CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2017; **00**:1–14
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Reducing Memory Requirements for Large Size LBM Simulations on GPUs

Pedro Valero-Lara*[1]

Barcelona Supercomputing Center (BSC), Barcelona, Spain.[1]

SUMMARY

The scientific community in its never-ending road of larger and more efficient computational resources is in need of more efficient implementations that can adapt efficiently on the current parallel platforms. GPUs are an appropriate platform that cover some of these demands. This architecture presents a high performance with a reduced cost and an efficient power consumption. However, the memory capacity in these devices is reduced and so expensive memory transfers are necessary to deal with big problems. Today, the Lattice-Boltzmann Method has positioned as an efficient approach for Computational Fluid Dynamics simulations. Despite this method is particularly amenable to be efficiently parallelized, it is in need of a considerable memory capacity, which is the consequence of a dramatic fall in performance when dealing with large simulations. In this work, we propose some initiatives to minimize such demand of memory, which allows us to execute bigger simulations on the same platform without additional memory transfers, keeping a high performance. In particular, we present two new implementations, LBM-Ghost and LBM-Swap, which are deeply analyzed, presenting the pros and cons of each of them. Copyright © 2017 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Computational Fluid Dynamics, Lattice-Boltzmann Method, GPU, CUDA

1. INTRODUCTION

GPUs are today an efficient alternative to other architectures, in particular due to theirs computational capacity and efficient power consumption. Many software packages have already been ported to take advantage of GPU computing. Although there are some applications or solvers that can be difficult to tune [29, 31] for GPUs. Fortunately, other solvers are particularly well suited for GPU acceleration and are able to achieve significant performance improvements. Lattice Boltzmann method (LBM) [22] is one of these examples due to its inherently data-parallel nature. The parallelism is abundant in LBM which is also amenable to fine grain parallelization. This is particularly interesting for GPU computing. The benefit of using LBM on parallel computers is consistently confirmed in many works [2, 20], for a large number of different problems and computing platforms. In [18], T. Pohl et al. improved the temporal locality for cache-based multicore architectures. P. R. Rinaldi et al. [20] reduced the number of accesses to global memory by using a different ordering for the LBM-steps causing a high impact on performance for GPU computing. LBM has been tested in multiples parallel platforms, such as multicore [18], hardware accelerators [20],[2],[26] and distributed-memory computers [6, 9, 15]. Also we can find many tools [6],[9],[34],[17] based on LBM, which has consolidated LBM in academia and industry. In this work, we use one of then, the LBM-HPC package [9].

The demand of computational resources from scientific community is constantly increasing in order to simulate more and more complex scenarios. One of the most important challenges

*Correspondence to: pedro.valero@bsc.es.

to deal with such scenarios is the large memory capacity that the scientific applications need. Multiple works have explored new techniques to reduce the impact of some applications on memory capacity [35, 10, 11, 36, 37]. Although LBM is amenable to be efficiently parallelized, it is in need of a high memory capacity. Our main motivation is the developing of two new approaches, *LBM-Ghost* and *LBM-Swap*, which minimize the demand of memory for LBM simulations over NVIDIA GPUs. In *LBM-Ghost*, we propose the use of *ghost-cells* to minimize the memory requirements. The implementation of this idea is in needs of non-trivial optimizations in comparison to the state-of-the-art implementations, which make difficult its implementation. The present work extends the previously published works [25],[4] with additional contributions. Ihis work includes a new approach to minimize the memory demand for LBM simulations, *LBM-swap*. This approach is based on the work developed by J. Latt [8], which is adapted to NVIDIA GPUs in the present work. Unlike the *LBM-Ghost*, the *LBM-Swap* is much easier to implement. Both initiatives allow us to execute bigger problems over the same platform, avoiding computationally expensive memory transfers.

The remainder of this paper is organized as follows. In Sec. 2 we introduce the general numerical and implementation framework for LBM. After that Sec. 3 and 4 describe the different optimizations and parallel strategies envisaged to achieve high-performance when dealing with large simulation domains. Finally, we discuss the performance results of the proposed techniques in Sec. 5. We conclude in Sec. 6 with a summary of the main contributions of this work.

2. LATTICE-BOLTZMANN METHOD

Most of the current solvers simulate the transport equations (heat, mass, and momentum) at macroscopic scale [33]. Otherwise, the medium can be also seen from a microscopic viewpoint where tiny particles (molecules and atoms) collide with each other (molecular dynamic) [14]. In this scale, where there is no definition of viscosity, heat capacity, temperature, pressure, etc., the inter-particle forces as well as the location, velocity, and trajectory of each of the particles must be computed, being extremely expensive computationally [14]. Other methods use statistical mechanisms to connect the microscopic and macroscopic worlds. The use of these methods does not require the management of every individual particle, obtaining the important macroscopic effects with manageable computer resources. This is the main idea of the Boltzmann equation and the mesoscopic scale [14].

Previous works have compared the numerical accuracy of the LBM with respect to other methods based on Navier-Stokes (see [1, 7]). They showed that LBM presents an equivalent precision over a large number of applications. There are multiple applications where LBM has been used; high Reynolds turbulent flows [12], aeroacoustics problems [13], bio-engineering applications [2], among others. Also, LBM can be integrated with other methods, such as the Immersed Boundary Method for Fluid-Solid Interaction problems [26],[30].

