Efficient GPU Implementation of the Linearly Interpolated Bounce-Back Boundary Condition

C. Obrecht, F. Kuznik, Bernard Tourancheau, J.-J. Roux

Efficient GPU Implementation of the Linearly Interpolated Bounce-Back Boundary Condition

Christian Obrecht[a,b,c,*], Frédéric Kuznik[b,c], Bernard Tourancheau[d], Jean-Jacques Roux[b,c]

[a] *EDF R&D, Département EnerBAT, 77818 Moret-sur-Loing Cedex, France*
[b] *Université de Lyon, 69361 Lyon Cedex 07, France*
[c] *INSA-Lyon, CETHIL UMR5008, 69621 Villeurbanne Cedex, France*
[d] *UJF-Grenoble, INRIA, LIG UMR5217, 38041 Grenoble Cedex 9, France*

Abstract

Interpolated bounce-back boundary conditions for the lattice Boltzmann method (LBM) make the accurate representation of complex geometries possible. In the present work, we describe an implementation of a linearly interpolated bounce-back (LIBB) boundary condition for graphics processing units (GPUs). To validate our code, we simulated the flow past a sphere in a square channel. At low Reynolds numbers, results are in good agreement with experimental data. Moreover, we give an estimate of the critical Reynolds number for transition from steady to periodic flow. Performance recorded on a single node server with eight GPU based computing devices ranged up to 2.63×10^9 node updates per second. Comparison with a simple bounce-back version of the solver shows that the impact of LIBB on performance is fairly low.

Key words: Lattice Boltzmann method, GPU programming, CUDA, Interpolated bounce-back boundary condition, TheLMA project

1. Introduction

From a computational perspective, the lattice Boltzmann method (LBM) can be seen as a data parallel algorithm with local synchronisation constraints. It is therefore well-adapted to massively parallel architectures such as graphics processing units (GPUs). Since the advent of the CUDA technology in 2007 [8], several efficient implementations of the LBM for the GPU were reported [7, 18]. Recent multi-GPU implementations [11] make the use of large computation domains possible, which otherwise would be bound by the limited amount of onboard emory. Nevertheless, several other issues, such as accurate representation of complex geometries, remain to be addressed in order to improve the practical interest of GPU LBM solvers. Implementing LBM boundary conditions for the

*christian.obrecht@insa-lyon.fr

GPU is quite challenging since it often leads to branch divergences and therefore may have dramatic impact on performance.

In this contribution, we ~~shall~~ describe the multi-GPU implementation of an extension to the simple bounce-back boundary condition. This approach introduced in 2001 by Bouzidi [2], uses interpolations to take the exact location of the solid boundaries ~~into account~~. For validation purpose, we simulated the flow past a sphere in a square channel and compared our results with experimental data. The paper is organised as follows. First, we briefly introduce the LBM and present the boundary condition we implemented. Then, we outline the TheLMA framework, on which our solver is based, and describe the proposed implementation. Next, we report and discuss our simulation results and last, we present some performance measurements.

2. Lattice Boltzmann method

With the continuous Boltzmann equation, fluid dynamics is represented through the evolution in time of a single-particle distribution function f in phase space. As shown by He and Luo [5], lattice Boltzmann models are based on discretised versions of the Boltzmann equation in both time and phase space. In general, the LBM uses a regular orthogonal lattice of mesh size δx and constant time steps δt. The velocity space is replaced by a finite set of $N+1$ particle velocities $\{\boldsymbol{\xi}_\alpha \,|\, \alpha = 0, \ldots, N\}$. The lattice Boltzmann analogue of the distribution function f is a set of functions $\{f_\alpha \,|\, \alpha = 0, \ldots, N\}$ associated to the particle velocities. Using the former notations, the lattice Boltzmann equation (LBE), i.e. the governing equation of the LBM, is written:

$$\big|f_\alpha(\boldsymbol{x} + \boldsymbol{c}_\alpha, t + \delta t)\big\rangle - \big|f_\alpha(\boldsymbol{x}, t)\big\rangle = \Omega\Big[\big|f_\alpha(\boldsymbol{x}, t)\big\rangle\Big]. \tag{1}$$

where Ω is the collision operator. The mass density ρ and the momentum $\boldsymbol{j}$ of the fluid are given by:

$$\rho = \sum_\alpha f_\alpha \,, \qquad \boldsymbol{j} = \sum_\alpha f_\alpha \boldsymbol{\xi}_\alpha \,. \tag{2}$$

