Computers and Mathematics with Applications 72 (2016) 375–385

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Thermal link-wise artificial compressibility method: GPU implementation and validation of a double-population model

Christian Obrecht [a,*], Pietro Asinari [b], Frédéric Kuznik [a], Jean-Jacques Roux [a]

[a] INSA de Lyon, CETHIL UMR 5008, Université de Lyon, France

[b] SMaLL, Dipartimento Energia, Politecnico di Torino, Italy

ARTICLE INFO

Article history:
Available online 9 June 2015

Keywords:
Computational fluid dynamics
Link-wise artificial compressibility method
High-performance computing
Differentially heated cubic cavity
CUDA

ABSTRACT

The link-wise artificial compressibility method (LW-ACM) is a novel formulation of the artificial compressibility method for the incompressible Navier–Stokes equations showing strong analogies with the lattice Boltzmann method (LBM). The LW-ACM operates on regular Cartesian meshes and is therefore well-suited for massively parallel processors such as graphics processing units (GPUs). In this work, we describe the GPU implementation of a three-dimensional thermal flow solver based on a double-population LW-ACM model. Focusing on large scale simulations of the differentially heated cubic cavity, we compare the present method to hybrid approaches based on either multiple-relaxation-time LBM (MRT-LBM) or LW-ACM, where the energy equation is solved through finite differences on a compact stencil. Since thermal LW-ACM requires only the storing of fluid density and velocity in addition to temperature, both double-population thermal LW-ACM and hybrid thermal LW-ACM reduce the memory requirements by a factor of 4.4 compared to a D3Q19 hybrid thermal LBM implementation following a two-grid approach. Using a single graphics card featuring 6 GiB[1] of memory, we were able to perform single-precision computations on meshes containing up to 536^3 nodes, i.e. about 154 million nodes. We show that all three methods are comparable both in terms of accuracy and performance on recent GPUs. For Rayleigh numbers ranging from 10^4 to 10^6, the thermal fluxes as well as the flow features are in similar good agreement with reference values from the literature.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

Today's computational fluid dynamics (CFD) softwares are often designed to operate on unstructured meshes in order to take into account complex physical interfaces. During the last years however, there has been a resurgence of interest in numerical methods based on regular Cartesian grids. Although the representation of complex geometries with such approaches can be challenging, they usually benefit from a good asymptotic behaviour and are well-suited for high-performance implementations. A cardinal example of CFD methods operating on Cartesian grids is the well-acknowledged lattice Boltzmann method (LBM) which is nowadays used in several industry-grade softwares such as PowerFLOW, X-Flow, Fluidyna, and LaBS.

* Corresponding author.
E-mail address: christian.obrecht@insa-lyon.fr (C. Obrecht).

[1] Instead of the widespread but ambiguous GB and KB notations, we use the notations of the International System of Quantities, namely 1 GiB = 2^{30} B, 1 KiB = 2^{10} B, and 1 kB = 10^3 B.

http://dx.doi.org/10.1016/j.camwa.2015.05.022
0898-1221/© 2015 Elsevier Ltd. All rights reserved.

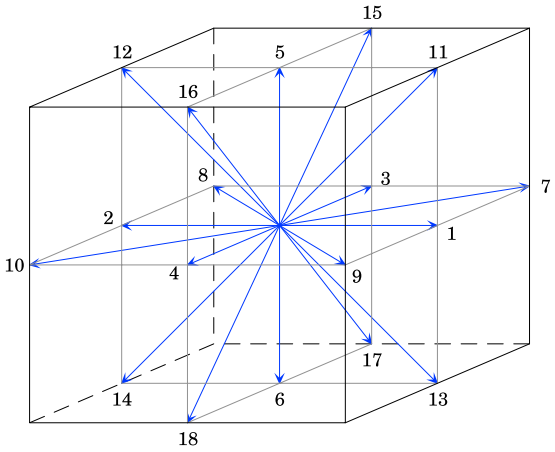


Fig. 1. The D3Q19 stencil—the arrows represent the $\boldsymbol{\xi}_\alpha$ velocities. This stencil links any bulk node to 18 of its nearest neighbours.

The link-wise artificial compressibility method (LW-ACM), introduced in [1], is a novel approach for solving the incompressible Navier–Stokes equations on Cartesian grids. Asymptotic analysis demonstrates that the LW-ACM yields the artificial compressibility equation [2], first proposed by Chorin in 1967. Although it primarily involves hydrodynamic macroscopic variables instead of discrete distribution functions, the LW-ACM exhibits strong formal similarities with LBM, making possible the use of both finite difference or LBM procedures, in particular regarding the treatment of boundary conditions. Recent work [3] has shown that the LW-ACM is well-suited for implementations on graphics processing units (GPUs), leading to a speed-up factor of 1.8 on recent hardware with respect to a state-of-the-art three-dimensional isothermal LBM solver. The large scale simulations of the lid-driven cubic cavity performed with this GPU implementation reveal that the LW-ACM is almost as accurate as multiple-relaxation-time LBM [4]. The chosen reference GPU LBM implementation uses a two-grid data layout in order to avoid read-after-write issues. With respect to this approach, the LW-ACM reduces memory consumption by a factor of $(19 \times 2 + 4)/(4 \times 2) = 5.25$. It should be mentioned that several approaches suited for GPU implementations make possible to use a single grid (see e.g. [5]) and that in this case the memory reduction factor is only $(19 + 4)/(4 \times 2) = 2.875$. These techniques, however, lead to fairly more intricate codes than the straightforward two-grid method and might be challenging to implement in conjunction with complex boundary conditions or couplings.

