HAL open science

High performance lattice Boltzmann solvers on massively parallel architectures with applications to building aeraulics

Christian Obrecht

▶ To cite this version:

Christian Obrecht. High performance lattice Boltzmann solvers on massively parallel architectures with applications to building aeraulics. Other [cond-mat.other]. INSA de Lyon, 2012. English. NNT : 2012ISAL0134 . tel-00776986v3

HAL Id: tel-00776986
https://theses.hal.science/tel-00776986v3

Submitted on 12 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre 2012ISAL0134
Année 2012

Thèse

High Performance Lattice Boltzmann Solvers on Massively Parallel Architectures with Applications to Building Aeraulics

Présentée devant
L'institut national des sciences appliquées de Lyon

Pour obtenir
Le grade de docteur

Formation doctorale
Génie civil
École doctorale
École doctorale MEGA (mécanique, énergétique, génie civil, acoustique)

Par
Christian Obrecht

Soutenue le 11 décembre 2012 devant la commission d'examen

Jury

J.-L. Hubert	Ingénieur-Chercheur (EDF R&D)	*Examinateur*
Ch. Inard	Professeur (Université de La Rochelle)	*Président*
M. Krafczyk	Professeur (TU Braunschweig)	*Rapporteur*
F. Kuznik	Maître de conférences HDR (INSA de Lyon)	*Co-directeur*
J. Roman	Directeur de recherche (INRIA)	*Rapporteur*
J.-J. Roux	Professeur (INSA de Lyon)	*Directeur*
B. Tourancheau	Professeur (UJF – Grenoble)	*Co-directeur*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Insa : R. GOURDON	M. Jean Marc LANCELIN Université de Lyon – Collège Doctoral Bât ESCPE 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72.43 13 95 directeur@edchimie-lyon.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Secrétariat : M.C. HAVGOUDOUKIAN eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 60 97 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://e2m2.universite-lyon.fr** Insa : H. CHARLES	Mme Gudrun BORNETTE CNRS UMR 5023 LEHNA Université Claude Bernard Lyon 1 Bât Forel 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cédex Tél : 04.72.43.12.94 e2m2@biomserv.univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** **http://ww2.ibcp.fr/ediss** Sec : Safia AIT CHALAL Insa : M. LAGARDE	M. Didier REVEL Hôpital Louis Pradel Bâtiment Central 28 Avenue Doyen Lépine 69677 BRON Tél : 04.72.68 49 09 Fax :04 72 35 49 16 Didier.revel@creatis.uni-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHEMATIQUES** http://infomaths.univ-lyon1.fr	M. Johannes KELLENDONK Université Claude Bernard Lyon 1 INFOMATHS Bâtiment Braconnier 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72. 44.82.94 Fax 04 72 43 16 87 infomaths@univ-lyon1.fr
Matériaux	**MATERIAUX DE LYON** Secrétariat : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry Ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA de Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cédex Tél : 04.72.43 83 18 Fax 04 72 43 85 28 Jean-yves.buffiere@insa-lyon.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** Secrétariat : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry mega@insa-lyon.fr	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél :04.72.43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	**ScSo*** M. OBADIA Lionel Sec : Viviane POLSINELLI Insa : J.Y. TOUSSAINT	M. OBADIA Lionel Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Tél : 04.78.69.72.76 Fax : 04.37.28.04.48 Lionel.Obadia@univ-lyon2.fr

*ScSo : Histoire, Geographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

High Performance Lattice Boltzmann Solvers on Massively Parallel Architectures with Applications to Building Aeraulics

Christian Obrecht

2012

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

I dedicate this work to my sons Adrien and Nicolas, for:

The lyf so short, the craft so longe to lerne,
Th'assay so hard, so sharp the conquerynge, ...

Geoffrey Chaucer — The Parlement of Foules

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Acknowledgements

First and foremost, I wish to acknowledge my deepest gratitude to my advisors Jean-Jacques Roux, Frédéric Kuznik, and Bernard Tourancheau whose trust and constant care made this whole endeavour possible. As well, I would like to express my particular appreciation to the EDF company for funding my research and to Jean-Luc Hubert for his kind consideration. I also wish to thank my fellows at the CETHIL: Miguel, Samuel, Andrea, Kévyn, and many others for their joyous company and the administrative staff, especially Christine, Agnès, and Florence, for their diligent work. Special acknowledgements go to Gilles Rusaouën for sharing his deep understanding of many topics related to my research work, and to the members of the ICMMES community for the so many fruitful discussions we had during our annual meetings. Last and most of all, I thank my beloved wife and sons for their everyday affection and support.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Résumé

Avec l'émergence des bâtiments à haute efficacité énergétique, il est devenu indispensable de pouvoir prédire de manière fiable le comportement énergétique des bâtiments. Or, à l'heure actuelle, la prise en compte des effets thermo-aérauliques dans les modèles se cantonne le plus souvent à l'utilisation d'approches simplifiées voire empiriques qui ne sauraient atteindre la précision requise. Le recours à la simulation numérique des écoulements semble donc incontournable, mais il est limité par un coût calculatoire généralement prohibitif. L'utilisation conjointe d'approches innovantes telle que la méthode de Boltzmann sur gaz réseau (LBM) et d'outils de calcul massivement parallèles comme les processeurs graphiques (GPU) pourrait permettre de s'affranchir de ces limites. Le présent travail de recherche s'attache à en explorer les potentialités.

La méthode de Boltzmann sur gaz réseau, qui repose sur une forme discrétisée de l'équation de Boltzmann, est une approche explicite qui jouit de nombreuses qualités : précision, stabilité, prise en compte de géométries complexes, etc. Elle constitue donc une alternative intéressante à la résolution directe des équations de Navier-Stokes par une méthode numérique classique. De par ses caractéristiques algorithmiques, elle se révèle bien adaptée au calcul parallèle. L'utilisation de processeurs graphiques pour mener des calculs généralistes est de plus en plus répandue dans le domaine du calcul intensif. Ces processeurs à l'architecture massivement parallèle offrent des performances inégalées à ce jour pour un coût relativement modéré. Néanmoins, nombre de contraintes matérielles en rendent la programmation complexe et les gains en termes de performances dépendent fortement de la nature de l'algorithme considéré. Dans le cas de la LBM, les implantations GPU affichent couramment des performances supérieures de deux ordres de grandeur à celle d'une implantation CPU séquentielle modérément optimisée.

Le présent mémoire de thèse est constitué d'un ensemble de neuf articles de revues internationales et d'actes de conférences internationales (le dernier étant en cours d'évaluation). Dans ces travaux sont abordés les problématiques liées tant à l'implantation mono-GPU de la LBM et à l'optimisation des accès en mémoire, qu'aux implantations multi-GPU et à la modélisation des communications inter-GPU et inter-nœuds. En complément, sont détaillées diverses extensions à la LBM indispensables pour envisager une utilisation en thermo-aéraulique des bâtiments. Les cas d'études utilisés pour la validation des codes permettent de juger du fort potentiel de cette approche en pratique.

Mots-clefs : calcul intensif, méthode de Boltzmann sur gaz réseau, processeurs graphiques, aéraulique des bâtiments.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Abstract

With the advent of low-energy buildings, the need for accurate building performance simulations has significantly increased. However, for the time being, the thermo-aeraulic effects are often taken into account through simplified or even empirical models, which fail to provide the expected accuracy. Resorting to computational fluid dynamics seems therefore unavoidable, but the required computational effort is in general prohibitive. The joint use of innovative approaches such as the lattice Boltzmann method (LBM) and massively parallel computing devices such as graphics processing units (GPUs) could help to overcome these limits. The present research work is devoted to explore the potential of such a strategy.

The lattice Boltzmann method, which is based on a discretised version of the Boltzmann equation, is an explicit approach offering numerous attractive features: accuracy, stability, ability to handle complex geometries, etc. It is therefore an interesting alternative to the direct solving of the Navier-Stokes equations using classic numerical analysis. From an algorithmic standpoint, the LBM is well-suited for parallel implementations. The use of graphics processors to perform general purpose computations is increasingly widespread in high performance computing. These massively parallel circuits provide up to now unrivalled performance at a rather moderate cost. Yet, due to numerous hardware induced constraints, GPU programming is quite complex and the possible benefits in performance depend strongly on the algorithmic nature of the targeted application. For LBM, GPU implementations currently provide performance two orders of magnitude higher than a weakly optimised sequential CPU implementation.

The present thesis consists of a collection of nine articles published in international journals and proceedings of international conferences (the last one being under review). These contributions address the issues related to single-GPU implementations of the LBM and the optimisation of memory accesses, as well as multi-GPU implementations and the modelling of inter-GPU and internode communication. In addition, we outline several extensions to the LBM, which appear essential to perform actual building thermo-aeraulic simulations. The test cases we used to validate our codes account for the strong potential of GPU LBM solvers in practice.

Keywords: high performance computing, lattice Boltzmann method, graphics processing units, building aeraulics.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Foreword

The present document is a thesis by publication consisting of a general introduction and the nine following articles:

[A] A New Approach to the Lattice Boltzmann Method for Graphics Processing Units. *Computers and Mathematics with Applications*, 12(61):3628–3638, June 2011.

[B] Global Memory Access Modelling for Efficient Implementation of the Lattice Boltzmann Method on Graphics Processing Units. *Lecture Notes in Computer Science 6449*, High Performance Computing for Computational Science – VECPAR 2010 Revised Selected Papers, pages 151–161, February 2011.

[C] The TheLMA project: a thermal lattice Boltzmann solver for the GPU. *Computers and Fluids*, 54:118–126, January 2012.

[D] Multi-GPU Implementation of the Lattice Boltzmann Method. *Computers and Mathematics with Applications*, published online March 17, 2011.

[E] The TheLMA project: Multi-GPU Implementation of the Lattice Boltzmann Method. *International Journal of High Performance Computing Applications*, 25(3):295–303, August 2011.

[F] Towards Urban-Scale Flow Simulations Using the Lattice Boltzmann Method. In *Proceedings of the Building Simulation 2011 Conference*, pages 933–940. IBPSA, 2011.

[G] Multi-GPU Implementation of a Hybrid Thermal Lattice Boltzmann Solver using the TheLMA Framework. *Computers and Fluids*, published online March 6, 2012.

[H] Efficient GPU Implementation of the Linearly Interpolated Bounce-Back Boundary Condition. *Computers and Mathematics with Applications*, published online June 28, 2012.

[I] Scalable Lattice Boltzmann Solvers for CUDA GPU Cluster. Submitted to *Parallel Computing*, August 22, 2012.

All the papers are authored by Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. They are hereinafter referred to as Art. A, Art. B, and so forth. The page numbering is specific to each article, e.g. B–1, B–2, etc.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Contents of the General Introduction

Introduction 1

I General purpose computing on graphics processors 3
- a. A brief history 3
- b. CUDA hardware and execution model 4
- c. Memory hierarchy and data transfer 5

II Principles of the lattice Boltzmann method 7
- a. Origins of the lattice Boltzmann method 7
- b. Isothermal fluid flow simulation 8
- c. Algorithmic aspects 9

III GPU implementations of the LBM 12
- a. Early implementations 12
- b. Direct propagation schemes 14
- c. Global memory access modelling 15

IV Extensions to the lattice Boltzmann method 17
- a. Hybrid thermal lattice Boltzmann method 17
- b. Large eddy simulations 18
- c. Interpolated bounce-back boundary conditions 19

V Large scale lattice Boltzmann simulations 22
- a. The TheLMA framework 22
- b. Multi-GPU implementations 23
- c. GPU cluster implementation 25

VI Applications and perspectives 27

Bibliography 32

Extended abstract (in French) 33

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

General introduction

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

D.E. Knuth — Computer Programming as an Art, 1974

IS ACCURATE building performance simulation possible? Taking into account the tremendous computational power of modern computers, one might be tempted to give an immediate positive answer to this question. Yet, buildings are complex systems interacting in numerous ways with their environment, at various time and length scales, and accurate simulations often appear to be of prohibitive computational cost.

The design of low or zero energy buildings even increases the need for accurate modelling of the heat and mass transfer between the outdoor or indoor environment and the envelope of a building. The commonplace practice regarding building aeraulics is to use simplified empirical or semi-empirical formulae. However, as outlined in [4], such approaches only provide a crude indication of the relevant parameters.

Regarding indoor simulations, nodal [6] and zonal [22] models are also widespread. In the former approach, the air in a given room is represented by a single node; in the latter, the room is divided into macroscopic cells. These methods, which require low computational efforts but simulate large volumes of air using single nodes, often lead to unsatisfactory results when compared to experimental data [9].

To achieve adequate accuracy, the use of computational fluid dynamics (CFD) in building aeraulics appears therefore to be mandatory. Yet, the computational cost and memory requirements of CFD simulations are often so high that they may not be carried out on personal computers. Since the access to high performance computing (HPC) facilities is limited and expensive, alternative approaches improving the performance of CFD solvers are of major practical interest in numerous engineering fields.

The present work explores the potential of graphics processing units (GPUs) to perform CFD simulations using the lattice Boltzmann method (LBM). This method is a rather recent approach for fluid flow simulation which is well-suited to HPC implementations as we shall see later. The investigations led to the design and creation of a framework going by the name of TheLMA, which stands for "Thermal LBM on Many-core Architectures". Several LBM solvers based on this framework were developed, ranging from single-GPU to GPU

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

cluster implementations, and addressing issues such as thermal flow simulation, turbulent flow simulation, or complex geometries representation.

The remainder of this introduction is organised as follows. Section I is an overview of general purpose computation technologies for the GPU. In section II, a description of the principles and algorithmic aspects of the lattice Boltzmann method is given. Section III focuses on single-GPU implementations of the LBM. Section IV outlines several extensions to the LBM, regarding thermal simulations, high Reynolds number flows and representation of complex geometries. Section V focuses on multi-GPU implementations of the LBM, either single-node or multi-node. Section VI concludes, giving a summary of the potential applications of the TheLMA framework in building simulation, and some perspective on the remaining issues.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

I General purpose computing on graphics processors

a. A brief history

Graphics accelerators are meant to alleviate the CPU's workload in graphics rendering. These specialised electronic circuits became commonplace hardware in consumer-level computers during the 1990s. In these early years, graphics accelerators were chiefly rasterisation devices using fixed-function logic, and had therefore limited interest from a computing standpoint, although some attempts were made [21]. The first electronic circuit marketed as "graphics processing unit", was Nvidia's GeForce 256 released in 1999, which introduced dedicated hardware to process transform and lighting operations, but was still not programmable.

In 2001, Nvidia implemented a novel architecture in the GeForce 3, based on programmable shading units. With this technology, the graphics pipeline incorporates *shaders*, which are programs responsible for processing vertices and pixels according to the properties of the scene to render. From then on, GPUs could be regarded as *single instruction multiple data* (SIMD) parallel processors. At first, shaders had to be written in hardware-specific assembly language; nonetheless the new versatility of GPUs increased their potential for numerical computations [41].

The two leading 3D graphics APIs both introduced a high level shading language in 2002: HLSL for Microsoft's Direct3D, which is also known as Nvidia Cg [30], and GLSL for OpenGL. Both languages use a C-like syntax and a stream-oriented programming paradigm: a series of operations (the *kernel*) is applied to each element of a set of data (the *stream*). High level languages contributed to significantly enlarge the repertoire of general purpose computing on GPU (GPGPU) [35]. In many situations, because of their massively parallel architecture, GPUs would outperform CPUs. Nevertheless, GPGPU development remained cumbersome because of the graphics processing orientation of both Cg and GLSL. It further evolved with the release of BrookGPU, which provides a run-time environment and compiler for Brook, a general purpose stream processing language [7].

During the last decade, because of several issues such as the "frequency wall", the computational power of CPUs per core grew modestly, to say the least. Meanwhile, the improvements of the CMOS fabrication process made possible to constantly increase the number of shading units per GPU, and therefore, to increase the raw computational power. This situation led the Nvidia company to consider the HPC market as a new target and to develop a novel technology labeled "Compute Unified Device Architecture" (CUDA) [24].

The advent of the CUDA technology in 2007 is probably the most significant breakthrough in the GPGPU field up to now. The core concepts consist of a set of abstract hardware specifications, together with a parallel programming model. CUDA often refers to the "C for CUDA" language, which is an extension

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

to the C/C++ language (with some restrictions) based on this programming model. Compared to previous technologies, CUDA provides unprecedented flexibility in GPGPU software development, which contributed to bring GPU computing to the forefront of HPC.

The proprietary status of the CUDA technology is a major drawback: only Nvidia GPUs are capable of running CUDA programs. The OpenCL framework [17], first released in 2008, provides a more generic approach, since it is designed for heterogeneous platforms. OpenCL is supported by most GPU vendors today. It is worth noting that the OpenCL programming model and the OpenCL language share many similarities with their CUDA counterparts. From an HPC standpoint, OpenCL is a promising standard which might override CUDA in future.

The present work focuses on CUDA platforms, mainly because, at the time it began, the CUDA technology was by far more mature than OpenCL. However, it should be mentioned that, since CUDA and OpenCL are closely related, several contributions of this research work might also be valuable on OpenCL platforms.

b. CUDA hardware and execution model

The CUDA hardware model consists of abstract specifications that apply to each Nvidia GPU architecture since the G80. The differences in features between GPU generations is represented by the "compute capability" which consist of a major and a minor version number. The GT200 GPU featured in the Tesla C1060 computing board, which was used for most of the present work, has compute capability 1.3. The GF100 GPU, also known as "Fermi", has compute capability 2.0.

A CUDA capable GPU is described in the hardware model as a set of "streaming multiprocessors" (SMs). An SM consists merely of an instruction unit, several "scalar processors" (SPs) (namely 8 for the GT200 and 32 for the GF100), a register file partitioned among SPs, and shared memory. Both register memory and shared memory are rather scarce, the later being for instance limited to 64 KB with the GF100. In the CUDA terminology, the off-chip memory associated to the GPU is called "device memory". In order to reduce latency when accessing to device memory, SMs contain several caches: textures, constants, and—as of compute capability 2.0—data.

The CUDA execution model, coined by Nvidia as "single instruction multiple threads" (SIMT), is rather complex because of the dual level hardware hierarchy. The SMs belong to the SIMD category, although they do not reduce to pure vector processors, being e.g. able to access scattered data. At global level, execution is better described as *single program multiple data* (SPMD) since SMs are not synchronised.[1] It should also be mentioned that, whereas efficient com-

[1] Starting with the Fermi generation, the execution at GPU level might even be described as *multiple program multiple data* (MPMD) since several kernels can be run concurrently.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

munication within an SM is possible through on-chip shared memory, sharing information at global level requires the use of device memory and may therefore suffer from high latency.

