2015 IEEE 18th International Conference on Computational Science and Engineering

# Multi-Domain Grid Refinement for Lattice-Boltzmann Simulations on Heterogeneous Platforms

Pedro Valero-Lara
*CFD & Computational Technology Unit*
*Basque Center for Applied Mathematics (BCAM)*
*Bibao, Spain*
*pvalero@bcamath.org*

Johan Jansson
*CFD & Computational Technology Unit and*
*Computational Technology Laboratory*
*Basque Center for Applied Mathematics (BCAM) and*
*KTH Royal Institute of Technology*
*Bilbao, Spain and Stockholm, Sweden*
*jjansson@bcamath.org*

***Abstract*—The main contribution of the present work consists of several parallel approaches for grid refinement based on a multi-domain decomposition for lattice-Boltzmann simulations. The proposed method for discretizing the fluid incorporates different regular Cartesian grids with no homogeneous spatial domains, which are in need to be communicated each other.**

**Three different parallel approaches are proposed, homogeneous Multicore, homogeneous GPU, and heterogeneous Multicore-GPU. Although, the homogeneous implementations exhibit satisfactory results, the heterogeneous approach achieves up to 30% extra efficiency, in terms of Millions of Fluid Lattice Updates per Second (MFLUPS), by overlapping some of the steps on both architectures, Multicore and GPU.**

***Keywords*-Parallel Computing; Lattice-Boltzmann Method; Multi-Domain Grid Refinement; Heterogeneous (Multicore-GPU) Platforms;**

## I. Introduction

The main objective of this work consists of obtaining a fast grid-refinement implementation based on lattice-Boltzmann method (LBM) by using Multicore and GPU. In particular, it is proposed an heterogeneous approach which distributes either to GPU or Multicore the different parts of the whole solver depending on their computational cost.

The LBM is a clever discretization of the Boltzmann equation, which is widely used in numerous numerical tools for Computational Fluid Dynamics (see, for instance [6], [8]). In particular, we have considered the LBM-HPC framework [8] as our reference software tool. LBM is an efficient and fast method, however the usage of uniform Cartesian grids is expensive. Although scientific problems exist for which a uniform grid is a reasonable choice, it is usually desirable to resolve regions of high geometrical complexity with a finer grid to minimize the computational cost. Nevertheless, the refinement operation induces a strong discontinuity in the physical quantities at the grid transition and can therefore give rise to artifacts in the solution.

Several refinement techniques have been implemented for LBM-based solvers, such as adaptive mesh refinement (AMR) [19], multi-grid [15], and multi-domain [14]. Each of these techniques exhibit its own advantages and disadvantages. For AMR and multi-grid the coarse grid is present all over the simulation domain. In the multi-domain refinement, the regions where refined patches are inserted are taken off the coarse grid. The approaches based on AMR present the most complex scenario for data management, due mainly to its dynamic data structure, while the multi-grid operations are more profitable for programming and data management. In particular, we choose the multi-domain approach in order to have better performance and higher memory savings. However, the coupling between grids is more complex.

Classical fluid solvers based on the unsteady incompressible Navier-Stokes equations may turn out to be inefficient or difficult to tune to achieve maximum performance on heterogeneous platforms [11], [12]. A choice that better meets the GPU hardware is based on modeling the fluid flow through the Lattice Boltzmann method (LBM). Several recent works have shown that the combination of GPU-based platforms and methods based on the LBM algorithm can achieve impressive performance due to the intrinsic characteristics of the algorithm [1], [5], [2], [3]. Certainly, the computing stages of LBM are amenable to fine grain paralelization (see for example [4], [5] and references therein). Recently, the features of the LBM were efficiently ported on other co-processors, such as Intel Xeon Phi (see [17]).

Not many works extend the parallel efficiency of LBM to cases involving multi-domain refinement techniques. A very recent work that covers a subject closely related with the present contribution is [15],

978-1-4673-8297-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSE.2015.9

where a new and efficient 2D implementation of LBM method for muti-grid flows is presented. Here, we focus on a different approach based on multi-domain coupled with LBM introduced in the work presented by D. Lagrava et al. [14]. This methodology has been analyzed deeply and validated in several numerical scenarios (see [14], [7]), so that we focus on the implementation techniques adopted to keep the solver highly efficient on Multicore-GPU heterogeneous platforms. The present work extends the previously published work [2] with additional contributions. In particular, it includes a comparative study among the two standard LBM implementations, pull and push, over our GPU platform, and different approaches for memory mapping applied to LBM on Multicore. We include the use of two grid-refined levels. We also follow the CUDA extension *Dynamic Parallelism* to deal with multiple domains on GPU.