Regarding thermal simulations, it is well known that energy conserving LBM models suffer from spurious couplings between energy and shear modes of the linearised collision operator [6]. To overcome this flaw, thermal LBM models must separate the resolution of the energy equation from the resolution of the continuity and momentum equations, which is usually achieved by using either a finite difference scheme or a second set of distribution functions. Both of these approaches also apply to LW-ACM. The former has been successfully tested in a recent work [7]. In the present article, we report our experiments following the later approach in order to implement and validate a three-dimensional thermal flow solver on GPUs using the CUDA technology [8].

The remainder of the paper is organised as follows. Section 2 presents the implemented model starting with a summary of isothermal LW-ACM, followed by a description of the chosen double-population thermal LW-ACM model (DPTLA). Section 3 describes the algorithmic aspects of DPTLA as well as the GPU implementation principles. Section 4 presents the methodology we followed in our validation study for both our performance results and our simulation data for differentially heated cubic cavity. Section 5 reports and discusses the results. A throughout comparison of thermal fluxes and characteristic flow features is carried out for the present model and reference data [9], as well as for results obtained using hybrid thermal LBM (HTLBM) [10,11] and hybrid thermal LW-ACM (HTLA) [7]. Section 6 provides some concluding remarks.

2. Model

2.1. Link-wise artificial compressibility method

Similarly to standard isothermal LBM, the isothermal LW-ACM operates on a *lattice*, i.e. a regular Cartesian grid of mesh size δx with a constant time step δt associated to a finite set of Q velocities $\{\boldsymbol{\xi}_\alpha\}$ with $\boldsymbol{\xi}_0 = \mathbf{0}$. In the present work, we assume *diffusive scaling*, i.e. we have $\delta x = \varepsilon$ and $\delta t = \varepsilon^2$ for some small parameter ε. The set of velocities, which we will refer to as the *stencil* of the model, is chosen such as to link neighbouring nodes of the mesh in one time step. For our simulations, we used the three-dimensional D3Q19 stencil displayed in Fig. 1.

The simulated fluid is represented at each node by a set of Q independent variables $\{f_\alpha\}$ which are the formal equivalents of the LBM distribution functions. Let us denote by ρ and $\boldsymbol{u}$ the density and velocity of the fluid, the variables f_α

obey:

$$\rho = \sum_\alpha f_\alpha \quad \text{and} \quad \rho \boldsymbol{u} = \sum_\alpha f_\alpha \boldsymbol{\xi}_\alpha. \tag{1}$$

The fundamental updating rule of isothermal LW-ACM is given by:

$$f_\alpha(\boldsymbol{x},\ t+1) = f_\alpha^{(\mathrm{e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t) + 2\left(\frac{\omega - 1}{\omega}\right)\left(f_\alpha^{(\mathrm{e,o})}(\boldsymbol{x},\ t) - f_\alpha^{(\mathrm{e,o})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)\right), \tag{2}$$

where ω is a relaxation frequency related to the kinematic viscosity ν, $f_\alpha^{(\mathrm{e})}$ are the local equilibrium functions and $f_\alpha^{(\mathrm{e,o})}$ are the odd parts of the equilibrium functions defined as:

$$f_\alpha^{(\mathrm{e,o})}(\rho, \boldsymbol{u}) = \frac{1}{2}\left(f_\alpha^{(\mathrm{e})}(\rho, \boldsymbol{u}) - f_\alpha^{(\mathrm{e})}(\rho, -\boldsymbol{u})\right). \tag{3}$$

The formal similarity of LW-ACM and LBM becomes more explicit when reformulating Eq. (2) in:

$$f_\alpha(\boldsymbol{x},\ t+1) = f_\alpha^{*}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t) + 2\left(\frac{\omega - 1}{\omega}\right) f_\alpha^{(\mathrm{e,o})}(\boldsymbol{x},\ t), \tag{4}$$

$$f_\alpha^{*}(\boldsymbol{x},\ t+1) = f_\alpha^{(\mathrm{e})}(\boldsymbol{x},\ t+1) - 2\left(\frac{\omega - 1}{\omega}\right) f_\alpha^{(\mathrm{e,o})}(\boldsymbol{x},\ t+1), \tag{5}$$

the former relation being the equivalent of the streaming step in its pull version, whereas the latter is the equivalent of the collision step.

It should however be noted that to apply the updating rule of LW-ACM at a given node $\boldsymbol{x}$, it is only necessary to compute the equilibria at $\boldsymbol{x}$ and some of its neighbours, these equilibria being known functions of the macroscopic variables of the fluid. Contrary to LBM where the distribution functions are stored at each time step, the LW-ACM only requires to save the density and the velocity, thus leading to significant memory savings. The artificial speed of sound being set to $c = 1/\sqrt{3}$, the equilibrium functions are given by:

$$f_\alpha^{(\mathrm{e})}(\rho, \boldsymbol{u}) = w_\alpha \rho \left(1 + 3\boldsymbol{\xi}_\alpha \cdot \boldsymbol{u} + \frac{9}{2}\left(\boldsymbol{\xi}_\alpha \cdot \boldsymbol{u}\right)^2 - \frac{3}{2}\boldsymbol{u}^2\right) \tag{6}$$

with the weights w_α depending on the stencil. For the D3Q19 stencil, we have:

$$w_\alpha = \begin{cases} 1/3 & \alpha = 0, \\ 1/18 & \alpha = 1, \ldots, 6, \\ 1/36 & \alpha = 7, \ldots, 18. \end{cases} \tag{7}$$

As shown in Appendix B of [1], under the assumption of diffusive scaling and with appropriate scaling of the moments, asymptotic expansion of Eq. (2) yields the artificial compressibility equation and the momentum equation up to second order in space and first order in time. The kinematic viscosity obeys:

$$\nu = \frac{1}{3}\left(\frac{1}{\omega} - \frac{1}{2}\right). \tag{8}$$

As isothermal LBM, isothermal LW-ACM is therefore an effective numerical procedure to solve the isothermal incompressible Navier–Stokes equations.