From an algorithmic perspective, Eq. 1 naturally breaks in two elementary step:

$$\big|\widetilde{f}_\alpha(\boldsymbol{x}, t)\big\rangle = \big|f_\alpha(\boldsymbol{x}, t)\big\rangle + \Omega\Big[\big|f_\alpha(\boldsymbol{x}, t)\big\rangle\Big] \tag{3}$$

$$\big|f_\alpha(\boldsymbol{x} + \boldsymbol{c}_\alpha, t + \delta t)\big\rangle = \big|\widetilde{f}_\alpha(\boldsymbol{x}, t)\big\rangle \tag{4}$$

where $\boldsymbol{c}_\alpha = \delta t\, \boldsymbol{\xi}_\alpha$. Equation 3 describes the *collision* step in which an updated particle distribution is computed. Equation 4 describes the *propagation* step in which the updated particle populations are transferred to the neighbouring nodes. The particle velocity set is usually chosen such as to link the nodes to some of their nearest neighbours, as the three-dimensional D3Q19 stencil illustrated by Fig. 1.

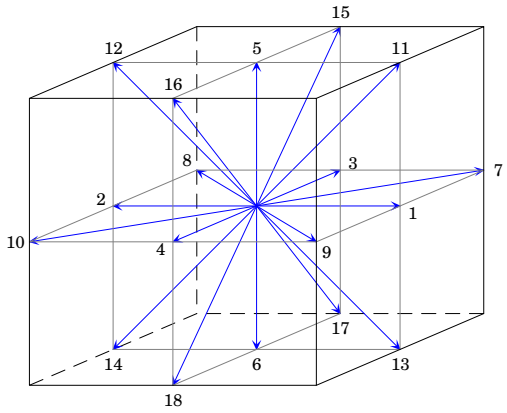

Figure 1: The D3Q19 stencil

For the present work, we used the D3Q19 multiple-relaxation-time (MRT) lattice Boltzmann model described in [3]. In the MRT approach, collision is performed in moment space. The particle distribution is mapped to a set of moments $\{m_\alpha \,|\, \alpha = 0, \ldots, N\}$ by an orthogonal matrix M:

$$\big|m_\alpha(\boldsymbol{x}, t)\big\rangle = \mathsf{M}\,\big|f_\alpha(\boldsymbol{x}, t)\big\rangle \tag{5}$$

where $\big|m_\alpha(\boldsymbol{x}, t)\big\rangle$ is the moment vector. The LBE becomes:

$$\big|f_\alpha(\boldsymbol{x} + \boldsymbol{c}_\alpha, t + \delta t)\big\rangle - \big|f_\alpha(\boldsymbol{x}, t)\big\rangle = -\mathsf{M}^{-1}\Lambda\Big[\big|m_\alpha(\boldsymbol{x}, t)\big\rangle - \big|m_\alpha^{(\mathrm{eq})}(\boldsymbol{x}, t)\big\rangle\Big] \tag{6}$$

where Λ is a diagonal collision matrix and the $m_\alpha^{(\mathrm{eq})}$ are the equilibrium values of the moments.

3. Bounce-back boundary conditions

Lattice Boltzmann boundary conditions for solid walls basically divide up into wet node conditions and bounce-back conditions. In the former category, the boundary nodes, i.e. the nodes on which the condition is applied, are supposed to be both located on the solid boundary and part of the fluid [6]. In the later category, the boundary nodes are in general the fluid nodes next to the solid nodes and the solid boundary is located somewhere in between.

An elementary version of bounce-back is the so-called simple bounce-back (SBB). With SBB, an unknown particle population f_α at a boundary node obeys the following equation:

$$f_\alpha(\boldsymbol{x}, t) = \widetilde{f}_{\bar{\alpha}}(\boldsymbol{x}, t - \delta t) \tag{7}$$

where $\bar{\alpha}$ is the direction opposite to α. Algorithmic simplicity of SSB is obvious when considering Eq. 7. The only information required for a given node is the list of unknown particle populations. Moreover, it is known that (asymptotically) the solid boundary is located half-way between the solid and the fluid nodes [4]. Simple bounce-back is therefore convenient in many situations. However, to handle complex geometries, a more elaborate approach is needed.