LBM combines some features of 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 [5]. The standard LBM [19] 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 scale. At this level, the fluid is modeled by a distribution function of the microscopic particle (f). LBM solves the particles 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

DOI: 10.1002/cpe

$f_i(x,t)$, 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 next 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 study we use the standard two-dimensional 9-speed *lattice D2Q9* [5].

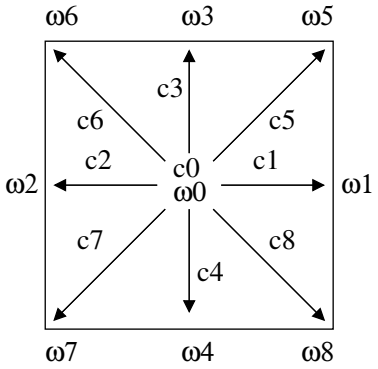

Figure 1. The standard two-dimensional 9-speed lattice (*D2Q9*) [16].

The operator Ω computes the changes caused by the collision between particles at microscopic scale, which is defined by the function (f). To calculate this operator we consider the *BGK* (Bhatnagar-Gross-Krook) formulation [16] 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 [19]:

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

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:

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

which can be compactly written as:

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

DOI: 10.1002/cpe

Algorithm 1 LBM *pull*.

1: **for** $ind = 1 \rightarrow Nx \cdot Ny$ **do**
2: **Streaming**
3: **for** $i = 0 \rightarrow 8$ **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** $i = 0 \rightarrow 8$ **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** $i = 0 \rightarrow 8$ **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**

The macroscopic velocity u in equation 4 must satisfy a Mach number requirement $\mid u \mid /c_s \approx M << 1$, which can be seen as the Courant Friedrichs Lewy (CFL) number 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:

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

Streaming stage:

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

LBM exhibits a high degree of parallelism, which is amenable to fine granularity (one thread per lattice node), as the computing of each of the *lattice* points is completely independent. To carry out LBM-streaming in parallel, we need two different distribution functions (f_1 and f_2 in Algorithm 1).

We decided to work with the *pull* approach (introduced by [32]). This approach has been widely analyzed in many studies in [20],[26],[30]. This is implemented via a single-loop where each *lattice* node can be independently computed by performing one complete time step of LBM. This implementation is given in Algorithm 1. Basically, the *pull* approach fuses in a single loop (that iterates over the entire domain), the computation of both LBM-operations, LBM-collision and LBM-streaming, to improve temporal locality. Furthermore it is not in need of any synchronization among these operations. Also, it minimizes the pressure on memory with respect to other approaches, as the macroscopic level can be completely computed on the highest levels of memory hierarchy (registers/L1 cache).

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. We exploit

DOI: 10.1002/cpe

coalescence by using a Structure of Array (SoA) approach. This idea (*pull-coalescing*) has proven to be a fast implementation in multicore and GPUs architectures [20],[26],[3],[30]. The discrete distribution function f_i is stored sequentially in the same array (see Fig. 2, where *Nx* and *Ny* are the number of horizontal and vertical fluid nodes respectively). In this way, consecutive threads access to contiguous memory locations.

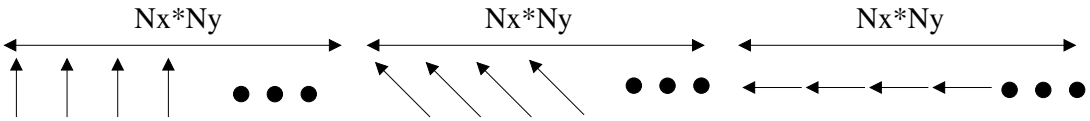

Figure 2. SoA data layout to store the discrete distribution function f_i in memory.

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 [26]. *Lattice* nodes are distributed across GPU cores using a fine-grained distribution (Fig. 3).

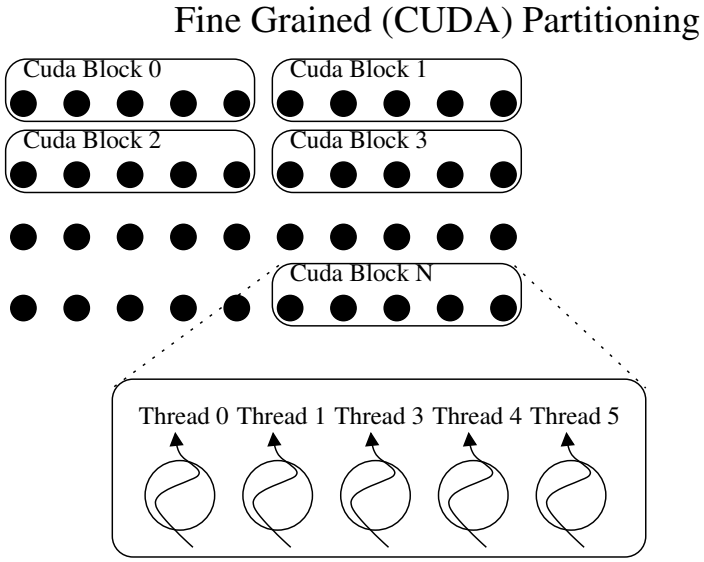

Figure 3. Fine-grained distributions of the *lattice* nodes.

In order to exploit the parallelism found in the LBM, previous studies make use of two different data set [20],[6],[9],[17],[30]. Basically, it consists of using an *AB* scheme [26] 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 propose two alternatives that follow an *AA* scheme to reduce such high memory requirements, one by adapting the use of *ghost cell* to LBM, and one by adapting the *LBM-swap* approach to our platform (NVIDIA GPUs).