This paper is structured as follows. Section II briefly introduces the physical problem at hand and the general numerical framework that has been selected to cope with it: LBM coupled with multi-domain refinement technique. We also detail the specific potential parallel features of LBM and the parallel strategies envisaged to optimally enhance the performance (Section III) of the multi-domain LBM algorithm on homogeneous Multicore/GPU and Multicore-GPU heterogeneous platforms. Finally, Section IV contains a performance analysis of the proposed techniques and in Section V some conclusions are outlined.

## II. Multi-Domain Grid Refinement on Lattice-Boltzmann Method

LBM has been extensively used in past decades (see [9] for a complete overview) and now is regarded as a powerful and efficient alternative to classical Navier Stokes solvers. In what follows, we briefly recall the basic formulation of the method. The LBM is based on an equation that governs the evolution of a discrete distribution function $f_i(\mathbf{x},t)$ describing the probability of finding a particle at Lattice site $\mathbf{x}$ at time $t$ with speed $\mathbf{v}=\mathbf{e}_i$. In this work, we consider the *BGK* formulation [10] that relies upon an unique relaxation time $\tau$ toward the equilibrium distribution $f_i^{eq}$:

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

The particles can move only along the links of a regular Lattice defined by the discrete speeds ($e_0 = c(0,0)$; $e_i = c(\pm1,0), c(0,\pm1), i = 1\cdots4$; $e_i = c(\pm1,\pm1), c(\pm1,\pm1), i = 5\cdots8$ with $c = \Delta x/\Delta t$) so that the synchronous particle displacements $\Delta\mathbf{x}_i = \mathbf{e}_i\Delta t$ never take the fluid particles away from the Lattice. For the present study, the standard two-dimensional 9-lattice directions *D2Q9* is used, but all the techniques that will be presented can be extended in a straightforward manner to three dimensional lattices. Essentially, the three-dimensional lattice requires more lattice directions (for instance, 27 directions instead of 9 for the three-dimensional LBM standard *D3Q27*), but the two major LBM steps, stream and collide, are computed identically equal than in the two-dimensional LBM. The equilibrium function $f^{eq}(\mathbf{x},t)$ can be obtained by Taylor series expansion of the Maxwell-Boltzmann equilibrium distribution [13]:

$$f_i^{eq}\left(\mathbf{x},t\right)=\rho\varpi_i\left[1+\frac{\mathbf{e_i}\cdot\mathbf{u}}{c_s^2}+\frac{\left(\mathbf{e_i}\cdot\mathbf{u}\right)^2}{2c_s^4}-\frac{\mathbf{u}^2}{2c_s^2}\right] \quad (2)$$

In equation 2, $c_s$ is the speed of sound ($c_s = 1/\sqrt{3}$) and the weight coefficients $\varpi_i$ are $\varpi_0 = 4/9$, $\varpi_i = 1/9$, $i = 1\cdots4$ and $\varpi_5 = 1/36$, $i = 5\cdots8$ according to the current normalization.

The equation 1 is typically advanced in time in two stages: collision and streaming.

Given $f_i(\mathbf{x},t)$ compute:

$\rho = \sum f_i(\mathbf{x},t)$ and
$\rho\mathbf{u} = \sum \mathbf{e}_i f_i(\mathbf{x},t)$

Collision stage:

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

Streaming stage:

$$f_i\left(\mathbf{x}+\mathbf{e}_i\Delta t,t+\Delta t\right)=f_i^*\left(\mathbf{x},t+\Delta t\right)$$

When using multi-resolution approaches [14], a communication between the grids is needed. In the case of multi-domain methods the communication is done on the boundaries connecting the grids. The coupling is made in two directions: from coarse to fine and from fine to coarse grids. On the boundaries of each refinement level, after a "collide-and-stream" operation there will be some missing information (some populations $f_i$ are unknown on the coarse and on the fine grids) that one needs to reconstruct. For the sake of clarity, let us call $C$ the ensemble of coarse sites and $F$ the ensemble of all fine sites. Let us now define $x_{f\to c}$ the fine sites that are contained in $F$ and $C$ where the coupling from fine to coarse is performed and $x_{c\to f}$ all the sites contained in

$F$ and $C$ where the coupling goes from coarse to fine. Let us also define $x^c_{f\to c} = \{x | x \in x_{f\to c}$ and $x \notin F\}$, $x^c_{c\to f} = \{x | x \in x_{c\to f}$ and $x \notin F\}$ and $x^f_{c\to f} = \{x | x \in x_{c\to f}$ and $x \notin x^c_{c\to f}\}$. The coupling proposed in this work requires the grids to overlap themselves by a domain of at least one coarse cell width, as Figure 1 illustrates.