In 2001, Bouzidi *et al.* [2] introduced an extension to SBB based on either linear interpolation (LIBB) or quadratic interpolation, which allows the solid boundary to take any desired position. In the present work, we implemented the LIBB as formulated by Pan *et al.* [15]. Let $\boldsymbol{x}$ denote a boundary node such that $\boldsymbol{x} + \boldsymbol{c}_\alpha$ is a solid node, and q be the number such that $\boldsymbol{x} + q\boldsymbol{c}_\alpha$ is on the solid boundary. For $q < 1/2$,

$$f_\alpha(\boldsymbol{x}, t) = (1 - 2q) f_{\bar{\alpha}}(\boldsymbol{x}, t) + 2q \widetilde{f}_{\bar{\alpha}}(\boldsymbol{x}, t - \delta t) \tag{8}$$

and for $q \geq 1/2$,

$$f_\alpha(\boldsymbol{x}, t) = \left(1 - \frac{1}{2q}\right) \widetilde{f}_\alpha(\boldsymbol{x}, t - \delta t) + \frac{1}{2q} \widetilde{f}_{\bar{\alpha}}(\boldsymbol{x}, t - \delta t) \tag{9}$$

It should be noted that both equations only require informations local to the boundary node. Moreover, it is worth mentioning that for $q = 1/2$, LIBB reduces to SBB.

4. The TheLMA framework

The proposed implementation of the LIBB boundary condition was carried out within the TheLMA framework [1]. The design of graphics processing units is guided by their primary use which is rather different from general purpose computations. As a matter of fact, several limitations of the CUDA technology, like the inlining of device functions[1], are induced by hardware characteristics of the Nvidia GPUs. The former limitation, which forbids the linking of CUDA object files, makes library oriented development not relevant in many situations, and more specifically for LBM solvers. We therefore decided to create a framework, in order to improve code reusability.

TheLMA stands for *Thermal LBM on Many-core Architectures*, thermal simulations being our main topic of interest. The framework consists in a set of modules which are designed such as to minimise code modifications when setting up a new simulations or implementing a new model. It provides native single-node multi-GPU support based on POSIX threads. The core collision and propagation kernel is derived from the single-GPU code described in [10]. The execution grid of the core kernel is two-dimensional with one-dimensional

[1] With the latest CUDA enabled GPU generation, i.e. the Fermi generation, inlining is the default behaviour but is not mandatory any more. However, the CUDA 4.0 compilation tool-chain is still unable to link several CUDA object files.

blocks, each node of the lattice being associated to a thread. In order to ensure global synchronisation, two instances of the particle distribution are kept in global memory, corresponding to even and odd time steps.

For each computation sub-domain, the particle distribution is stored in a four-dimensional array. The fastest varying dimension corresponds to the direction of the blocks which allows memory transactions to be coalesced. The second fasted varying dimension corresponds to the velocity set index. When using the D3Q19 stencil, the size of the second dimension is therefore 19 which is prime. The number of device memory banks being a power of two, this layout has a positive impact on partition camping effects. Instead of using the usual out-of-place propagation, our core kernel performs in-place propagation which consists in carrying out propagation before collision instead of after. This propagation scheme is illustrated by Fig. 2. The represented case is only two-dimensional for the sake of clarity. It was shown in [9] that this simple optimisation minimises the cost of misaligned memory transactions, which may have dramatic effects on performance with pre-Fermi hardware.

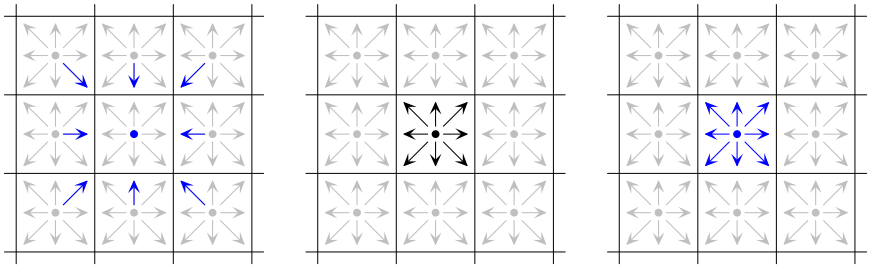

Figure 2: Out-of-place propagation

When running several GPUs in parallel, inter-GPU communication is performed by zero-copy transactions on pinned buffers in CPU memory. This approach leads to excellent overlapping of communication and computations [12]. Figure 3 outlines the communication scheme. For the sake of clarity, only one GPU with a single sub-domain interface is displayed. In the figure, G denotes the GPU, L0 and L1 denote the particle distribution arrays for even and odd time steps, R0 and R1 in-coming data, S0 and S1 out-going data.

