Computers & Fluids 80 (2013) 269–275

Contents lists available at SciVerse ScienceDirect

Computers & Fluids

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

Multi-GPU implementation of a hybrid thermal lattice Boltzmann solver using the TheLMA framework

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

[a] *EDF R&D, Département EnerBAT, F-77818 Moret-sur-Loing Cedex, France*
[b] *Université de Lyon, F-69361 Lyon Cedex 07, France*
[c] *INSA-Lyon, CETHIL, UMR 5008, F-69621 Villeurbanne Cedex, France*
[d] *INSA-Lyon, CITI, INRIA, F-69621 Villeurbanne Cedex, France*
[e] *Université Lyon 1, F-69622 Villeurbanne Cedex, France*

A R T I C L E   I N F O

Article history:
Received 15 September 2011
Received in revised form 18 February 2012
Accepted 20 February 2012
Available online 7 March 2012

Keywords:
Thermal lattice Boltzmann method
GPU computing
CUDA

A B S T R A C T

In this contribution, a single-node multi-GPU thermal lattice Boltzmann solver is presented. We implement a simplified version of the hybrid model developed by Lallemand and Luo in 2003, which combines multiple-relaxation-time lattice Boltzmann for the fluid flow with a finite-difference method for temperature. The program is based on the TheLMA framework which was developed for that purpose. The chosen implementation and optimisation strategies are described, both for inter-GPU communication and for coupling with the thermal component of the model. Validation and performance results are provided as well.

1. Introduction

Since its introduction in the late eighties, the lattice Boltzmann method (LBM) has proven to be an effective approach in computational fluid dynamics (CFDs). It has been successfully applied to a wide range of engineering issues such as multiphase flows, porous media, or free surface flows. Despite of these achievements, the use of the LBM for thermal flow simulation is not very widespread yet. A possible reason for this situation is the relatively high computational cost of most thermal LBM models, such as multi-speed models [18] or double-population models [5] which both largely increase the required amount of data per node. In the present work, we implement a simplified version of a hybrid thermal LBM model [6] in which fluid flow is simulated using multiple-relaxation-time LBM and temperature is solved using a finite-difference scheme. The overhead due to the thermal part is likely to be rather small since this approach only introduces one additional scalar to the usual isothermal model.

The use of emerging many-core architectures such as graphics processing units (GPUs) in CFD is fairly promising [2]. Being a regular data-parallel algorithm, the LBM is especially well adapted to such hardware. Nevertheless, implementing the lattice Boltzmann method for the GPU is still a pioneering task.

Several issues, such as multi-physics applications and efficient multi-GPU implementations, are of cardinal importance to reinforce the usability of GPU LBM solvers in practical situations. Regarding the later issue, several successful attempts were reported recently, either single-node [16], heterogeneous [3], or multi-node [21]. To the best of our knowledge, the present work, is the first one describing the implementation of a multi-GPU thermal LBM solver.

The remaining of the paper is organised as follows. In the first section, we briefly present the thermal lattice Boltzmann model we chose to implement. Next, we give an overview of the TheLMA framework on which our program is based. In the third section, we describe our implementation and our optimisation strategies. Last, we provide some simulation results for validation purpose and discuss performance issues.

2. Hybrid thermal lattice Boltzmann model

The lattice Boltzmann equation (LBE), i.e. the governing equation of the LBM is interpreted as a discrete version of the Boltzmann equation [8]. In the LBM, as for the Boltzmann equation, a fluid is represented through the distribution of a single particle in phase space (i.e. position $\mathbf{x}$ and particle velocity ξ). Space is commonly discretised using a uniform orthogonal lattice of mesh size δx and time using constant time steps δt. Moreover, the particle velocity space is discretised into a finite set of particle velocities ξ_α. The LBM counterpart of the distribution function $f(\mathbf{x}, \xi, t)$ is a

* Corresponding author at: INSA-Lyon, CETHIL, UMR 5008, F-69621 Villeurbanne Cedex, France.
E-mail address: christian.obrecht@insa-lyon.fr (C. Obrecht).

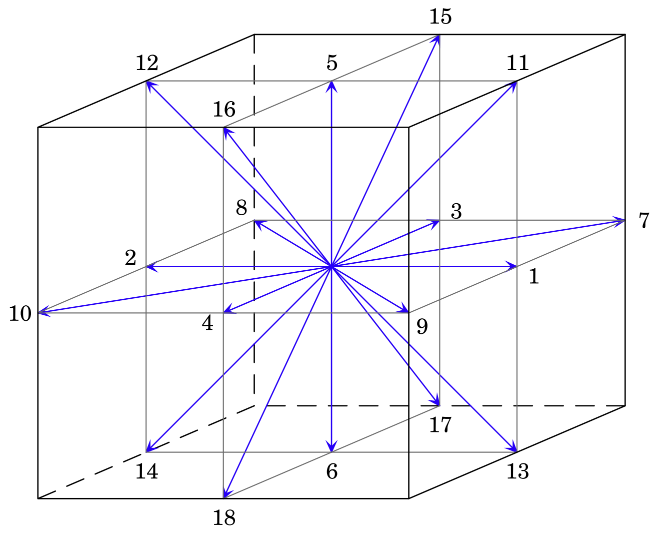


Fig. 1. Outline of the D3Q19 stencil for one single node.