3. LBM-GHOST CELLS

This section explains how we have adapted the use of *ghost cells* to LBM to reduce the memory requirements for GPU-based implementations.

Although, the *ghost cells* are usually used for communication in distributed memory systems [27], 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 *ghost cells* we use the biggest size of CUDA

DOI: 10.1002/cpe

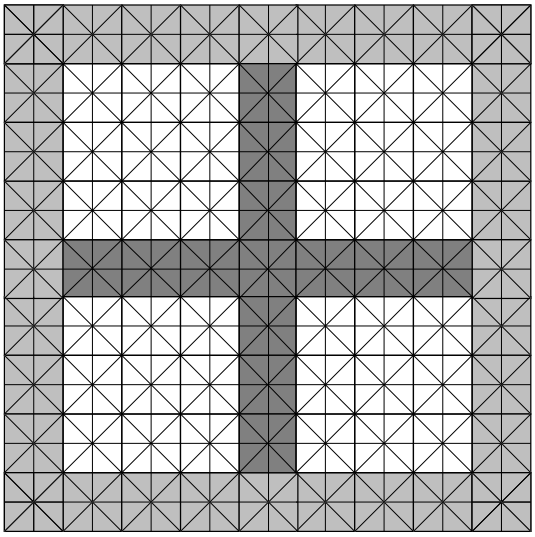

Figure 4. A simple scheme for our LBM approach composed by four *fluid blocks* (CUDA blocks) composed by ghost-cells (dark-gray background), boundary (light-gray background), and fluid (white-background) units.

block 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. 4 illustrates a simple scheme for our interpretation of the *ghost cell* strategy applied to LBM (*LBM-Ghost*).

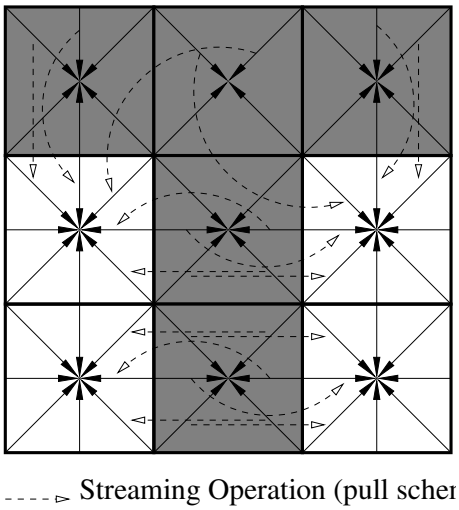

Figure 5. Streaming operation from *ghost cells* to fluid units.

In LBM-streaming operation (Fig. 5), some of the lattice-speed in 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 needs to be computed for the LBM-streaming operation. For instance, if one fluid element is located in one of the corners of the *fluid block*, this requires to take 5 lattice-speed from 3 different *ghost cells*. However, if it is in other position of the boundary, it have to take 3 lattice-speed from one *ghost cell* (Fig. 5).

The information of the *ghost cells* have to be updated once per time step. The updating is computing via a second kernel before computing LBM. This kernel moves some lattice-speed from

DOI: 10.1002/cpe

lattice units to *ghost cells*. This CUDA kernel is computed 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. 6 shows the differences between each of the cases regarding the location. Similarly to the LBM-streaming, a different number of memory movements are necessary depending on the position of the *ghost cells*. In particular, if one *ghost cell* is located in one of the *ghost cell* rows or columns (Vertical and Horizontal cases in Fig. 6), 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. 6), then this *ghost cell* requires 4 lattice-speed from 4 fluid units.

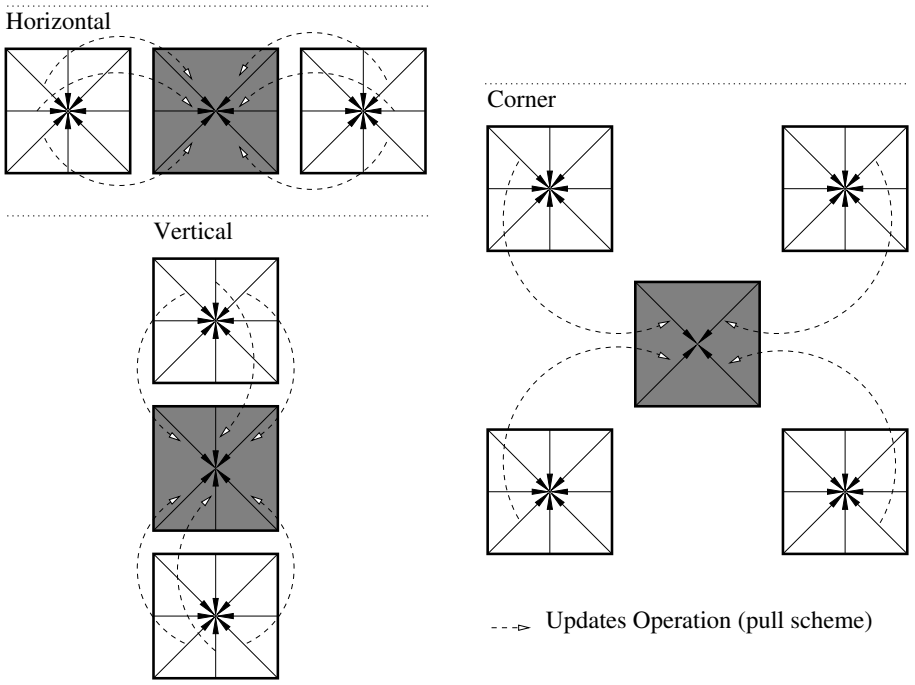

Figure 6. Update operation from fluid units (white background) to *ghost cells* (gray background), depending on *ghost cells* position.

