{stream}$
7: $f[i] = f_1[i][ind_{stream}]$
8: **end for**
9: **for** $i = 1 \rightarrow 9$ **do**
10: $\rho+ = f[i]$
11: $u_x+ = c_x[i] \cdot f[i]$
12: $u_y+ = c_y[i] \cdot f[i]$
13: **end for**
14: $u_x = u_x/\rho$
15: $u_y = u_y/\rho$
16: *Synchronization point (only) for our approach based on Ghost Cell*
17: *syncthreads*()
18: **Collision**
19: **for** $i = 1 \rightarrow 9$ **do**
20: $cu = c_x[i] \cdot u_x + c_y[i] \cdot u_y$
21: $f_{eq} = \omega[i] \cdot \rho \cdot (1 + 3 \cdot cu + cu^2 - 1.5 \cdot (u_x)^2 + u_y)^2$
22: $f_2[i][ind] = f[i] \cdot (1 - \frac{1}{\tau}) + f_{eq} \cdot \frac{1}{\tau}$
23: **end for**
24: **end for**

Memory management plays a crucial role in LBM implementations. The information of the fluid domain should be stored in memory in such way that reduces the number of memory accesses and keeps the implementation highly efficient by taking advantages of vector units. In this work, we consider a coalescing memory access pattern by using a Structure of Array (SoA) approach. This strategy (*pull-coalescing*) has proven to be very efficient in multicore and GPUs architectures [5],[8],[7],[20]. The discrete distribution function f_i is stored sequentially in the same array (see Fig. 2-top, where Nx and Ny are the number of horizontal and vertical fluid nodes respectively). This way, consecutive threads access adjacent memory locations (coalescing).

Parallelism is abundant in the LBM update and can be exploited in different ways. The recommendable parallelization of LBM over GPUs consists of using a single *kernel* by using a 1D Grid of 1D CUDA block, in which each CUDA-thread performs a complete LBM update on a single *lattice* node [8]. *Lattice* nodes are distributed across GPU cores using a fine-grained distribution (Fig. 2-bottom).

In order to exploit the parallelism found in the LBM, previous studies make use of two different data set [5],[9],[10],[13],[20]. Essentially, it follows an AB scheme [8] which holds the data of two successive time steps (A and B) and the simulation alternates between reading from A and writing to B, and vice-versa. In this work, we proposed an alternative to reduce such high memory

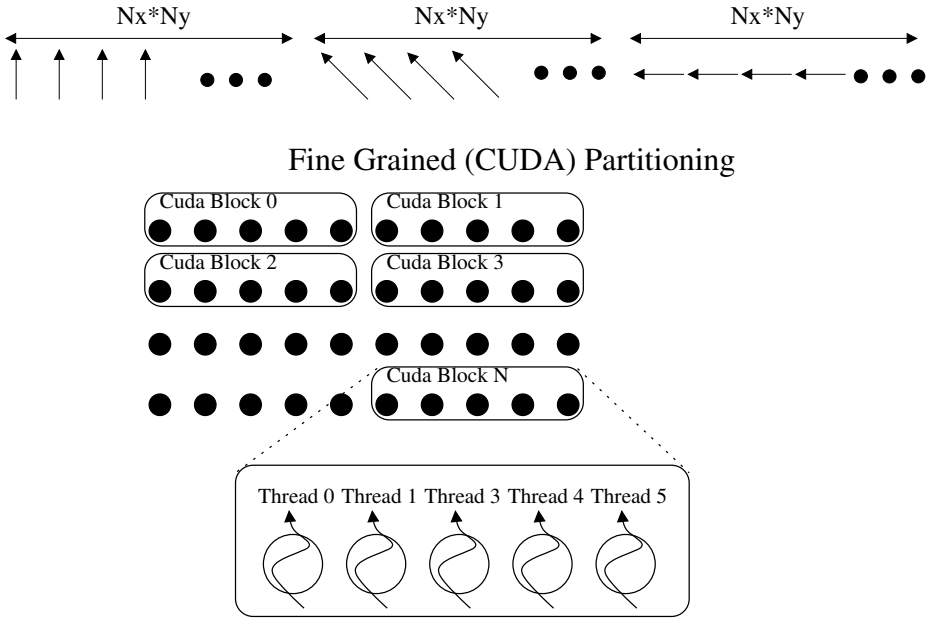

Fig. 2. SoA data layout to store the discrete distribution function f_i in memory (top) and fine-grained distributions of the *lattice* nodes (bottom).

requirements, adapting the use of *ghost cell* to our target problem (LBM) and platform (GPUs).