2.2. Double-population link-wise artificial compressibility method

Double-population thermal LW-ACM is an extension of isothermal LW-ACM involving a second set of Q dependent variables $\{g_\alpha\}$ such that:

$$T = \sum_\alpha g_\alpha \tag{9}$$

where T is the temperature. Although one might use different stencils for the two variable sets, in our work we have chosen to use the D3Q19 stencil for both, which is more convenient in terms of implementation.

Since the purpose of the second set is to recover the energy equation, its updating rule slightly differs from the one of the first set. It may be written as:

$$g_\alpha(\boldsymbol{x},\ t+1) = g_\alpha^{(\mathrm{e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t) + 2\left(\frac{\omega_t - 1}{\omega_t}\right)\left(g_\alpha^{(\mathrm{e,e})}(\boldsymbol{x},\ t) - g_\alpha^{(\mathrm{e,e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)\right) \tag{10}$$

where ω_t is a relaxation frequency linked to the thermal diffusivity κ, $g_\alpha^{(e)}$ are the local equilibrium functions and $g_\alpha^{(e,e)}$ are the even parts of the equilibrium functions defined as:

$$g_\alpha^{(e,e)}(T,\ \boldsymbol{u}) = \frac{1}{2}\left(g_\alpha^{(e)}(T,\ \boldsymbol{u}) + g_\alpha^{(e)}(T,\ -\boldsymbol{u})\right). \tag{11}$$

The definition of the equilibrium functions for g_α is similar to the one of the equilibrium functions for f_α, namely:

$$g_\alpha^{(e)}(T, \boldsymbol{u}) = w_\alpha T\left(1 + 3\boldsymbol{\xi}_\alpha \cdot \boldsymbol{u} + \frac{9}{2}\left(\boldsymbol{\xi}_\alpha \cdot \boldsymbol{u}\right)^2 - \frac{3}{2}\boldsymbol{u}^2\right). \tag{12}$$

It should be noted that, in principle, because energy equation is linear with regard to $\boldsymbol{u}$, one could remove all terms proportional to $\boldsymbol{u}^2$ in this equilibrium.

With appropriate scaling of the moments, asymptotic expansion of Eq. (10) shows that the temperature field is governed by an advection–diffusion equation (see Appendix B of [1] for details), the thermal diffusivity being:

$$\kappa = \frac{1}{3}\left(\frac{1}{\omega_t} - \frac{1}{2}\right). \tag{13}$$

Under the assumption of weak coupling between fluid-dynamic and energy equations, the double-population thermal LW-ACM is thus an effective approach to perform thermal flow simulations.

3. Implementation

3.1. Algorithmic aspects

As shown in [3], an implementation of isothermal LW-ACM may use either the single-step formulation (Eq. (2)) or the two-step formulation (Eqs. (4) and (5)) of the updating rule. The latter approach is interesting since it makes possible to easily adapt an existing LBM code to LW-ACM. It proves also to be well-suited to LBM specific boundary conditions such as simple bounce-back or interpolated bounce-back. The data access pattern is almost similar to LBM except in Eq. (4) where there is an additional access to $\rho(\boldsymbol{x},\ t)$ and $\boldsymbol{u}(\boldsymbol{x},\ t)$ in order to compute $f_\alpha^{(e,o)}(\boldsymbol{x},\ t)$. In a memory bound context, a two-step LW-ACM implementation might therefore be slightly less efficient than the corresponding LBM solver.

In terms of memory consumption however, the two-step version is equivalent to LBM, whereas the single-step version may lead to a considerable diminution. As already stated, to apply the updating rule expressed in Eq. (2), it is only necessary to store the density and velocity fields at each time step. From a local point of view, in the case of three-dimensional simulations, updating a single node using single-step LW-ACM requires $4Q$ read operations and 4 write operations whereas LBM requires Q read operations and Q write operations (or possibly $Q+4$ write operations when saving the density and velocity fields). In a memory bound context, the main optimisation goal for single-step LW-ACM is therefore the reduction of read redundancy. This might be achieved for CPU implementations by an appropriate tiling, in order to optimise cache reuse, and for GPU implementations by an efficient use of shared memory.

In the case of the double-population thermal LW-ACM, the updating rule of the second population expressed in Eq. (10) may also be split in two steps. As for isothermal LW-ACM, a double-population thermal LBM solver may thus easily be adapted to DPTLA. The memory requirements are however even higher than for isothermal LBM. We therefore chose to follow the single-step approach for both updating rules which only requires the additional storing of the temperature field. The general formulation of our implementation is outlined in Algorithm 1.

The D3Q19 HTLBM GPU solver we chose for our tests uses a two-grid data layout. With respect to this approach, the memory consumption is reduced by a factor of $(20 \times 2 + 4)/(5 \times 2) = 4.4$. A GPU computing device fitted with 6 GiB of memory is able to handle about 37 million nodes using our chosen HTLBM implementation and over 160 million nodes using our DPTLA implementation. Considering a cubic cavity, the largest available size is therefore 320 for HTLBM and 536 for DPTLA. For the sake of completeness, it should be mentioned that with respect to a HTLBM implementation following a one-grid approach, the memory consumption is only reduced by a factor of $(19 + 5)/(5 \times 2) = 2.4$.

In a computational perspective, HTLBM compared to double population thermal LBM, even when using a reduced velocity set such as the D3Q7 stencil, has the advantage of lower memory and data transfer requirements, thus yielding slightly better performance. However, when using HTLBM with simple bounce-back, the finite difference thermal boundary conditions must take into account that the physical interface between fluid and solid is located halfway between two nodes.