finite set $f_\alpha(\mathbf{x},t)$ of particle distribution functions associated to the particle velocities ξ_α. The LBE writes:

$$|f_\alpha(\mathbf{x}+\delta t\xi_\alpha, t+\delta t)\rangle - |f_\alpha(\mathbf{x},t)\rangle = \Omega[|f_\alpha(\mathbf{x},t)\rangle]. \quad (1)$$

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

$$\rho = \sum_\alpha f_\alpha, \qquad \mathbf{j} = \sum_\alpha f_\alpha \xi_\alpha. \quad (2)$$

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. It is well-known that such basic models are not energy conserving. To address this issue, several approaches such as multi-speed models [17] or double-population models [5] have been developed. In the former category, a larger set of particle velocities is defined allowing multiple particle speeds along some directions. In the later category, an additional set of energy distribution functions is used. Both approaches suffer from inherent numerical instabilities [6]. Moreover, from a computational standpoint, both methods lead to a markedly higher memory consumption.

Hybrid thermal lattice Boltzmann models constitute an alternative approach in which the flow simulation is decoupled from the solution of the heat equation. These models are free from the aforementioned drawbacks. In the present work, we implemented a simplified version of the hybrid thermal lattice Boltzmann model developed in [6]. Flow simulation is performed by multiple-relaxation-time LBM [1], using the D3Q19 stencil. In the multiple-relaxation-time approach, collision is performed in moment space and the LBE writes:

$$|f_\alpha(\mathbf{x}+\delta t\xi_\alpha, t+\delta t)\rangle - |f_\alpha(\mathbf{x},t)\rangle = -\mathsf{M}^{-1}\mathsf{S}\left[|m_\alpha(\mathbf{x},t)\rangle - |m_\alpha^{(\mathrm{eq})}(\mathbf{x},t)\rangle\right] \quad (3)$$

where M is an orthogonal matrix mapping the set of particle distributions to a set of moments m_α, and S is a diagonal matrix containing the relaxation rates. Matrices M and S as well as the equilibria of the moments for the D3Q19 stencil can be found in Appendix A of [1].

In our simulations, unlike Lallemand and Luo in [6], we set the ratio of specific heats $\gamma = C_P/C_V$ to $\gamma = 1$, which simplifies the coupling of the temperature T to the fluid momentum. Temperature is therefore obtained by solving the following finite-difference equation:

$$\partial_t^* T = \kappa\Delta^* T - \mathbf{j}\cdot\nabla^* T \quad (4)$$

where κ denotes the thermal diffusivity, which we assume being constant. For the sake of simplicity, we set $\delta x = 1$ and $\delta t = 1$, and define the finite-difference operators as:

$$\partial_t^* T(t) = T(t+1) - T(t) \quad (5)$$

$$\begin{aligned}\partial_x^* T(i,j,k) = {} & T(i+1,j,k) - T(i-1,j,k) - \frac{1}{8}(T(i+1,j+1,k)\\ & - T(i-1,j+1,k) + T(i+1,j-1,k) - T(i-1,j-1,k)\\ & + T(i+1,j,k+1) - T(i-1,j,k+1) + T(i+1,j,k-1)\\ & - T(i-1,j,k-1))\end{aligned} \quad (6)$$

$$\begin{aligned}\Delta^* T(i,j,k) = {} & 2(T(i+1,j,k) + T(i-1,j,k) + T(i,j+1,k)\\ & + T(i,j-1,k) + T(i,j,k+1) + T(i,j,k-1))\\ & - \frac{1}{4}(T(i+1,j+1,k) + T(i-1,j+1,k)\\ & + (i+1,j-1,k) + T(i-1,j-1,k) + T(i,j+1,k+1)\\ & + T(i,j-1,k+1) + T(i,j+1,k-1) + T(i,j-1,k-1)\\ & + T(i+1,j,k+1) + T(i-1,j,k+1)\\ & + T(i+1,j,k-1) + T(i-1,j,k-1)) - 9T(i,j,k)\end{aligned} \quad (7)$$

It should be mentioned that these operators share the same symmetries as the D3Q19 stencil. The coupling of the temperature to the momentum is explicit in Eq. (4). The coupling of the momentum to the temperature is carried out in the equilibrium of the second order moment m_2 related to internal energy:

$$m_2^{(\mathrm{eq})} = -11\rho + 19\mathbf{j}^2 + T \quad (8)$$

3. The TheLMA framework

Since the introduction of the CUDA technology [11] by the Nvidia company in 2007, several successful attempts to implement the LBM on the GPU were reported [19,13]. Yet, constraints induced by low-level hardware specificities make GPU programming fairly different from usual software engineering. Beside other limitations, it is worth mentioning the fact that all symbols (e.g. device functions, device constants...) appearing in a kernel must be in the same compilation unit, which is due to the inability of the compilation tool chain to link several GPU binaries. Moreover, with hardware of compute capability 1.3 (which is the target architecture in the present work), device functions are of limited interest since inlining is compulsory in most cases.[1] Because of these constraints, library oriented development seems not relevant up to now for CUDA LBM solvers.