2.4 Ghost Cells

In this subsection, we focus on explaining how we have adapted the use of *ghost cells* to LBM to reduce the memory requirements for GPU-based implementations.

Although, the *ghost cells* strategy is usually used for communication in distributed memory systems [25], we use this strategy to reduce memory requirements and avoid race conditions among the set of CUDA blocks (*fluid blocks*). To minimize the number of *g*host cells we use the biggest size of CUDA (*fluid block*) blocks possible. The use of *ghost cells* consists of replicating the borders of all immediate neighbors blocks, in our case *fluid blocks*. These *ghost cells* are not updated locally, but provide stencil values when updating the borders of local blocks. Every *ghost cell* is a duplicate of a piece of memory located in neighbors nodes. To clarify, Fig. 3-left illustrates a simple scheme for our interpretation of the *ghost cell* strategy applied to LBM.

In spread operation (Fig. 3-right), some of the lattice-speed into each *ghost cell* are used by adjacent fluid (*lattice*) elements located in neighbors *fluid blocks*. Depending on the position of the fluid units, a different pattern for spread operation is required. For instance, if one fluid element is positioned in one of

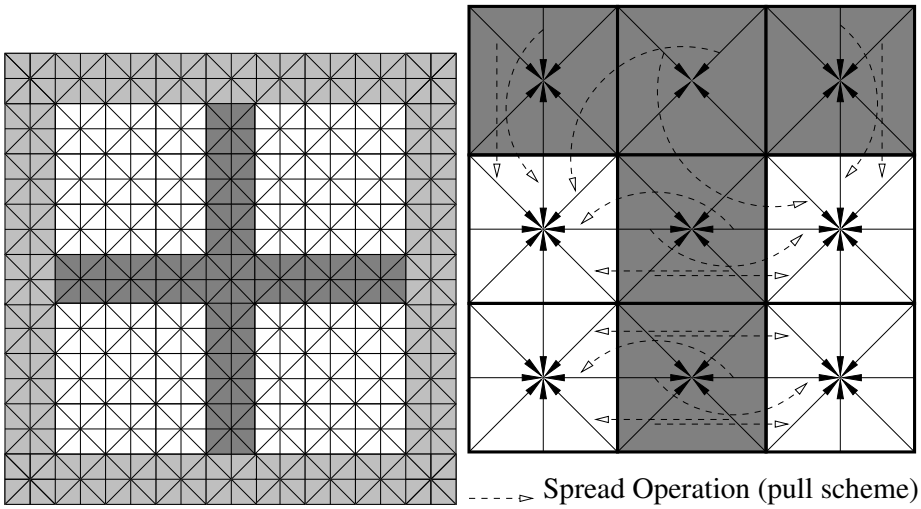

Fig. 3. A simple scheme (left) for our LBM approach composed by four *fluid blocks* (CUDA blocks). Spread operation (right) from *ghost cells* to fluid units. Ghost (dark gray background), boundary (light gray background), and fluid (white background) units.

the corners of fluid block, this takes 5 lattice-speed from 3 different *ghost cells*. Otherwise, if one fluid element is located in other position of the fluid block boundary, it needs to take 3 lattice-speed from one *ghost cell*.

The information stored in the *ghost cells* are in need of being updated once per time step. The update operation is performed via a second kernel before computing LBM. Basically, this kernel moves some lattice-speed from lattice units to *ghost cells*. This CUDA kernel is composed by as many threads as *ghost cells*. To optimize memory management and minimize divergence, continuous CUDA blocks compute each of the updating cases. To clarify Fig. 4 shows the different data movements applied to each of the cases. In this regard, depending on the location of *ghost cells*, a different number of memory movements are necessary. In particular, if one *ghost cell* is located in one of the *ghost cell* rows or columns (Vertical and Horizontal cases in Fig. 4), this needs to take 6 lattice-speed from 2 different fluid units (3 lattice-speed per fluid unit). However, if one *ghost cell* is positioned in one of the corners (Corner case in Fig. 4), then this *ghost cell* requires 4 lattice-speed from 4 fluid units.

Unlike the standard LBM implementation (*pull* approach) on GPU, the *CUDA* blocks need to be synchronized before computing collision. This is possible via a *__syncthreads()* call from the kernel side (see Algorithm 1). This synchronization and the use of *ghost cells* among CUDA (*fluid*) blocks guarantees the absence of race conditions.