To achieve satisfactory performance, data exchanges at the interfaces must be coalescent [11]. The computation domain is therefore split in balanced sub-domains along the direction corresponding to the slowest varying dimension of the particle distribution array.

5. Proposed implementation

In the TheLMA framework, geometry is represented using bit-fields. To process a node, the corresponding thread first loads a 32-bit integer. The first

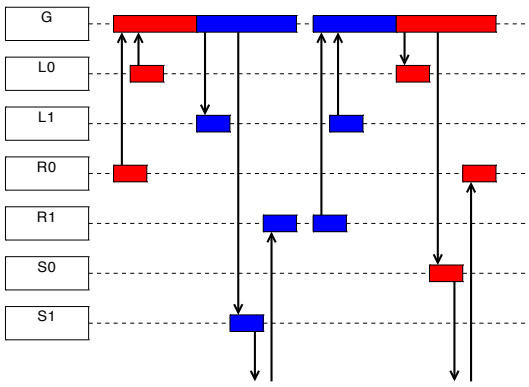

Figure 3: Inter-GPU communication scheme

N bits of the integer are used to indicate whether the node in the corresponding direction is solid. This technique makes the implementation of SBB rather straightforward. Since we use in-place propagation, some of the particle populations loaded for a boundary node are invalid, but these values are discarded when applying the boundary condition. Our tests have shown that it is of little interest to avoid loading these invalid populations. As a matter of fact, it may have a positive impact to cancel invalid loads when a whole half-warp is involved, e.g. for cavity walls parallel to the blocks or within very large obstacles. Yet, the overhead of branching decisions together with surface to volume effects make the benefits negligible in practice.

Our implementation of the LIBB takes advantage of these unnecessary memory accesses. At initialisation, the distance information for the solid boundaries are computed and stored in the unused particle population array cells of the relevant solid nodes. At each time step, the distance information are retrieved by the threads processing boundary nodes during propagation. To perform interpolation, the threads need in addition to fetch some of the local updated particle populations of the former time step. The data access scheme is outlined by Fig. 4. Blue is used for the particle populations involved in collision, red for the distance information, and black for the particle populations involved in interpolation. Again, the displayed case is two-dimensional for the sake of clarity.

It is worth stressing that, in practice, the proposed implementation of LIBB only slightly increases the overall number of memory accesses compared to our implementation of SBB. The implemented initialisation and simulation kernel are summarised in Pseudo-Codes 1 and 2.

6. Flow past a sphere

For validation purposes, we performed single precision simulations of a uniform flow past a sphere in a square channel. Figure 5 outlines the computation

```
1.  if node x is solid then
2.    set flag solid for x
3.    for each direction α do
4.      if node x + c_α is fluid then
5.        compute q for x and x + c_α
6.        store q in f_ᾱ(x + c_α, 0)
7.      end if
8.    end for
9.  else
10.   for each direction α do
11.     if node x + c_α is solid then
12.       set flag α for x
13.     end if
14.   end for
15. end if
```

Pseudo-Code 1: Initialisation kernel

```
1.  read bit-field for x
2.  if node x is fluid then
3.    for each direction α do
4.      read f̃_α(x − c_α, t − δt)
5.    end for
6.    for each direction α do
7.      if flag α is set then
8.        set q to f_ᾱ(x, t)
9.        read f̃_α(x, t − δt) and f̃_ᾱ(x, t − δt)
10.       interpolate f_ᾱ(x, t)
11.     end if
12.   end for
13.   compute distribution f̃_α(x, t)
14.   store distribution f̃_α(x, t)
15. end if
```