To improve code reusability, we designed the TheLMA framework [12], which is outlined in Fig. 2. TheLMA stands for *Thermal LBM on Many-core Architectures*, thermal flow simulations being our main topic of interest. The framework consists in a set of C and CUDA source files. The C files provide a set of utility functions to retrieve simulation parameters, initialise computation devices, extract statistical informations, and export data in various output formats. The CUDA files are included at compile time in the `thelma.cu` file which is mainly a container, additionally providing some general-purpose macros. Implementing a new lattice Boltzmann model within the framework mostly requires to alter the `compute.cu` file.

[1] Only device functions with a short list of parameters containing no pointers are actually callable and are likely to be not inlined by the compiler. Starting with hardware of compute capability 2.0, i.e. the Fermi generation, it is possible to perform actual function calls in any case, yet inlining is still the default behaviour.

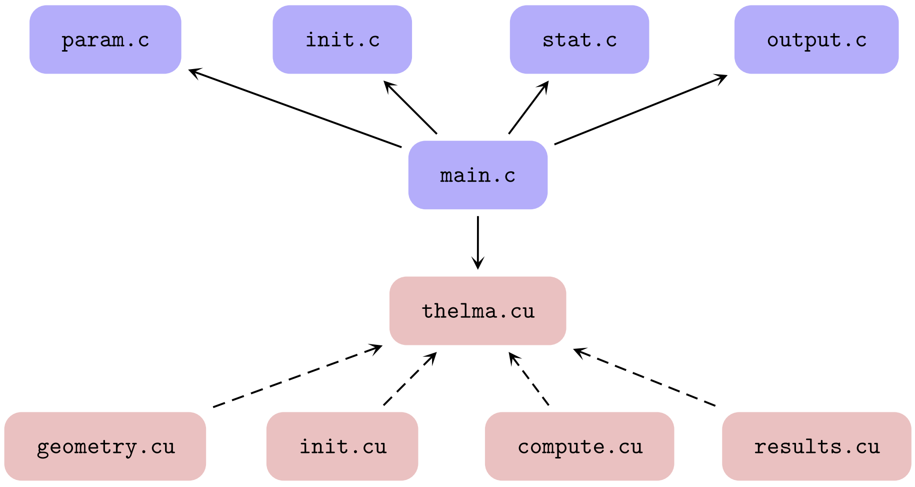


Fig. 2. Overall structure of the TheLMA framework.

Our framework provides native single-node multi-GPU management based on POSIX threads [15,16]. Each computing device is managed by a specific thread which is responsible for creating the appropriate CUDA context. Communication between sub-domains is carried out using zero-copy transactions on pinned exchange buffers in CPU memory. During initialisation, one-dimensional domain decomposition is performed. The interfaces between the sub-domains are set such as to be normal to the major dimension of the lattice in memory. The read and store accesses to CPU memory can therefore be coalesced. Moreover, the chosen configuration leads to an excellent communication and computations overlapping. Fig. 3 describes the inter-GPU communication scheme. For the sake of simplicity, only one GPU associated to a single sub-domain interface is presented. In order to address data dependency issues, since computations at grid level are asynchronous, our solver uses two instances of the lattice to store even time step data (in red) and odd time step data (in blue). The top line represents the GPU computations, Lattice 0 and Lattice 1 stand for the instances of the lattice stored in global memory, Read buffer 0 and Read buffer 1 for the buffers containing in-coming data, Store buffer 0 and Store buffer 1 for the buffers containing out-going data.

4. Implementation

For the implementation of a hybrid thermal lattice Boltzmann model on the GPU, there is the alternative of using a single kernel or two distinct kernels for solving the fluid flow and the thermal part. Since Eqs. (3) and (4) are tightly coupled, the two kernels option would increase the communication needs, not mentioning the overhead of kernel switching. As a matter of fact, handling temperature in a separate kernel, would require momentum to be stored and read back at each time step, thus increasing the amount of