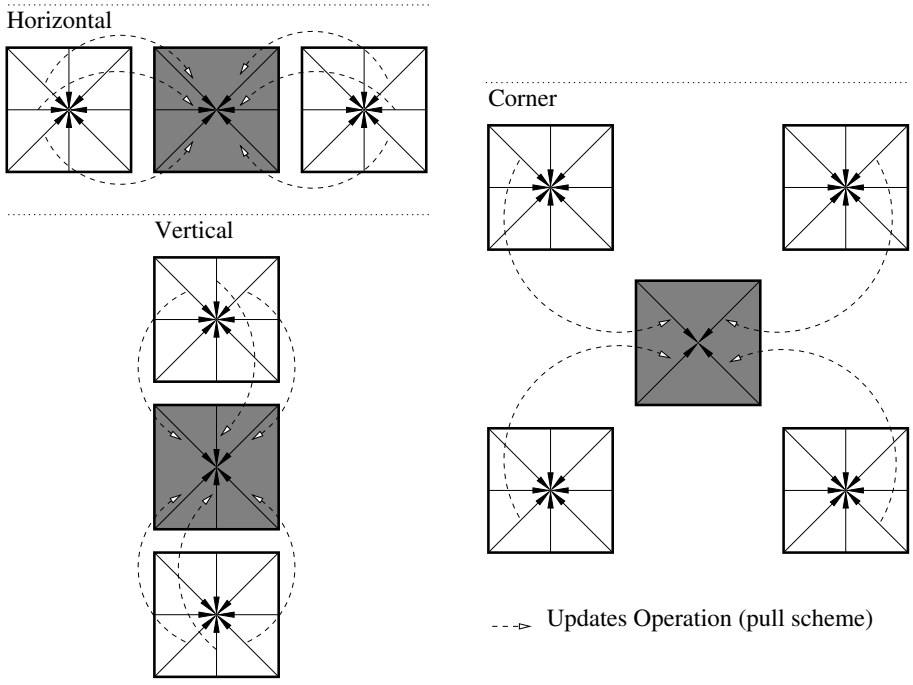

Fig. 4. Update operation from fluid units (white background) to *ghost cells* (gray background), depending on *ghost cells* position.

2.5 Memory Management

It is well known that the memory management has an important influence on performance of parallel computing, in particular on those parallel computers that suffer of a high latency such as Nvidia GPUs or Intel MIC [8]. Furthermore, LBM is a memory-bound algorithm, so that, another important optimization problem is to maximize data locality.

The previous thread-data distribution shown in Fig. 2-bottom does not allow us to exploit coalescence (contiguous threads access to continuous memory locations), when dealing with *ghost cells*, so we proposed a new memory mapping which fits better our particular data structure. Essentially, we follow the same aforementioned strategy (*SoA*), adapting it to our approach based on *ghost cells*. Instead of mapping every lattice-speed in consecutive memory location for the whole fluid domain (Fig. 2), we map the set of lattice-speed of every bi-dimensional fluid (*CUDA*) block in consecutive memory locations, as graphically illustrated by Fig. 5.