Unlike streaming languages, which are data centred, the CUDA programming paradigm is task centred. An elementary process is designated as a *thread*, the corresponding series of operations is named *kernel*. To execute a CUDA kernel, it is mandatory to specify an "execution grid". A *grid* is a multidimensional array of identical *blocks*, which are multidimensional arrays of threads.[2] This dual level scheme is induced by the architecture: a block is to be processed by a single SM; consequently, the number of threads a block can hold is limited by the local resources.

Depending on the available resources, an SM may run several blocks concurrently. For computationally intensive application, it might be of great interest to tailor the dimensions of the blocks in order to achieve high arithmetic intensity, i.e. for the SMs, to host as many active threads as possible. It is worth mentioning that grid dimensions are less constraint than block dimensions. As described in [8], blocks are processed asynchronously in batches. A grid may therefore contain far more blocks than a GPU is actually able to run concurrently.

Because of the SIMD nature of an SM, the threads are not run individually but in groups named *warps*. Up to now, a warp contains 32 threads; yet, this value is implementation dependent and might change for future hardware generations. The warp being an atomic unit, it is good practice to ensure that the total number of threads in a block is a multiple of the warp size. Warps induce several limitations such as in conditional branching, for instance: when branch divergence occurs within a warp, the processing of the branches is serialised. Whenever possible, branch granularity should be greater than the warp size. In many situations, the design of an execution grid is not a trivial task, and might be of cardinal importance to achieve satisfactory performance.

c. Memory hierarchy and data transfer

As outlined in the preceding subsection, the CUDA memory hierarchy is fairly complex from an architectural standpoint, but it is even more intricate from a programming standpoint. In CUDA C, a kernel is merely a void-valued function. Grid and block dimensions as well as block and thread indices are accessible within the kernel using built-in read-only variables. CUDA memory spaces fall into five categories: local, shared, global, constant, and textures.

Automatic variables in a kernel, which are proper to each thread, are stored in registers whenever possible. Arrays and structures that would consume too much registers, as well as arrays which are accessed using unpredictable in-

[2] As of compute capability 2.0, grids and blocks may have up to three dimensions, prior to this, grids were limited to two dimensions.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

dices, are stored in local memory.[3] Local memory, which is hosted in device memory, is also used to spill registers if need be. Prior to the Fermi generation, local memory was not cached. Greatest care had therefore to be taken in kernel design in order to avoid register shortage.

Threads may also access to shared and global memory. The scope and lifetime of shared memory is limited to the local block. Being on chip, it may be as fast as registers, provided no bank conflicts occurs.[4] Global memory is visible by all threads and is persistent during the lifetime of the application. It resides in device memory which is the only accessible by the host system. Transferring from and to global memory is the usual way for an application to provide data to a kernel before launch and to retrieve results once kernel execution has completed.

Constant and shared memory are additional memory spaces hosted in device memory, which may be accessed read-only by the threads. Both are modifiable by the host. The former is useful to store parameters that remain valid across the lifetime of the application; the later is of little use in GPGPU and shall not be discussed further.

With devices of compute capability 1.3, global memory is not cached. Transactions are carried out on aligned segments[5] of either 32, 64, or 128 bytes. At SM level, global memory requests are issued by half-warps and are serviced in as few segment transactions as possible. Devices of compute capability 2.0 feature L1 and L2 caches for global memory. L1 is local to each SM and may be disabled at compile time, whereas L2 is visible to all SMs. Cache lines are mapped to 128 bytes aligned segments in device memory. Memory accesses are serviced with 128-byte memory transactions when both caches are activated and with 32-byte memory transactions when L1 is disabled, which may reduce over-fetch in case of scattered requests.

It is worth stressing that for both architectures, although being significantly different, global memory bandwidth is best used when consecutive threads access to consecutive memory locations. Data layout is therefore an important optimisation issue when designing data-intensive application for CUDA.

[3] Registers being not addressable, an array may only be stored in the register file when its addressing is known at compile time.

[4] The shared memory is partitioned in several memory banks. A bank conflict occurs when several threads try to access concurrently to the same bank. To resolve the conflict, the transactions must be serialised.

[5] An *aligned* segment is a block of memory whose start address is a multiple of its size.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

II Principles of the lattice Boltzmann method

a. Origins of the lattice Boltzmann method

The path followed by usual approaches in CFD may be sketched as "top-down": a set of non-linear partial differential equations is discretised and solved using some numerical technique such as finite differences, finite volumes, finite elements, or spectral methods. As pointed out in [51], attention is often drawn on truncation errors induced by the discretisation. However, from a physical standpoint, the fact that the desired conservation properties still hold for the discretised equations is of major importance, especially for long-term simulations in which small variations may by accumulation lead to physically unsound results. The preservation of these conservation properties is generally not guaranteed by the discretisation method.

Alternatives to the former methods may be described as "bottom-up" approaches, molecular dynamics (MD) being the most emblematic one. In MD, the macroscopic behaviour of fluids is obtained by simulating as accurately as possible the behaviour of individual molecules. Because of the tremendous computational effort required by even small scale MD simulations, the scope of applications is limited. Lattice-gas automata (LGA) and LBM fall into the same bottom-up category than MD although they act at a less microscopic scale, which is often referred to as *mesoscopic*.

LGA are a class of cellular automata attempting to describe hydrodynamics through the discrete motion and collision of fictitious particles. In LGA, space is represented by a regular lattice in which particles hop from one node to another at each time step. Most LGA models obey to an *exclusion principle*, which allows only one particle for each lattice direction to enter a node at a time. A node may therefore only adopt a finite number of states, which are usually represented by a bit field. The simulation process consist of an alternation of *collisions*, in which the state of each node determines the local set of out-going particles, and *propagation*, during which the out-going particles are advected to the appropriate neighbour nodes.

The first two-dimensional LGA model was proposed by Hardy, Pomeau, and de Pazzis in 1973 [18]. It is known as HPP after its inventors. HPP operates on a square lattice with collision rules conserving mass and momentum. Yet, HPP lacks rotational invariance and therefore cannot yield the Navier-Stokes equations in the macroscopic limit, which drastically reduces its practical interest. In the two-dimensional case, this goal was first achieved with the FHP model introduced in 1986 by Frisch, Hasslacher, and Pomeau [15]. Increased isotropy is obtained by using an hexagonal grid instead of a square lattice.

In the three-dimensional case, achieving sufficient isotropy is by far more difficult than with two dimensions, and this could only be solved by using a four-dimensional face-centred hypercube (FCHC) lattice [12]. FCHC yields 24-bit wide states which in turn causes the look-up tables of the collision rules to be

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

very large. This issue, beside the complexity induced by the fourth dimension, is a serious drawback for the use of 3D LGA models in practice.

Another general disease of LGA models is the statistical noise caused by the use of Boolean variables. To address this issue, McNamara and Zanetti in 1988 [31], proposed the first lattice Boltzmann model, replacing the Boolean fields by continuous distributions over the FHP and FCHC lattices. Later on, in 1992, Qian, d'Humières, and Lallemand [37], replaced the Fermi-Dirac by the Maxwell–Boltzmann equilibrium distribution and the collision operator inherited from the LGA models by a linearised operator based on the Bhatnagar–Gross–Krook (BGK) approximation [2]. This model, often referred to as LBGK (for lattice BGK), is still widely in use today.

A conclusion to this emergence period was brought by the work of He and Luo in 1997 [20]. The lattice Boltzmann equation (LBE), i.e. the governing equation of the LBM, may be derived directly from the continuous Boltzmann equation. From a theoretical standpoint, the LBM is therefore a standalone approach sharing similarities with LGA but no actual dependency.

b. Isothermal fluid flow simulation

The Boltzmann equation describes the hydrodynamic behaviour of a fluid by the means of a one-particle distribution function f over phase space, i.e. particle position $\boldsymbol{x}$ and velocity ξ, and time:

$$\partial_t f + \xi \cdot \nabla_{\boldsymbol{x}} f + \frac{\boldsymbol{F}}{m} \cdot \nabla_{\xi} f = \Omega(f). \tag{1}$$

In the former equation, m is the particle mass, $\boldsymbol{F}$ is an external force, and Ω denotes the collision operator. In a three-dimensional space, the macroscopic quantities describing the fluid obey:

$$\rho = \int f \, \mathrm{d}\xi \tag{2}$$

$$\rho \boldsymbol{u} = \int f \xi \mathrm{d}\xi \tag{3}$$

$$\rho \boldsymbol{u}^2 + 3\rho\theta = \int f \xi^2 \mathrm{d}\xi \tag{4}$$

where ρ is the fluid density, $\boldsymbol{u}$ the fluid velocity, and $\theta = k_{\mathrm{B}} T/m$ with T the absolute temperature and k_{B} the Boltzmann constant.

The LBM is based on a discretised form of Eq. 1 using a constant time step δt and a regular orthogonal lattice of mesh size δx. A finite set of velocities $\{\xi_\alpha \,|\, \alpha = 0, \ldots N\}$ with $\xi_0 = \mathbf{0}$, is substituted to the velocity space. The velocity set or *stencil*, is chosen in accordance with the time step and mesh size: given a lattice site $\boldsymbol{x}$, $\boldsymbol{x} + \delta t \xi_\alpha$ is on the lattice for any α. In the three-dimensional

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

case, three velocity sets are commonly considered (see Fig. 1). These stencils are named D3Q15, D3Q19, and D3Q27, following the notation proposed by Qian *et al.* in 1992.[6]

The discrete counterpart of the distribution function f is a set of particle populations $\{f_\alpha \,|\, \alpha = 0, \dots N\}$ corresponding to the velocities. With T being the transpose operator, let us denote: $|a_\alpha\rangle = (a_0, \dots a_N)^\mathsf{T}$. In the absence of external forces, Eq. 1 becomes:

$$|f_\alpha(\boldsymbol{x} + \delta t \boldsymbol{\xi}_\alpha, t + \delta t)\rangle - |f_\alpha(\boldsymbol{x}, t)\rangle = \Omega\left(|f_\alpha(\boldsymbol{x}, t)\rangle\right). \tag{5}$$

Regarding the macroscopic variables of the fluid, Eqs. 2 and 3 become:

$$\rho = \sum_\alpha f_\alpha, \tag{6}$$

$$\rho \boldsymbol{u} = \sum_\alpha f_\alpha \boldsymbol{\xi}_\alpha. \tag{7}$$

It is possible to show, as for instance in [29], that Eqs. 6 and 7 are exact quadratures. This demonstration uses Gauss-Hermite quadratures on the low Mach number second-order expansion of the Maxwellian equilibrium distribution function, which is sufficient to derive the Navier-Stokes equation. As a consequence, mass and momentum conservation are preserved by the numerical scheme. It should be mentioned that the Gaussian quadrature method does not yield energy conserving models and is therefore only suitable to build isothermal models.

Besides LBGK, which is also known as single-relaxation-time LBM, numerous alternative models have been proposed over the past years, e.g. multiple-relaxation-time (MRT) [10], entropic lattice Boltzmann [23], or regularised lattice Boltzmann [27]. The discussion of the advantages and drawbacks of the various approaches is beyond the scope of this introduction. It should nevertheless be mentioned that MRT collision operators are implemented in all but one solver described in the present collection of articles. MRT operators are explicit, like LBGK operators, and provide increased stability and accuracy at the cost of a slightly higher arithmetic complexity. An overview of MRT is to be found e.g. in Art. D, whereas a comprehensive presentation is given in [11].

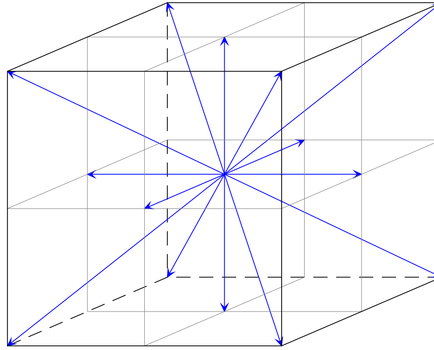
c. Algorithmic aspects

Like LGA, from an algorithmic standpoint, the LBM consists of an alternation of collision and propagation steps. The collision step is governed by:

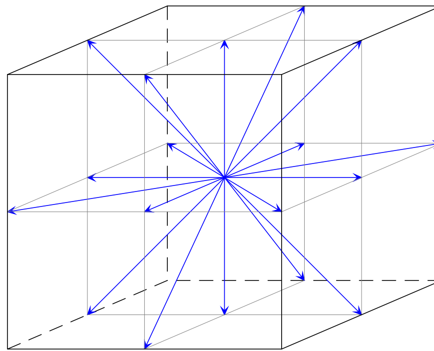
$$|\widetilde{f}_\alpha(\boldsymbol{x}, t)\rangle = |f_\alpha(\boldsymbol{x}, t)\rangle + \Omega\left(|f_\alpha(\boldsymbol{x}, t)\rangle\right), \tag{8}$$

[6] In the DmQn notation, m is the spatial dimension and n is the number of velocities including the rest velocity $\boldsymbol{\xi}_0$.

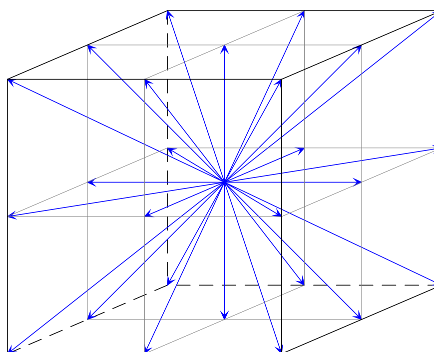
Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés



(a) D3Q15



(b) D3Q19



(c) D3Q27

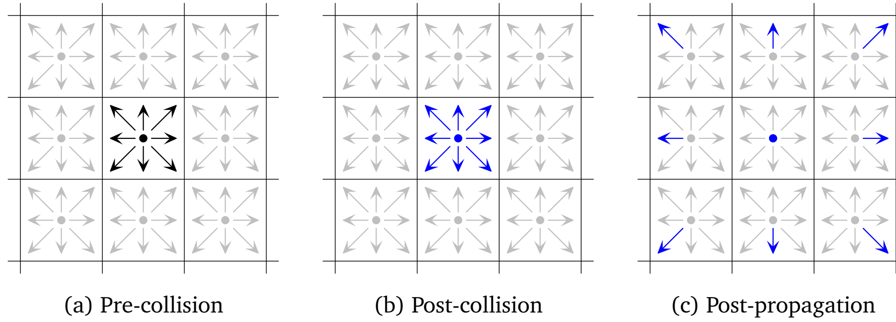
Figure 1: Usual velocity sets in 3D LBM. — *The D3Q27 stencil links a given node to its 26 nearest neighbours in the lattice, D3Q15 and D3Q19 are degraded versions of the former.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

where $\widetilde{f}_\alpha$ denotes the post-collision particle populations. It is worth noting that the collision step is purely local to each node. The propagation step is described by:

$$\left|f_\alpha(\boldsymbol{x} + \delta t\, \xi_\alpha, t + \delta t)\right\rangle = \left|\, \widetilde{f}_\alpha(\boldsymbol{x}, t)\right\rangle, \tag{9}$$

which reduces to mere data transfer. This two phase process is outlined for the two-dimensional D2Q9 stencil by Fig. 2.



(a) Pre-collision (b) Post-collision (c) Post-propagation

Figure 2: Collision and propagation. — *The nine pre-collision particle populations of the central cell are drawn in black, whereas the nine post-collision populations are drawn in blue. After collision, these populations are advected to the neighbouring nodes in accordance with the velocity set.*

The breaking of Eq. 5 makes the data-parallel nature of LBM obvious. The LBM is therefore well-suited for massively parallel implementations, provided the next-neighbours synchronisation constraint is taken care of. The simplest way to address this issue is to use two instances of the lattice, one for even time steps and the other for odd time steps. A now commonplace method, known as *grid compression* [50], makes possible to save almost half of the memory by overlaying the two lattices with a diagonal shift of one unit in each direction. Yet, this technique requires control over the schedule of node updates in order to enforce data dependency. It does therefore not apply in an asynchronous execution model.

In a shared memory environment, when using non-overlaid lattices, the data layout for 3D LBM simulations often reduces to a five-dimensional array: three dimensions for space, one for velocity indices, and one for time. Depending on the target architecture and memory hierarchy, the ordering of the array may have major impact on performance. The design of the data layout is therefore a key optimisation phase for HPC implementations of LBM.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

III GPU implementations of the LBM

a. Early implementations

As mentioned in the previous section, data-parallel applications such as the LBM are usually well-suited for massively parallel hardware. Attempts to implement the LBM for the GPU were made in the very early days of GPGPU, starting with the contribution of Li *et al.* in 2003 [28]. At that time, because of the unavailability of a general purpose programming framework, the particle distribution had to be stored as a stack of two-dimensional texture arrays. With this approach, the LBE needs to be translated into rendering operations of the texturing units and the frame buffer, which is of course extremely awkward and hardware dependent. Beside issues such as the rather low accuracy of the computations, the authors of [28] had also to face the limited amount of on-board memory which only made possible very coarse simulations.[7] This work was later extended by Fan *et al.* in 2004 [14], which describes the first GPU cluster implementation of the LBM. The authors report a simulation on a $480 \times 400 \times 80$ lattice with 30 GPUs, demonstrating the practical interest of GPU LBM solvers.

In 2008, Tölke reported the first CUDA implementation of 2D LBM [43], and later the same year, with Krafczyk, the first CUDA implementation of 3D LBM [44]. The general implementation principles are the same in both works and remain, for the most part, valid until today. The authors focus on minimising the cost of data transfer between GPU and global memory, since the target architecture, i.e. the G80, like the later GF100 does not provide cache for global memory. The following list gives an outline of these principles:

1. Fuse collision and propagation in a single kernel to avoid unnecessary data transfer.
2. Map the CUDA execution grid to the lattice, i.e. assign one thread to each lattice node. This approach, by creating a large number of threads, is likely to take advantage of the massive parallelism of the GPU and to hide the latency of the global memory.
3. Use an appropriate data layout in order for the global memory transactions issued by the warps to be coalesced whenever possible.
4. Launch the collision and propagation kernel at each time step and use two instances of the lattice in order to enforce local synchronisation.