Pseudo-Code 2: Simulation kernel

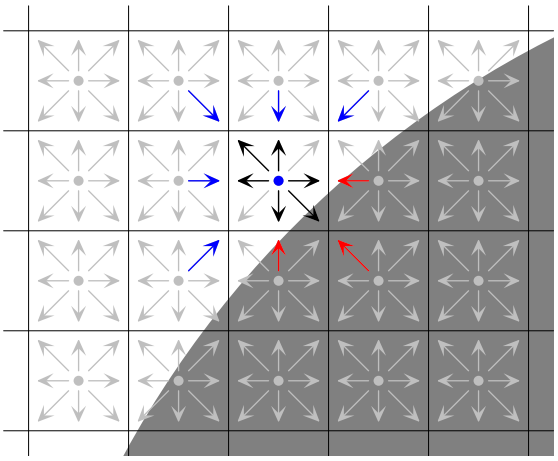

Figure 4: Implementation of the LIBB

domain. For the sake of simplicity, we set $\delta x = 1$ and $\delta t = 1$. The overall dimensions we chose for the channel are $L \times \ell \times \ell = 768 \times 352 \times 352$ and the radius of the sphere is $r = 21$. The blockage ratio $\beta = 2r/\ell$ is therefore $\beta \approx 0.112$. The center of the sphere is positioned at a distance $d = 224$ from the inlet, with $y' = 171$, $y'' = 181$, $z' = z'' = 176$. The distance from the back of the sphere to the outlet is therefore greater than twelve times the diameter. The slight asymmetry in the y direction contributes to stabilise the flow pattern.

In order to study the vortex shedding frequency, we recorded the flow velocity components at points A, B, and C such that $OA = 171$ and $AB = BC = r$. We performed frequency analysis using fast Fourier transform on a 2^{19} sample, the overall number of time steps being at least 10^6. The size of the sample is greater than one hundred shedding periods considering the typical values of the Strouhal number (St) reported for the Reynolds numbers (Re) we investigated.

7. Simulation results

7.1. Comparison with experimental data

In order to compare our results with experimental data provided by Sakamoto and Haniu [17], and by Ormières and Provansal [13], we computed the Roshko number $\mathrm{Ro} = \mathrm{St} \times \mathrm{Re}$ for Reynolds numbers ranging from $\mathrm{Re} = 280$ to $\mathrm{Re} = 360$. In both cases, the measurements were carried out in wide wind tunnels, whereas for our simulations the blockage ratio is not negligible. We therefore applied a correction to the computed Strouhal number St in order to obtain the corresponding Strouhal number St^* in an unbounded flow. Following Ota *et al.* [14], St^* obeys:

$$\mathrm{St}^* = (1 - \beta\xi_{\mathrm{St}})\mathrm{St} \tag{10}$$

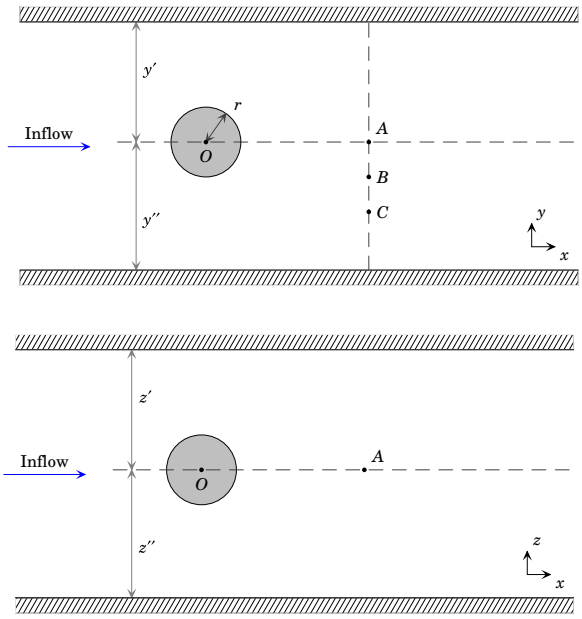

Figure 5: Simulation set-up

where ξ_{St} is the correction factor depending on the shape of the obstacle. We used the value $\xi_{\mathrm{St}} = 0.95$ obtained thanks to the least square method with the fit given by Ormières and Provansal:

$$\mathrm{Ro} = -48.2 + 0.391 \times \mathrm{Re} - 3.6 \times 10^{-4} \times \mathrm{Re}^2 \tag{11}$$

In the considered interval of Reynolds numbers, the obtained power spectra are dominantly unimodal and independent from the chosen tracking point (either A, B, or C). Figure 6 gives a sample for $\mathrm{Re} = 300$ of the power spectral density of the cross-stream component v of the velocity in the y direction.