One of the interesting features of LW-ACM is its ability to use both finite difference or LBM approaches. For convenience, in our implementations of thermal LW-ACM, we chose finite difference formulations for both fluid-dynamic and thermal boundary conditions. The fluid–solid interface may therefore be on-grid, leading to simpler expressions for the thermal boundary conditions. It should however be noted that with DPLA, it is possible to use LBM boundary conditions for both fluid-dynamics and energy. Such approach appears to be promising since it provides the versatility and consistency of the boundary conditions used in double population thermal LBM without increasing memory consumption.

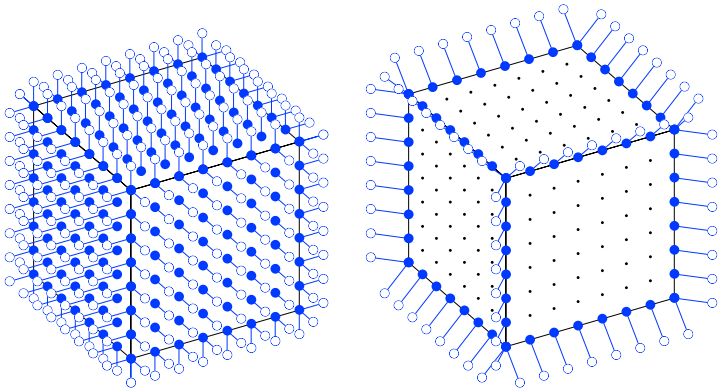
1. **for all** time step t **do**
2. **for all** mesh node $\boldsymbol{x}$ **do**
3. **for all** index α **do**
4. **if** $\boldsymbol{x} - \boldsymbol{\xi}_\alpha$ is a boundary node **then**
5. apply boundary conditions
6. **else**
7. load $\rho(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$, $\boldsymbol{u}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$, and $T(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$
8. **end if**
9. compute $f_\alpha^{(\mathrm{e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$ and $f_\alpha^{(\mathrm{e,o})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$
10. compute $g_\alpha^{(\mathrm{e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$ and $g_\alpha^{(\mathrm{e,e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha,\ t)$
11. **end for**
12. **for all** index α **do**
13. compute $g_\alpha(\boldsymbol{x},\ t+1)$
14. **end for**
15. compute $T(\boldsymbol{x},\ t+1)$ and buoyancy force $\boldsymbol{F}(\boldsymbol{x},\ t)$
16. **for all** index α **do**
17. compute $f_\alpha(\boldsymbol{x},\ t+1)$
18. **end for**
19. compute $\rho(\boldsymbol{x},\ t+1)$ and $\boldsymbol{u}(\boldsymbol{x},\ t+1)$
20. store $\rho(\boldsymbol{x},\ t+1)$, $\boldsymbol{u}(\boldsymbol{x},\ t+1)$, and $T(\boldsymbol{x},\ t+1)$
21. **end for**
22. **end for**

Algorithm 1: General formulation of the double-population thermal LW-ACM.

3.2. GPU implementation

Our implementation of the DPTLA was developed using the CUDA technology, introduced by the Nvidia company in 2007 as a framework for general purpose computation on graphics processing units. A detailed description of CUDA programming principles being beyond the scope of this article, we refer the interested reader to [8]. There are however a few aspects regarding the CUDA framework that should be mentioned before proceeding to the description of our program. A CUDA program mainly consists of host code, which are portions of the program run by the host system, and *kernels*, which are functions run in parallel by the GPU device in several *threads*. Each thread has access to its own instance of the local variables of the kernel. The whole of the threads running a kernel is called the *execution grid*, which is subdivided into *blocks* of threads of identical size. On modern Nvidia hardware, grids and blocks may have up to three dimensions. Threads within a block are executed synchronously in groups named *warps* and may exchange data through a local shared memory, which is fast but limited in size. At grid level, blocks are executed asynchronously, and threads belonging to different blocks may only exchange data through the off-chip device memory. Although cached, the device memory is by far slower than registers and shared memory. Its size ranges from 1 GiB on basic commodity graphics cards to 12 GiB on high-end dedicated computing devices.

For our GPU implementation of the DPTLA, we followed a strategy similar to the one described in [3]. We chose to map the computation domain by the execution grid, each thread being assigned to a single node of the mesh, which enables to take advantage of the massive parallelism of GPUs. Each block of threads is therefore associated to a block of nodes in the computation domain. For the sake of clarity, let us call *halo* of a block the set of all next neighbour nodes of this block. When processing the updating rules on a block of nodes, the associated threads must first load from device memory into shared memory the density, velocity, and temperature for the block and its halo. Each thread carries out the read operations for its assigned node. In addition, as illustrated by Fig. 2, the threads at the faces of the block are responsible for the nodes at the faces of the halo and the threads at the edges of the block are responsible for the nodes at the edges of the halo. The



(a) Faces of the halo. (b) Edges of the halo.

Fig. 2. Data access to the halo of a block for the main computation kernel—the plain discs represent the active threads whereas the hollow ones represent the nodes from which ρ, $\boldsymbol{u}$, and T are read.

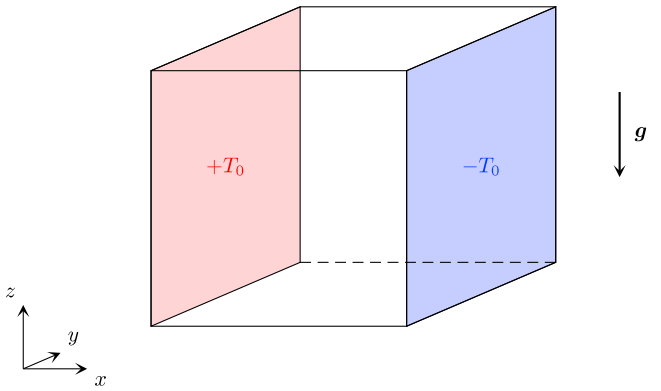


Fig. 3. The DHCC test case—the cavity consists of two isothermal vertical walls and four adiabatic walls.

density and velocity components are grouped into a `float4` structure[2] whereas an additional `float` number is used for the temperature. After all load operations have completed, which is enforced by a synchronisation barrier, the threads proceed to the computations and store the updated values for the block in device memory.