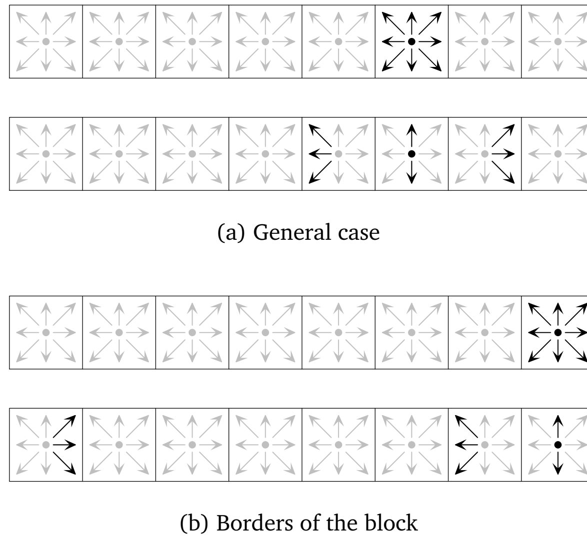
The implementations described in [43] and [44] use one-dimensional blocks of threads and specific two- or three-dimensional arrays for each velocity index. It should be noted that, because of the constraints on the block size, the size

[7] The Nvidia GeForce4 used for the implementation has only 128 MB of memory.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

of the lattice along the blocks' direction is better chosen to be a multiple of the warp size. Such a set-up allows the memory accesses to be coalesced when fetching the pre-collision distributions. Yet, special care is taken for the propagation step. Propagation reduces to data shifts along the spatial directions, including the minor dimension of the distribution arrays. Thus, directly writing back to global memory leads to memory transactions on non-aligned segments. With the G80, this would have a dramatic impact on performance, since in this case memory accesses are serviced individually.

To address the former issue, Tölke proposes to store the particle distribution within a block in shared memory and to break propagation in two steps. At first, a partial propagation is performed along the blocks' direction within shared memory, then propagation is completed while storing the distribution back to global memory. Fig. 3 outlines the method (using eight nodes wide blocks for the sake of clarity). By suppressing shifts along the minor dimension of the distribution arrays, this approach fully eliminates misalignments for global memory write accesses during propagation. However, when the blocks do not span the entire width of the domain, which may occur at least with large two-dimensional simulations, nodes located at the borders of the blocks but not on the domain's boundary need special handling: the particle populations leaving the block are temporarily stored in the unused places at the opposite border, as illustrated in Fig. 3b. After the execution of the main kernel, a second kernel is then needed to reorder the distribution arrays.



(a) General case

(b) Borders of the block

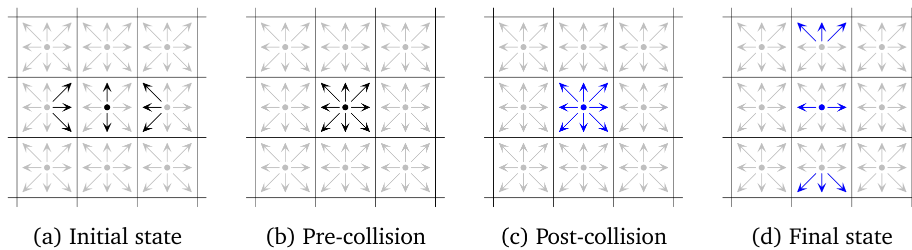
Figure 3: Partial propagation in shared memory. — *In the general case, the relevant particle populations are simply advected along the blocks' direction. For the border nodes, the populations leaving the block are stored in the unused array cells corresponding to the populations entering the block at the opposite border.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

b. Direct propagation schemes

Implementing the LBM for CUDA devices of compute capability 1.0, as the aforementioned G80, is quite a challenging task because of the constraints induced by hardware limitations on data transfer. The work of Tölke and Krafczyk achieves this goal in a rather efficient way since the authors report up to 61% of the maximum sustained throughput in 3D. Transition from compute capability 1.0 to 1.3 significantly decreased the cost of misaligned memory transactions. For 32-bit word accesses, e.g. single precision floating point numbers, a stride of one word yields an additional 32 B transaction by half-warp, i.e. a 50% increase of transferred data. It seems therefore reasonable to consider direct propagation schemes as alternatives to the shared memory approach. The results of our experiments are reported in Art. A.

Basic investigations on a GeForce GTX 295 led us to the conclusion that misaligned memory transactions still have noticeable influence on data transfer for CUDA devices of compute capability 1.3, but that misaligned read accesses are significantly less expensive than misaligned write accesses. We therefore experimented two alternatives to the elementary out-of-place propagation scheme. The first one, which we named *split scheme*, consist in breaking the propagation in two half-steps as illustrated in Fig. 4. After collision, the updated particle distributions are advected in global memory in all directions except along the minor dimension. The remainder of the propagation is carried out before collision at the next time step. With such a procedure, misalignement occurs only when loading data.



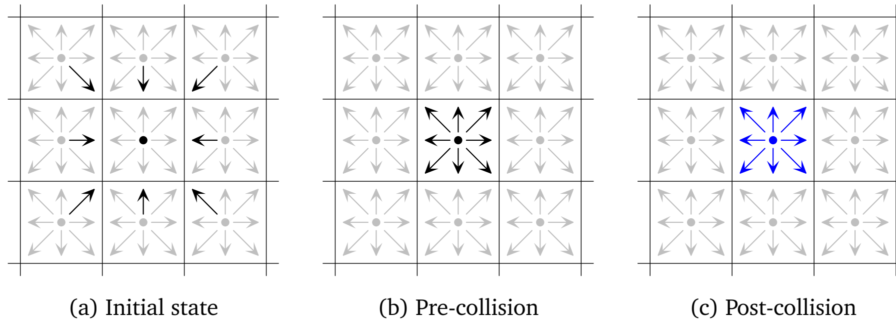
(a) Initial state (b) Pre-collision (c) Post-collision (d) Final state

Figure 4: Split propagation scheme. — *The data transfer corresponding to the propagation along the minor dimension is represented by the transition between (a) and (b) whereas the second propagation half-step is represented by the transition between (c) and (d).*

The second alternative propagation scheme is the *in-place scheme*, also referred to as *reversed scheme* in Art. A and B. As shown in Fig. 5, the in-place scheme consists in reversing collision and propagation. At each time step, the kernel performs propagation while gathering the local particle distribution with the appropriate strides. After collision, the updated particle distribution is stored back in global memory without any shift. As for the split scheme, the

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

in-place scheme avoids any misaligned memory transaction when writing data to global memory.



(a) Initial state (b) Pre-collision (c) Post-collision

Figure 5: In-place propagation scheme. — *Post-collision particle populations from the former time step are gathered to the current node. Collision is then carried out and the new particle distribution is written back to global memory.*

In Art. A, we describe two implementations of 3D LBM; the first one is based on the split scheme and the LBGK collision operator, and the second one is based on the in-place scheme and the MRT collision operator. Both solvers show similar performance with about 500 million lattice node updates per second (MLUPS) in single precision, the data throughput being over 80% of the maximum sustained throughput. Hence, communication appears to be the limiting factor as for the implementation of Tölke and Krafczyk. For 3D simulations, provided no auxiliary kernel is needed, the shared memory approach is likely to provide even better performance than the split propagation or the in-place propagation approach. However, these direct propagation schemes are of genuine practical interest, since they lead to significantly simpler code and exert less pressure on hardware, leaving the shared memory free for additional computations as we shall see in the case of thermal simulations.

The performance obtained by CUDA LBM solvers (in single precision) using a single GPU is more than one order of magnitude higher than the performance (in double precision) reported for HPC systems such as the NEC SX6+ or the Cray X1 [49]. Considering furthermore the moderate cost of GPU based computing devices makes obvious the great potential of GPU LBM solvers for realistic engineering applications.

c. Global memory access modelling

CUDA implementations of the LBM tend to be communication-bound. Although some information is given in the CUDA programming guide [33], most of the data transfer mechanisms between GPU and device memory remain undocumented. Art. B reports the investigations we undertook to gain a better understanding of these aspects. Our study aims at devising possible optimisation

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

strategies for future CUDA LBM implementations, and at providing a performance model suitable for a wide range of LBM based simulation programs.

The GT200 GPU we used for our benchmarks is divided into ten texture processing clusters (TPCs), each containing three SMs. Within a TPC, the SMs are synchronised and the corresponding hardware counter is accessible via the CUDA `clock()` function. We implemented a program able to generate benchmark kernels mimicking the data transfer performed by an actual LBM kernel. At start, the benchmark kernel loads a given number N of registers from global memory which are afterwards stored back. Several parameters such as the number of misaligned load or store transactions, or the number k of warps per SM are tunable. The generated code incorporates appropriate data dependencies in order to avoid aggressive compiler optimisations. The `clock()` function is used to measure the durations of both the loading phase and the storing phase, making possible to evaluate the actual throughput in both directions.

The results reported in Art. B show that the behaviour of the memory interface is not uniform, depending on whether $N \leq 20$ or not. In the first case, we estimate the time to process k warps per SM with $T = \ell + T_R + T_W$, where ℓ is the launch time of the last warp, T_R is the average read time, and T_W is the average write time. We show that ℓ only depends on k, and that, for a given number of misalignments, T_R and T_W depend linearly on N. The knowledge of T for the appropriate parameters, makes possible to evaluate the expected performance P (in MLUPS) of an LBM kernel: $P = (K/T) \times F$, where K is the global number of active threads and F is the GPU frequency in MHz. Comparing this performance model to actual performance values is useful to evaluate the opportunity of additional optimisations.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

IV Extensions to the lattice Boltzmann method

a. Hybrid thermal lattice Boltzmann method

Although isothermal fluid flow solvers are of practical interest in building aeraulics, it is often desirable to take thermal effects into account. Art. C and G report our endeavour to implement thermal LBM solvers for the GPU. Usual lattice Boltzmann models such as the D3Q19 MRT fail to conserve energy and numerous attempts to apply the LBM to thermo-hydrodynamics are to be found in literature. Beside others, one should mention multi-speed models [36], using larger velocity sets in order to enforce energy conservation, double-population models [19], using an energy distribution in addition to the particle distribution, or hybrid models, in which the energy equation is solved by other means such as finite differences. As shown in [26], both multi-speed models and double-population models suffer from inherent numerical instabilities. Moreover, from an algorithmic standpoint, both approaches significantly increase the requirements in global memory as well as the volume of transfered data which has a direct impact on performance for communication-bound applications such as GPU LBM solvers.

We therefore chose to implement the hybrid thermal lattice Boltzmann model described in [26]. The hydrodynamic part is solved using a slightly modified MRT model in which a temperature coupling term is added to the equilibrium of the internal energy moment. The temperature T is solved using a finite difference equation. In the case where the ratio of specific heats $\gamma = C_P/C_V$ is set to $\gamma = 1$, this equation may be written as:

$$\partial_t^* T = \kappa \Delta^* T - \boldsymbol{j} \cdot \nabla^* T \tag{10}$$

where $\boldsymbol{j}$ is the momentum, κ is the thermal diffusivity, and where ∂_t^*, Δ^*, and ∇^* denote the finite difference operators, the two later using the same stencil as the associated lattice Boltzmann equation.

The single-GPU implementation of this model, which is described in Art. C, is derived from our single-GPU D3Q19 MRT solver. We use a two-dimensional execution grid of one-dimensional blocks spanning the simulation domain in the x-direction. Processing Eq. 10 for a given node requires to have access to the nineteen temperatures corresponding to the D3Q19 stencil, but leads to only one additional store operation for the local temperature. In order to reduce read redundancy, the kernel fetches at start the temperatures of all the nodes neighbouring the current block into shared memory. As illustrated by Fig. 6, each thread is responsible for reading the temperatures of the nodes sharing the same abscissa. By reducing the amount of additional reads by more than one half, this approach leads to rather satisfying performance with around 300 MLUPS in single precision using a single Tesla C1060 computing device.

The single-GPU solver described in Art. C and the multi-GPU solver described in Art. G were both tested using the differentially heated cubic cavity.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

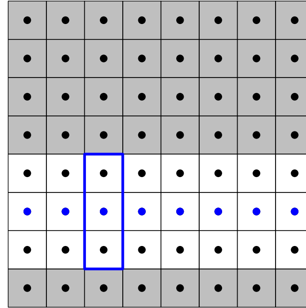


Figure 6: Read access pattern for temperature. — *The nodes associated to the current block are represented by blue dots. The white background cells stand for the nodes whose temperature is needed to solve the finite difference equation. The blue frame represents the zone assigned to a thread for temperature fetching.*

For validation purpose, we computed the Nusselt numbers at the isothermal walls. The obtained values are in good agreement with previously published results [42, 46]. Using the multi-GPU version, we could perform simulations for Rayleigh numbers up to 10^9.

b. Large eddy simulations

Multi-GPU LBM solvers, such as the one described in Art. D and E, make possible to perform fluid simulations on very large computation domains. When using a Tyan B7015 server equipped with eight Tesla C1060, each providing 4 GB of memory, a cubic computation domain may be as large as 576^3 in single precision, i.e. contain more than 190 million nodes. Such a fine resolution allows the Reynolds number of the simulation to be of the order of 10^3 to 10^4. In external building aeraulics, however, the typical Reynolds numbers are of the order of 10^6. Because of turbulence phenomena, performing direct numerical simulation for such applications appears therefore to be out of reach at the present time.

We see that, to be of practical interest in external building aeraulics, LBM solvers need to incorporate a sub-grid scale model such as large eddy simulation (LES). As a first attempt, we chose to implement the most elementary LES model, proposed by Smagorinsky in 1963 [40]. The simulation results we obtained using this extended version of our multi-GPU isothermal LBM solver are reported in Art. F. In the Smagorinsky model, a turbulent viscosity ν_t is added to the molecular viscosity ν_0 to obtain the kinematic viscosity ν of the simulation: $\nu = \nu_0 + \nu_t$. The turbulent viscosity is given by:

$$\nu_t = |\mathsf{S}|\,(C_S\delta x)^2, \quad |\mathsf{S}| = \sqrt{2\,\mathsf{S}:\mathsf{S}}, \tag{11}$$

where C_S is the Smagorinsky constant, which is usually set to $C_S = 0.1$, and S is

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

the strain rate tensor. As shown in [25], the MRT approach has the interesting feature that the strain rate tensor can be determined directly from the moments computed when applying the collision operator. The computations being fully local, the impact on the performance of a communication-bound program such as our fluid flow solver is negligible.

In Art. F, we report the simulation of the flow around a group of nine wall-mounted cubes at Reynolds number $\mathrm{Re} = 10^6$. In order to obtain relevant time-averaged pressure and velocity fields, the computations were carried out for a duration of $200T_0$, where T_0 is turn-over time corresponding to the obstacle size and the inflow velocity. These results could not be validated against experiments, due to the lack of data. However, the overall computation time being less than eighteen hours, this study demonstrates the feasibility of simulating the flow around a small group of buildings in reasonable time.

The original Smagorinsky LES model is a rather simplistic turbulence modelling approach with well-known defects such as the inability to fulfil the wall boundary law [32]. More elaborate methods such as the wall-adapting local eddy-viscosity (WALE) model proposed in [32] were reported to be suitable for the LBM [48]. Further investigations seem therefore to be required before considering using our LBM solver for actual building aeraulics simulations.

c. Interpolated bounce-back boundary conditions

With the LBM, fluid-solid boundaries result in unknown particle populations. Considering a boundary node, i.e. a fluid node next to at least one solid node, it is easily seen that the populations, which should be advected from the solid neighbouring nodes, are undefined. A common way to address this issue is the simple bounce-back (SBB) boundary condition, which consists in replacing the unknown populations with the post-collision values at the former time step for the opposite directions. Let $\boldsymbol{x}$ denote a boundary node, the SBB obeys:

$$f_{\bar{\alpha}}(\boldsymbol{x}, t) = \widetilde{f}_{\alpha}(\boldsymbol{x}, t - \delta t) \tag{12}$$

where $f_{\bar{\alpha}}(\boldsymbol{x}, t)$ is an unknown particle population and $\bar{\alpha}$ is the direction opposite to α. As shown in [16], SBB is second-order accurate in space. The fluid-solid interface is located about half-way between the boundary node and the solid nodes.

One sees from Eq. 12 that applying SBB to a boundary node is straightforward, provided the list of solid neighbours is known. Moreover, it should be noted that the SBB is well-suited not only for straight walls but for any stationary obstacle. Because of its versatility and simplicity, we chose SBB for most of our LBM implementations. We generally use an array of bit-fields to describe the neighbourhood of the nodes[8]. This array is set up at start by a specific

[8] In the case of an empty cavity, the neighbourhood bit-fields are simply determined from the coordinates in order to avoid global memory accesses.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

kernel, according to the simulation layout. This approach allowed us to easily implement new test cases, even when complex geometries are involved as in the nine wall-mounted cubes simulation described in Art. F.

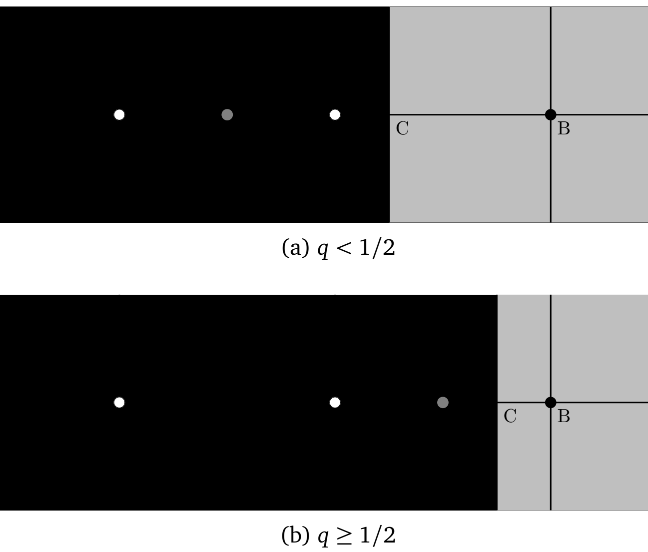
Since the fluid-solid interface has a fixed location, the use of the SBB is convincing as long as the considered bodies have flat faces parallel to the spatial directions. A curved boundary or an inclined straight wall leads to stepping effects that may have an impact on the simulated flow. In order to study these aspects, we chose to implement an extension to the SBB known as linearly interpolated bounce-back (LIBB) which takes into account the exact location of the fluid-solid interface [5].