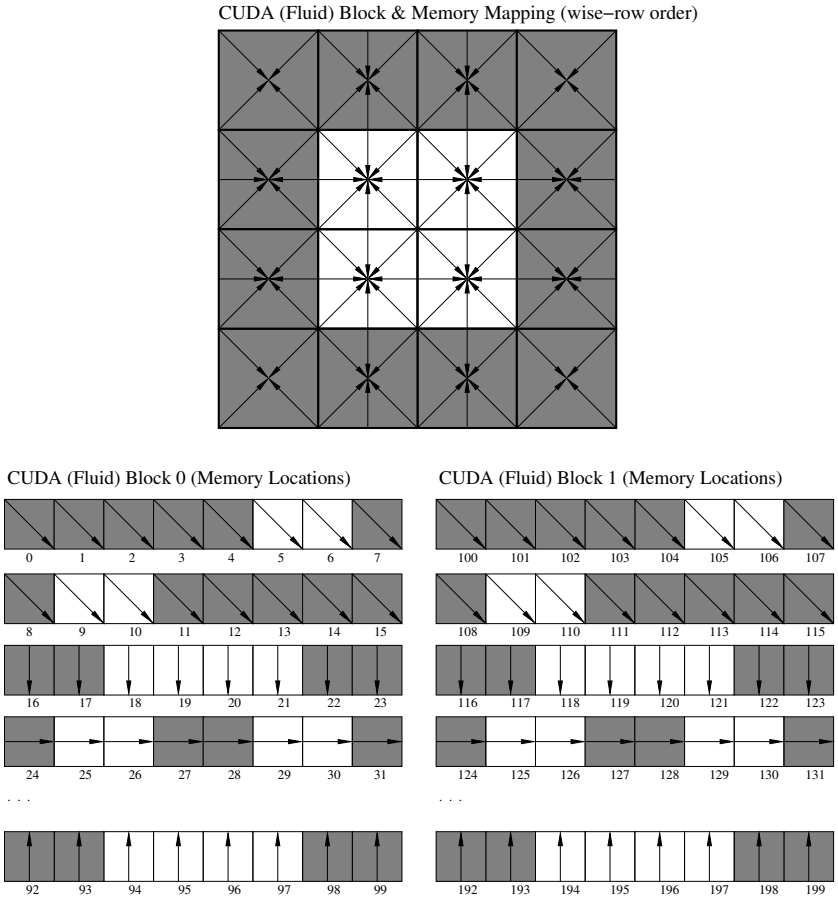

Fig. 5. Memory and CUDA block mapping for the *1 lattice + ghost* approach.

3 Performance Evaluation

To critically evaluate the performance of the proposed LBM solver, next we consider a number of tests executed on one Nvidia Kepler (K20c) GPU with 2496 CUDA cores at 706 Mhz and 5GB GDDR5 of memory. According to the memory requirements of the GPU kernels, the memory hierarchy of the GPU has been configured as 16KB shared memory and 48KB L1, since our codes do not benefit from a higher amount of shared memory on the investigated tests.

Given the restrictions in terms of number of threads per CUDA (*fluid*) block, we have considered the most appropriate size of fluid block for each of the implementations. For our testbed platform the maximum number of threads per CUDA (*fluid*) block is 2048.

In the following, we analyse the time consumed by both approaches, *2 lattice* (*pull* approach) and *1 lattice + ghost*. Fig. 6-left graphically illustrates the execu-

tion time for both approaches increasing the size of the problem to be computed. As expected, the *2 lattice* achieves a low execution time against the *1 lattice + ghost* in those problems which can be stored completely in GPU memory (from 9 to 36 millions of fluid nodes) using f_1 and f_2. However, for bigger fluid domains (from 45 millions of nodes), the limit of memory forces us to execute our problem in two-steps when using the *2 lattice* approach. It requires additional memory transfers regarding data dependency of our simulation. In particular, the whole subdomains of both LBM blocks (LBM-Block1/2 in Fig. 6-left) must be transfered from GPU to CPU and vice-versa (CPU/GPU->GPU/CPU-Block1/2 in Fig. 6-left) every time-step. As graphically illustrated by Fig. 6-left, this additional overhead consumes most of the execution time, being the main bottleneck and causing an important fall in performance.

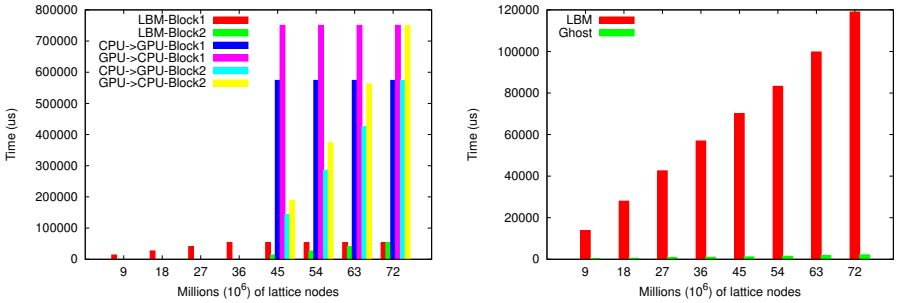

Fig. 6. Time consumed by the *2 lattice* approach (left) and by the *1 lattice+ghost* approach (right).

On the other hand, the *1 lattice + ghost* shows a better performance for large problems (Fig. 6-right). Despite this approach requires 2 kernels, the time consumed by the additional kernel (*Ghost* in Fig. 6-right), which updates the information in *ghost cells*, does not represent an important overhead. Actually, it is less than 2% of the total consumed time.

As previously introduced, one of our main motivations of this work consists of reducing the memory requirements for LBM simulations on GPUs. Fig. 7-left illustrates the memory consumed by both approaches. The reduction achieved by our approach (*1 lattice + ghost*) represents almost the half of the memory consumed by the *2 lattice* approach. It allows us to launch bigger simulations obviating additional and computationally expensive memory transfers.