The shape and size of the blocks must be chosen carefully in order to minimise read redundancy. On one hand, to obtain the minimal ratio between halo and block, the blocks must be cubic and as large as possible. On the other hand, the corresponding data must fit into shared memory and number of threads within the block should be a multiple of the warp size (which is equal to 32 for all hardware generations up to now). Our experiments show that, with a maximal shared memory size of 48 KiB as with our test hardware, the optimal dimensions for a block are $8 \times 8 \times 8$. In this case, the read redundancy ratio is $(10^3 - 2^3)/8^3 \approx 1.94$ (the values at the vertices of the halo are not needed when using the D3Q19 stencil). In the bulk of the computation domain, the average amount of data read is therefore about 9.69 words per node for D3Q19 DPTLA, whereas it is 26 words per node for the D3Q19 HTLBM GPU implementation described in [10].

4. Methodology

4.1. Differentially heated cubic cavity

To validate our DPTLA solver, we simulated the differentially heated cubic cavity (DHCC) represented in Fig. 3. This test case consists of a cubic cavity in a constant gravity field $\boldsymbol{g}$, with two opposite isothermal vertical walls, one at $T = +T_0$ and the other at $T = -T_0$, and four adiabatic walls. In our simulations, the kinematic viscosity is set to $\nu = 0.05$ in lattice units, i.e. using δx and δt as units of length and time respectively, and the Prandtl number $\mathrm{Pr} = \nu/\kappa$ is set to $\mathrm{Pr} = 0.71$. The Rayleigh number $\mathrm{Ra} = \mathrm{Pr} \cdot \mathrm{Gr}$ ranges from 10^4 to 10^7, Gr being the Grashof number defined as:

$$\mathrm{Gr} = \frac{2T_0\beta g N^3}{\nu^2}, \tag{14}$$

[2] The CUDA `float4` structure contains four `float` fields referenced by `x`, `y`, and `z`, which are used for velocity components, and `w`, which is used for density.

where β is the thermal expansion coefficient and N is the size of the cavity in lattice units. We assume the Boussinesq approximation for the buoyancy force $\boldsymbol{F}$, which is therefore expressed as:

$$\boldsymbol{F} = -\rho T \beta \boldsymbol{g}. \tag{15}$$

4.2. Convergence criterion

Let us denote $T^l_{i,j,k}$ the temperature at time step l and node (i, j, k) in lattice units. In numerous works devoted to the DHCC, the convergence criterion chosen for the temperature field is equivalent to:

$$\frac{1}{N^3} \sum_{i,j,k} \left(T^{l+p}_{i,j,k} - T^l_{i,j,k} \right)^2 < B, \tag{16}$$

where p is a constant number of time steps and B is a constant convergence threshold. However, such criterion is not satisfactory since it is resolution-dependent. As a matter of fact, in the case of diffusive scaling, setting $\varepsilon = 1/N$, leads to $\delta x = \varepsilon = 1/N$ and $\delta t = \varepsilon^2 = 1/N^2$. Hence, the duration in physical units corresponding to p time steps decreases as the resolution increases. In other words, the higher the resolution, the less severe the convergence criterion.

We chose instead to use the L^2-norm of the temperature evolution rate, which is expressed as:

$$\left\| \dot{T}^{\,l} \right\|_2 = \left[\delta x^3 \sum_{i,j,k} \left(\frac{T^{l+1}_{i,j,k} - T^l_{i,j,k}}{\delta t} \right)^2 \right]^{1/2} = \left[N \sum_{i,j,k} \left(T^{l+1}_{i,j,k} - T^l_{i,j,k} \right)^2 \right]^{1/2} \tag{17}$$

and we declare convergence when the criterion $\left\| \dot{T}^{\,l} \right\|_2 < 10^{-2}$ is met.

4.3. Performance evaluation

We carried out performance measures using an Nvidia GeForce GTX Titan Black graphics card featuring a GK110 GPU with 2880 cores and 6 GiB of device memory. The host system is an Intel Xeon workstation running the Debian 7.7 GNU/Linux operating system and the CUDA 6.5 framework. We tested our DPTLA solver as well as the HTLA solver presented in [7] and the single-GPU HTLBM solver described in [10], for cavities of size ranging from 96 to 512 for the LW-ACM based programs, and from 96 to 320 for the HTLBM program. While carrying out actual simulations on this hardware, we noticed that the performance achieved for long runs showed significant fluctuations. Since adaptive power management on the Kepler is temperature-driven, we assume that these fluctuations are mainly due to the variations of the temperature of the room where the workstation is located.

Since our primary goal regarding performance evaluation was to compare the three mentioned codes, we chose to carry out short runs in order to get uniform experimental conditions and reproducible results. The runs lasted approximatively 60 s whatever the size of the cavity. We estimated the required number of time steps by launching preliminary runs on cavities of various sizes. Moreover, since the results significantly depend on the initial temperature of the GPU, we monitored this temperature in order to launch the next run once a threshold temperature of 42 °C was reached.

5. Results and discussion

5.1. Performance

The performance results using single-precision on the DHCC for increasing cavity size are reported in Fig. 4. The three programs have close performance, ranging from 1197 to 1394 million lattice node updates per second (MLUPS). On average, the DPTLA and the HTLA implementations perform better than the HTLBM solver by 5% and 7% respectively. These small differences seem to be mostly hardware-related, since former tests on a GeForce GTX Titan, which is slightly less efficient than the device used in the present work, showed almost identical performance results for the three solvers.