Unlike the *LBM-Standard* implementation (*pull* approach) on GPU, the CUDA blocks need to be synchronized before computing collision. This is possible using *__syncthreads()* (see Algorithm 1). The synchronizations and *ghost cells* make possible the absence of race conditions.

It is well known that the memory management has an impressive impact on performance, in particular on those parallel computers that suffer from a high latency such as, NVIDIA GPUs or Intel Xeon Phi [26]. 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. 3 does not 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. 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 locations for the whole simulation domain (Fig. 2), we map the set of lattice-speed of every bi-dimensional CUDA (*fluid*) block in consecutive memory locations, as graphically illustrated by Fig. 7.

DOI: 10.1002/cpe

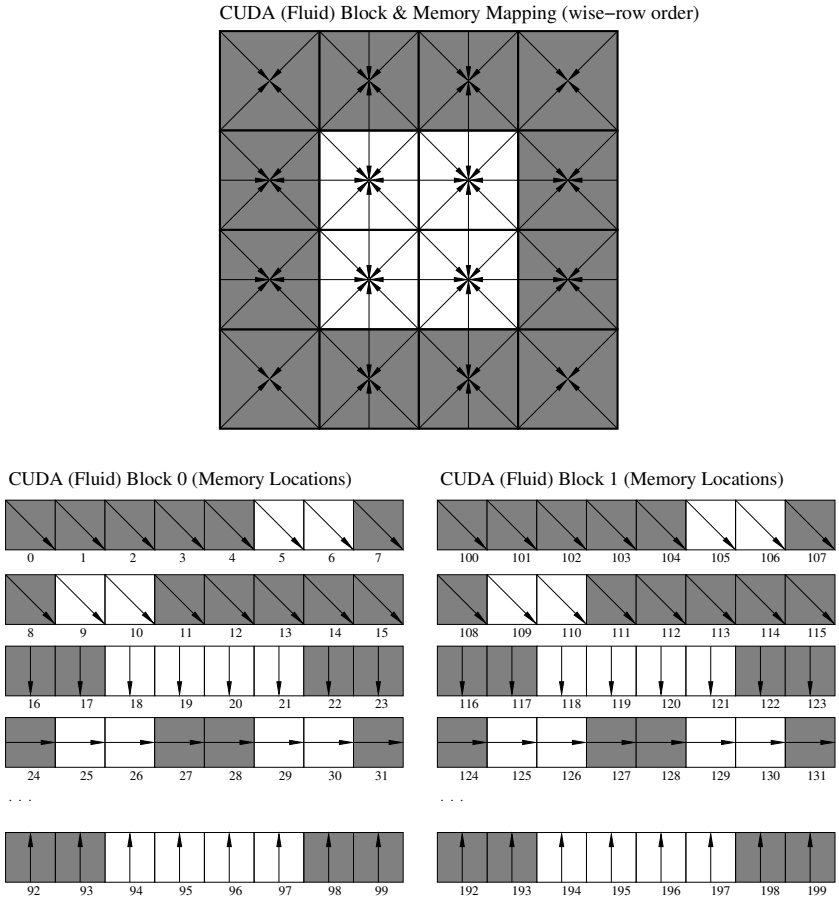

Figure 7. Memory and CUDA block mapping for the *1 lattice + ghost* approach.

4. LBM-SWAP

In this section, we explore other strategy to minimize the memory requirements for LBM simulations on NVIDIA GPUs. Unlike the previous strategy, this approach (*LBM-swap*) does not need more memory (*ghost cells*) or change the data layout. This makes much easier the implementation and the integration with the CPU for heterogeneous implementations [26, 23, 30, 24, 28]. The *LBM-swap* algorithm only needs one lattice-speed data space. For sake of clarity and make easier the understanding in the rest of this section, let's define the *opposite* relation as follow:

$$c_{opposite(i)} = -c_i \tag{7}$$

The *LBM-swap* consists of swapping the lattice-speed after computing the two-main LBM steps, collide and streaming. In this way we avoid race conditions among neighbor lattice. The *swap* function can be implemented as Pseudocode 8 describes.

Algorithm 2 Swap implementation.

```
1: void swap(double * a, double * b){
2: double tmp = a;
3: a = b;
4: b = tmp;
5: }
```

Basically, this approach is based in the next property:

DOI: 10.1002/cpe

$$f_i\left(x + e_i\Delta t, t + \Delta t\right) \leftarrow f_{opposite(i)}(x, t) \tag{8}$$

which is symmetric and then it can be reverted by using the property $i = opposite(opposite(i))$ and Eq. 7 obtaining:

$$f_{opposite(i)}(x, t + \Delta t) \leftarrow f_i\left(x + e_i\Delta t, t\right) \tag{9}$$

As the *LBM-Ghost*, here we need two kernels, one LBM-collision and one for LBM-streaming. Due to GPU programming and architecture, it is necessary a strong point of synchronism among both steps to guarantee the absence of race conditions among them. This is because of the use of one lattice (*AA scheme*) instead of 2-lattice (*AB scheme*). In each kernel we have as many threads as number of lattice (fluid) nodes. We use the same CUDA thread and memory mapping used for the *LBM-Standard* approach (Figures 2 and 3).