Most of the LBM studies consider the conventional MFLUPS (Millions of Fluid Lattice Updates Per Second ratio) as a metric. As a reference, we also estimate the ideal MFLUPS [26]:

$$MFLUPS_{ideal} = \frac{B \times 10^9}{10^6 \times n \times 6 \times 8} \tag{7}$$

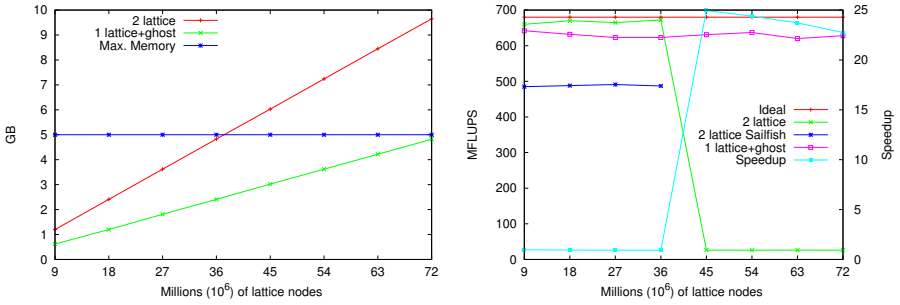

Fig. 7. Memory consumed (left) by each of the implementations. MFLUPS reached by both approaches and speedup (pink line) achieved by the *1 lattice + ghost* against the *2 lattice* (right).

where $B \times 10^9$ is the memory bandwidth (GB/s), n depends on LBM model (DxQn), for our framework $n = 9$, D2Q9. The factor 6 is for the memory accesses, three read and write operations in the spreading step and three read and write operations in the collision step, and the factor 8 is for double precision (8 bytes).

Fig. 7-right illustrates the MFLUPS achieved by both approaches and an estimation for the ideal MFLUPS for our platform. The *2 lattice* approach is near ideal performance when dealing with "small" problems (until 36 millions of fluid units), being the *1 lattice + ghost* approach almost a 10% slower, due to a more complex implementation (synchronization before computing collision, one additional kernel for ghost cell updates and a more complex memory access pattern). However, when bigger domains are considered (from 45 to 72 millions of fluid units), the *2 lattice* approach turns out to be very inefficient, suffering a dramatic fall in performance. In contract, the performance achieved by *1 lattice + ghost* is kept constant for the rest of tests. Also, as reference, we included the performance achieved by the GPU based implementation provided in the sailfish package [9], which is slower than the other implementations (*2 lattice* and *1 lattice + ghost*).

Additionally Fig. 7-right illustrates the speedup achieved by our new approach against the *2 lattice* implementation. Our approach is slower than the *2 lattice* approach for those problems with a domain equal or smaller than 36 millions of units, however our approach turns out to be faster (speedup near of 25) when dealing with bigger domains.

4 Conclusions

The limitation found in the memory capacity of GPUs and the amount of memory demanded by LBM supposes an important drawback when dealing with large problems. This work presents a new alternative based on *ghost cells*, reducing considerably the memory requirements and keeping a high MFLUPS ratio for

large simulations. It was carried out a detailed performance analysis in terms of time, memory requirements, speedup and MFLUPS ratio. Furthermore, the implementation proposed (ghost cell, additional kernel, memory access pattern, synchronizations points, etc) has been thoroughly detailed.