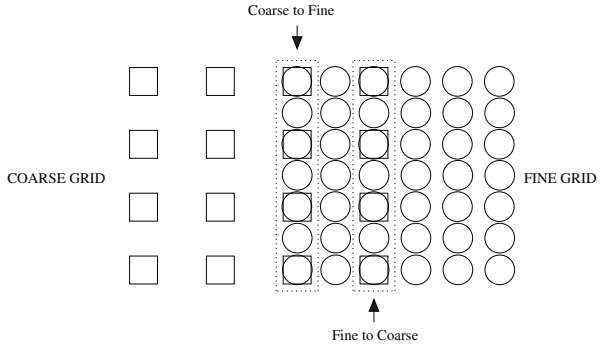


Figure 1. Example of a multi-domain scheme.

In LBM a regular *Cartesian* grid is used. Therefore an abrupt transition occurs when refining the computational domain. This change of scales induces a need for a rescaling of the physical quantities between the grids. To clarify, we chose to refine the grids by a factor of two.

Thus defining $\delta x_c$ and $\delta x_f$ the spatial discretization of the coarse and fine grids respectively one has the following relation between them [14]:

$$\delta x_f = \delta x_c / 2 \tag{3}$$

The temporal loop in the fine grid must do twice the iterations of the coarse grid. Another consequence of the convective scaling, is that the velocity and the pressure in lattice units are continuous fields on the grid transition, while the relaxation time $\tau$ must be rescaled [14].

$$\omega_f = \frac{2\omega_c}{4 - \omega_c} \tag{4}$$

The rescaling of the distribution function $f_i$ now needs to be discussed. The basic ideas of the algorithm are explained in the following. Each $f_{i,n}$ can be written as [14]:

$$f_{i,n} = f^{eq}_i(\rho_n, \mathbf{u}_n) + f^{neq}_{i,n}(\bigtriangledown \mathbf{u}) \tag{5}$$

$f^{eq}_{i,n}$ does not need any rescaling [14], as it only depends on $\rho$ and $\mathbf{u}$ and both are continuous between the grids. On the other hand, the non-equilibrium part $f^{neq}_{i,n} = f_{i,n} - f^{eq}_{i,n}$ is proportional to the gradient of the velocity, it is therefore necessary to rescale it when it is moved between grids with different resolutions [14].

$$f^{neq}_{i,c} = \frac{2\omega_f}{\omega_c} f^{neq}_{i,f} \tag{6}$$

We are now going to discuss in more detail the actual coupling procedure between the coarse and fine grids. In the $F \to C$ boundary, the fine grid has more sites than the coarse one. The necessary steps are: restrict the values, rescale them and copy them to the coarse grid. The proposed coupling is over the sites marked as $x_{f\to c}$. It is expressed by the following equation

$$\begin{aligned} f_{i,c}(x^c_{f\to c}, t) = f^{eq}_i(\rho_f(x^c_{f\to c}, t), \mathbf{u}_f(x^c_{f\to f}, t)) \\ + \frac{2\omega_f}{\omega_c} \overline{f^{neq}}_{i,f}(x^c_{f\to c}, t) \end{aligned} \tag{7}$$

where $\rho_f = \sum_i f_{i,f}$ and $\mathbf{u}_f = \sum_i \mathbf{e}_i f_{i,f}$ and $\overline{f^{neq}}_{i,f}(x^c_{f\to c}, t)$ is the result of applying the restriction to the incoming fine grid values. We carry out only a filter on the non-equilibrium part of the populations $f^{neq}_i$ averaging over all the $q$ lattice directions, thus obtaining the following restriction

$$\overline{f^{neq}}_{i,f}(x^c_{f\to c}, t) = \frac{1}{q} \sum_{i=0}^{q-1} f^{neq}_{i,f}(x^c_{f\to c} + \mathbf{e}_i, t) \tag{8}$$

The coupling over the $C \to F$ boundary ($x_{c\to f}$) is given by two different operations. If a point has a corresponding coarse site in $x_{c\to f}$ (i.e. if a computational node has both a coarse and a fine site, or in a mathematical notation if $x_f \in x^c_{c\to f}$ ) then

$$\begin{aligned} f_{i,f}(x^c_{c\to f}) = f^{eq}_i(\rho_c(x^c_{c\to f}), \mathbf{u}_c(x^c_{c\to f})) \\ + \frac{\omega_c}{2\omega_f} f^{neq}_{i,c}(x^c_{c\to f}), \end{aligned} \tag{9}$$

where $\rho_c = \sum_i f_{i,c}$ and $\mathbf{u}_c = \sum_i \mathbf{e}_i f_{i,c}$ and $f^{neq}_{i,c}$ are computed from the populations of the coarse grid. However, if the fine site does not correspond to a coarse site in $x_{c\to f}$

$$f_{i,f}(x^f_{c\to f}) = f^{eq}_i(\overline{\rho}_c, \overline{\mathbf{u}}_c) + \frac{\omega_c}{2\omega_f} \overline{f^{neq}}_{i,c}, \tag{10}$$

where $\overline{\rho}_c$,$\overline{\mathbf{u}}_c$ and $\overline{f^{neq}}_{i,c}$ are interpolated from the values where the fine and coarse sites are coincident. Next we present a detailed version of the coupling algorithm that we implemented.