Pseudocode 3 describes the first kernel of the *LBM-Swap*. As shown, apart of using one lattice-space (f in pseudocode 3), the only difference with respect to the *LBM-Standad* consists of computing a swap operation after *LBM-collision* on all lattice nodes.

Algorithm 3 LBM-swap, kernel collision.

1: $ind = threaIdx$
2: **for** $i = 0 \rightarrow 8$ **do**
3: $\quad \rho+ = f[i][ind]$
4: $\quad u_x+ = c_x[i] \cdot f[i][ind]$
5: $\quad u_y+ = c_y[i] \cdot f[i][ind]$
6: **end for**
7: $u_x = u_x/\rho$
8: $u_y = u_y/\rho$
9: **for** $i = 0 \rightarrow 8$ **do**
10: $\quad cu = c_x[i] \cdot u_x + c_y[i] \cdot u_y$
11: $\quad f_{eq} = \omega[i] \cdot \rho \cdot (1 + 3 \cdot cu + cu^2 - 1.5 \cdot (u_x)^2 + u_y)^2$
12: $\quad f[i][ind] = f[i][ind] \cdot (1 - \frac{1}{\tau}) + f_{eq} \cdot \frac{1}{\tau}$
13: **end for**
14: **Swapping**
15: **for** $i = 0 \rightarrow 4$ **do**
16: $\quad swap(f[i][ind], f[i+4][ind])$
17: **end for**

The second kernel is implemented as Pseudocode 4 describes. Basically it consists of computing streaming and swap, as described in Eq. 9.

Algorithm 4 LBM-swap, kernel streaming.

1: $ind = threaIdx$
2: **for** $i = 0 \rightarrow 4$ **do**
3: $\quad x_{stream} = x + c_x[i]$
4: $\quad y_{stream} = y + c_y[i]$
5: $\quad ind_{stream} = y_{stream} \cdot Nx + x_{stream}$
6: $\quad$ **Streaming and Swapping**
7: $\quad swap(f[i+4][ind], f[i][ind_{stream}])$
8: **end for**

For sake of clarity, Figure 8 graphically illustrates the swapping carried out in both LBM steps, collision (top) and streaming (bottom):

5. PERFORMANCE EVALUATION

To carry out the experiments we have used one NVIDIA Kepler (K20c) GPU with 2496 CUDA cores at 706 Mhz and 5GB GDDR5 of memory. More details about the specific architecture that

DOI: 10.1002/cpe

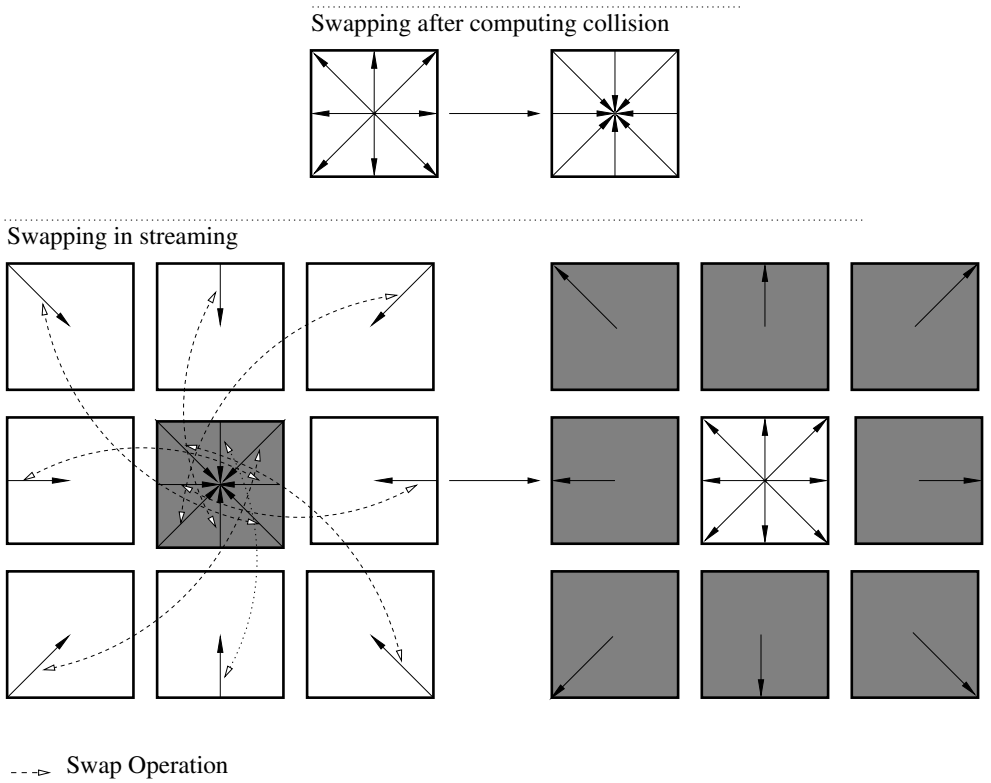

Figure 8. Swap operation in LBM-collision (top) and in the LBM-streaming (bottom).

have been used for performance evaluation are given in Table I. The memory hierarchy of the GPU has been configured as 16KB shared memory and 48KB L1, since our codes can not take advantages form a bigger shared memory. We have considered the most appropriate size of fluid block for each of the tests.