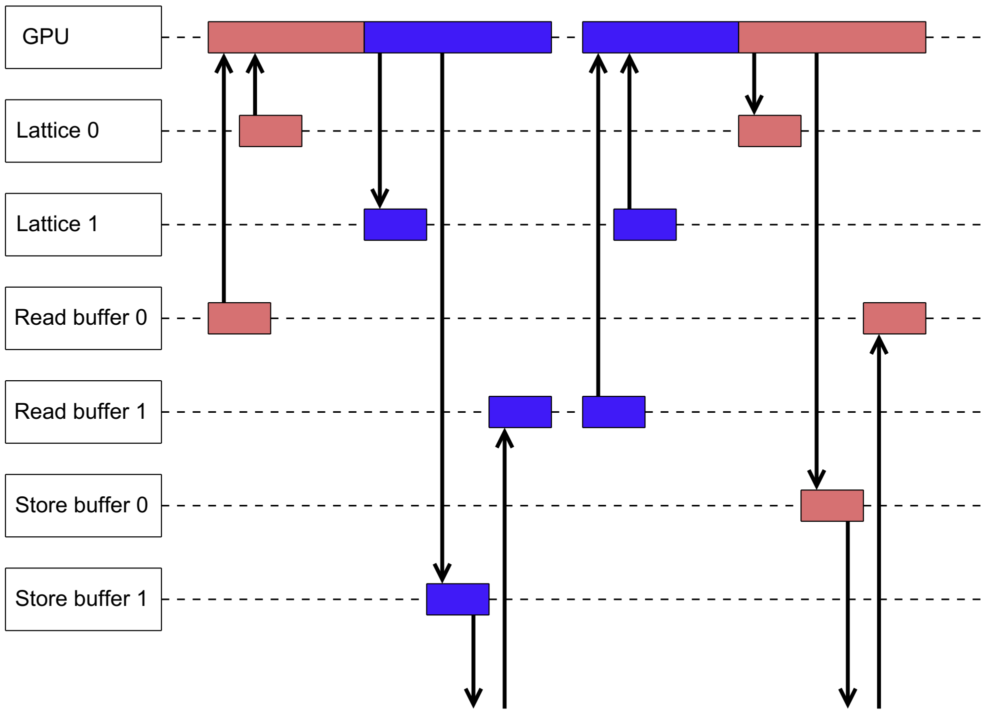


Fig. 3. Inter-GPU communication scheme.

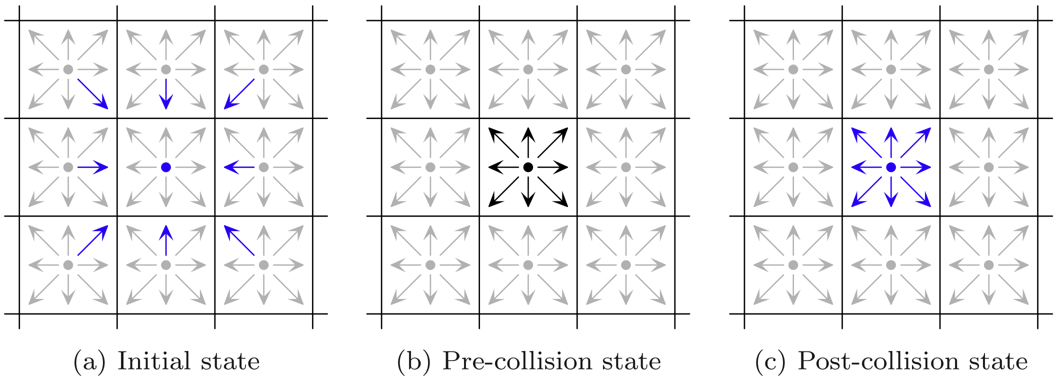


Fig. 4. In-place propagation.

exchanged data by about 15%, for both read and store accesses. Since our isothermal multi-GPU solver is communication bound, we chose to process both parts in the same kernel.

The fluid flow component is derived from the one described in [13]. Beside other optimisations, the kernel uses in-place propagation instead of the usual out-of-place propagation, which consists in performing propagation when reading the particle densities at the beginning of the kernel execution. Fig. 4 illustrates this method in the two-dimensional case for the sake of clarity. Fig. 4a shows the initial state before propagation, the particle densities required by the central node are displayed in blue. Fig. 4b shows the state after propagation and before collision, the pre-collision particle densities belonging to the central node are displayed in black. Fig. 4c shows the state after collision, the post-collision particle densities belonging to the central node are displayed in blue. The in-place propagation approach allows to minimise the cost of misaligned global memory transactions, since misaligned read accesses are less penalising than misaligned write accesses. [14]. Misalignment may have a dramatic impact on performance with pre-Fermi hardware, since the device's main memory is not cached.

CUDA implementations of the LBM generally assign a thread to each node in order to take advantage of the massive parallelism of the architecture. This approach often leads to the use of a two-dimensional grid of one-dimensional blocks, which allows a straightforward determination of the coordinates. The grid and block dimensions are therefore identical to the size of the computation sub-domain. The direction of the blocks corresponds to the slowest varying dimension in memory in order to enable coalesced memory accesses.