1) A "collide-and-stream" operation is performed on the coarse grid bringing it to time $t + \delta t_c$.

At this point the populations in $x_{f\to c}$ that were supposed to be streamed from the fine grid are unknown.

2) A "collide-and-stream" cycle is performed on the fine grid bringing it at time $t + \delta t_c/2$. The grid lacks information in $x_{c\to f}$.
3) $Coarse \rightarrow Fine$ communication. One then performs a double interpolation, one in time and one in space. First the values of $\rho_c$, $\mathbf{u}_c$ and $f_{i,c}^{neq}$ of the coarse sites in $x_{c\to f}$ are interpolated at time $t + \delta t_c/2$. Then the values of the fine sites $\rho_c(t + \delta t/2)$, $\mathbf{u}_c(t + \delta t/2)$ and $f_{i,c}^{neq}(t + \delta t/2)$ are interpolated in space.
4) A second "collide-and-stream" operation is performed on the fine grid, bringing it to time $t + \delta t_c$. At this point we have the information from the coarse grid to complete the fine grid in $x_{c\to f}$.
5) $Fine \rightarrow Coarse$ communication. All the populations of the coarse grid in $x_{c\to f}$ are replaced according Eqs. 7 and 8.

## III. LBM Implementaton

The actual computational scheduling of LBM is based on the works by [5], [3], a novel efficient implementation based on a *pull* single-loop strategy. This approach reduces the number of accesses to main memory by computing the macroscopic variables, velocities and density, in top regions of the hierarchy of memory. To exploit the high data parallelism presented in LBM, we keep in memory two copies of lattice. Each time step inputs from one copy and writes results to the other.

The information for each *lattice* node should be stored in sequential memory locations that reflect their geometrical ordering to improve coalescing. In contrast, other approaches are more efficient for Multicore processors and low latency memories. We have proposed several strategies adapted to LBM.

The first strategy (*Uncoalesced*) does not suppose an elaborate management of memory. Basically, the set of 9 speeds associated to each lattice unit is stored in consecutive locations of memory. The lattice is stored in memory as an Array of Structure (AoS). Although, it can be efficiently implemented for systems which exploit a coarse grain parallelism, this memory mapping makes it difficult to use vector (intrinsic) instructions due to the displacement of data-memory among the lattice-velocities for consecutive lattice units.

To mitigate the inconvenience found in the previous approach, other alternative (*Coalesced*) is given, which consists of storing each of the 9 lattice-directions consecutively. Henceforth, we consider to use a Structure of Array (SoA) approach instead of using AoS. This approach has proven to be a very efficient memory mapping on CUDA-based implementations [5], [3], [16], [15], [20]. Despite that this approach is more amenable to vectorized instructions, we find a large space of memory, as big as the size of the fluid domain, among the different lattice-velocities.

Finally in order to take advantage of the main features of both aforementioned strategies, another approach arises: *Blended*, it consists of joining both features by introducing a chunk among lattice speeds to exploit vectorial instructions efficiently. The size of vector unit imposes the maximum number of elements to be grouped (chunk).

The homogeneous Multicore implementation consists of using OpenMP pragmas, which orchestrates the distribution of the workload over the set of threads. These pragmas are placed before iterative sentences, being very transparent from programmer point of view.

Next, we introduce the homogeneous GPU implementation. First, the *coarse* grid is computed in one kernel. The number of threads is equal to the number of *lattice* nodes. Then, a second *Stream-Collide* step is carried out on the nodes of the *fine* grid. After that, the next step consists of computing the *coarse to fine* communication. In particular this step is carried out on both set of points, *coarse* and *fine*, located in the *coarse to fine* region of the *fine* grid (Figure 1). This step is divided into three different interpolation operations: temporal interpolation on *coarse* points, spatial and temporal interpolations on *fine* points. The first kernel corresponds to the Equation 9. The next two interpolations correspond, first, to the elements $\overline{\rho}_c, \overline{\mathbf{u}}_c$ and $\overline{f^{neq}}_{i,c}$ (spatial-interpolation), and, second (temporal-interpolation), of the Equation 10. The two first interpolations are independent between them, and so, we use a single kernel. The third interpolation is carried out by a separate kernel. After computing the first communication step, the fine grid is completed for a second *Stream-Collide* step. Finally, the $F \rightarrow C$ communication is carried out on the points of the *coarse* grid of the *fine to coarse* region (Equation 7) in a separate kernel. We have used the *Dynamic Parallelism* CUDA extension [18] to implement our homogeneous GPU approach. Dynamic Parallelism enables a CUDA kernel to create and synchronize new nested work. It avoids the use of CPU for synchronizing the different steps for computing multiple domains

on GPU.