Platform	NVIDIA GPU Kepler K20c
Model	Kepler K20c
Frequency	0.706
Cores	2496
On-chip Mem.	SM 16/48KB (per MP) L1 48/16KB (per MP) L2 768KB (unified)
Memory	5GB GDDR5
Bandwidth	208 GB/s
Compiler	nvcc 6.0.67
Compiler Flag	-O3 -arch = sm 35

Table I. Summary of the main features of the platforms used.

Big fluid domains (from 45 millions of nodes) can not be fully stored in global memory, which forces us to execute our problem in two-steps, when using *LBM-Standard*, requiring additional memory transfers. In this case, several sub-domains must be transfered from GPU to CPU and vice-versa every temporal iteration, causing a big fall in performance. Otherwise, the *LBM-Ghost* is able to achieve a better performance when dealing with big problems.Although this approach is in need of 2 kernels, instead of 1 as in the *LBM-Standard*, the time required by the new kernel (*Ghost* in Fig. 9), which is in charge of updating the information in the *ghost elements*, does not cause a

DOI: 10.1002/cpe

significant overhead. Indeed the time consumed is less than 2% with respect to the total consumed time.

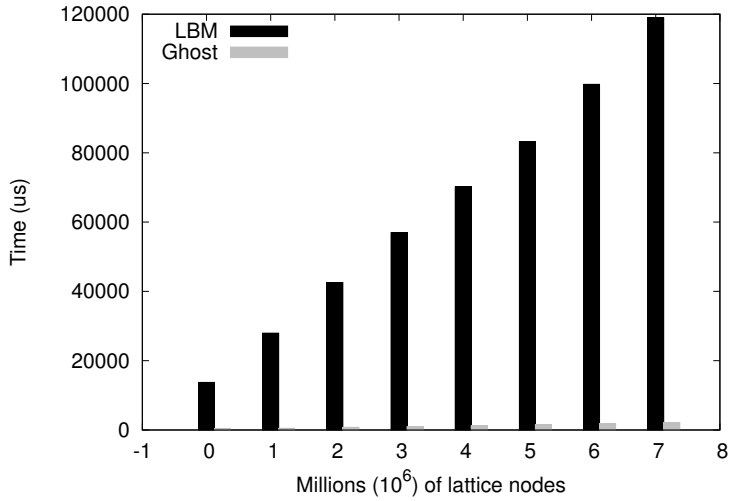

Figure 9. Execution time for the *LBM-Ghost* approach.

The main motivation of this work consists of reducing the memory requirements for LBM simulations on GPUs. The reduction achieved by *LBM-Ghost* represents about the 55% of the memory consumed by the *LBM-Standard*. On the other hand, the *LBM-Swap* is in need of the lowest memory requirements with respect to the other two implementations, needing the half of the memory required by the *LBM-Standard* and about a 5% less than the *LBM-Ghost*. Using both approaches, bigger simulations can be computed without additional and computationally expensive memory transfers.

Most of the LBM studies include the MFLUPS (Millions of Fluid Lattice Updates Per Second ratio) as a metric. As a reference, we also estimate the ideal MFLUPS [21] regarding our platform (K20c):

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

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 streaming step and three read and write operations in the collision step, and the factor 8 is for double precision (8 bytes).

Fig. 10 illustrates the MFLUPS achieved by all the implementations tested and an estimation for the ideal MFLUPS for our platform. The *LBM-Standard* approach is close to ideal performance for "small" problems (until 36 millions of fluid units), being the *LBM-Ghost* almost a 10% slower, due to a more complex implementation The *LBM-swap* is positioned as the slowest implementation tested. This implementation is about a 30% and 40% slower than the *LBM-Ghost* and *LBM-Standard* respectively. This is mainly because of the swap operation carried out at the end of the collision-kernel (Pseudocode 3).

However, when bigger domains are considered (from 45 to 72 millions of fluid units), the *LBM-Standard* turns out to be very inefficient, causing an important fall in performance. In contract, the performance achieved by the other two approaches, *LBM-Ghost* and *LBM-Swap*, keeps constant for the rest of tests. Also, as reference, we included the performance achieved by the GPU based implementation provided in the sailfish package [6], which is slower than *LBM-Standard* and *LBM-Ghost* and faster than *LBM-Swap* for small simulations.

Fig. 11 illustrates the speedup, in terms of MFLUPS, achieved by the *LBM-Ghost* and *LBM-Swap* implementations over the *LBM-Standard*. Both approaches (*LBM-Ghost* and *LBM-Swap*) are slower

DOI: 10.1002/cpe

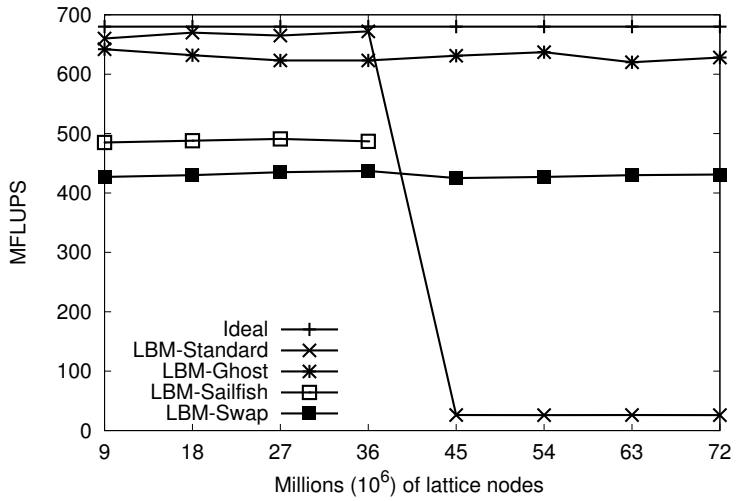

Figure 10. MFLUPS reached by each of the approaches.