In our case, these common sense optimisation principles had to be altered. Since the implemented kernel takes care of both the fluid flow part and the thermal part, the register consumption is fairly higher than for usual isothermal LBM kernels. For compute capability 1.3, we could not achieve less than 124 registers per thread. Thus, assigning one thread to each node and using blocks spanning the entire cavity width causes register shortage with large cavities. In order to avoid this issue, we instead use small blocks containing one to four warps (i.e. 32, 64, 96 or 128 threads), each one associated to a one-dimensional zone spanning the cavity width. The kernel loops over the zone, in case of the block size being smaller than the zone size. Fig. 5a outlines the chosen configuration (in two dimensions for the sake of simplicity). The blue dots represent the nodes belonging to the zone; the red frame represents the nodes being processed by the block of threads at a given step; the white background is used for the nodes whose temperature is required by the finite-difference computations in the zone.

The associated grid is two-dimensional, its size corresponding to the remaining dimensions of the sub-domain. It is worth mentioning that we assign the first rather than the second field of the `blockIdx` structure to the fastest varying dimension in memory. This option appears to improve the overlapping of computation and inter-GPU communication.

When implementing stencil computations on the GPU, reducing read redundancy is a key optimisation target [9]. In our case, with the chosen finite-difference stencil, two consecutive nodes belonging to the same one-dimensional zone share ten required temperatures. We therefore chose to store the temperature of the active nodes and their neighbours in shared memory. In the case of boundary nodes, the surplus cells in the temperature array may be used to store shadow values determined by extrapolation. During the read phase, each thread is responsible for gathering the temperatures of all the nodes sharing the same abscissa, as outlined in Fig. 5b.

Not taking the boundaries into account, the chosen approach reduces the read redundancy in the D3Q19 case from 19 to at most[2] 9.3125. Moreover, it should be noted that this data access pattern induces no misalignement at all.

5. Results and discussion

5.1. Test case

To test our code, we simulated the well-known differentially heated cubic cavity illustrated in Fig. 6. In this test case, two vertical opposite walls have imposed temperatures $\pm T_0$ whereas the four remaining walls are adiabatic. The buoyancy force $\mathbf{F}$ is computed using the Boussinesq approximation:

$$\mathbf{F} = -\rho\beta T\mathbf{g} \tag{9}$$

where β is the thermal expansion coefficient, and $\mathbf{g}$ the gravity vector of magnitude g.

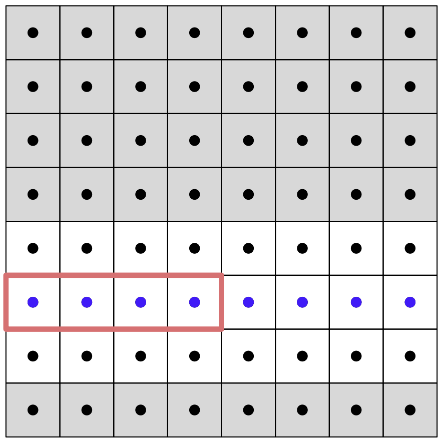
The simple bounce-back scheme is used for the flow field boundary conditions. As shown by [4], the solid boundary is asymptotically located half-way between the wall nodes and the fluid nodes. This aspect must be taken into account when imposing thermal boundary conditions. In our implementation, we use halo temperatures computed by second-order extrapolations. For imposed temperature T_0, we have:

$$T(-1) = \frac{8}{3}T_0 - 2T(0) + \frac{1}{3}T(1) \tag{10}$$

For adiabatic walls, we have:

$$T(-1) = \frac{21}{23}T(0) + \frac{3}{23}T(1) - \frac{1}{23}T(2) \tag{11}$$

[2] Five additional temperatures for both the first and the last node are read, thus the worst case is for blocks of size 32 and the read redundancy equals $(32 \times 9 + 2 \times 5)/32$.



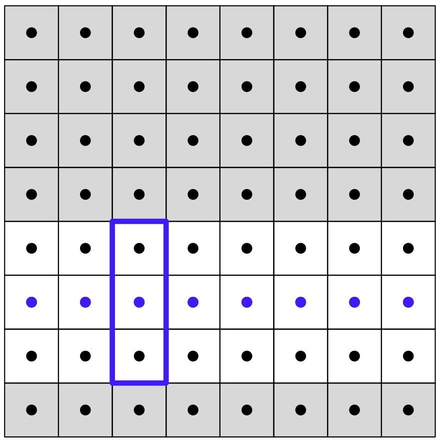


Fig. 5. Kernel execution pattern.

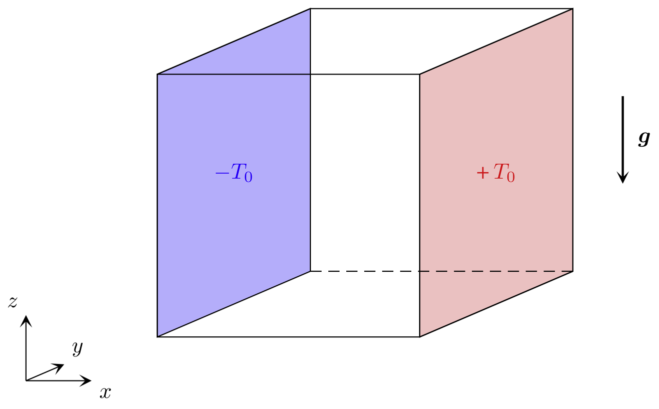


Fig. 6. Differentially cubic heated cavity.