As illustrated in Fig. 7, the LIBB is based on the determination of a fictitious particle population entering the boundary node. Two different interpolation formulae are used, depending on the location of the interface. Keeping the same notations as in Eq. 12, let q be the number such that $\boldsymbol{x} + q\delta t \boldsymbol{\xi}_\alpha$ is on the fluid-solid interface. For $q < 1/2$,

$$f_{\bar{\alpha}}(\boldsymbol{x}, t) = (1 - 2q) f_\alpha(\boldsymbol{x}, t) + 2q \widetilde{f}_\alpha(\boldsymbol{x}, t - \delta t) \tag{13}$$

and for $q \geq 1/2$,

$$f_{\bar{\alpha}}(\boldsymbol{x}, t) = \left(1 - \frac{1}{2q}\right) \widetilde{f}_{\bar{\alpha}}(\boldsymbol{x}, t - \delta t) + \frac{1}{2q} \widetilde{f}_\alpha(\boldsymbol{x}, t - \delta t). \tag{14}$$



(a) $q < 1/2$

(b) $q \geq 1/2$

Figure 7: Interpolation schemes of the LIBB boundary condition. — *In the first case, the fictitious particle population leaving D and entering A is interpolated from the post-collision values at E and A. In the second case, the particle population leaving A ends up at D. The population entering A is interpolated from the post-propagation values at E and D.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

The LIBB is interesting from an algorithmic standpoint since it can take advantage from the in-place propagation scheme, thus only slightly increasing the volume of communications. The details of our implementation strategy are given in Art. H. This work also reports the results obtained when simulating the flow past a sphere using either SBB or LIBB. Comparing the Strouhal numbers computed from our simulations to experimental results [39, 34] shows that LIBB improves the stability as well as the accuracy of the vortex shedding frequency. However, this conclusion does not seem to hold in general. In further investigations, yet unpublished, we simulated the flow past an inclined flat plate. For this test case, we could not see significant differences in the obtained Strouhal numbers.

The process of validating our approach in the perspective of realistic engineering applications is still at an early stage. Regarding LIBB, additional studies focusing on possibly more relevant parameters such as the drag coefficient should be carried out. Moreover, a wide variety of alternative boundary conditions for the LBM is to be found in literature, some of which being well-suited for efficient GPU implementations,could also be tested. As shown in Art. F and H, GPU LBM solvers make large scale validation studies possible, and thus may contribute to evaluate novel modelling strategies.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

V Large scale lattice Boltzmann simulations

a. The TheLMA framework

CUDA, being a recent technology, imposes strong constraints to the programmer, mainly induced by hardware limitations. Although this situation has evolved with the increasing capabilities of the different hardware generations, as well as the progresses made in the compilation tool-chain, some of the most acknowledged practices of software engineering, such as library-oriented development, are still not relevant. However, our research work led us to develop multiple versions of our LBM solver, implementing a wide variety of models and simulation layouts, and targeted for several different hardware configurations. We therefore had to devise an appropriate strategy to enforce both code reusability and maintainability, which resulted in the design and creation of the TheLMA framework.

A CUDA program usually consists of a set of CUDA C source files together with some additional plain C or C++ files. Although it is possible to build a program from sole CUDA C files, it is in general good practice to split the GPU related functions from the remainder of the code. In addition to kernels, a CUDA C source file may contain *device* functions and *host* functions. The first category corresponds to functions that are run by the GPU and may only be called by CUDA threads, i.e. from within a kernel or another device function. In practice, most device functions are inlined at compile-time, which restricts their use to short auxiliary functions[9]. The second category corresponds to functions that are run by the CPU and may be called by external C or C++ function as well as launch kernels using a specific syntax to stipulate the execution grid. Host device thus provide a convenient way to launch GPU computations from an external plain C module[10].

Up to now, the most severe limitation of the CUDA compilation tool-chain is the absence of an actual linker[11]. Consequently, all compilation symbols (e.g. device functions, device constants…) related to a kernel must lie in the same compilation unit. A commonplace way to achieve some degree of modularity consist in using inclusion directives in order to merge separate CUDA source files into a single one before compilation.

The overall structure of the TheLMA framework is outlined in Fig. 8. It consists in a set of data structure definitions, C and CUDA C source files. The C files provide a collection of functions useful to process the configuration parameters, initialise the global data structures, post-process the simulation results,

[9] As of compute capability 2.0, CUDA devices are able to perform actual function calls. However, inlining is still the default behaviour.

[10] An alternative way to launch a kernel from an external module consist in using the CUDA device API, which shares many similarities with the OpenCL API. This approach gives more control over the low-level details but leads to significantly more complex codes.

[11] The CUDA 5.0 software development kit, still unreleased at the time of this writing, should provide such a feature.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

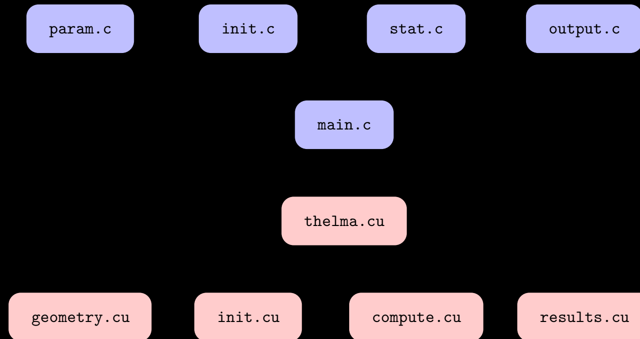


Figure 8: Structure of the TheLMA framework. — *The plain arrows symbolise the call dependencies, whereas the dashed arrows represent the inclusion operations performed by the preprocessor in order to provide a single file to the CUDA compiler.*

export data in various formats, or produce graphical outputs. The `thelma.cu` file contains some general macro definitions and device functions as well as the directives to include the other CUDA files. Each of them contains a specific kernel with the appropriate launching function. The TheLMA framework has been thought to confine code modifications to definite places in the source tree when implementing new models or new simulation layouts, thus increasing the development efficiency.

b. Multi-GPU implementations

The memory shipped with CUDA computing devices reaches up to 6 GB on recent hardware such as the Tesla C2075. However, this on-board memory is not extensible. For single precision D3Q19 LBM, such an amount allows the computation domain to contain as much as 3.7×10^7 nodes, which is fairly large but might not be sufficient to carry out large scale simulations. To be of practical interest in many applications, GPU LBM solvers should therefore be able to run on multiple GPUs in parallel, which implies to address both communication and partitioning issues.

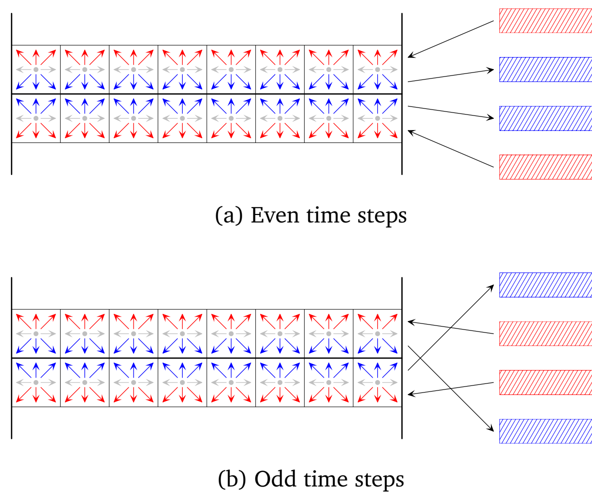
GPUs communicate with their host system through PCI Express (PCIe) links. Fermi based devices, for instance, may use up to 16 PCIe 2.0 links in parallel, which yields a maximum sustained throughput exceeding 3 GB/s in each direction. Although considerable, this performance is limited by non-negligible latencies. In the case of LBM, the simple approach, which consists in performing inter-GPU communication through host driven data transfer once kernel execution has completed, fails to give satisfactory performance [38]. A possible way to overlap communication and computations would be to process the

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

interface nodes using CUDA streams, but this would lead to a rather complex memory layout and—to our knowledge—no such attempt was ever reported in literature.

A more convenient method to overlap communication and computations arose with the introduction of the zero-copy feature, which enables the GPU to access directly to host memory, but requires the use of page-locked buffers to avoid interferences with the host operating system. As for global memory transactions, zero-copy transactions are issued per warp (or half-warp, depending on the compute capability), and should therefore be coalescent. For a multi-GPU implementation of the LBM using the same execution configuration and data layout as our generic single-GPU version, the former constraint implies that efficient data exchange is possible only for nodes located at the faces of the sub-domain parallel to the blocks' direction. Implementations based on this approach are thus limited to one- or two-dimensional domain partitions.

Recent motherboards may hold up to eight GPU computing devices. For such a small number of sub-domains, the advantages of a 2D partition over a 1D partition are dubious: the volume of transferred data is yet reduced by 43% in the best case (i.e. the cubic cavity), but the number of sub-domain interfaces raises from 7 to 10. For single node implementations, we therefore chose to restrict ourselves to one-dimensional partitioning which greatly simplifies the code. The isothermal version of our single-node multi-GPU solver is presented in Art. D and E. The thermal version, for which we had to modify our execution pattern in order to avoid shared memory shortage, is described in Art. G.



(a) Even time steps

(b) Odd time steps

Figure 9: Inter-GPU communication scheme. — *The out-going particle populations are drawn in blue whereas the in-coming populations are drawn in red. The arrows represent the data transfer occurring between the global memory of the two involved GPUs and the host memory.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Our single-node solvers use POSIX threads to manage each GPU separately. The implemented inter-GPU communication scheme is outlined in Fig. 9. Each interface is associated to four page-locked buffers needed to load in-coming and store out-going particle populations. At each time step, the pointers to the buffers are swapped. For a cubic cavity, our single-node solvers currently reach up to 2,000 MLUPS in single precision, using eight Tesla C1060. As mentioned in Art. E, such performance is comparable to the one achieved on a Blue Gene/P computer with 4,096 cores by an optimised double precision code. Our studies in Art. D. and G show that the chosen approach yields good overlapping of communication and computations, with usually more than 80% parallelisation efficiency. In Art. E, we furthermore demonstrate that in most cases, even for fairly small computation domains, inter-GPU communication is not a limiting factor.

c. GPU cluster implementation

The single node implementation strategy outlined in the former section does not directly apply to GPU cluster systems. The inability to communicate efficiently using 3D partitions is the major issue. Beside providing increased flexibility, 3D partitioning considerably decreases the volume of communications. In the case of a cubic computation domain containing n^3 nodes and split into N^3 balanced sub-domains, the number of interface nodes is $2(N^3-1)n^2$ for the 1D partition and, with sufficiently large n, is approximately $6(N-1)n^2$ for the 3D partition. With $N = 4$ for instance, the volume of communication is thus divided by a factor of nearly 7. Moreover, with the possibility of a large number of sub-domains arises the need for a convenient way to specify the execution configuration. The MPI CUDA implementation proposed in Art. I attempts to address both issues.

With our usual data layout and execution pattern, the particle populations at the interfaces perpendicular to the blocks' direction would lie in non-contiguous memory locations. We therefore chose to store these data in auxiliary arrays. In order to enable coalesced accesses to these arrays, we use one-dimensional blocks (containing a single warp) that are not mapped any more to a single row of nodes, but to a square tile. The execution pattern at block level is summarised in Fig. 10. Before processing row by row the nodes of the tile, the in-coming populations from the auxiliary arrays are copied to shared memory. The resulting out-going populations are temporarily stored in shared memory and written back once the processing of the node has completed. Such a method allows the zero-copy transactions to be coalesced for all six possible interfaces of the sub-domain.

The execution set-up is described by a configuration file in JSON[12] format. Beside providing general parameters, such as the physical parameters of the

[12]JavaScript Object Notation. This format has the advantage of being both human-readable and easy to parse.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

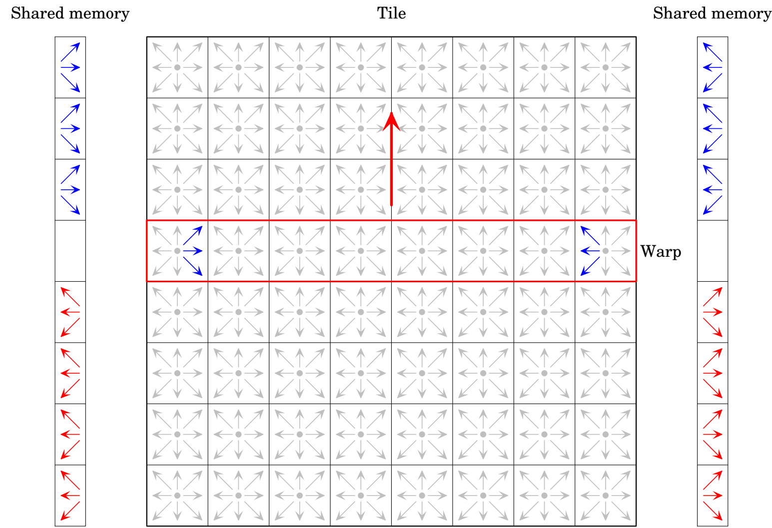


Figure 10: Kernel execution pattern. — *The current row of nodes is framed in red, the bold red arrow representing the direction of processing. The in-coming populations are drawn in red, whereas the out-going ones are drawn in blue. Once a row has been processed, the out-going post-collision populations are written over the outdated in-coming ones.*

simulation, this file is mainly used to specify for each sub-domain to which cluster node and GPU it is assigned and to which neighbouring sub-domains it is linked. A set of MPI routines is responsible for inter-GPU communication. During kernel execution, the out-going particle population located at the faces of the sub-domain are written to page-locked buffers using zero-copy transactions. The corresponding MPI process then copies the populations located at the edges to specific buffers and proceeds with message passing. Once completed, the receive buffers for faces and edges are gathered into page-locked read buffers, performing partial propagation at the same time.

We tested our solver on a nine-node cluster, each node hosting three Tesla M2070 (or M2090) computing devices. Using all 27 GPUs, we recorded up to 10,280 MLUPS in single precision with about 796 million nodes. Our performance results compare favourably with recently published works. On a 768^3 cavity for instance, we manage, using only 24 GPUs, to outperform the solver tested on the TSUBAME cluster with 96 GT200 [47]. In Art. I, our studies show that both weak and strong scalability are quite satisfactory. However, because of the limited number of sub-domains, further investigations on larger systems should be carried out. Although fully functional, our code still requires comprehensive validation studies, and may benefit from further enhancements. Nevertheless, this GPU cluster implementation of the LBM appears to be a promising tool to perform very large scale simulations.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

VI Applications and perspectives

The multi-GPU thermal LBM solver described in Art. G, or possibly an extension to thermal simulations of our recent GPU cluster implementation, could be a major component for realistic indoor environment simulations as in [45]. However, the requirements in terms of spatial resolution are so high that, even with the computational power provided by GPUs, direct numerical simulations are beyond reach. The only practicable way for the time being seems to be the use of grid refinement together with a turbulence model suited for thermal LBM. Adding the grid refinement feature to our solvers is therefore of major importance, although it may have a significant impact on performance. Radiative effects should also be taken into account, especially when simulating low energy buildings, for which the direct solar contribution is usually considerable. Several approaches to simulate radiative heat transfers with GPUs are available and could be coupled to GPU LBM solvers.

Regarding external building aeraulics, the multi-GPU isothermal LES-LBM solver described in Art. F appears to be a convincing tool for simulating the flow around a building or even a group of buildings, although alternative boundary conditions and turbulence models should be tested. It may be useful to evaluate parameters such as pressure coefficients in various configurations or to carry out pedestrian wind environment studies. In this perspective, the interfacing between a geographic information system and the geometry module of the TheLMA framework would be of great interest. For external thermo-aeraulics, the situation is even more challenging than for indoor simulations, since at this level of Reynolds number, the thickness of the viscous sub-layer can go down to 100 μm [3]. The evaluation of parameters such as convective heat transfer coefficients may thus require very large computational efforts.

The TheLMA framework has been implemented in order to be as generic as possible despite of the limitations induced by the CUDA technology. Adapting it to solve similar schemes such as the recently introduced link-wise artificial compressibility method [1], or to solve PDEs such as the diffusion or the Laplace equation [52], should therefore be straightforward. Alternative many-core processors, e.g. the Intel MIC or the Kalray MPPA, share many similarities with GPUs, especially regarding data transfer mechanisms. The extension of our framework to such architectures, although it would probably necessitate to rewrite large portions of the code [13], should benefit from the general optimisation strategies we devised for the GPU implementation of LBM.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Bibliography

[1] P. Asinari, T. Ohwada, E. Chiavazzo, and A.F. Di Rienzo. Link-wise Artificial Compressibility Method. *Journal of Computational Physics*, 231(15):5109–5143, 2012.

[2] P. L. Bhatnagar, E. P. Gross, and M. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*, 94(3):511–525, 1954.

[3] B. Blocken, T. Defraeye, D. Derome, and J. Carmeliet. High-resolution CFD simulations for forced convective heat transfer coefficients at the facade of a low-rise building. *Building and environment*, 44(12):2396–2412, 2009.

[4] B. Blocken, T. Stathopoulos, J. Carmeliet, and J.L.M. Hensen. Application of computational fluid dynamics in building performance simulation for the outdoor environment: an overview. *Journal of Building Performance Simulation*, 4(2):157–184, 2011.

[5] M. Bouzidi, M. Firdaouss, and P. Lallemand. Momentum transfer of a Boltzmann-lattice fluid with boundaries. *Physics of Fluids*, 13(11):3452–3459, 2001.

[6] H. Boyer, J.-P. Chabriat, B. Grondin-Perez, C. Tourrand, and J. Brau. Thermal building simulation and computer generation of nodal models. *Building and environment*, 31(3):207–214, 1996.

[7] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: stream computing on graphics hardware. In *ACM Transactions on Graphics*, volume 23, pages 777–786. ACM, 2004.

[8] S. Collange, D. Defour, and A. Tisserand. Power Consumption of GPUs from a Software Perspective. In *Lecture Notes in Computer Science 5544, Proceedings of the 9th International Conference on Computational Science, Part I*, pages 914–923. Springer, 2009.

[9] B. Crouse, M. Krafczyk, S. Kühner, E. Rank, and C. Van Treeck. Indoor air flow analysis based on lattice Boltzmann methods. *Energy and buildings*, 34(9):941–949, 2002.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

[10] D. d'Humières. Generalized lattice-Boltzmann equations. In *Proceedings of the 18th International Symposium on Rarefied Gas Dynamics*, pages 450–458. University of British Columbia, Vancouver, Canada, 1994.

[11] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.S. Luo. Multiple-relaxation-time lattice Boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society A*, 360:437–451, 2002.

[12] D. d'Humières, P. Lallemand, and U. Frisch. Lattice gas models for 3D hydrodynamics. *EPL (Europhysics Letters)*, 2(4):291–297, 1986.

[13] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing*, 38(8):391–407, 2012.

[14] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. GPU cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, pages 47–58. IEEE, 2004.

[15] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier-Stokes equation. *Physical review letters*, 56(14):1505–1508, 1986.

[16] I. Ginzbourg and D. d'Humières. Local second-order boundary methods for lattice Boltzmann models. *Journal of statistical physics*, 84(5):927–971, 1996.

[17] Khronos OpenCL Working Group. *The OpenCL specification*. 2008.

[18] J. Hardy, Y. Pomeau, and O. De Pazzis. Time evolution of a two-dimensional classical lattice system. *Physical Review Letters*, 31(5):276–279, 1973.

[19] X. He, S. Chen, and G. D. Doolen. A Novel Thermal Model for the Lattice Boltzmann Method in Incompressible Limit. *Journal of Computational Physics*, 146(1):282–300, 1998.