As shown, by Fig. 7, agreement of the corrected values with experimental data is satisfactory, the deviation from the fit being within 5%.

7.2. Transition to periodic flow

In their aforementioned work [13], Ormières and Provansal investigate the transition from steadiness to periodicity for the flow past a sphere. The reported value for the critical Reynolds number is $\mathrm{Re}_c = 280 \pm 5$. Our simulations lead us to lower this estimate to $\mathrm{Re}_c = 265 \pm 5$. To determine this critical value, we computed σ_v, the standard deviation of v for the last 5×10^5 time steps. For $\mathrm{Re} = 270$, we obtain $\sigma_v/U_0 \approx 3 \times 10^{-6}$ at points A and B, where U_0 denotes the bulk velocity at the inflow. At point C, $\sigma_v = 0$ to the extent of machine precision. The corresponding Strouhal number is $\mathrm{St} = 0.1122$. For $\mathrm{Re} = 260$, σ_v equals to zero at each tracking point (again, to the extent of machine precision).

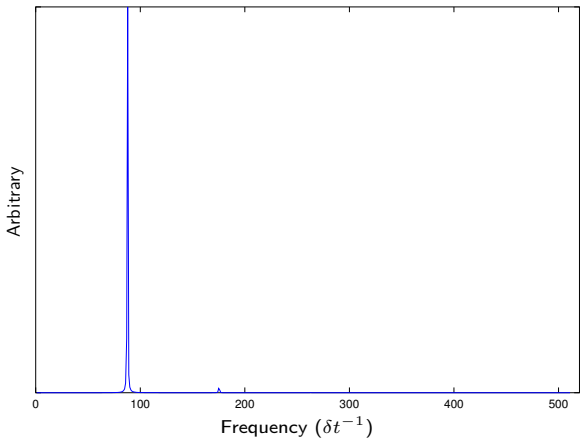

Figure 6: Power spectral density for Re = 300

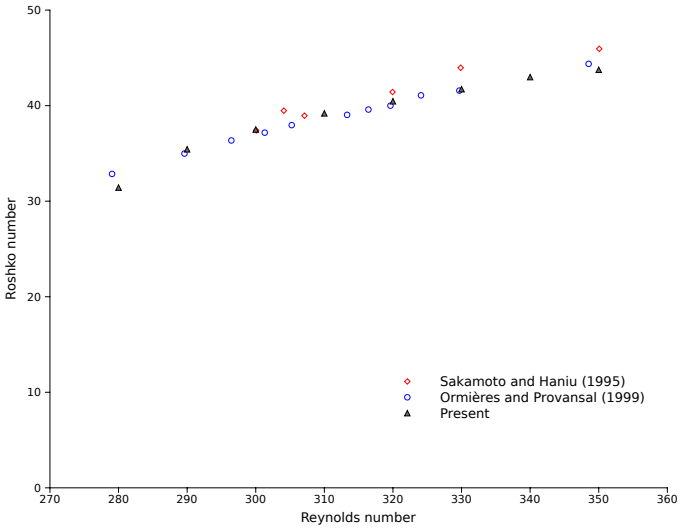

Figure 7: Variation of the Roshko number with the Reynolds number

7.3. Simulations at higher Reynolds numbers

In further investigations, we performed simulations for Reynolds numbers ranging from 400 to 1,000 with a stride of 100. With 10^6 time steps, we could not determine coherent frequencies from our samples. The obtained values generally depend from the chosen tracking point. Moreover, the frequencies appear to be unstable when sliding the sampling window. Figure 8 illustrates the former observation for Re = 1,000. The diagram displays the estimated Strouhal number with respect to the lower bound of the window. One possible

reason for this situation could be an insufficient duration of the initial run. Yet, considering the required computational effort, we decided to focus on the upper part of the regular mode region of the flow pattern [16] located between Re = 360 and Re = 420. For each chosen Reynolds number, we computed 2×10^6 time steps and checked for invariance of the obtained frequencies.