In the following, we present our heterogeneous scheduler as an alternative to the homogeneous approaches. The strategy is based on a temporal segmentation of our problem. It takes advantages of both, the independence among some steps and the coupling of non balanced features of our numerical algorithm on our non homogeneous system. Depending on the size of grids, the computational cost concerning different steps is different. To clarify, Figure 2 graphically illustrates the proposed strategy. Our overlapped scheduler is now introduced. The execution of the temporal interpolation on the *coarse* points of the *coarse to fine* region (Equation 9) can be overlapped with the first LBM step of the *fine* grid, since the output of this interpolation is required by the temporal interpolation on *fine* points of the same region (Equation 10), which has to be computed after the first fine-LBM step. The *fine to coarse* communication step is lightly modified with respect the homogeneous GPU approach. In particular, it is implemented by a single OpenMP-function/CUDA-kernel which computes, first, the spatial interpolation on *fine* points, and then, the temporal interpolation on the same points (Equation 10). Additionally, it is possible to compute a *prediction* operation for the next coarse-LBM step, while all the previous steps on *fine* grid are being computed. However, it is necessary to compute a *Stream-Collide* step on points which lack information concerning *coarse* points of the *fine to coarse* region. Although, this step is not overlapped with others, it does not suppose an important overhead, as it is only carried out on the points located in the *fine to coarse* region of the *coarse* grid.

The heterogeneous approach requires more memory transfers with respect to the homogeneous GPU counterpart. In particular, the boundary regions among grids have to be transfered from (to) both memories, main (Multicore) and global (GPU). As Figure 2 shows, after computing the interpolation on the boundary region the missing information is transfered from (to) both grids.

Two different heterogeneous approaches arise (Figure 2), *Top-GPU*, in which GPU computes the top pipeline (Figure 2) and Multicore the bottom pipeline, and *Top-Multicore*, in which Multicore computes the top pipeline and the GPU the bottom pipeline. We have extended this strategy to problems with 2 refined levels, in which we consider the next grid distribution; the two refined domains on Multicore/GPU and the coarsest level on GPU/Multicore. As in the homogeneous GPU implementation, we follow the *Dynamic Parallelism* CUDA extension [18] when dealing with multiple domains on GPU.

## IV. Performance Evaluation

To critically evaluate the performance of our multi-domain LBM implementation, next we consider a number of tests executed on our Multicore-GPU system. More details of the specific architectures are given in Table I. According to memory requirements, the GPU memory hierarchy has been configured as 16KB shared memory and 48KB L1, since our codes do not take advantage of a higher amount of shared memory. All the simulations have been performed using double precision. We use the conventional MFLUPS metric (Millions of Fluid Lattice Updates per Second) reported in most LBM studies. We have used a CUDA block size equal to 256.

| **Platform** | Xeon E5520 | Kepler K20c |
|---|---|---|
| **Cores** | 8 | 2496 |
| **(on-chip** | L1 32KB (per core) | SM 16/48KB (per MP) |
| **Memory)** | L2 512KB (unified) | L1 48/16KB (per MP) |
| | L3 20MB (unified) | L2 768KB (unified) |
| **Memory** | 64GB DDR3 | 5GB GDDR5 |
| **Bandwidth** | 51.2 GB/s | 208 GB/s |
| **Compiler** | gcc 4.6.2 | nvcc 5.5 |

Table I
Details of the test platform.

Based on insights extracted from previous works [1], [2], [3], [4], [5], [6], we have focused on analyzing the performance of LBM on GPU by exploiting the coalesced memory mapping presented in Section 3, and a fine-grain scheme (one thread per lattice unit). We analyze two standard approaches, push (Collide-Stream) [1] and pull (Stream-Collide) [3], for computing LBM over GPU. This study also includes the performance achieved by the sailfish framework [6], which exploits a push-based LBM approach. It is CUDA compatible. We also analyze the three memory storage strategies proposed for our Intel Xeon processor. Vectorization is the main recommendation for boosting the performance over Intel Multicore processors. We have explored Auto-vectorization as a way to maintain a common code baseline among others architectures, guiding the compiler towards an efficient SIMD exploitation.