[20] 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):6811–6817, 1997.

[21] K.E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM, 1999.

[22] C. Inard, H. Bouia, and P. Dalicieux. Prediction of air temperature distribution in buildings with a zonal model. *Energy and Buildings*, 24(2):125–132, 1996.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

[23] I.V. Karlin, A. Ferrante, and H.C. Öttinger. Perfect entropy functions of the Lattice Boltzmann method. *EPL (Europhysics Letters)*, 47(2):182–188, 1999.

[24] D. Kirk. Nvidia cuda software and gpu parallel computing architecture. In *Proceedings of the 6th International Symposium on Memory Management*, pages 103–104. ACM, 2007.

[25] M. Krafczyk, J. Tölke, and L.S. Luo. Large-eddy simulations with a multiple-relaxation-time LBE model. *International Journal of Modern Physics B*, 17(1):33–40, 2003.

[26] P. Lallemand and L. S. Luo. Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions. *Physical review E*, 68(3):36706(1–25), 2003.

[27] J. Latt and B. Chopard. Lattice Boltzmann method with regularized pre-collision distribution functions. *Mathematics and Computers in Simulation*, 72(2):165–168, 2006.

[28] W. Li, X. Wei, and A. Kaufman. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19(7):444–456, 2003.

[29] S. Marié. *Étude de la méthode Boltzmann sur réseau pour les simulations en aéroacoustique*. PhD thesis, Université Pierre-et-Marie-Curie, Paris, 2008.

[30] W.R. Mark, R.S. Glanville, K. Akeley, and M.J. Kilgard. Cg: A system for programming graphics hardware in a C-like language. In *ACM Transactions on Graphics*, volume 22, pages 896–907. ACM, 2003.

[31] G. R. McNamara and G. Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Physical Review Letters*, 61(20):2332–2335, 1988.

[32] F. Nicoud and F. Ducros. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. *Flow, Turbulence and Combustion*, 62(3):183–200, 1999.

[33] NVIDIA. *Compute Unified Device Architecture Programming Guide version 4.0*, 2011.

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

[35] M. Pharr, editor. *GPU Gems 2 : programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley, 2005.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

[36] Y. H. Qian. Simulating thermohydrodynamics with lattice BGK models. *Journal of scientific computing*, 8(3):231–242, 1993.

[37] Y. H. Qian, D. d'Humières, and P. Lallemand. Lattice BGK models for Navier-Stokes equation. *EPL (Europhysics Letters)*, 17(6):479–484, 1992.

[38] E. Riegel, T. Indinger, and N.A. Adams. Implementation of a Lattice–Boltzmann method for numerical fluid mechanics using the nVIDIA CUDA technology. *Computer Science – Research and Development*, 23(3):241–247, 2009.

[39] 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(7):151–172, 1995.

[40] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, 1963.

[41] C.J. Thompson, S. Hahn, and M. Oskin. Using modern graphics architectures for general-purpose computing: a framework and analysis. In *Proceedings of the 35th annual ACM/IEEE International Symposium on Microarchitecture*, pages 306–317. IEEE, 2002.

[42] E. Tric, G. Labrosse, and M. Betrouni. A first incursion into the 3D structure of natural convection of air in a differentially heated cubic cavity, from accurate numerical solutions. *International Journal of Heat and Mass Transfer*, 43(21):4043 – 4056, 2000.

[43] J. Tölke. Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA. *Computing and Visualization in Science*, 13(1):29–39, 2010.

[44] 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.

[45] C. van Treeck, E. Rank, M. Krafczyk, J. Tölke, and B. Nachtwey. Extension of a hybrid thermal LBE scheme for Large-Eddy simulations of turbulent convective flows. *Computers & Fluids*, 35(8):863–871, 2006.

[46] S. Wakashima and T.S. Saitoh. Benchmark solutions for natural convection in a cubic cavity using the high-order time-space method. *International Journal of Heat and Mass Transfer*, 47(4):853–864, 2004.

[47] X. Wang and T. Aoki. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. *Parallel Computing*, 37(9):521–535, 2011.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

[48] M. Weickert, G. Teike, O. Schmidt, and M. Sommerfeld. Investigation of the LES WALE turbulence model within the lattice Boltzmann framework. *Computers & Mathematics with Applications*, 59(7):2200–2214, 2010.

[49] G. Wellein, T. Zeiser, G. Hager, and S. Donath. On the single processor performance of simple lattice Boltzmann kernels. *Computers & Fluids*, 35(8):910–919, 2006.

[50] J. Wilke, T. Pohl, M. Kowarschik, and U. Rüde. Cache performance optimizations for parallel lattice Boltzmann codes. *Euro-Par 2003, LNCS 2790*, pages 441–450, 2003.

[51] D.A. Wolf-Gladrow. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*, volume 1725 of *Lecture Notes in Mathematics*. Springer, 2000.

[52] Y. Zhao. Lattice Boltzmann based PDE solver on the GPU. *The Visual Computer*, 24(5):323–333, 2008.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Résumé détaillé

Est-il possible de simuler avec précision le comportement énergétique d'un bâtiment ? La puissance de calcul considérable offerte par les ordinateurs actuels laisse à penser que la réponse à cette question est positive. Cela dit, les bâtiments forment des systèmes complexes interagissants de nombreuses manières avec leur environnement, à différentes échelles spatiales et temporelles. Ainsi, effectuer des simulations précises conduit souvent à des temps de calcul prohibitifs.

La conception des bâtiments à haute efficacité énergétique accroit encore les besoins en termes de modélisation des transferts de masse et de chaleur entre l'enveloppe d'un bâtiment et son environnement. La pratique usuelle en aéraulique des bâtiments est de recourir à des modèles simplifiés empiriques ou semi-empiriques. Ces approches, néanmoins, ne sauraient fournir davantage que des indications quant aux grandeurs étudiées.

En ce qui concerne l'aéraulique interne, les modèles nodaux et zonaux sont également répandus. Dans la première approche, l'air contenu dans une pièce est représenté par un nœud unique, dans la seconde, le volume correspondant à une pièce est divisé en cellules de dimensions macroscopiques. Ces méthodes, dont le coût calculatoire est faible mais qui simulent de larges volumes d'air à l'aide d'un seul nœud, conduisent souvent à des résultats éloignés des données expérimentales.

Afin d'atteindre la précision requise, le recours à la mécanique des fluides numérique (CFD[1]) en aéraulique des bâtiments semble donc incontournable. Néanmoins, la simulation numérique des écoulements a souvent un tel coût calculatoire qu'elle ne saurait être menée à bien sur une station de travail individuelle. L'accès aux centres de calculs étant limité et coûteux, la mise au point d'approches alternatives conduisant à des programmes de simulation des écoulements plus performants est donc un objectif majeur, susceptible d'avoir des retombées importantes dans de nombreux domaines de l'ingénierie.

Le travail de recherche présenté dans ces pages s'attache à explorer le potentiel lié à l'utilisation de processeurs graphiques (GPU[2]) pour mener à bien des simulations en mécanique des fluides numérique basées sur la méthode de Boltzmann sur gaz réseau (LBM[3]). Cette méthode, qui constitue une approche assez récente dans le domaine de la CFD, est bien adaptée au calcul intensif (HPC[4]) comme il sera montré plus avant. Ce travail a conduit à la conception et la création d'une plateforme logicielle nommée TheLMA[5]. Plusieurs solveurs

[1] Computational fluid dynamics.
[2] Graphics processing unit.
[3] Lattice Boltzmann method.
[4] High-performance computing.
[5] Thermal LBM for Many-core Architectures.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

LBM ont été développés à partir de cette plateforme, allant de l'implantation mono-GPU à celle pour grappe de GPU, et abordant divers aspects tels que la simulation de fluides anisothermes, la simulation d'écoulements turbulents ou encore la simulation d'obstacles présentant des géométries complexes.

Le présent résumé est organisé de la façon suivante. La première section donne une vue d'ensemble des technologies de calcul généraliste sur processeurs graphiques. Dans la deuxième section, une description des principes et des aspects algorithmiques de la méthode de Boltzmann sur gaz réseau est donnée. La section III se concentre sur les implantations mono-GPU de la LBM. La section IV décrit diverses extensions à la LBM essentielles pour envisager une utilisation en aéraulique des bâtiments. La section V est consacrée aux implantations multi-GPU de la LBM, simple nœud et multi-nœuds. La section VI apporte une conclusion en donnant un aperçu des applications potentielles de la plateforme TheLMA à la simulation des bâtiments ainsi que des questions restant en suspens.

I Calcul généraliste sur processeurs graphiques

Les circuits graphiques sont destinés à soulager le travail du processeur central en ce qui concerne le rendu graphique. Ces circuits spécialisés sont devenus courants dans les ordinateurs grand public durant les années 1990. Dans un premier temps, les accélérateurs graphiques étaient pour l'essentiel des outils de rastérisation conçus pour déterminer les caractéristiques des pixels à afficher à partir d'un schéma électronique fixé. Ces circuits étaient donc d'un intérêt limité en termes de calcul bien que des essais aient été menés dès 1999. C'est au cours de cette même année que le fabricant Nvidia créa la dénomination GPU à l'occasion de la sortie de la GeForce 256.

En 2001, Nvidia a introduit une architecture innovante pour la GeForce 3, basée sur des unités de rendu programmables. Avec cette technologie, le pipeline graphique incorpore des *shaders*, c'est-à-dire des programmes affectés à la détermination des facettes et des pixels suivant les caractéristiques des scènes à afficher. À partir de là, il devint possible de considérer les GPU comme des processeurs parallèles de type SIMD[6]. Au départ, les *shaders* devaient être écrits dans un assembleur spécifique à l'architecture cible, jusqu'à l'introduction de langages de haut niveau : HLSL, également connu sous le nom de Cg, lié à l'API[7] graphique Direct3D de Microsoft et GLSL lié à l'API graphique OpenGL. Ces deux langages utilisent une syntaxe dérivée du C et un paradigme de programmation centré sur la notion de flux de données : une séquence d'opérations (le *noyau*) est effectuée pour chaque élément d'un ensemble de données (le *flux*). Ces langages de haut niveau ont largement contribué à élargir le réper-

[6] Single instruction multiple data.
[7] Application programming interface.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

toire du calcul généraliste sur processeurs graphiques (GPGPU[8]). Néanmoins, le développement d'applications généralistes en Cg ou en GLSL reste délicat de par l'orientation principalement graphique de ces langages.

Durant la dernière décennie, la puissance de calcul par cœur des processeurs généralistes n'a progressé que très modestement, alors que l'amélioration de la finesse de gravure des circuits intégrés a permis aux fondeurs de multiplier le nombre d'unités de rendu graphique des GPU, augmentant d'autant la puissance de calcul théorique de ces processeurs. Cette situation a conduit le fabricant Nvidia à considérer le marché du calcul intensif comme une cible potentielle et à développer une nouvelle technologie baptisée CUDA[9]. L'arrivée de CUDA en 2007 constitue probablement le progrès le plus significatif dans le domaine du GPGPU à ce jour. Les concepts clefs consistent en un ensemble de spécifications matérielles génériques liées à un modèle de programmation parallèle. CUDA désigne souvent le langage associé à cette technologie qui dérive du C/C++ (avec quelques restrictions) mettant en œuvre ce modèle de programmation. Comparé aux technologies antérieures, CUDA apporte une flexibilité sans précédent pour le développement de logiciels destinés aux processeurs graphiques, augmentant de fait leur potentiel dans le domaine du calcul intensif.

Le statut propriétaire de la technologie CUDA est un inconvénient majeur : seuls les GPU produits par Nvidia sont à même d'exécuter des programmes CUDA. Le standard OpenCL, diffusé à partir de 2008 suit une approche plus portable dans la mesure où il se destine à tout type de plateforme. Il est à noter que le modèle et le langage de programmation OpenCL présentent de nombreuses similitudes avec leurs équivalents CUDA. Du point vue du calcul intensif, OpenCL est un standard prometteur susceptible de remplacer CUDA à l'avenir. Bien qu'énoncées dans le cadre de la technologie CUDA, les contributions du travail de recherche résumé dans ces pages devraient conserver leur pertinence avec OpenCL.

Le modèle architectural CUDA repose sur des spécifications matérielles abstraites. Un GPU CUDA est décrit comme un ensemble de multiprocesseurs (SM[10]), chacun d'entre eux contenant : un ordonnanceur, un groupe de processeur scalaires (SP[11]) disposant de leurs propres registres, ainsi que d'une mémoire partagée. La quantité de registres et de mémoire partagée par SM est assez limitée, avec par exemple au plus 64 Ko de mémoire partagée par SM sur le GF100. Les SM embarquent également de la mémoire cache destinée aux constantes, textures et, selon les générations de GPU, aux données.

Le modèle d'exécution CUDA, nommé SIMT[12] par Nvidia, est relativement complexe de par le double niveau d'organisation matérielle. Le paradigme de

[8]General purpose computing on graphics processing units.
[9]Compute Unified Device Architecture.
[10]Streaming multiprocessor.
[11]Scalar processor.
[12]Single instruction multiple threads.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

programmation CUDA est centré sur la notion de tâche. Un processus élémentaire est désigné par le terme *thread*, que nous traduirons en *fil d'exécution* ou plus simplement *fil* dans la suite de ce texte, la séquence d'instruction associée étant nommée *noyau*. Pour exécuter un noyau, il est nécessaire de spécifier une *grille* d'exécution. Une grille consiste en un tableau multidimensionnel de *blocs*, qui eux-mêmes sont des tableaux multidimensionnels de fils d'exécution. Ce schéma à deux niveaux est induit par les caractéristiques architecturales : un bloc ne peut être traité qu'au sein d'un seul SM ; le nombre de fils figurant dans un bloc est par conséquent relativement limité.

Les SM pris séparément sont des processeurs de type SIMD, ce qui implique que les fils d'exécution ne sont pas traités de manière autonome, mais en groupes nommés *warp*, que nous traduirons par *trame*[13]. Pour l'instant, une trame contient 32 fils quelle que soit l'architecture considérée, mais cette valeur n'est pas fixée et est susceptible de changer pour les générations à venir. Une trame constitue un ensemble insécable, il est donc préférable de faire en sorte que le nombre de fils dans un bloc soit un multiple de la taille d'une trame. Cette organisation entraîne de sévères limitations, comme par exemple pour les branchements conditionnels : lorsqu'un chemin d'exécution diverge au sein d'une trame, le traitement des branches est séquentiel. Dans de nombreux cas, déterminer une grille d'exécution adéquate n'est pas trivial, mais peut se révéler d'une importance cardinale quant aux performances de l'application considérée.

Les variables automatiques d'un noyau, qui sont spécifiques à chaque fil, sont stockées autant que possible dans les registres. Les tableaux en général et les structures de taille trop importante, en revanche, résident en mémoire *locale*. Cet espace mémoire, qui sert également aux déchargements de registres, est hébergé par les circuits de mémoire de la carte de calcul. Les fils d'exécution ont également accès à la mémoire partagée et à la mémoire *globale*. La portée et la durée de vie de la mémoire partagée est limitée au bloc courant. La mémoire globale, quant à elle, est visible par l'ensemble des fils et persiste durant toute la durée de l'application. Comme la mémoire locale, elle est hébergée dans la mémoire externe au GPU qui est la seule accessible au système hôte. Il est à noter que pour les GPU des générations précédant le Fermi, ni la mémoire locale ni la mémoire globale ne sont pas associées à un cache.

Les accès à la mémoire globale se font par segments de taille allant de 32 à 128 octets, selon l'architecture. Pour être efficaces, les opérations de lecture et d'écriture en mémoire globale au sein d'une trame donnée doivent porter sur des adresses consécutives. L'organisation des données en mémoire est donc un aspect important pour l'optimisation des applications CUDA.

[13]Le terme *warp* appartient au vocabulaire du tissage. Sa traduction exacte en français est « chaîne ».

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

II Principes de la méthode de Boltzmann sur gaz réseau

Les démarches suivies en CFD peuvent dans la plupart des cas être qualifiées de « descendantes » : un système d'équations aux dérivées partielles non linéaires est discrétisé et résolu à l'aide d'une méthode d'analyse numérique telle les différences finies, volumes finis, éléments finis ou encore les approches spectrales. L'attention se porte en général sur les erreurs de troncature liées à la discrétisation. Néanmoins, d'un point de vue physique, la préservation des propriétés de conservation est essentielle, tout particulièrement pour les simulations de longue durée où de légères variations peuvent, par accumulation, conduire à des résultats totalement incorrects.

D'autres approches que l'on pourrait qualifier de « montantes » constituent des alternatives aux méthodes précédentes. Dans cette catégorie, il convient de citer la dynamique moléculaire, les automates sur gaz réseau et la méthode de Boltzmann sur gaz réseau. Alors que la première tente de simuler de la manière la plus précise possible le comportement individuel de chaque molécule, ce qui induit un coût calculatoire extrêmement élevé, les deux autres se placent à une échelle plus importante et sont souvent, de fait, qualifiées de *mésoscopiques*.

L'équation de Boltzmann décrit le comportement hydrodynamique d'un fluide par une fonction de distribution d'une particule dans l'espace des phases, regroupant la position $\boldsymbol{x}$ et la vitesse particulaire $\boldsymbol{\xi}$, et le temps :

$$\partial_t f + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} f + \frac{\boldsymbol{F}}{m} \cdot \nabla_{\boldsymbol{\xi}} f = \Omega(f). \tag{1}$$

Dans l'équation précédente, m désigne la masse de la particule, $\boldsymbol{F}$ est une force extérieure et Ω correspond à l'opérateur de collision. En trois dimensions, les grandeurs macroscopiques associées au fluide sont données par :

$$\rho = \int f \,\mathrm{d}\boldsymbol{\xi} \tag{2}$$

$$\rho \boldsymbol{u} = \int f \boldsymbol{\xi} \mathrm{d}\boldsymbol{\xi} \tag{3}$$

$$\rho \boldsymbol{u}^2 + 3\rho\theta = \int f \boldsymbol{\xi}^2 \mathrm{d}\boldsymbol{\xi} \tag{4}$$

où ρ désigne la densité du fluide, $\boldsymbol{u}$ sa vitesse et θ est défini par $\theta = k_{\mathrm{B}} T/m$ avec T la température absolue et k_{B} la constante de Boltzmann.