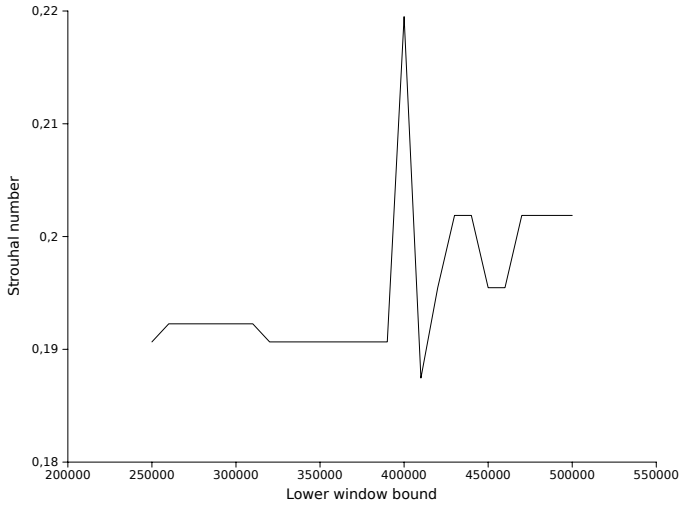

Figure 8: Variation of the estimated Strouhal number at Re = 1,000

For Re = 360, we observe a stable value of St = 0.1394 at points A and B, whereas the Strouhal number obtained at point C oscillates between the former value and St = 0.1826. For Re = 380, we observe a stable value of St = 0.139 at point A and a stable value of St = 0.1810 at point C. The Strouhal number obtained at point B oscillates between the two former values. This observation, which suggests that the flow pattern is split in two independent regions, requires further investigation. For Re = 400, we obtain the same stable value St = 0.2019 for each tracking point. Last, for Re = 420, we could not observe a stable shedding frequency, regardless of the tracking point.

8. Performance results

We carried out our computations on a Tyan B7015 server with eight Tesla C1060 computing devices. For the purpose of evaluating the efficiency of our LIBB implementation, we ran single precision simulations of the flow past a sphere with both a SBB version and a LIBB version of our multi-GPU solver on a computation domain of size $1{,}024 \times \ell \times \ell$ with increasing ℓ. The diameter of the sphere was set to 30. Figure 9 displays the obtained performance in million lattice node updates per second (MLUPS), which is the usual metrics for LBM. As shown by the diagram, the impact of LIBB is in general negligible, with at

most 11% performance loss. The maximum recorded performance for LIBB is 2,630 MLUPS.

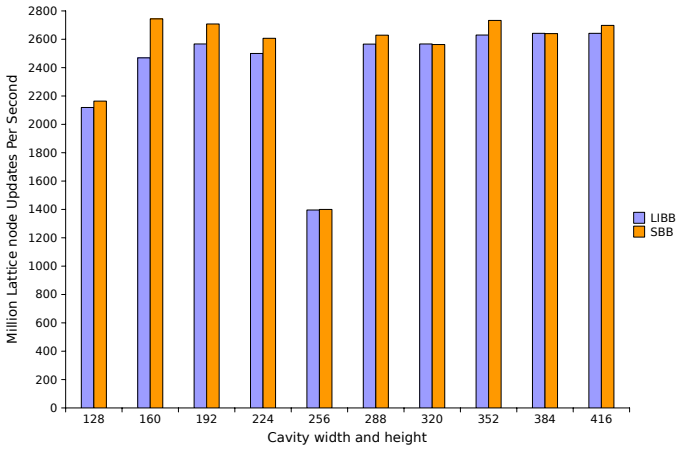

Figure 9: Performance comparison of SBB and LIBB implementations

With the Tesla C1060, the maximum sustained throughput for communication between GPU and device memory is 73.3 GB/s. Except for $\ell = 128$ and $\ell = 256$, the data throughput with both versions is thus above 61% of the maximum. For LIBB, we computed the number of additional memory transactions required using a simple program derived from the solver code. A sphere of diameter 30 yields 3,960 boundary nodes and 32,736 additional memory accesses. Taking the inter-GPU communication overhead into account, the achieved performance is therefore rather satisfactory. The quite low performance obtained with $\ell = 128$ and $\ell = 256$ is most likely due to partition camping effects, considering the dimensions of both computation domains.