Figure 3-left graphically illustrates the performance achieved by each of the LBM schedulers, being the pull-LBM (Stream-Collide) more efficient than the push-based approaches counterparts. As in GPU, the pull-LBM scheduler is more efficient when dealing

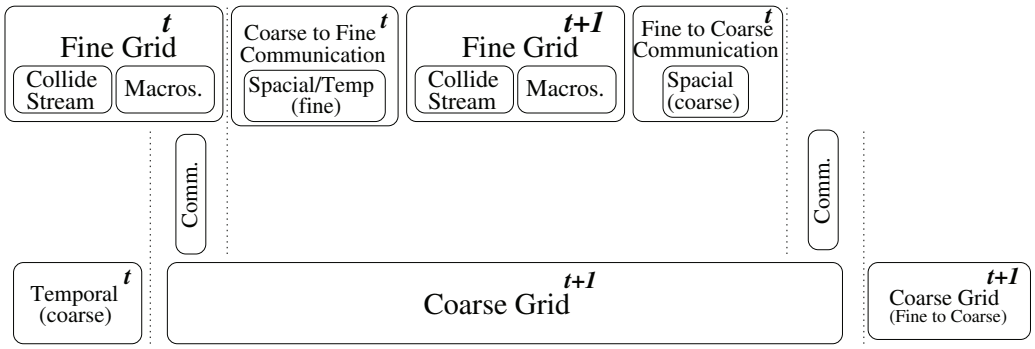


Figure 2. Steps of the multi-domain LBM code for our heterogeneous (Multicore-GPU) approach.

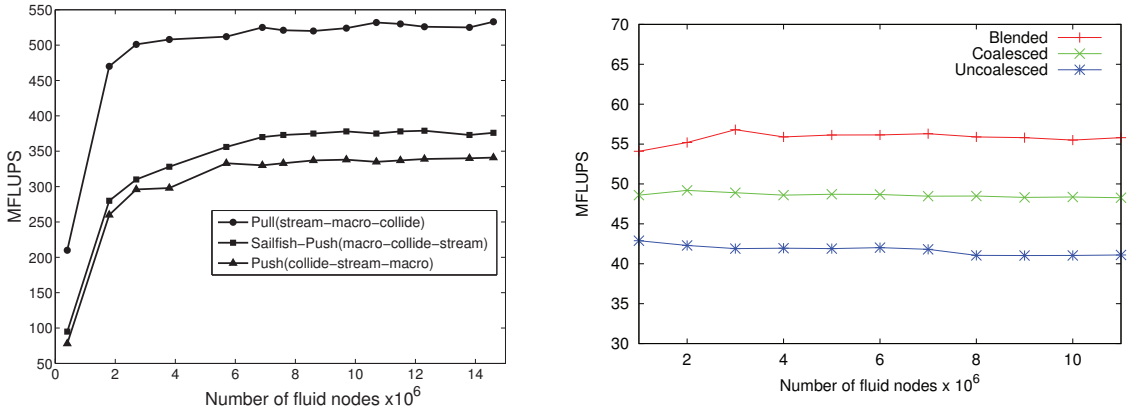


Figure 3. Performance, in terms of MFLUPS, for the two LBM schedulers over GPU (left) and for the three memory mappings implemented over Multicore (right).

with Multicore. Unlike GPU, the gain gap by using the pull-LBM scheme over Multicore is not so high with respect to the use of the push-LBM. The MFLUPS reached by each memory-storage strategy over our Multicore processor is graphically illustrated in Figure 3-right. The uncoalesced approach turns to be the worst strategy, whereas the blended approach is the most efficient strategy followed by the coalesced. Our parallel implementations achieved a speedup around $4\times$ and $6\times$ by implementing the coalesced and blended approaches with respect to the sequential counterpart respectively. As consequence, in our heterogeneous implementation, the memory access pattern for those steps computed on Multicore follows the *Blended* mapping, while steps computed on GPU and regions of the fluid domain to be communicated follow the *Coalesced* mapping.

Taking into account the high number of case-study with particular requirements in terms of size of the fine domains over the size of coarse domain/s, we have carried out several synthetic cases, which are composed by 4 different ratios, which simulates several real scenarios of academia and industrial interest [14], [7], [21]; $fine_{size}/coarse_{size}$ for one refined-level ($0.25\times$, $0.5\times$, $1\times$, $2\times$), and $(fine^{2^{nd}}_{size}/fine^{1^{st}}_{size}, fine^{1^{st}}_{size}/coarse_{size})$ for two refined-levels (($0.1\times$,$0.25\times$),($0.25\times$,$0.5\times$),($0.5\times$,$1\times$), and ($1\times$,$2\times$)). A ratio equals $0.25\times$ means that the coarse domain is $4$ times bigger than fine domain and a ratio equals $2\times$ means that the fine grid is $2$ times bigger than the coarse domain. Three implementations are studied, one homogeneous GPU and two heterogeneous Multicore-GPU, *Top-GPU* and *Top-Multicore*, previously introduced. Figure 4-left graphically illustrates the performance achieved in terms of MFLUPS. A larger fine domain exhibits a much lower performance, as fine grids are computed twice (first refined level) or four times (second refined level) per time step. Better results are reached for smaller fine domains.