La méthode de Boltzmann sur gaz réseau est basée sur une forme discrétisée de l'équation 1 utilisant un pas de temps constant δt et un maillage orthogonal régulier de pas δx. Un ensemble fini de vitesses $\{\boldsymbol{\xi}_\alpha \,|\, \alpha = 0, \dots N\}$ avec $\boldsymbol{\xi}_0 = \boldsymbol{0}$, tient lieu d'espace des vitesses particulaires. Cet ensemble, ou *stencil*, est choisi de telle sorte que pour un nœud $\boldsymbol{x}$ donné, $\boldsymbol{x} + \delta t \boldsymbol{\xi}_\alpha$ est également un nœud quel que soit α. En trois dimensions, le stencil le plus couramment employé

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

est nommé D3Q19. Il lie un nœud donné à 18 de ses plus proches voisins et compte donc 19 éléments en tout.

L'équivalent discret de la fonction de distribution f est un ensemble de densités particulaires $\{f_\alpha \,|\, \alpha = 0, \ldots N\}$ associées aux vitesses. Notons $\top$ l'opérateur de transposition et $|a_\alpha\rangle = (a_0, \ldots a_N)^\top$. En l'absence de force extérieure, l'équation 1 devient :

$$|f_\alpha(\boldsymbol{x} + \delta t \boldsymbol{\xi}_\alpha, t + \delta t)\rangle - |f_\alpha(\boldsymbol{x}, t)\rangle = \Omega\left(|f_\alpha(\boldsymbol{x}, t)\rangle\right). \tag{5}$$

En ce qui concerne les grandeurs macroscopiques du fluide, les équations 2 et 3 deviennent :

$$\rho = \sum_\alpha f_\alpha, \tag{6}$$

$$\rho \boldsymbol{u} = \sum_\alpha f_\alpha \boldsymbol{\xi}_\alpha. \tag{7}$$

Il convient de mentionner le fait que les modèles LBM les plus couramment utilisés ne satisfont pas à la conservation de l'énergie. Ils ne sont donc appropriés que pour la simulation de fluides isothermes. Un composant essentiel de ces modèles est l'opérateur de collision utilisé. De nombreuses versions ont été proposées, parmi lesquelles le LBGK basé sur l'approximation de Bhatnagar–Gross–Krook, le MRT[14] recourant à des temps de relaxation multiples, les opérateurs entropiques ou encore les opérateurs régularisés. L'ensemble des solveurs LBM développés dans le cadre de ce travail de recherche, à l'exception d'un seul, intègrent l'opérateur MRT, dont une description relativement complète est donnée dans l'article D, entre autres. Par rapport au LBGK, le plus souvent employé, le MRT apporte une précision et une stabilité plus importantes au prix d'une complexité arithmétique légèrement supérieure.

D'un point de vue algorithmique, la LBM revient à une succession d'opérations de *collision* et de *propagation*. La phase de collision est décrite par :

$$|\widetilde{f}_\alpha(\boldsymbol{x}, t)\rangle = |f_\alpha(\boldsymbol{x}, t)\rangle + \Omega\left(|f_\alpha(\boldsymbol{x}, t)\rangle\right), \tag{8}$$

où $\widetilde{f}_\alpha$ désigne les densités post-collision. Il est à noter que cette étape est purement locale. La phase de propagation obéit à :

$$|f_\alpha(\boldsymbol{x} + \delta t\, \boldsymbol{\xi}_\alpha, t + \delta t)\rangle = |\widetilde{f}_\alpha(\boldsymbol{x}, t)\rangle, \tag{9}$$

ce qui correspond à de simples transferts de données. La partition de l'équation 5 en deux relations met en lumière le parallélisme de données présent dans la LBM. Cette approche est donc bien adaptée au calcul intensif sur architectures massivement parallèles, le point essentiel étant de garantir la synchronisation des opérations entre voisins immédiats. Le moyen le plus simple

[14] Multiple relaxation time.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

de gérer cette contrainte consiste à utiliser deux instances des tableaux de données, une pour les pas de temps pairs et l'autre pour les pas de temps impairs. Une méthode à présent largement utilisée et nommée *compression de grille*, permet de diminuer l'occupation mémoire presque de moitié, mais nécessite d'avoir le contrôle sur l'ordre de traitement des nœuds. Elle ne s'applique donc pas dans un contexte d'exécution asynchrone.

Dans le cadre de systèmes à mémoire partagée, l'organisation des données pour des simulations LBM en trois dimensions se réduit généralement à un tableau à cinq dimensions : trois pour l'espace, une pour les indices de vitesses particulaires et une pour le temps. Selon l'architecture cible et la hiérarchie mémoire, l'ordre des dimensions dans le tableau peut avoir un impact majeur sur les performances. La conception des structures de données est de fait une étape essentielle pour l'optimisation des implantations HPC de la LBM.

III Implantations GPU de la LBM

Les premières tentatives d'implanter la LBM sur GPU remontent aux débuts du calcul généraliste sur GPU en 2003. Aucune plateforme de programmation généraliste n'étant encore disponible, les densités particulaires devaient être stockées dans des piles de tableaux bidimensionnels de textures. Avec cette approche, les calculs associés à l'équation 5 doivent être traduits en opérations destinées aux unités de rendu graphique, ce qui, à l'évidence, est très contraignant et dépendant du matériel. La première implantation CUDA de la LBM en trois dimensions a été décrite par Tölke et Krafczyk en 2008. Les principes d'implantation proposés demeurent pour l'essentiel valables aujourd'hui. On retiendra essentiellement :

1. La fusion des étapes de collision et de propagation en un seul noyau pour éviter les transferts de données inutiles.

2. L'identification de la grille d'exécution CUDA au maillage, c'est-à-dire l'affectation d'un fil d'exécution à chaque nœud. Cette approche, créant un grand nombre de fils, permet de profiter du parallélisme massif des GPU et de masquer éventuellement la latence de la mémoire globale en l'absence de cache.

3. Le recours à une organisation des données permettant aux accès en mémoire globale effectués par les trames d'être coalescents.

4. Le lancement du noyau de collision et propagation à chaque pas de temps et l'utilisation de deux instances des tableaux de données afin de garantir la synchronisation locale.

Pour leur implantation de la LBM, Tölke et Krafczyk utilisent une grille bidimensionnelle de blocs unidimensionnels et des tableaux de distribution spécifiques à chaque vitesse particulaire. Ce dispositif permet des accès coalescents

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

lors de la lecture des densités avant d'opérer la collision. Néanmoins, un soin particulier doit être apporté à la réalisation de la propagation, qui consiste en des déplacements de données dans toutes les directions spatiales, y compris celle qui correspond à la dimension mineure des tableaux de distribution. Ainsi, une écriture directe en mémoire globale induit nécessairement des défauts d'alignement, ce qui, avec le G80 utilisé alors, a des conséquences délétères sur les performances, dans la mesure où tous les accès mémoire sont traités individuellement. La solution proposée par Tölke et Krafczyk consiste à utiliser la mémoire partagée pour opérer une propagation partielle le long des blocs et d'achever la propagation lors de l'écriture en mémoire globale.

La méthode de la propagation en mémoire partagée, supprimant entièrement les défauts d'alignement, semble incontournable pour les implantations destinées au G80. Pour les générations suivantes en revanche, le coût des défauts d'alignement est nettement moins conséquent, bien que toujours non négligeable. En ce qui concerne le GT200, par exemple, les accès à la mémoire globale se font par segments alignés de 32, 64 ou 128 octets, une transaction non alignée étant réalisée en un minimum d'opérations. Il apparaît donc raisonnable de considérer des approches consistant à effectuer la propagation directement en mémoire globale. Les résultats de nos recherches à ce sujet sont consignés dans l'article A.

Quelques investigations élémentaires sur une GeForce GTX 295 nous ont permis de constater que les défauts d'alignement sont significativement plus coûteux à l'écriture qu'à la lecture. Nous avons donc testé deux schémas de propagation alternatifs. Le premier, baptisé schéma *scindé*, consiste à effectuer une propagation partielle dans les directions perpendiculaires aux blocs durant la phase d'écriture, puis à compléter cette propagation durant la phase de lecture du pas de temps suivant. Le second, nommé schéma *inversé*, revient à réaliser la propagation durant la phase de lecture, aucun décalage n'étant effectué lors de l'écriture.

Les implantations de ces deux schémas obtiennent des performances comparables, de l'ordre de 500 millions de nœuds traités par seconde (MLUPS[15]) en simple précision, soit plus de 80 % du débit maximal effectif entre GPU et mémoire globale. La méthode de propagation en mémoire partagée est susceptible de mener à des performances encore supérieures à celles obtenues avec des approches de propagation directe en mémoire globale. Néanmoins, ces dernières sont d'un grand intérêt en pratique car elles conduisent à des codes plus simples et laissent la mémoire partagée vacante, ce qui permet d'envisager une utilisation pour mener à bien des calculs additionnels comme nous le verrons plus avant.

Les communications entre GPU et mémoire globale tendent à être le facteur limitant les performances des solveurs LBM pour CUDA. Bien que quelques informations soient livrées dans le guide de programmation CUDA, l'essentiel des

[15]Million lattice node updates per second.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

mécanismes de transfert de données demeure non documenté. Afin de gagner en compréhension dans ce domaine et affiner nos stratégies d'optimisation, nous avons réalisé un ensemble de bancs d'essais sur l'architecture GT200 dont les résultats sont rapportés dans l'article B. Ces études nous ont permis d'énoncer un modèle pour les communications entre GPU et mémoire globale prenant en compte divers paramètres comme le nombre de défauts d'alignement éventuels. Du précédent modèle, nous dérivons une estimation de l'optimum de performance pour les implantations CUDA de la LBM, susceptible de fournir des indications utiles dans le processus d'optimisation.

IV Extensions à la LBM

En aéraulique des bâtiments, bien que les simulations d'écoulements isothermes ne soient pas dénuées d'intérêt, il se révèle souvent nécessaire de prendre en compte les aspects thermiques. Les articles C et G retracent nos tentatives d'implantation de solveurs LBM thermiques sur GPU. Comme nous l'avons déjà mentionné, les modèles de Boltzmann sur gaz réseau usuels tels le D3Q19 MRT ne satisfont pas à la conservation de l'énergie. De nombreuses voies ont été explorées pour appliquer la LBM en thermo-hydrodynamique, parmi lesquelles il convient de citer les modèles multi-vitesses, utilisant un jeu de vitesses particulaires agrandi, les modèles à double population, utilisant une distribution d'énergie en complément de la distribution particulaire, ou encore les modèles hybrides, pour lesquels l'équation de la chaleur est traitée par une méthode d'analyse numérique classique.

Notre choix s'est porté sur l'approche hybride développée par Lallemand et Luo en 2003. Elle repose sur une version légèrement modifiée du D3Q19 MRT, l'équation de la chaleur étant résolue par une méthode aux différences finies. Comparée aux modèles multi-vitesses ou double population, l'approche hybride apporte une stabilité et une précision accrues, tout en présentant un surcoût réduit en termes de communication. En effet, une seule opération d'écriture et dix-neuf opérations de lecture supplémentaires par nœud sont nécessaires. Nos implantations, utilisant la mémoire partagée afin de réduire les accès redondants, parvient à diminuer de plus de moitié le volume additionnel en lecture. Le solveur mono-GPU décrit dans l'article C et le solveur multi-GPU décrit dans l'article G ont tous deux été testés sur la cavité cubique différentiellement chauffée. Les nombres de Nusselt calculés aux parois isothermes sont en bon accord avec les données issues de la littérature. En utilisant la version multi-GPU, nous avons pu effectuer des simulations pour des nombres de Rayleigh allant jusqu'à 10^9.

Les solveurs multi-GPU tels ceux décrits dans les articles D et E permettent d'effectuer des simulations sur des domaines de calcul de tailles considérables. Avec un serveur Tyan B7015 équipé de huit Tesla C1060, comportant chacune 4 Go de mémoire vive, il est possible de travailler sur un domaine de calcul

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

cubique contenant jusqu'à 576^3 nœuds en simple précision, soit plus de 190 millions de nœuds. De telles résolutions permettent d'effectuer des simulations à des nombres de Reynolds de l'ordre de 10^3 à 10^4, selon les cas. Néanmoins, les valeurs typiquement atteintes en aéraulique externe des bâtiments sont plutôt de l'ordre de 10^6. À cause des phénomènes de turbulence, réaliser des simulations numériques directes pour de telles applications semble donc hors de portée pour l'instant.

Ainsi, pour être d'un intérêt pratique en aéraulique externe des bâtiments, les solveurs LBM doivent intégrer un modèle de sous-maille tel que, par exemple, la simulation aux grandes échelles (LES[16]). Afin d'évaluer la faisabilité et la pertinence d'une telle démarche, nous avons choisi d'implanter la version la plus élémentaire de la LES, à savoir le modèle originel proposé par Smagorinsky en 1963. Les résultats des simulations effectuées à l'aide d'une version étendue de notre solveur multi-GPU isotherme sont présentés dans l'article F. Dans le modèle de Smagorinsky, une viscosité turbulente obtenue à partir du tenseur des contraintes est ajoutée à la viscosité moléculaire afin d'obtenir la viscosité cinématique. L'un des intérêts de l'opérateur MRT en l'occurrence est qu'il permet aisément de retrouver les composantes du tenseur des contraintes. Les calculs étant purement locaux, l'impact sur les performances est négligeable.

Dans l'article F, nous décrivons la simulation des écoulements au voisinage d'un groupe de neuf cubes montés sur une paroi pour un nombre de Reynolds $\mathrm{Re} = 10^6$. Nous avons déterminé les champs de pression et de vitesse moyens en intégrant sur un intervalle de temps suffisamment long pour être statistiquement pertinent. En l'absence de données expérimentales, ces résultats n'ont pu être validés. En revanche, le temps de calcul étant inférieur à 18 h, notre étude démontre la possibilité de simuler les écoulements au voisinage d'un groupe de bâtiments en un temps raisonnable.

Avec la LBM, l'interface entre fluide et solide conduit à un certain nombre de densités particulaires indéterminées. En considérant un nœud à l'interface, c'est-à-dire un nœud fluide ayant au moins un nœud solide pour voisin immédiat, il est aisé de voir que les densités qui devraient être propagées depuis les nœuds solides voisins sont indéfinis. Une manière communément employée de régler ce problème consiste à utiliser la condition aux limites de *simple rebond* (SBB[17]) qui consiste à remplacer les densités inconnues par les valeurs post-collision du pas de temps précédent pour les directions opposées. Soit $\boldsymbol{x}$ un nœud à l'interface, la SBB obéit à :

$$f_{\bar{\alpha}}(\boldsymbol{x}, t) = \widetilde{f}_{\alpha}(\boldsymbol{x}, t - \delta t) \qquad (10)$$

où $f_{\bar{\alpha}}(\boldsymbol{x}, t)$ est une densité particulaire inconnue et $\bar{\alpha}$ est la direction opposée à α. Il est possible de montrer que la SBB est du second ordre en espace et que la limite entre fluide et solide est située approximativement à mi-chemin entre

[16] Large eddy simulation.
[17] Simple bounce-back.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

le nœud à l'interface et le nœud solide. On constate à partir de l'équation 10 que l'application de cette condition aux limites est simple, puisqu'il suffit de connaître la liste des voisins solides du nœud considéré. La SBB est utilisée pour la plupart de nos solveurs, le voisinage d'un nœud étant généralement décrit à l'aide d'un champ de bits. Cette approche nous permet de représenter aisément des obstacles possédant une géométrie complexe, comme le groupe de neuf cubes étudié dans l'article F.

Comme la position de l'interface entre fluide et solide est fixée, l'utilisation de la SBB n'est convaincante que dans le cas où les solides considérés possèdent des faces planes parallèles aux directions spatiales. Une surface incurvée voire une paroi plane inclinée conduisent à des effets de marches d'escalier qui peuvent avoir un impact non négligeable sur la simulation. Pour tenter de palier ces effets, nous avons également considéré la condition aux limites de *rebond linéairement interpolé* (LIBB[18]) qui prend en compte la position exacte de l'interface entre fluide et solide. Notre implantation met à profit des transferts de données qui demeurent inexploités avec le schéma de propagation inversé. Elle se révèle donc relativement efficace puisqu'elle n'accroit que légèrement la quantité de données à lire. Pour le cas test de l'écoulement autour d'une sphère, la comparaison avec des données expérimentales montre que la LIBB apporte une précision et une stabilité supérieures à la SBB en ce qui concerne la fréquence de détachement des vortex. Cette conclusion ne paraît cependant pas pouvoir se généraliser ; des simulations récentes et non publiées pour l'instant, portant sur les écoulements au voisinage d'une plaque plane inclinée, ne montrent pas de différences significatives entre les deux conditions aux limites. Des études plus poussées portant sur des paramètres plus pertinents tels que le coefficient de traînée, ou mettant en œuvre d'autres formes de conditions aux limites, méritent d'être menées. Comme il est montré dans les articles F et H, les solveurs LBM sur GPU rendent possibles des campagnes de validation à grande échelle et peuvent donc contribuer à l'évaluation de stratégies de modélisation innovantes.

V Simulations LBM à grande échelle

CUDA est une technologie récente qui impose des contraintes fortes au programmeur, liées principalement à des aspects matériels. Bien que la situation ait évolué favorablement avec les capacités croissantes des générations successives de GPU, ainsi que les améliorations apportées à la chaîne de compilation, de nombreuses pratiques issues l'ingénierie logicielle, tels que le développement de bibliothèques, ne sont pas pertinentes. Néanmoins, nos travaux nous ont conduits à la réalisation de nombreuses versions de nos solveurs LBM, correspondant à des modèles physiques et à des configurations très variés. Il en découle la nécessité de disposer de stratégies appropriées favorisant la réuti-

[18]Linearly interpolated bounce-back.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

lisation et la maintenance du code existant. Cette situation nous a amenés à la conception et la création de la plateforme logicielle TheLMA. Elle consiste en un ensemble cohérent de définitions de structures de données et de fichiers source C et CUDA. Les fichiers C fournissent une collection de fonctions utilitaires destinées au traitement des paramètres de configuration, à l'initialisation des structures de données globales, au traitement des résultats de simulation, ou encore à la production de sorties graphiques. La partie CUDA de la plateforme se divise en plusieurs modules, chacun étant centré sur un noyau aux attributions spécifiques. La plateforme TheLMA a été conçue pour confiner les modifications à des endroits bien définis du code source lors de l'implantation de nouveaux modèles ou la mise en place d'une nouvelle simulation, permettant au développement de gagner en efficacité.