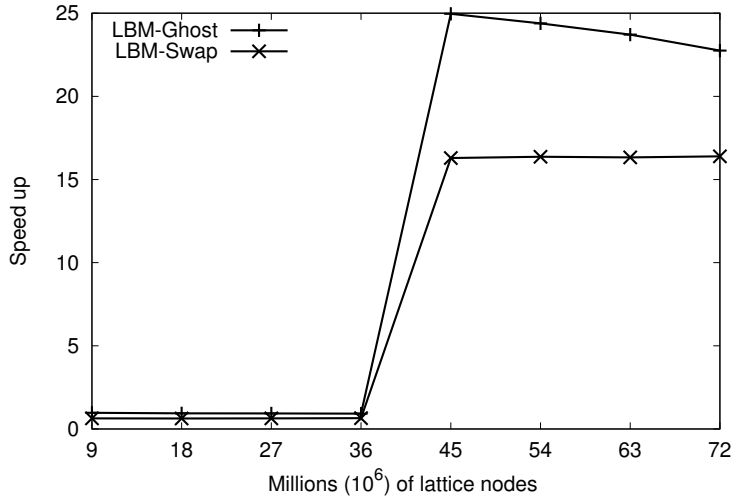

Figure 11. Speedup achieved by the *LBM-Ghost* and *LBM-Swap* on the *LBM-Standard*.

than the *LBM-Standard* counterpart when executing simulations equal or smaller than 36 millions of fluid units, however the *LBM-Standard* turns to be the slowest when dealing with bigger domains. The *LBM-Ghost* is able to achieve a peak speedup equal to 25, while the peak speedup achieved by the *LBM-Swap* is about 16.

6. 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 two new alternatives, *LBM-Ghost* and *LBM-Swap*, which reduce the memory used and keep a high performance for large simulations. It was carried out a detailed performance analysis in terms of time, memory requirements, speedup and MFLUPS ratio. The implementation proposed are thoroughly detailed.

DOI: 10.1002/cpe

Although the *LBM-Ghost* achieves a high performance when dealing with big simulations, it makes use of non-trivial optimizations, which makes difficult its implementation. Otherwise, the *LBM-Swap* is straight-forward. It basically consists of swapping the lattice-units of each fluid node after computing LBM-collision and LBM-streaming.

Also, it is important to note the *LBM-Ghost* is in need of a different data-layout which may suppose additional overheads (not considered in this work) regarding pre/post-processing to adapt the standard data-layout to/from the particular data-layout used by this approach. On the other hand, the *LBM-Swap* is not in need of a different data layout with respect to the *LBM-Standard*, so that no pre/post-processing is needed.

ACKNOWLEDGMENTS.

This project was founded by the Spanish Ministry of Economy and Competitiveness (MINECO): BCAM Severo Ochoa accreditation SEV-2013-0323, MTM2013-40824, Computación de Altas Prestaciones-VII TIN2015-65316-P, by the Basque Excellence Research Center (BERC 2014-2017) program by the Basque Government, and by the Departament d' Innovació, Universitats i Empresa de la Generalitat de Catalunya, under project MPEXPAR: Models de Programació i Entorns d' Execució Paral·lels (2014-SGR-1051). We also thank the support of the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT) and NVIDIA GPU Research Center program for the provided resources, as well as the support of NVIDIA through the BSC/UPC NVIDIA GPU Center of Excellence.

REFERENCES

1. Lilit Axner, Alfons G. Hoekstra, Adam Jeays, Pat Lawford, Rod Hose, and Peter MA Sloot. Simulations of time harmonic blood flow in the mesenteric artery: comparing finite element and lattice boltzmann methods. *BioMedical Engineering OnLine*, 2000.
2. M. Bernaschi, M. Fatica, S. Melchiona, S. Succi, and E. Kaxiras. A flexible high-performance lattice boltzmann gpu code for the simulations of fluid flows in complex geometries. *Concurrency Computa.: Pract. Exper.*, 22:1–14, 2010.
3. M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras. 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):114, 2010.
4. Jesús Carretero, Javier García Blas, Ryan K. L. Ko, Peter Mueller, and Koji Nakano, editors. *Algorithms and Architectures for Parallel Processing - 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings*, volume 10048 of *Lecture Notes in Computer Science*. Springer, 2016.
5. Xiaoyi He and Li-Shi Luo. A priori derivation of the lattice boltzmann equation. *Phys. Rev. E*, 55:R6333–R6336, Jun 1997.
6. M. Januszewski and M. Kostur. Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method. *Computer Physics Communications*, 185(9):2350–2368, September 2014.
7. S. Kollmannsberger, S. Geller, A. Dster, J. Tlke, C. Sorger, M. Krafczyk, and E. Rank. 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):817–845, 2009.
8. Jonas Latt. Technical report: How to implement your ddqq dynamics with only q variables per node (instead of 2q). In *Tufts University*, pages 1–8, 2007.
9. LBM-HPC. Last access on 26-04-2016 to http://www.bcamath.org/en/research/lines/CFDCT/software.
10. K. Li, W. Yang, and K. Li. Performance analysis and optimization for spmv on gpu using probabilistic modeling. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):196–205, Jan 2015.
11. K. Li, W. Yang, and K. Li. A hybrid parallel solving algorithm on gpu for quasi-tridiagonal system of linear equations. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2795–2808, Oct 2016.
12. O. Malaspinas and P. Sagaut. Consistent subgrid scale modelling for lattice boltzmann methods. *Journal of Fluid Mechanics*, 700:514–542, 2012.
13. Simon Marié, Denis Ricot, and Pierre Sagaut. Comparison between lattice boltzmann method and navier-stokes high order schemes for computational aeroacoustics. *J. Comput. Phys.*, 228(4):1056–1070, March 2009.
14. A. A. Mohamad. *The Lattice Boltzmann Method - Fundamental and Engineering Applications with Computer Codes*. Springer, 2011.
15. Christian Obrecht, Frdric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. Scalable lattice boltzmann solvers for {CUDA} {GPU} clusters. *Parallel Computing*, 39(67):259 – 270, 2013.
16. E. Gross P. Bhatnagar and M. Krook. A model for collision processes in gases. i: small amplitude processes in charged and neutral one-component system. *Phys. Rev. E*, 94:511–525, 1954.