The *Top-Multicore* approach reaches a good perfor-

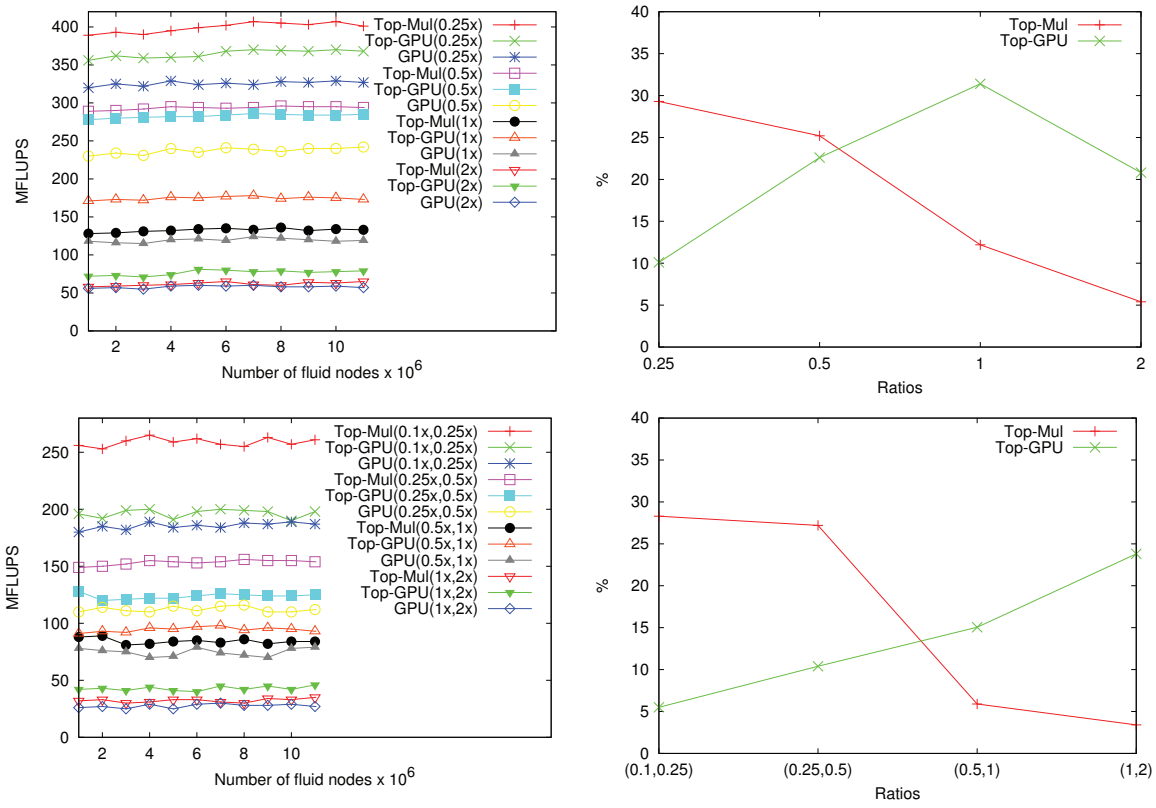


Figure 4. One refined level (top) and Two refined levels (bottom). Performance (left), in terms of MFLUPS, achieved by the three approaches, homogeneous GPU, heterogeneous *Top-Multocore* and heterogeneous *Top-GPU*. Gain (right), in terms of percentage, of each of the both heterogeneous implementations, *Top-Multicore* and *Top-GPU*, against the homogeneous GPU implementation.

mance for small fine domains, achieving a good balancing for ratios equal to $0.25\times$ and $(0.1\times, 0.25\times)$. However, worse gains are achieved in the rest of experiments. A different trend arises for the *Top-GPU* approach. This load distribution is very beneficial for greather fine grids. For balanced domains ($1\times$ and $(1\times,2\times)$), the heterogeneous *Top-GPU* implementation is approximately $30\%$ and $15\%$ faster over the homogeneous GPU counterpart for one refined level and two refined levels respectively. The difference among the regions of both grids has an important impact in performance, which degrades or increases the whole performance, so that both heterogeneous implementations, Top-Multicore and Top-GPU, present a better performance with respect to the other depending on this ratio. Figure 4-right graphically illustrates the gain of both heterogeneous approaches over the homogeneous GPU, for each ratio. The dealing of multiple domains over GPU degrades the benefit of using the *Top-GPU* approach. This reduces the benefit of using our heterogeneous approach, at least in those case-study with greater fine grids. Despite the overheads aforementioned, our heterogeneous implementation is able to outperform the homogeneous counterpart in all cases evaluated.

## V. Conclusions

In this paper, we have investigated the performance of a mesh refinement algorithm for lattice-Boltzmann solvers that simulates regular Cartesian grids with multiples space domains. While the lattice-Boltzmann method has been widely studied on heterogeneous platforms, the parallelization of mesh refinement algorithms based on this method is an emerging topic.