For long runs, such as the simulations carried for our validation study, performance is lower than the reported measurements. For a low room temperature, i.e. less than 20 °C, the performance loss is typically around 5%, e.g. 1230 MLUPS instead of 1295 MLUPS on a 256^3 cavity, whereas for high room temperature, i.e. higher than 30 °C, we saw the performance loss going up to 15%.

Although all three thermal GPU solvers achieve similar performance, the corresponding data throughput for the DPTLA and HTLA codes is considerably lower than for the HTLBM code. As a matter of fact, on our test hardware the average data throughput is 71.0 GiB/s and 72.9 GiB/s for the DPTLA and HTLA implementations respectively, whereas the HTLBM implementation reaches 223.5 GiB/s. If the usual GPU performance model for LBM applied to LW-ACM, then performance of the latter should be considerably higher. The same observation can be made for isothermal LW-ACM which shows a speedup of 1.8 with respect to isothermal LBM [3], which is less than expected.

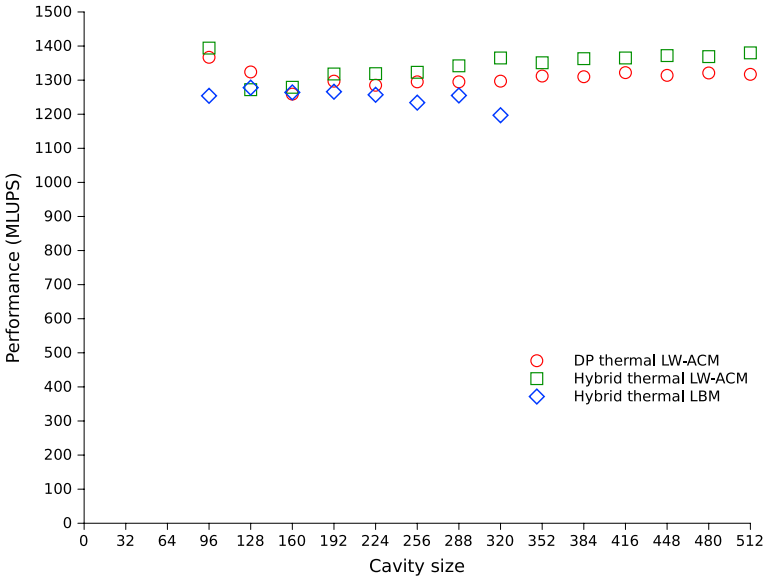


Fig. 4. Performance comparison for the DPTLA, HTLA, and HTLBM solvers—performance is reported in million lattice node updates per second (MLUPS). The programs were tested on the same GeForce GTX Titan Black graphics card.

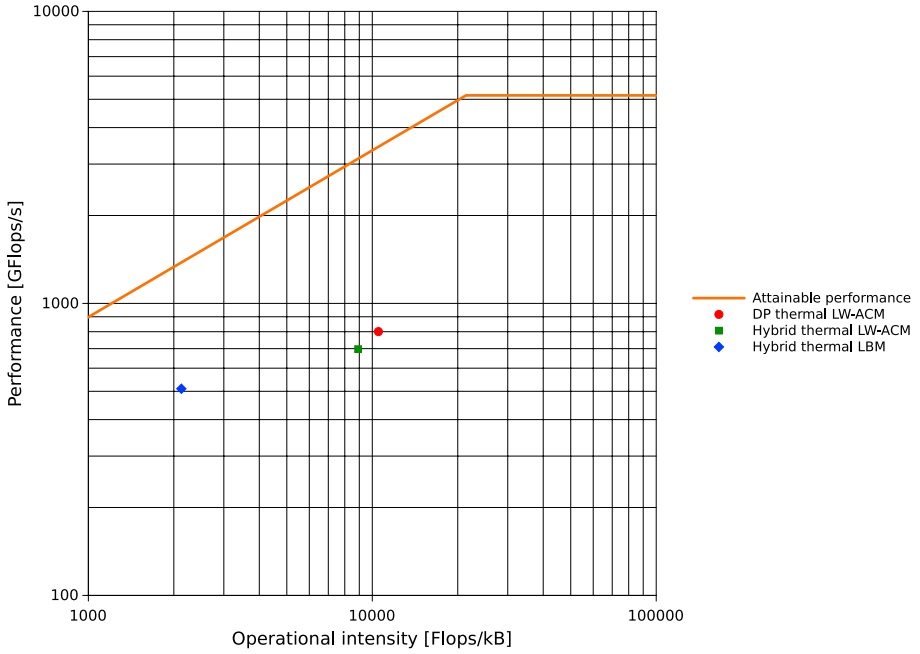


Fig. 5. Roofline diagram for the DPTLA, HTLA, and HTLBM solvers—The diagram represents the computational throughput with respect to the operational intensity, i.e. the number of floating point operations per amount of transferred data. The 'roofline' represents the maximum attainable performance on the GeForce GTX Titan Black graphics card.

In order to get a better understanding of the computational behaviour of the three codes on the GeForce GTX Titan Black, we have placed each of them in a *roofline diagram* [12] according to their algorithmic characteristics and average performance (see Fig. 5). The 'roofline' in Fig. 5, which corresponds to the maximum attainable performance, was determined using the maximum sustained computational throughput and data throughput obtained with the SHOC benchmark [13]. Furthermore, we used the `cuobjdump` utility to determine the number of floating point operations (Flops) in each main computation kernel. We obtained 612 Flops for DPTLA, 519 Flops for HTLA, and 408 for HTLBM.

In the 'roofline model' [12], programs located on the left-hand side of the inflexion point are memory-bound and programs located on the right-hand side are computation bound. According to this model, all three solvers are memory-bound. We furthermore see that the operational intensity of the LW-ACM codes is considerably higher than the one of the

Table 1
Nusselt numbers (Nu) at the isothermal hot wall obtained by DPTLA, HTLA, and HTLBM, and relative discrepancy (Δ) with respect to reference data.