17. CFD Complex Physics Palabos. Last access on 26-04-2016 to urlhttp://www.palabos.org/.
18. T. Pohl, M. Kowarchik, J. Wilke, and U. Rüde K. Iglberger. Optimization and profiling of the cache performance of parallel lattice boltzmann codes. *Parallel Processing Letters*, 13(4):549560, 2003.
19. Y. H. Qian, D. D'Humires, and P. Lallemand. Lattice bgk models for navier-stokes equation. *EPL (Europhysics Letters)*, 17(6):479, 1992.
20. P.R. Rinaldi, E.A. Dari, M.J. Vnere, and A. Clausse. A lattice-boltzmann solver for 3d fluid simulation on {GPU}. *Simulation Modelling Practice and Theory*, 25(0):163 – 171, 2012.
21. Aniruddha G. Shet, Shahajhan H. Sorathiya, Siddharth Krithivasan, Anand M. Deshpande, Bharat Kaul, Sunil D. Sherlekar, and Santosh Ansumali. Data structure and movement for lattice-based simulations. *Phys. Rev. E*, 88:013314, Jul 2013.
22. Sauro Succi. *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.
23. Pedro Valero-Lara. Accelerating solidfluid interaction based on the immersed boundary method on multicore and gpu architectures. *The Journal of Supercomputing*, pages 1–17, 2014.
24. Pedro Valero-Lara. A fast multi-domain lattice-boltzmann solver on heterogeneous (multicore-gpu) architectures. *14th International Conference Computational and Mathematical Methods in Science and Engineering*, 4:1239–1250, 2014.
25. Pedro Valero-Lara. Leveraging the performance of LBM-HPC for large sizes on gpus using ghost cells. In *Algorithms and Architectures for Parallel Processing - 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings*, pages 417–430, 2016.
26. Pedro Valero-Lara, Francisco D. Igual, Manuel Prieto-Matas, Alfredo Pinelli, and Julien Favier. Accelerating fluidsolid simulations (lattice-boltzmann & immersed-boundary) on heterogeneous architectures. *Journal of Computational Science*, 10:249–261, 2015.
27. Pedro Valero-Lara and Johan Jansson. 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*, pages 318–321, 2015.
28. Pedro Valero-Lara and Johan Jansson. Heterogeneous cpu+ gpu approaches for mesh refinement over lattice-boltzmann simulations. *Concurrency and Computation: Practice and Experience*, 2016.
29. Pedro Valero-Lara, Alfredo Pinelli, Julien Favier, and Manuel Prieto Matias. Block tridiagonal solvers on heterogeneous architectures. In *Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, ISPA '12, pages 609–616, Washington, DC, USA, 2012. IEEE Computer Society.
30. Pedro Valero-Lara, Alfredo Pinelli, and Manuel Prieto-Matias. Accelerating solid-fluid interaction using lattice-boltzmann and immersed boundary coupled simulations on heterogeneous platforms. *Procedia Computer Science*, 29(0):50 – 61, 2014. 2014 International Conference on Computational Science.
31. Pedro Valero-Lara, Alfredo Pinelli, and Manuel Prieto-Matias. Fast finite difference poisson solvers on heterogeneous architectures. *Computer Physics Communications*, 185(4):1265 – 1272, 2014.
32. G. Wellein, T. Zeiser, G. Hager, and S. Donath. On the single processor performance of simple lattice boltzmann kernels. *Computers & Fluids*, 35(89):910 – 919, 2006. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.
33. John F. Wendt and John David Anderson. *Computational Fluid Dynamics: An Introduction*. Springer, 2008.
34. Next Generation of CFD XFlow. Last access on 26-04-2016 to urlhttp://www.xflowcfd.com/.
35. W. Yang, K. Li, Z. Mo, and K. Li. Performance optimization using partitioned spmv on gpus and multicore cpus. *IEEE Transactions on Computers*, 64(9):2623–2636, Sept 2015.
36. Wangdong Yang, Kenli Li, and Keqin Li. A hybrid computing method of spmv on cpugpu heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 104:49 – 60, 2017.
37. Yu Ye, Kenli Li, Yan Wang, and Tan Deng. Parallel computation of entropic lattice boltzmann method on hybrid cpugpu accelerated system. *Computers & Fluids*, 110:114 – 121, 2015. ParCFD 2013.

DOI: 10.1002/cpe