Table 1
Comparison of Nusselt numbers at the isothermal walls.

Rayleigh number	10^4	10^5	10^6	10^7
Tric et al. [20]	2.054	4.337	8.640	16.342
Present (384^3)				
Nusselt number	2.055	4.339	8.652	16.481
Time steps	420,000	304,000	210,500	148,500
Computation time (min)	365	245	177	123
Relative deviation	0.05%	0.05%	0.14%	0.84%
Present (448^3)				
Nusselt number	2.050	4.335	8.645	16.432
Time steps	485,000	380,000	266,000	182,000
Computation time (min)	394	309	216	148
Relative deviation	0.19%	0.07%	0.06%	0.55%

The wall temperature T_0 is set to $T_0 = 1$ and the Prandtl number is set to $\mathrm{Pr} = 0.71$. The parameters for the simulations are the Rayleigh number Ra and the kinematic viscosity ν, which determine the thermal diffusivity κ and the value of $\beta\, g$.

We ran our program on a Tyan B7015 server fitted with eight Tesla C1060 computing devices. We could therefore perform computations on cavities as large as 512^3 in single precision.

5.2. Simulations

For validation purpose, we performed several simulations of the differentially heated cubic cavity using a 384^3 lattice and a 448^3 lattice in single precision. The kinematic viscosity was set to $\nu = 0.05$ and the Rayleigh number ranged from 10^4 to 10^7. The computations were carried out until convergence to steadiness, which is assumed to be reached when:

$$\max_{\mathbf{x}} |T(\mathbf{x}, t_{n+1}) - T(\mathbf{x}, t_n)| < 10^{-5} \qquad (12)$$

where $t_n = n \times 500\delta t$.

The obtained Nusselt numbers at the isothermal walls for both grid configurations are in good agreement as shown in Table 1, thus assuring grid independence. The simulation results are also close to previously published data [20], the maximum relative deviation being 0.84% for the coarser grid.

Using lattices of size 512^3 allowed us to run simulation for Rayleigh numbers up to 10^9 without facing numerical instabilities. From a phenomenological standpoint, although unsteady, the flow rapidly leads to a rather stable vertical stratification. We furthermore observe quasi-symmetric and quasi-periodic flow patterns near the bottom edge of the cold wall and the top edge of the hot wall. Fig. 7 shows the temperature field in the symmetry plane after 10^6 iterations. Further investigations on this simulation are required and will be published in a future contribution.

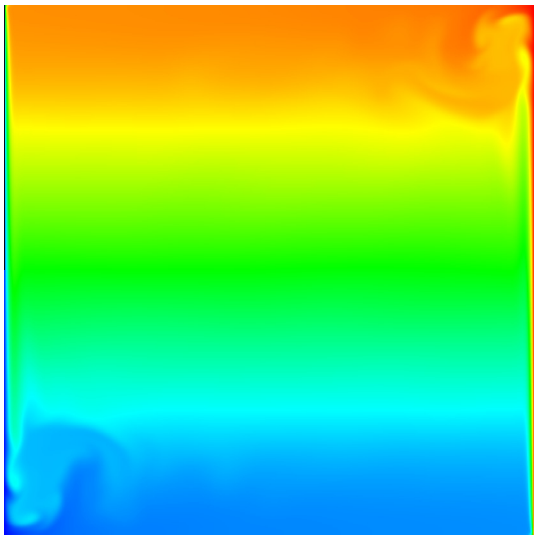


Fig. 7. Symmetry plane temperature field at $\mathrm{Ra} = 10^9$.