Ra	N	DPTLA		HTLA		HTLBM	
		Nu	Δ	Nu	Δ	Nu	Δ
10^4	128	2.06360	0.46%	2.06422	0.49%	2.05984	0.27%
	256	2.06131	0.35%	2.06147	0.35%	2.05916	0.24%
	384	2.05974	0.27%	2.05995	0.28%	2.05862	0.22%
	512	2.05683	0.13%	2.05836	0.20%	–	–
10^5	128	4.34411	0.16%	4.33908	0.05%	4.34320	0.14%
	256	4.33501	−0.05%	4.33364	−0.08%	4.33120	−0.13%
	384	4.33288	−0.10%	4.33243	−0.11%	4.33034	−0.15%
	512	4.33250	−0.10%	4.33248	−0.10%	–	–
10^6	128	8.68767	0.54%	8.56856	−0.83%	8.82661	2.15%
	256	8.65080	0.12%	8.61264	−0.32%	8.66098	0.23%
	384	8.64038	0.00%	8.62276	−0.21%	8.64082	0.00%
	512	8.63726	−0.04%	8.62787	−0.15%	–	–
10^7	256	16.27012	−0.44%	15.94519	−2.43%	16.71278	2.26%
	384	16.37431	0.19%	16.14409	−1.22%	16.46645	0.76%
	512	16.36820	0.16%	16.22397	−0.73%	–	–

LBM code. This is consistent with the nature of LW-ACM which can be considered as a version of LBM where populations can be reconstructed at each timestep instead of being stored. We can also see that the LW-ACM codes actually achieve better computational throughput than our reference LBM code, namely 700 to 800 GFlops/s instead of about 500 GFlops/s.

A close inspection of the LW-ACM based programs shows that the significantly lower data throughput with respect to our reference LBM implementation is due to the local synchronisation barriers required after data fetching. A cure to this issue would be to decrease the size of the thread blocks, however this would in turn increase read redundancy. Our experiments show that the retained size of $8 \times 8 \times 8$ is the best trade-off for the targeted hardware but this might be different for future CUDA hardware generations.

5.2. Nusselt numbers

Table 1 provides the Nusselt numbers at the isothermal hot wall obtained with DPTLA as well as the results for HTLA given in [7] for N ranging from 128 to 512. For HTLBM, in order to apply the same convergence criterion, we used a modified version of the multi-GPU solver described in [11] running on one to four Tesla C2075 computing devices for $N = 128$, 256, and 384. All the computations were carried out in single-precision. The temperature gradient in the direction normal to the wall is estimated by second-order finite differences. We furthermore report the relative discrepancy of these results with respect to the reference values published by Tric et al. in [9]. These benchmark data were obtained using an accurate pseudo-spectral method. We selected the values computed on a 81^3 grid for our comparisons.

For Rayleigh numbers ranging from 10^4 to 10^6, all three models are in good agreement with our reference. The relative discrepancy being typically less than 1%, and close to 0.1% at the highest resolution. For $\mathrm{Ra} = 10^7$, all simulations diverge at $N = 128$. In terms of agreement with the reference values, the DPTLA performs slightly better than the two other models, but this is inconclusive since at these levels of spatial resolution, the thermal boundary layer is unlikely to be accurately resolved.

5.3. Flow features

In addition to heat fluxes, the work of Tric et al. provides some information regarding the flow features in the DHCC, namely the value and location of the maxima of the velocity components u_m, v_m, and w_m. Tables 2, 3, and 4 report these results for all three models at $N = 384$, as well as comparisons with the reference data. For this purpose, the velocity components are scaled by κ/N and the coordinates range from $-1/2$ to $1/2$. Again, we selected the reference values computed on a 81^3 grid. It should be mentioned that there is a probable misprint in the table providing results for $\mathrm{Ra} = 10^4$ (see Table 4 of [9]). In the part concerning w_m, the line labelled x actually provides the y-coordinate and the line providing the x-coordinate is missing. There is therefore no comparison for this case.

Again, for Rayleigh numbers ranging from 10^4 to 10^6, all three models are in good agreement with our reference. For both the maximal velocity components and their location, the relative discrepancy is typically less than 1%. The same conclusion holds for $\mathrm{Ra} = 10^7$, except for u_m where all models disagree with the reference data.

6. Conclusions

In this contribution, we describe an effective approach to implement a double-population LW-ACW thermal flow solver on CUDA GPUs. Performing large scale simulations of the differentially heated cubic cavity, we show that DPTLA

Table 2
Value and location of the maxima of the velocity components obtained by DPTLA, relative discrepancy of the value and normalised distance of the location with respect to reference data.

Ra	10^4		10^5		10^6		10^7	
u_m	16.6728	0.28%	44.2835	0.86%	128.4355	1.14%	406.1065	5.50%
	0.0195		−0.1836		−0.3060		−0.3763	
	0.0039	0.0040	0.2227	0.0032	0.3008	0.0026	0.3711	0.3711
	0.3242		0.3893		0.4388		0.4674	
v_m	2.1720	0.08%	9.7936	0.98%	25.6421	0.30%	85.9780	3.00%
	0.3841		0.4180		0.4518		−0.3320	
	0.2826	0.0018	0.3398	0.0017	0.3971	0.0017	0.4102	0.0026
	0.3451		0.3815		0.4180		0.3971	
w_m	18.9823	0.71%	71.3399	0.38%	237.0305	0.13%	770.2316	0.28%
	0.3841		0.4310		0.4622		0.4779	
	0.2357	–	0.3737	0.0039	0.4310	0.0023	0.4596	0.0388
	0.0195		0.0039		0.0273		−0.0065	

Table 3
Value and location of the maxima of the velocity components obtained by DPTLA, relative discrepancy of the value and normalised distance of the location with respect to reference data.