La mémoire vive disponible sur des cartes de calcul récentes comme la Tesla C2075 atteint jusqu'à 6 Go, mais n'est pas extensible. Pour une simulation LBM utilisant le stencil D3Q19 en simple précision, une telle quantité permet de stocker des domaines de calcul contenant jusqu'à 3.7×10^7 nœuds. Ce nombre, certes conséquent, peut se révéler insuffisant pour mener à bien des simulations à grande échelle. Pour avoir une portée pratique, un solveur LBM pour processeurs graphiques doit être en mesure de recourir à plusieurs GPU en parallèle, ce qui implique de traiter à la fois les problèmes liés à la communication et au partitionnement. Pour assurer un bon recouvrement des calculs et des échanges de données entre sous-domaines, nous avons choisi d'utiliser des accès directs à la mémoire centrale initiés par les processeurs graphiques eux-mêmes durant l'exécution du noyau. Comme pour les opérations en mémoire globale, ces accès, pour être efficaces, doivent être coalescents, ce qui n'est pas le cas pour les faces du sous-domaine perpendiculaires aux blocs. Cette approche ne permet donc pas d'envisager l'utilisation de partitions tridimensionnelles du domaine de calcul.

Les cartes mères actuelles sont capables de gérer jusqu'à huit cartes de calcul simultanément. Avec un nombre de sous-domaines aussi faible, les avantages d'une partition bidimensionnelle sont sujets à caution. Notre implantation multi-GPU simple nœud de la LBM isotherme, décrite dans les articles D et E se restreint volontairement aux partitions unidimensionnelles, ce qui a pour effet de simplifier le code. La version thermique, présentée dans l'article G, a nécessité une refonte partielle du schéma d'exécution habituel pour éviter les problèmes liés à la pénurie de mémoire partagée. Les performances sont de l'ordre de 2 000 MLUPS en simple précision en utilisant huit Tesla C1060, ce qui est comparable aux performances réalisées sur un ordinateur Blue Gene/P équipé de 4096 cœurs par un code optimisé en double précision. Nos études montrent que l'efficacité de parallélisation dépasse généralement les 80 % et que dans la plupart des cas, même pour des domaines de taille modeste, la communication inter-GPU n'est pas un facteur limitant.

Dès que le nombre de sous-domaines employés devient conséquent, l'utilisation d'une partition tridimensionnelle permet de réduire de façon drastique le

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

volume des communications. Le procédé d'échange de données employé pour les versions simple nœud n'est donc pas pertinent pour les implantations de la LBM sur grappe de GPU. Dans l'article I, nous présentons une nouvelle approche permettant une communication efficace sur toutes les faces des sous-domaines. Au lieu d'affecter un fil d'exécution à chaque nœud, nous utilisons des blocs unidimensionnels contenant une seule trame et associés à des groupes de nœuds formant une tuile carrée. Le recours à des tableaux auxiliaires permet d'échanger efficacement des données quelle que soit la direction spatiale considérée. Notre implantation pour grappe de processeurs graphiques utilise un jeu de routines MPI[19] pour assurer la communication inter-GPU. L'affectation des sous-domaines aux cartes de calcul de la grappe se fait par l'intermédiaire d'un fichier de configuration au format JSON[20].

Nous avons testé notre solveur sur une grappe de calcul regroupant neuf nœuds, chacun contenant trois Tesla M2070 (ou M2090). En utilisant 27 GPU, nous avons pu atteindre 10 280 MLUPS en simple précision sur environ 796 millions de nœuds. Nos résultats en termes de performances dépassent largement ceux, publiés récemment, qui ont été obtenus avec le super-ordinateur TSUBAME. Nos études montrent que la scalabilité, tant au sens fort qu'au sens faible, est très satisfaisante, bien que des mesures additionnelles sur des grappes plus importantes restent à effectuer. Notre solveur LBM pour grappe de GPU paraît d'ores et déjà constituer un outil prometteur pour la réalisation de simulations à très grande échelle.

VI Applications et perspectives

Le solveur LBM thermique multi-GPU décrit dans l'article G, ou éventuellement une extension à la thermique de notre récente implantation sur grappe de GPU, pourrait être un composant majeur dans la mise en œuvre de simulations réalistes en aéraulique interne. Néanmoins, les contraintes en termes de résolution spatiale sont si élevées que, même avec la puissance de calcul apportée par les processeurs graphiques, la simulation numérique directe n'est pas envisageable. Le seul chemin actuellement praticable semble être l'utilisation de maillages raffinés couplés à un modèle de turbulence adapté à la LBM thermique. L'ajout de la prise en charge du raffinement de maillage dans nos solveurs est donc d'une importance cardinale, quand bien même l'impact sur les performances pourrait être significatif. Les effets radiatifs devraient eux aussi être pris en compte, tout particulièrement dans les simulations de bâtiments à haute efficacité énergétique pour lesquels l'apport solaire est habituellement considérable. Plusieurs approches permettant de simuler les transferts radiatifs sur GPU ont été développées et devraient pouvoir être couplées à des solveurs LBM sur GPU.

[19] Message Passing Interface.
[20] JavaScript Object Notation.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Résumé détaillé

Concernant l'aéraulique externe des bâtiments, le solveur LES-LBM isotherme multi-GPU décrit dans l'article F se révèle être un outil convaincant pour la simulation des écoulements au voisinage d'un bâtiment voire d'un groupe de bâtiments, quand bien même des conditions aux limites et des modèles de turbulence alternatifs devraient être testés. Il pourrait contribuer à l'évaluation de paramètres tels que des coefficients de pression pour diverses configurations ou à la réalisation d'études d'environnement aéraulique urbain. Dans cette perspective, le couplage entre des systèmes d'information géographique et le module de géométrie de la plateforme TheLMA serait d'un grand intérêt. En ce qui concerne la thermo-aéraulique externe, la situation semble d'un abord encore plus difficile que dans le cas de simulations internes, étant donné qu'à ce niveau de nombres de Reynolds, l'épaisseur de la sous-couche visqueuse peut descendre jusqu'à 100 μm. L'évaluation de paramètres tels que les coefficients d'échanges convectifs pourrait donc nécessiter des temps de calculs très conséquents.

La plateforme TheLMA a été développée de façon à être la plus générique possible en dépit des limitations induites par la technologie CUDA. Adapter TheLMA à la résolution de schémas similaires comme la méthode de compressibilité artificielle sur réseau (LW-ACM[21]) récemment proposée, ou à la résolution d'équations aux dérivées partielles telles que l'équation de diffusion ou l'équation de Laplace, ne devrait pas présenter de difficultés majeures. Les architectures massivement parallèles alternatives, comme le MIC d'Intel ou le MMPA de Kalray, possèdent de nombreux points communs avec les GPU, en particulier en ce qui concerne les mécanismes de transfert de données. L'extension de notre plateforme à ces processeurs, bien que nécessitant probablement la réécriture de larges parties du code, devrait bénéficier des stratégies générales d'optimisation mises en place pour l'implantation de la LBM sur processeurs graphiques.

[21] Link-wise artificial compressibility method.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article A
A New Approach to the Lattice Boltzmann Method for Graphics Processing Units

Computers and Mathematics with Applications, 61(12):3628–3638, June 2011
Accepted January 29, 2010

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article B

Global Memory Access Modelling for Efficient Implementation of the Lattice Boltzmann Method on Graphics Processing Units

Lecture Notes in Computer Science 6449, High Performance Computing for Computational Science – VECPAR 2010 Revised Selected Papers, pages 151–161, February 2011
Accepted October 10, 2010

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article C
The TheLMA Project: A Thermal Lattice Boltzmann Solver for the GPU

Computers and Fluids, 54:118–126, January 2012
Accepted October 8, 2011

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article D
Multi-GPU Implementation of the Lattice Boltzmann Method

Computers and Mathematics with Applications, published online March 17, 2011
Accepted February 14, 2011

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article E
The TheLMA project: Multi-GPU Implementation of the Lattice Boltzmann Method

International Journal of High Performance Computing Applications,
25(3):295–303, August 2011
Accepted April 26, 2011

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article F
Towards Urban-Scale Flow Simulations using
the Lattice Boltzmann Method

Proceedings of the Building Simulation 2011 Conference, pages 933–940. IBPSA, 2011
Accepted September 6, 2011

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article G

Multi-GPU Implementation of a Hybrid Thermal Lattice Boltzmann Solver using the TheLMA Framework

Computers and Fluids, published online March 6, 2012
Accepted February 20, 2012

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article H

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

Computers and Mathematics with Applications, published online June 28, 2012
Accepted May 23, 2012

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Article I

Scalable Lattice Boltzmann Solvers for CUDA GPU Cluster

Submitted to *Parallel Computing*, August 22, 2012

Abstract

The lattice Boltzmann method (LBM) is an innovative and promising approach in computational fluid dynamics. From an algorithmic standpoint it reduces to a regular data parallel procedure and is therefore well-suited to high performance computations. Numerous works report efficient implementations of the LBM for the GPU, but very few mention multi-GPU versions and even fewer GPU cluster implementations. Yet, to be of practical interest, GPU LBM solvers need to be able to perform large scale simulations. In the present contribution, we describe an efficient LBM implementation for CUDA GPU clusters. Our solver consists of a set of MPI communication routines and a CUDA kernel specifically designed to handle three-dimensional partitioning of the computation domain. Performance measurement were carried out on a small cluster. We show that the results are satisfying, both in terms of data throughput and parallelisation efficiency.

Keywords: GPU clusters, CUDA, lattice Boltzmann method

1 Introduction

A single-GPU based computing device is not proper to solve large scale problems because of the limited amount of on-board memory. However, applications running on multiple GPUs have to face the PCI-E bottleneck, and great care has to be taken in design and implementation to minimise inter-GPU communication. Such constraints may be rather challenging; the well-known MAGMA [14] linear algebra library, for instance, only added support for single-node multiple GPUs with the latest version (i.e. 1.1), two years after the first public release.

The lattice Boltzmann method (LBM) is a novel approach in computational fluid dynamics (CFD), which, unlike most other CFD methods, does not consist in directly solving the Navier-Stokes equations by a numerical procedure [7]. Beside many interesting features, such as the ability to easily handle complex geometries, the LBM reduces to a regular data-parallel algorithm and therefore, is well-suited to efficient HPC implementations. As a matter of fact, numerous successful attempts to implement the LBM for the GPU have been reported in the recent years, starting with the seminal work of Li *et al.* in 2003 [8].

CUDA capable computation devices may at present manage up to 6 GB of memory. This capacity allows the GPU to process at most 8.5×10^7 nodes running a standard three-dimensional LBM solver in single-precision. Taking architectural constraints into account, the former amount is sufficient to store a 416^3 cubic lattice. Although large, such a computational domain is likely to be too coarse to perform direct numerical simulation of a fluid flow in many practical situations as, for instance, urban-scale building aeraulics or thermal modeling of electronic circuit boards.

To our knowledge, the few single-node multi-GPU LBM solvers described in literature all use a one-dimensional (1D) partition of the computation domain, which is relevant given the small number of involved devices. This option does not require any data reordering, provided the appropriate partitioning direction is chosen, thus keeping the computation kernel fairly simple. For a GPU cluster implementation, on the contrary, a kernel able to run on a three-dimensional (3D) partition seems preferable, since it would both provide more flexibility for load balancing and contribute to reduce the volume of communication.

In the present contribution, we describe an implementation of a lattice Boltzmann solver for CUDA

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

GPU clusters. The core computation kernel is designed so as to import and export data efficiently in each spatial direction, thus enabling the use of 3D partitions. The inter-GPU communication is managed by MPI-based routines. This work constitutes the latest extension to the TheLMA project [10], which aims at providing a comprehensive framework for efficient GPU implementations of the LBM.

The remainder of the paper is structured as follows. In Section 2, we give a description of the algorithmic aspects of the LBM as well as a short review of LBM implementations for the GPU. The third section consists of a detailed description of the implementation principles of the computation kernel and the communication routines of our solver. In the fourth section, we present some performance results on a small cluster. The last section concludes and discusses possible extensions to the present work.

2 State of the art

2.1 Lattice Boltzmann Method

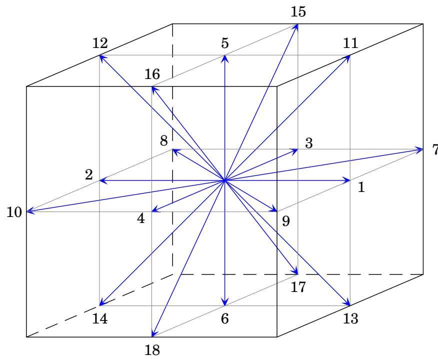


Figure 1: The D3Q19 stencil — *The blue arrows represent the propagation vectors of the stencil linking a given node to some of its nearest neighbours.*

The lattice Boltzmann method is generally carried out on a regular orthogonal mesh with a constant time step δt. Each node of the lattice holds a set of scalars $\{f_\alpha \,|\, \alpha = 0, \ldots N\}$ representing the local particle density distribution. Each particle density f_α is associated with a particle velocity ξ_α and a *propagation vector* $\mathbf{c}_\alpha = \delta t \cdot \xi_\alpha$. Usually the propagation vectors link a given node to one of its nearest neighbours, except for $\mathbf{c}_0$ which is null. For the present work, we implemented the D3Q19 propagation stencil illustrated in Fig. 1. This stencil, which contains 19 elements, is the most commonly used in practice for 3D LBM, being the best trade-off between size and isotropy. The governing equation of the LBM at node $\boldsymbol{x}$ and time t writes:

$$\left|f_\alpha(\boldsymbol{x}+\mathbf{c}_\alpha, t+\delta t)\right\rangle - \left|f_\alpha(\boldsymbol{x}, t)\right\rangle = \Omega\Big(\left|f_\alpha(\boldsymbol{x}, t)\right\rangle\Big), \quad (1)$$

where $\left|f_\alpha\right\rangle$ denotes the distribution vector and Ω denotes the so-called *collision operator*. The mass density ρ and the momentum $\boldsymbol{j}$ of the fluid are given by:

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

In our solver, we implemented the multiple-relaxation-time collision operator described in [3]. Further information on the physical and numerical aspects of the method are to be found in the aforementioned reference. From an algorithmic perspective, Eq. 1 naturally breaks in two elementary steps:

$$\left|\widetilde{f}_\alpha(\boldsymbol{x}, t)\right\rangle = \left|f_\alpha(\boldsymbol{x}, t)\right\rangle + \Omega\Big(\left|f_\alpha(\boldsymbol{x}, t)\right\rangle\Big), \quad (3)$$

$$\left|f_\alpha(\boldsymbol{x}+\mathbf{c}_\alpha, t+\delta t)\right\rangle = \left|\widetilde{f}_\alpha(\boldsymbol{x}, t)\right\rangle. \quad (4)$$

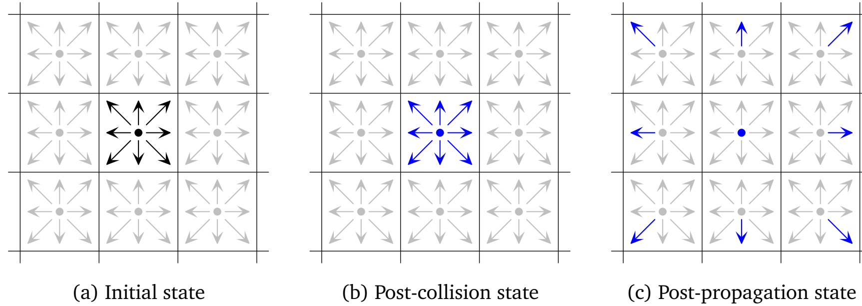
Equation 3 describes the *collision* step in which an updated particle distribution is computed. Equation 4 describes the *propagation* step in which the updated particle densities are transferred to the neighbouring nodes. This two-step process is outlined by Fig. 2 (in the two-dimensional case, for the sake of clarity).

2.2 GPU implementations of the LBM

Due to substantial evolution of hardware, the pioneering work of Fan *et al.* [4] reporting a GPU cluster LBM implementation is only partially relevant today. The GPU computations were implemented using pre-CUDA techniques that are now obsolete. Yet, the proposed optimisation of the communication pattern still applies, although it was only tested on Gigabyte Ethernet; in future work, we plan to evaluate its impact using InfiniBand interconnect.

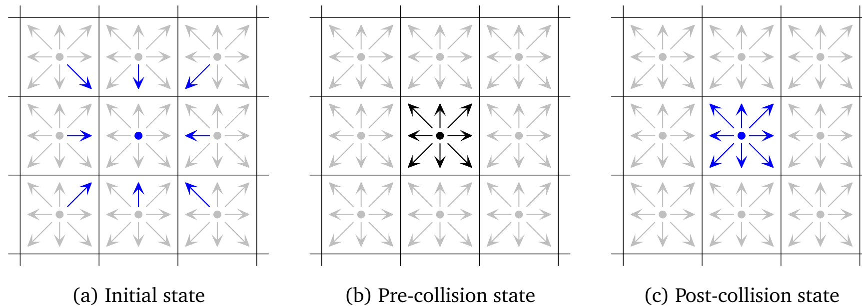
In 2008, Tölke and Krafczyk [15] described a single-GPU 3D-LBM implementation using CUDA. The authors mainly try to address the problem induced by misaligned memory accesses. As a matter

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés



(a) Initial state (b) Post-collision state (c) Post-propagation state

Figure 2: Collision and propagation — *The collision step is represented by the transition between (a) and (b). The pre-collision particle distribution is drawn in black whereas the post-collision one is drawn in blue. The transition from (b) to (c) illustrates the propagation step in which the updated particle distribution is advected to the neighbouring nodes.*



(a) Initial state (b) Pre-collision state (c) Post-collision state

Figure 3: In-place propagation — *With the in-place propagation scheme, contrary to the out-of-place scheme outlined in Fig. 2, the updated particle distribution of the former time step is advected to the current node before collision.*

of fact, with the NVIDIA G80 GPU available at this time, only aligned and ordered memory transactions could be coalesced. The proposed solution consists in partially performing propagation in shared memory. With the GT200 generation, this approach is less relevant, since misalignment has a lower—though not negligible—impact on performance. As shown in [11], the misalignment overhead is significantly higher for store operations than for read operations. We therefore suggested in [12] to use the in-place propagation scheme outlined by Fig. 3 instead of the ordinary out-of-place propagation scheme illustrated in Fig. 2. The resulting computation kernel is simpler and leaves the shared memory free for possible extensions.

Further work led us to develop a single-node multi-GPU solver, with 1D partitioning of the computation domain [13]. Each CUDA device is managed by a specific POSIX thread. Inter-GPU communication is carried out using zero-copy transactions to page-locked host memory. Performance and scalability are satisfying with up to 2,482 million lattice node updates per second (MLUPS) and 90.5% parallelisation efficiency on a 384^3 lattice using eight Tesla C1060 computing devices in single-precision.

In their recent paper [16], Wang and Aoki describe an implementation of the LBM for CUDA GPU clusters. The partition of the computation domain may be either one-, two-, or three-dimensional. Although the authors are elusive on this point, no special care seems to be taken to optimise data transfer between device and host memory, and as a matter of fact, performance is quite low. For instance, on a 384^3 lattice with 1D partitioning, the authors report

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

about 862 MLUPS using eight GT200 GPUs in single-precision, i.e. about one third of the performance of our own solver on similar hardware. It should also be noted that the given data size for communication per rank, denoted M_{1D}, M_{2D}, and M_{3D}, are at least inaccurate. For the 1D and 2D cases, no account is taken of the fact that for the simple bounce-back boundary condition, no external data is required to process boundary nodes. In the 3D case, the proposed formula is erroneous.