9. Conclusion

In the present work, we describe an implementation of the LIBB boundary condition within a multi-GPU LBM solver based on the TheLMA framework. The proposed approach proves to be efficient, with little impact on performance compared to a SBB version of the code. When simulating the flow past a sphere in a channel, our solver allowed to successfully compute the vortex shedding frequency for Reynolds numbers belonging to the regular mode flow pattern region. Moreover, we obtained a plausible value of the critical Reynolds number for transition from steady to periodic flow.

The instabilities observed at higher Reynolds numbers may be caused by either an insufficient simulation duration or by inaccuracies due to the boundary condition. The second hypothesis, which may lead to the conclusion that LIBB is not appropriate for curved solid boundaries past a certain Reynolds number,

will be tested by performing simulations at higher spatial resolution. In both cases, higher computational efforts are required. In order to investigate further, we are at present working on an extension of the TheLMA framework to multi-node multi-GPU hardware.

References

[1] Thermal LBM on Many-core Architectures. www.thelma-project.info.

[2] M. Bouzidi, M. Firdaouss, and P. Lallemand. Momentum transfer of a boltzmann-lattice fluid with boundaries. *Physics of Fluids*, 13:3452, 2001.

[3] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.S. Luo. Multiple-relaxation-time lattice Boltzmann models in three dimensions. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, pages 437–451, 2002.

[4] I. Ginzbourg and P. M. Adler. Boundary flow condition analysis for the three-dimensional lattice boltzmann model. *Journal de Physique II*, 4(2): 191–214, 1994.

[5] X. He and L.-S. Luo. Theory of the lattice boltzmann method: From the boltzmann equation to the lattice boltzmann equation. *Physical Review E*, 56(6):6811, 1997.

[6] T. Inamuro, M. Yoshina, and F. Ogino. A non-slip boundary condition for lattice Boltzmann simulations. *Physic of Fluids*, 7:2928–2930, 1995.

[7] F. Kuznik, C. Obrecht, G. Rusaouën, and J.-J. Roux. LBM Based Flow Simulation Using GPU Computing Processor. *Computers and Mathematics with Applications*, 59(7):2380–2392, April 2010.

[8] *Compute Unified Device Architecture Programming Guide version 4.0*. nVidia, June 2011.

[9] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. Global Memory Access Modelling for Efficient Implementation of the Lattice Boltzmann Method on Graphics Processing Units. In *High Performance Computing for Computational Science – VECPAR2010, Lecture Notes in Computer Science 6449*, pages 151–161. Springer, February 2011.

[10] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. A New Approach to the Lattice Boltzmann Method for Graphics Processing Units. *Computers and Mathematics with Applications*, 12(61):3628–3638, June 2011.

[11] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. The TheLMA project: Multi-GPU Implementation of the Lattice Boltzmann Method. *International Journal of High Performance Computing Applications*, 25(3): 295–303, August 2011.

[12] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. Multi-GPU Implementation of the Lattice Boltzmann Method. *Computers and Mathematics with Applications*, published online March 17, 2011.

[13] D. Ormières and M. Provansal. Transition to turbulence in the wake of a sphere. *Physical review letters*, 83(1):80–83, 1999.

[14] T. Ota, Y. Okamoto, and H. Yoshikawa. A correction formula for wall effects on unsteady forces of two-dimensional bluff bodies. *Journal of fluids engineering*, 116(3):414–418, 1994.

[15] C. Pan, L.-S. Luo, and C. T. Miller. An evaluation of lattice boltzmann schemes for porous medium flow simulation. *Computers & fluids*, 35(8-9): 898–909, 2006.

[16] H. Sakamoto and H. Haniu. A study on vortex shedding from spheres in a uniform flow. *ASME, Transactions, Journal of Fluids Engineering*, 112: 386–392, 1990.

[17] H. Sakamoto and H. Haniu. The formation mechanism and shedding frequency of vortices from a sphere in uniform shear flow. *Journal of Fluid Mechanics*, 287:151–172, 1995.

[18] J. Tölke and M. Krafczyk. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *International Journal of Computational Fluid Dynamics*, 22(7):443–456, 2008.