Ra	10^4		10^5		10^6		10^7	
u_m	16.6800	0.24%	44.2981	0.89%	128.6368	1.29%	408.2344	5.99%
	0.0195		−0.1836		−0.3060		−0.3763	
	0.0039	0.0040	0.2227	0.0032	0.3008	0.0026	0.0013	0.0043
	0.3242		0.3893		0.4388		0.4701	
v_m	2.1737	0.79%	9.7986	1.03%	25.8183	0.98%	83.0815	0.38%
	0.3841		0.4180		0.4518		−0.3294	
	0.2826	0.0018	0.3398	0.0017	0.3971	0.0017	0.4102	0.0073
	0.3451		0.3815		0.4180		0.4023	
w_m	18.9843	0.00%	71.3462	0.39%	237.2971	0.24%	774.7953	0.87%
	0.3841		0.4310		0.4622		0.4779	
	0.2357	–	0.3737	0.0039	0.4310	0.0023	0.4596	0.0206
	0.0195		0.0039		0.0273		0.0117	

Table 4
Value and location of the maxima of the velocity components obtained by HTLBM, relative discrepancy of the value and normalised distance of the location with respect to reference data.

Ra	10^4		10^5		10^6		10^7	
u_m	16.6707	0.29%	44.2512	0.79%	127.5759	0.47%	377.2339	1.74%
	0.0195		−0.1836		−0.3060		−0.3737	
	0.0013	0.0015	0.2201	0.0008	0.2982	0.0016	0.3659	0.3659
	0.3242		0.3867		0.4362		0.4648	
v_m	2.1661	0.44%	9.7902	0.95%	25.7571	0.75%	82.8791	0.63%
	0.3815		0.4180		0.4518		−0.3320	
	0.2826	0.0009	0.3398	0.0015	0.3971	0.0018	0.4076	0.0015
	0.3451		0.3789		0.4154		0.3945	
w_m	18.9468	0.19%	71.2223	0.0022	236.7128	0.00%	766.0296	0.27%
	0.3841		0.4310		0.4596		0.4779	
	0.2331	–	0.3737	0.0065	0.4310	0.0023	0.4596	0.0055
	0.0195		0.0065		0.0247		0.0378	

is comparable to HTLA and HTLBM, i.e. hybrid LW-ACW or LBM thermal flow solvers, both in terms of accuracy with respect to reference data, and in terms of performance on recent GPUs. LW-ACM based thermal flow solvers are promising, since in both cases the device memory consumption is significantly lower than HTLBM, allowing to manage substantially larger computation domains. Performance of the DPTLA implementation on a computing device featuring a GK110 GPU is about 1300 MLUPS on average, and might be considerably higher on future hardware such as the GK210 which will provide up to 112 KiB of local shared memory instead of 48 KiB with the GK110. For our DPTLA and HTLA implementations, this larger shared memory should increase the number of concurrently processed blocks and therefore may have a positive impact on the local synchronisation issue.

In future, we will focus on designing and implementing boundary conditions for DPTLA, with the aim of performing thermal flow simulations involving complex geometries. Special care will be taken in order to minimise the impact of these

boundary conditions on performance. We also plan to extend our single-GPU DPTLA solver to multi-GPU using an MPI-based approach similar to the one presented in [14].

References

[1] P. Asinari, T. Ohwada, E. Chiavazzo, A.F. Di Rienzo, Link-wise artificial compressibility method, J. Comput. Phys. 231 (15) (2012) 5109–5143.
[2] A.J. Chorin, A numerical method for solving incompressible viscous flow problems, J. Comput. Phys. 2 (1) (1967) 12–26.
[3] C. Obrecht, P. Asinari, F. Kuznik, J.-J. Roux, High-performance implementations and large-scale validation of the link-wise artificial compressibility method, J. Comput. Phys. 275 (2014) 143–153.
[4] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, L.S. Luo, Multiple-relaxation-time lattice Boltzmann models in three dimensions, Phil. Trans. R. Soc. A 360 (2002) 437–451.
[5] M. Wittmann, T. Zeiser, G. Hager, G. Wellein, Comparison of different propagation steps for lattice Boltzmann methods, Comput. Math. Appl. 65 (6) (2013) 924–935.
[6] P. Lallemand, L.-S. Luo, Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions, Phys. Rev. E 68 (3) (2003) 36706. 1–25.
[7] C. Obrecht, F. Kuznik, G. Rusaouën, J.-J. Roux, Towards high-performance thermal flow solvers based on the link-wise artificial compressibility method, in: Proceedings of the 15th International Heat Transfer Conference, 2014.
[8] Nvidia, Compute Unified Device Architecture Programming Guide version 6.5, August 2014.
[9] E. Tric, G. Labrosse, M. Betrouni, A first incursion into the 3D structure of natural convection of air in a differentially heated cubic cavity, from accurate numerical solutions, Int. J. Heat Mass Transfer 43 (21) (2000) 4043–4056.
[10] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, The TheLMA project: a thermal lattice Boltzmann solver for the GPU, Comput. & Fluids 54 (2012) 118–126.
[11] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, Multi-GPU implementation of a hybrid thermal lattice Boltzmann solver using the TheLMA framework, Comput. & Fluids 80 (2013) 269–275.
[12] S. Williams, A. Waterman, D. Patterson, Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76.
[13] A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter, The scalable heterogeneous computing (SHOC) benchmark suite, in: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM, 2010, pp. 63–74.
[14] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, Scalable lattice Boltzmann solvers for CUDA GPU clusters, Parallel Comput. 39 (6–7) (2013) 259–270.