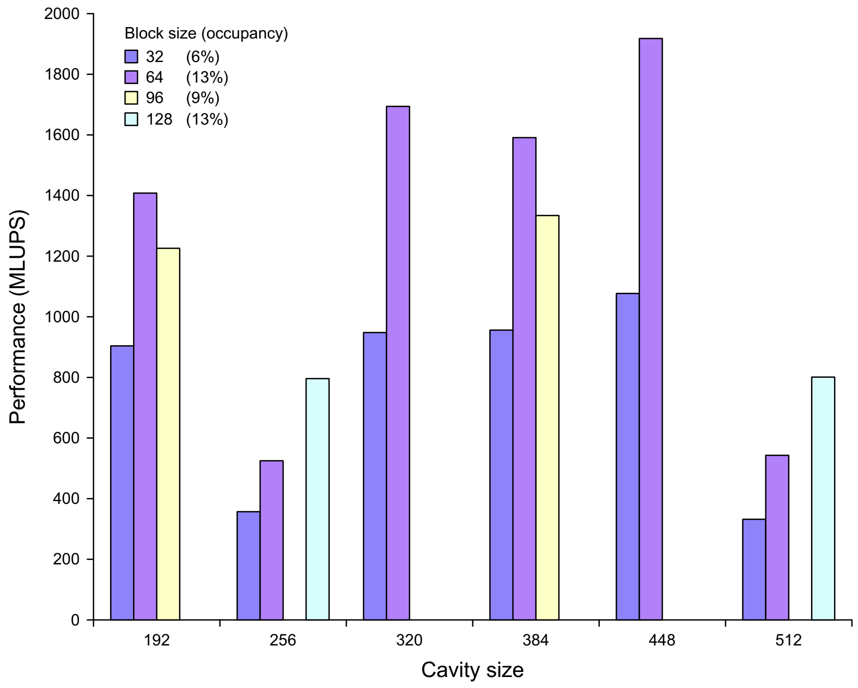


Fig. 8. Performance for increasing block size and cavity size.

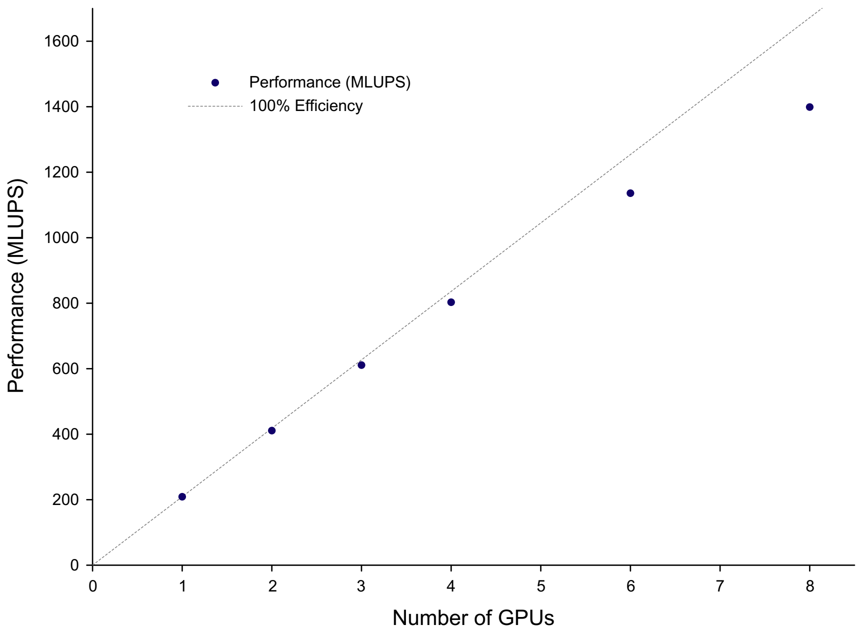


Fig. 9. Parallelisation efficiency on a 192^3 cavity.

5.3. Performance

We recorded performance results of our solver for increasing block size and cavity size (see Fig. 8). The chosen performance metric is the million lattice node updates per second (MLUPS). The cavity size has to be a multiple of the block size, hence several configurations are not available. Performance obtained with a given block size appears to be correlated to the corresponding occupancy. The poor efficiency occurring with cavity size 256 and 512 is probably caused by partition camping effects. With compute capability 1.3 hardware, global memory is split in eight 256 bytes wide memory banks [10]. The stride between corresponding nodes in distinct blocks is necessarily a multiple of the cavity size. Therefore, with cavity size 256 and 512, active blocks issue concurrent accesses to the same banks, which causes queuing and increases latency.

The maximum performance is 1920 MLUPS, achieved for cavity size 448 and block size 64. The corresponding GPU to device memory data throughput is 46.1 GB/s per GPU, which is about 62.3% of the maximum sustained throughput.[3] The multiprocessor occupancy, which is only 13%, appears to be the limiting factor since it is lower than the minimum required to properly hide the global memory latency, i.e. 18.75% for compute capability 1.3.

[3] Using the `bandwidthTest` program of the CUDA SDK, we estimate the GPU to device memory maximum sustained throughput to 73.3 GB/s for the Tesla C1060.

To evaluate scalability, we also ran our program on a 192^3 lattice using from one to eight GPUs (see Fig. 9). Parallelisation efficiency is very satisfactory with no less than 84% for a fairly small computation domain. As for our isothermal multi-GPU LBM solver [16], our implementation allows excellent overlapping of communication and computations. Moreover, the amount of data to exchange does not exceed the capacity of the PCI-E links.

6. Conclusion

In this contribution, we present a multi-GPU implementation of a thermal LBM solver, which to the best of our knowledge was never reported before. Using appropriate hardware, our program is able to run up to eight GPUs in parallel. With the latest generation of Nvidia computing devices, it is therefore possible to perform simulations on lattices containing as much as 3.2×10^8 nodes.