References

1. Valero-Lara, P., Pinelli, A., Favier, J., Matias, M.P.: Block tridiagonal solvers on heterogeneous architectures. In: Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications. ISPA '12, Washington, DC, USA, IEEE Computer Society (2012) 609–616
2. Valero-Lara, P., Pinelli, A., Prieto-Matias, M.: Fast finite difference poisson solvers on heterogeneous architectures. Computer Physics Communications **185**(4) (2014) 1265 – 1272
3. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond (Numerical Mathematics and Scientific Computation). Numerical mathematics and scientific computation. Oxford University Press, USA (August 2001)
4. Bernaschi, M., Fatica, M., Melchiona, S., Succi, S., Kaxiras, E.: A flexible high-performance lattice boltzmann gpu code for the simulations of fluid flows in complex geometries. Concurrency Computa.: Pract. Exper. **22** (2010) 1–14
5. Rinaldi, P., Dari, E., Vnere, M., Clausse, A.: A lattice-boltzmann solver for 3d fluid simulation on {GPU}. Simulation Modelling Practice and Theory **25**(0) (2012) 163 – 171
6. Pohl, T., Kowarchik, M., Wilke, J., K. Iglberger, U.R.: Optimization and profiling of the cache performance of parallel lattice boltzmann codes. Parallel Processing Letters **13**(4) (2003) 549560
7. Bernaschi, M., Fatica, M., Melchionna, S., Succi, S., Kaxiras, E.: A flexible high-performance lattice boltzmann gpu code for the simulations of fluid flows in complex geometries. Concurrency and Computation: Practice and Experience **22**(1) (2010) 114
8. Valero-Lara, P., Igual, F.D., Prieto-Matas, M., Pinelli, A., Favier, J.: Accelerating fluidsolid simulations (lattice-boltzmann & immersed-boundary) on heterogeneous architectures. Journal of Computational Science **10** (2015) 249–261
9. Januszewski, M., Kostur, M.: Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method. Computer Physics Communications **185**(9) (September 2014) 2350–2368
10. LBM-HPC. Last access on 26-04-2016 to url-http://www.bcamath.org/en/research/lines/CFDCT/software
11. Obrecht, C., Kuznik, F., Tourancheau, B., Roux, J.J.: Scalable lattice boltzmann solvers for {CUDA} {GPU} clusters. Parallel Computing **39**(67) (2013) 259 – 270
12. XFlow, N.G.o.C. Last access on 26-04-2016 to urlhttp://www.xflowcfd.com/
13. Palabos, C.C.P. Last access on 26-04-2016 to urlhttp://www.palabos.org/
14. Wendt, J.F., Anderson, J.D.: Computational Fluid Dynamics: An Introduction. Springer (2008)
15. Mohamad, A.A.: The Lattice Boltzmann Method - Fundamental and Engineering Applications with Computer Codes. Springer (2011)
16. Axner, L., Hoekstra, A.G., Jeays, A., Lawford, P., Hose, R., Sloot, P.M.: Simulations of time harmonic blood flow in the mesenteric artery: comparing finite element and lattice boltzmann methods. BioMedical Engineering OnLine (2000)

17. Kollmannsberger, S., Geller, S., Dster, A., Tlke, J., Sorger, C., Krafczyk, M., Rank, E.: Fixed-grid fluidstructure interaction in two dimensions based on a partitioned lattice boltzmann and p-fem approach. International Journal for Numerical Methods in Engineering **79**(7) (2009) 817–845
18. Malaspinas, O., Sagaut, P.: Consistent subgrid scale modelling for lattice boltzmann methods. Journal of Fluid Mechanics **700** (2012) 514–542
19. Marié, S., Ricot, D., Sagaut, P.: Comparison between lattice boltzmann method and navier-stokes high order schemes for computational aeroacoustics. J. Comput. Phys. **228**(4) (March 2009) 1056–1070
20. Valero-Lara, P., Pinelli, A., Prieto-Matias, M.: Accelerating solid-fluid interaction using lattice-boltzmann and immersed boundary coupled simulations on heterogeneous platforms. Procedia Computer Science **29**(0) (2014) 50 – 61 2014 International Conference on Computational Science.
21. He, X., Luo, L.S.: A priori derivation of the lattice boltzmann equation. Phys. Rev. E **55** (Jun 1997) R6333–R6336
22. Qian, Y.H., D'Humires, D., Lallemand, P.: Lattice bgk models for navier-stokes equation. EPL (Europhysics Letters) **17**(6) (1992) 479
23. P. Bhatnagar, E.G., Krook, M.: A model for collision processes in gases. i: small amplitude processes in charged and neutral one-component system. Phys. Rev. E **94** (1954) 511–525
24. Wellein, G., Zeiser, T., Hager, G., Donath, S.: On the single processor performance of simple lattice boltzmann kernels. Computers & Fluids **35**(89) (2006) 910 – 919 Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.
25. Valero-Lara, P., Jansson, J.: LBM-HPC - an open-source tool for fluid simulations. case study: Unified parallel C (UPC-PGAS). In: 2015 IEEE International Conference on Cluster Computing, CLUSTER 2015, Chicago, IL, USA, September 8-11, 2015. (2015) 318–321
26. Shet, A.G., Sorathiya, S.H., Krithivasan, S., Deshpande, A.M., Kaul, B., Sherlekar, S.D., Ansumali, S.: Data structure and movement for lattice-based simulations. Phys. Rev. E **88** (Jul 2013) 013314