We have implemented and analyzed two different heterogeneous approaches that take advantages of both architectures, Multicore and GPU, in a cooperative way. Our heterogeneous approach outperforms the performance achieved by the homogeneous GPU counterpart. We consider, as the main contribution of the present work, the study of the influence in performance of the ratio among the sizes of the set of domains. This factor is the key to choice what is

the best heterogeneous distribution for multi-domain LBM problems.

## ACKNOWLEDGMENT

This research has been supported by EU-FET grant EUNISON 308874, 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 and the Project of the Spanish Ministry of Economy and Competitiveness with reference MTM2013-40824. We also thank NVIDIA GPU Research Center program for the provided resources.

## REFERENCES

[1] J. Tölke, Implementation of a Lattice Boltzmann kernel using the compute unified device architecture developed by nVIDIA. Comput. Visual. Sci. 13(1):29-39, 2010.

[2] P. Valero-Lara. A Fast Multi-Domain Lattice-Boltzmann Solver on Heterogeneous Architectures, Proceedings of the 14th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE), 2014.

[3] P. Valero-Lara, A. Pinelli, M. Prieto-Matías. Accelerating Solid-fluid Interaction using Lattice-boltzmann and Immersed Boundary Coupled Simulations on Heterogeneous Platforms. The International Conference on Computational Science (ICCS), 50-61, 2014.

[4] M. Bernaschi, M. Fatica, S. Melchiona, S. Succi, 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.

[5] P. R. Rinaldi, E. A. Dari, M. J. Vénere, A. Clausse. A Lattice-Boltzmann solver for 3D fluid simulation on GPU. Simulation Modelling Practice and Theory, 25, 163-171, 2012.

[6] M. Januszewski and M. Kostur. Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method. Computer Physics Communications, 185(9):23502368, 2014.

[7] P. Valero-Lara, J. Jansson. A Non-uniform Staggered Cartesian Grid Approach for Lattice-boltzmann Method. The International Conference on Computational Science (ICCS), 296-305, 2015.

[8] LBM-HPC. Computational Fluid Dynamics & Computational Technology Unit at BCAM. http://www.bcamath.org/en/research/lines/CFDCT/software, 2015.

[9] S. Succi. The lattice Boltzmann equation: for fluid dynamics and beyond. Oxford university press New York, 2001.

[10] P. Bhatnagar, E. Gross and M. Krook. A model for collision processes in gases. I: small amplitude processes in charged and neutral one-component system. Physical Review, 94, 511-525, 1954.

[11] P. Valero-Lara, A. Pinelli, J. Favier, M. Prieto-Matías. Block Tridiagonal Solvers on Heterogeneous Architectures. The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 609-616, 2012.

[12] P. Valero-Lara, A. Pinelli, M. Prieto-Matías. Fast finite difference Poisson solvers on heterogeneous architectures. Computer Physics Communications, 185(4), 1265-1272, 2014.

[13] Y. Qian, D. D'Humieres and P. Lallemand. Lattice BGK Models for Navier-Stokes Equation. Europhysics Letters, 17 (6), 479-484, 1992.

[14] D. Lagrava, O. Malaspinas, J. Latt and B. Chopard. Advances in multi-domain lattice Boltzmann grid refinement. Journal of Computational Physics, 231, 4808-4822, 2012.

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

[16] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, E. Kaxiras. A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. Concurrency Computat.: Pract. Exper. 2010; 22: 1-14.

[17] G. Crimi, F. Mantovani, M. Pivanti, S.F. Schifano, R. Tripiccione. Early Experience on Porting and Running a Lattice Boltzmann Code on the Xeon-phi Co- Processor. Procedia Computer Science. 18, 551-560, 2013.

[18] Nvidia Corp. Dynamic Parallelism in CUDA - Nvidia technical report. http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/TechBrief_Dynamic_Parallelism_in_CUDA.pdf, 2013.

[19] A. Fakhari, T. Lee. Finite-difference lattice Boltzmann method with a block-structured adaptive-mesh-refinement technique. Phys. Rev. E. 89 (3), 12, 2014.

[20] P. Valero-Lara, F. D. Igual, M. Prieto-Matas, A. Pinelli, J. Favier. Accelerating fluid-solid simulations (lattice-boltzmann & immersed-boundary) on heterogeneous architectures. Journal of Computational Science, 10 (2015), 249-261.

[21] M. A. Mussa, S. Abdullah, C. S. Azwadi, N. Muhamad and K. Sopian. Numerical Simulation of Lid-driven Cavity Flow Using the Lattice Boltzmann Method. Proceedings of the 13th WSEAS International Conference on Applied Mathematics. 236-240, 2008.