Validation studies have been carried out, showing both the accuracy and the stability of the chosen thermal LBM model and the correctness of our implementation. Although slightly less efficient than the isothermal version of our solver, our program provides unrivaled performance compared to CPU implementations. Recent studies [7] have shown that optimised multi-threaded CPU implementations of isothermal LBM solver running on up-to-date hardware achieve at most 85 MLUPS, which is 22$\times$ less than our maximum performance. Desktop computers nowadays are commonly able to handle three or four GPU computing devices, which are rather affordable. Using our solver instead of a CPU based solver on such hardware would provide a performance increase of about one order of magnitude.

We furthermore study the performance bottlenecks, showing that the limiting factor is the low occupancy. Since the multiprocessor occupancy is bound by the amount of available registers there is little room for improvements using the same hardware. Yet, a more elaborate memory access pattern could avoid the partition camping effects we could observe in some cases.

We believe our work is a significant step towards the use of GPU based LBM solvers in practice. In near future, we intend to add specific optimisations for compute capability 2.0 and 2.1 hardware, i.e. the latest CUDA capable GPUs. We also plan to extend the TheLMA framework on which our program is based to multi-node implementations.

References

[1] d'Humières D, Ginzburg I, Krafczyk M, Lallemand P, Luo LS. Multiple-relaxation-time lattice Boltzmann models in three dimensions. Phil Trans Math Phys Eng Sci 2002:437–51.
[2] Dongarra J, Moore S, Peterson G, Tomov S, Allred J, Natoli V, et al. Exploring new architectures in accelerating CFD for Air Force applications. In: Proceedings of HPCMP users group conference, Citeseer; 2008. p. 14–7.
[3] Feichtinger C, Habich J, Hager G, Wellein G, et al. A flexible patch-based lattice Boltzmann parallelization approach for heterogeneous GPU–CPU clusters. Parallel Comput 2011.
[4] Ginzbourg I, Adler PM. Boundary flow condition analysis for the three-dimensional lattice Boltzmann model. J Phys II 1994;4(2):191–214.
[5] He X, Chen S, Doolen GD. A novel thermal model for the lattice Boltzmann method in incompressible limit. J Comput Phys 1998;146(1):282–300.
[6] Lallemand P, Luo LS. Theory of the lattice Boltzmann method: acoustic and thermal properties in two and three dimensions. Phys Rev E 2003;68(3):36706.
[7] Lee VW, Kim C, Chhugani J, Deisher M, Kim D, Nguyen AD, et al. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In: ACM SIGARCH computer architecture news, vol. 38. ACM; 2010. p. 451–60.
[8] McNamara GR, Zanetti G. Use of the Boltzmann equation to simulate lattice-gas automata. Phys Rev Lett 1988;61:2332–5.
[9] Micikevicius P. 3D finite difference computation on GPUs using CUDA. In: Proceedings of 2nd workshop on general purpose processing on graphics processing units. ACM; 2009. p. 79–84.
[10] Mudigere D. Data access optimized applications on the GPU using NVIDIA CUDA. Master's thesis, Technische Universität München; 2009.
[11] NVIDIA, Compute unified device architecture programming guide version 3.1.1; July 2010.
[12] Obrecht C, Kuznik F, Tourancheau B, Roux J-J. Thermal LBM on manycore architectures; 2010. <www.thelma-project.info>.
[13] Obrecht C, Kuznik F, Tourancheau B, Roux J-J. A new approach to the lattice Boltzmann method for graphics processing units. Comput Math Appl 2011;12(61):3628–38. June.
[14] Obrecht C, Kuznik F, Tourancheau B, Roux J-J. Global memory access modelling for efficient implementation of the LBM on GPUs. In: Palma JMLM et al., editors. High performance computing for computational science – VECPAR2010. Lecture notes in computer science 6449. Springer; 2011. p. 151–61.
[15] Obrecht C, Kuznik F, Tourancheau B, Roux J-J. Multi-GPU implementation of the lattice Boltzmann method. Comput Math Appl 2011. doi:10.1016/j.camwa.2011.02.020.
[16] Obrecht C, Kuznik F, Tourancheau B, Roux J-J. The TheLMA project: multi-GPU implementation of the lattice Boltzmann method. Int J High Performance Comput Appl 2011;25(3):295–303. August.
[17] Qian YH. Simulating thermohydrodynamics with lattice BGK models. J Sci Comput 1993;8(3):231–42.
[18] Teixeira C, Chen H, Freed DM. Multi-speed thermal lattice Boltzmann method stabilization via equilibrium under-relaxation. Comput Phys Commun 2000;129(1-3):207–26.
[19] Tölke J. Implementation of a lattice Boltzmann Kernel using the compute unified device architecture developed by *n*VIDIA. Comput Visual Sci 2008:1–11.
[20] Tric E, Labrosse G, Betrouni M. 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 2000;43(21):4043–56.
[21] Xian Wang, Takayuki Aoki. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. Parallel Comput 2011;37(9):521–35.