3 Proposed implementation

3.1 Computation kernel

To take advantage of the massive hardware parallelism, our single-GPU and our single-node multi-GPU LBM solvers both assign one thread to each node of the lattice. The kernel execution set-up consists of a two-dimensional grid of one-dimensional blocks, mapping the spatial coordinates. The lattice is stored as a four-dimensional array, the direction of the blocks corresponding to the minor dimension. Two instances of the lattice are kept in device memory, one for even time steps and one for odd time steps, in order to avoid local synchronisation issues. The data layout allows the fetch and store operations issued by the warps to be coalesced. It also makes possible, using coalesced zero-copy transactions, to import and export data efficiently at the four sub-domain faces parallel to the blocks, with partial overlapping of communication and computations. For the two sub-domain faces normal to the blocks, the data is scattered across the array and needs reordering to be efficiently exchanged.

A possible solution to extend our computation kernel to support 3D partitions would be to use a specific kernel to handle the interfaces normal to the blocks. Not mentioning the overhead of kernel switching, this approach does not seem satisfying since such a kernel would only perform non-coalesced read operations. However, the minimum data access size is 32 bytes for compute capability up to 1.3, and 128 bytes above, whereas only 4 or 8 bytes would be useful. The cache memory available in devices of compute capability 2.0 and 2.1 is likely to have small impact in this case, taking into account the scattering of accessed data.

We therefore decided to design a new kernel able to perform propagation and reordering at once. With this new kernel, blocks are still one-dimensional but, instead of spanning the lattice width, contain only one warp, i.e. $W = 32$ threads for all existing CUDA capable GPUs. Each block is assigned to a tile of nodes of size $W \times W \times 1$, which imposes for the sub-domain dimensions to be a multiple of W in the x- and y-direction. For the sake of clarity, let us call *lateral densities* the particle densities crossing the tile sides parallel to the y-direction. These lateral densities are stored in an auxiliary array in device memory. At each time step, the lateral densities are first loaded from the auxiliary array into shared memory, then the kernel loops over the tile row by row to process the nodes saving the updated lateral densities in shared memory, last the updated lateral densities are written back to device memory. This process is summarised in Fig. 4. Note that we only drew a 8×8 tile in order to improve readability.

Using this new kernel, the amount of 4-byte (or 8-byte) words read or written per block and per time step is :

$$Q_T = 2\left(19W^2 + 10W\right) = 38W^2 + 20W, \quad (5)$$

and the amount of data read and written in device memory per time step for a $S_x \times S_y \times S_z$ sub-domain is:

$$Q_S = \frac{S_x}{W} \times \frac{S_y}{W} \times S_z \times Q_T = S_x S_y S_z \frac{38W + 20}{W}. \quad (6)$$

We therefore see that this approach only increases the volume of device memory accesses by less than 2%, with respect to our former implementations [12], while greatly reducing the number of misaligned transactions. Yet, most important is the fact that it makes possible to exchange data efficiently at the interfaces normal to the blocks. The procedure simply consists in replacing, for the relevant tiles, accesses to the auxiliary array with coalesced zero-copy transactions to host memory. The maximum amount of data read and written to host memory per time step is:

$$\begin{aligned} Q_H &= 2(S_x S_y + S_y S_z + S_z S_x) \times 5 \\ &= 10(S_x S_y + S_y S_z + S_z S_x). \quad (7) \end{aligned}$$

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

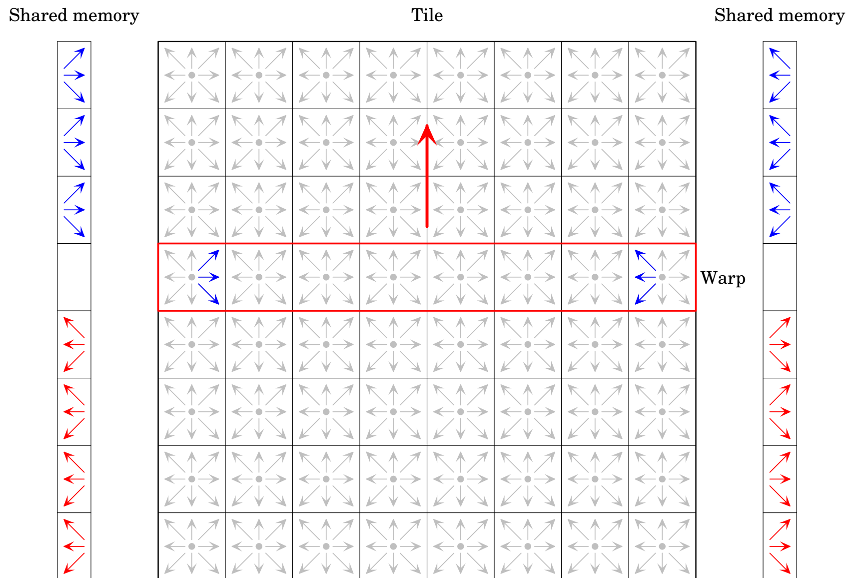


Figure 4: Processing of a tile — *Each tile of nodes is processed row by row by a CUDA block composed of a single warp. The current row is framed in red and the direction of the processing is indicated by the bold red arrow. The in-coming lateral densities are drawn in blue whereas the out-going ones are drawn in red. During the execution of the loop, these densities are stored temporarily in an auxiliary array hosted in shared memory.*

3.2 Multi-GPU solver

To enable our kernel to run across a GPU cluster, we wrote a set of MPI-based initialisation and communication routines. These routines as well as the new computation kernel were designed as components of the TheLMA framework, which was first developed for our single-node multi-GPU LBM solver. The main purpose of TheLMA is to improve code reusability. It comes with a set of generic modules providing the basic features required by a GPU LBM solver. This approach allowed us to develop our GPU cluster implementation more efficiently.

At start, the MPI process of rank 0 is responsible for loading a configuration file in JSON format. Beside general parameters, such as the Reynolds number for the flow simulation or the graphical output option flag, this configuration file mainly describes the execution set-up. Listing 1 gives an example file for a $2 \times 2 \times 1$ partition running on two nodes. The parameters for each sub-domains, such as the size or the target node and computing device, are given in the `Subdomains` array. The `Faces` and `Edges` arrays specify to which sub-domains a given sub-domain is linked, either through its faces or edges. These two arrays follow the same ordering as the propagation vector set displayed in Fig. 1. Being versatile, the JSON format is well-suited for our application. Moreover, its simplicity makes both parsing and automatic generation straightforward. This generic approach brings flexibility. It allows any LBM solver based on our framework to be tuned to the target architecture.

Once the configuration file is parsed, the MPI processes register themselves by sending their MPI processor name to the rank 0 process, which in turn assigns an appropriate sub-domain to each of them and sends back all necessary parameters. The processes then perform local initialisation, setting the assigned CUDA device and allocating the communication buffers, which fall into three categories: send buffers, receive buffers and read buffers. It is worth noting that both send buffers and read buffers consist of pinned memory allocated using the CUDA API, since they have to be made accessible by the GPU.

The steps of the main computation loop consist of a kernel execution phase and a communication phase. During the first phase, the out-going particle densities are written to the send buffers assigned to the

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

```
{
    "Path": "out",
    "Prefix": "ldc",
    "Re": 1E3,
    "U0": 0.1,
    "Log": true,
    "Duration": 10000,
    "Period": 100,
    "Images": true,
    "Subdomains": [
        {
              "Id": 0,
              "Host": "node00",
              "GPU": 0,
              "Offset": [0, 0, 0],
              "Size": [128, 128, 256],
              "Faces": [   1, null,    2, null, null, null],
              "Edges": [   3, null, null, null, null, null,
                        null, null, null, null, null, null]
        },
        {
              "Id": 1,
              "Host": "node00",
              "GPU": 1,
              "Offset": [128, 0, 0],
              "Size": [128, 128, 256],
              "Faces": [null,    0,    3, null, null, null],
              "Edges": [null,    2, null, null, null, null,
                        null, null, null, null, null, null]
        },
        {
              "Id": 2,
              "Host": "node01",
              "GPU": 0,
              "Offset": [0, 128, 0],
              "Size": [128, 128, 256],
              "Faces": [   3, null, null,    0, null, null],
              "Edges": [null, null,    1, null, null, null,
                        null, null, null, null, null, null]
        },
        {
              "Id": 3,
              "Host": "node01",
              "GPU": 1,
              "Offset": [128, 128, 0],
              "Size": [128, 128, 256],
              "Faces": [null,    2, null,    1, null, null],
              "Edges": [null, null, null,    0, null, null,
                        null, null, null, null, null, null]
        }
    ]
}
```

Listing 1: Configuration file

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

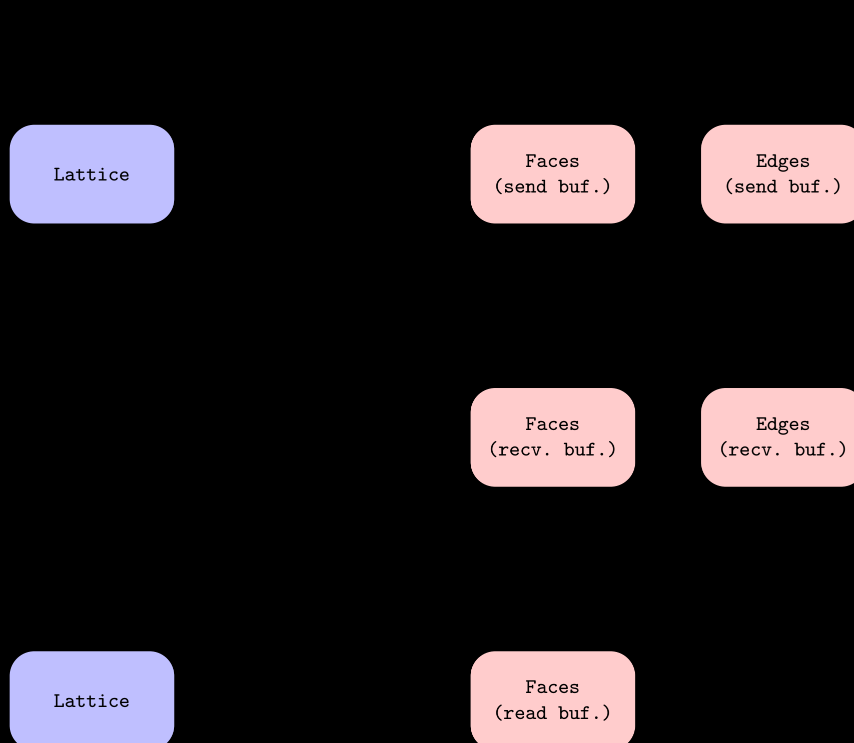


Figure 5: Communication phase — *The upper part of the graph outlines the path followed by data leaving the sub-domain handled by GPU 0. For each face of the sub-domain, the out-going densities are written by the GPU to pinned buffers in host memory. The associated MPI process then copies the relevant densities into the edge buffers and sends both face and edge buffers to the corresponding MPI processes. The lower part of the graph describes the path followed by data entering the sub-domain handled by GPU 1. Once the reception of in-coming densities for faces and edges is completed, the associated MPI process copies the relevant data for each face of the sub-domain into pinned host memory buffers, which are read by the GPU during kernel execution.*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

faces *without* performing any propagation as for the densities written in device memory. During the second phase, the following operations are performed:

1. The relevant densities are copied to the send buffers assigned to the edges.
2. Asynchronous send requests followed by synchronous receive requests are issued.
3. Once message passing is completed, the particle densities contained in the receive buffers are copied to the read buffers.

This communication phase is outlined in Fig. 5. The purpose of the last operation is to perform propagation for the in-coming particle densities. As a result, the data corresponding to a face and its associated edges is gathered in a single read buffer. This approach avoids misaligned zero-copy transactions, and most important, leads to a simpler kernel since only six buffers at most have to be read. It should be mentioned that the read buffers are allocated using the *write combined* flag to optimise cache usage. According to [9, 6, 5], this setting is likely to improve performance since the memory pages are locked.

4 Performance study

We conducted experiments on an eight-node GPU cluster, each node being equipped with two hexa-core X5650 Intel Xeon CPUs, 36 GB memory, and three NVIDIA Tesla M2070 computing devices; the network interconnect uses QDR InfiniBand. To evaluate raw performance, we simulated a lid-driven cavity [1] in single-precision and recorded execution times for 10,000 time steps using various configurations. Overall performance is good, with at most 8,928 million lattice node updates per second (MLUPS) on a 768^3 lattice using all 24 GPUs. To set a comparison, Wang and Aoki in [16] report at most 7,537 MLUPS for the same problem size using four time as many GPUs. However, it should be mentioned that these results were obtained using hardware of the preceding generation.

The solver was compiled using CUDA 4.0 and OpenMPI 1.4.4. It is also worth mentioning that the computing devices had ECC support enabled. From tests we conducted on a single computing device, we expect the overall performance to be about 20% higher with ECC support disabled.

4.1 Performance model

Our first performance benchmark consisted in running our solver using eight GPUs on a cubic cavity of increasing size. The computation domain is split in a $2 \times 2 \times 2$ regular partition, the size S of the sub-domains ranging from 128 to 288. In addition, we recorded the performance for a single-GPU on a domain of size S, in order to evaluate the communication overhead and the GPU to device memory data throughput. The results are gathered in Tables 1 and 2.

Table 1 shows that the data throughput between GPU and device memory is stable, only slightly increasing with the size of the domain. (Given the data layout in device memory, the increase of the domain size is likely to reduce the amount of L2 cache misses, having therefore a positive impact on data transfer.) We may therefore conclude that the performance of our kernel is communication bound. The last column accounts for the ratio of the data throughput to the maximum sustained throughput, for which we used the value 102.7 GB/s obtained using the `bandwidthTest` program that comes with the CUDA SDK. The obtained ratios are fairly satisfying taking into account the complex data access pattern the kernel must follow.

In Tab. 2, the parallel efficiency and the non-overlapped communication time were computed using the single-GPU results. The efficiency is good with at least 87.3%. The corresponding data throughput given in the last column is slightly increasing from 8.2 GB/s to 9.9 GB/s, which implies that the proportion of communication overhead decreases when the domain size increases. The efficiency appears to benefit from surface-to-volume effects. Figure 6 displays the obtained performance results.

4.2 Scalability

In order to study scalability, both weak and strong, we considered seven different partition types with increasing number of sub-domains. Weak scalability represents the ability to solve larger problems with larger resources whereas strong scalability accounts for the ability to solve a problem faster using more resources. For weak scalability, we used cubic sub-domains of size 128, and for strong scalability, we used a computation domain of constant size 384 with

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Domain size (S)	Runtime (s)	Performance (MLUPS)	Device throughput (GB/s)	Ratio to peak throughput
128	54.7	383.2	59.2	57.6%
160	100.6	407.2	62.9	61.2%
192	167.6	422.3	65.2	63.5%
224	260.3	431.8	66.7	64.9%
256	382.3	438.8	67.8	66.0%
288	538.7	443.4	68.5	66.7%

Table 1: Single-GPU performance

Domain size ($2S$)	Runtime (s)	Performance (MLUPS)	Parallel efficiency	Data transfer (s)	Throughput (GB/s)
256	62.7	2,678	87.3%	7.9	9.9
320	114.5	2,862	87.9%	13.9	8.9
384	186.9	3,030	89.7%	19.3	9.6
448	289.6	3,105	89.9%	29.3	8.2
512	418.7	3,206	91.3%	36.4	8.6
576	587.0	3,256	91.8%	48.3	8.2

Table 2: Performance for a $2 \times 2 \times 2$ regular partition

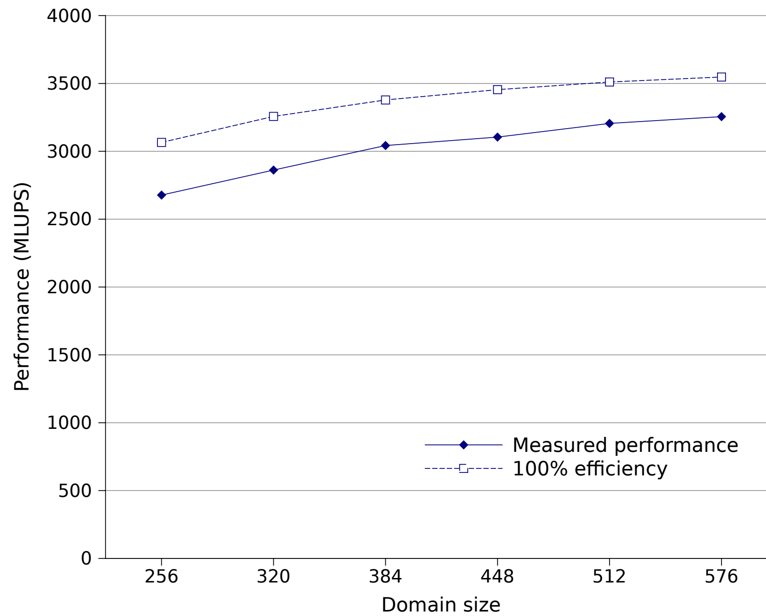


Figure 6: Performance for a $2 \times 2 \times 2$ regular partition

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

cuboid sub-domains. Table 3 gives all the details of the tested configurations.

For our weak scaling test, we use fixed size sub-domains so that the amount of processed nodes linearly increases with the number of GPUs. We chose a small, although realistic, sub-domain size in order to reduce as much as possible favourable surface-to-volume effects. Since the workload per GPU is fixed, perfect scaling is achieved when the runtime remains constant. The results of the test are gathered in Tab. 4. Efficiency was computed using the runtime of the smallest tested configuration. Figure 7 displays the runtime with respect to the number of GPUs. As illustrated by this diagram, the weak scalability of our solver is satisfying, taking into account that the volume of communication increases by a factor up to 11.5.

In our strong scalability test, we consider a fixed computation domain processed using an increasing number of computing devices. As a consequence the volume of the communication increases by a factor up to three, while the size of the sub-domains decreases, leading to less favourable configurations for the computation kernel. The results of the strong scaling test are given in Tab. 5. The runtime with respect to the number of GPUs is represented in Fig. 8 using a log-log diagram. As shown by the trend-line, the runtime closely obeys a power law, the correlation coefficient for the log-log regression line being below -0.999. The obtained scaling exponent is approximately -0.8, whereas perfect strong scalability corresponds to an exponent of -1. We may conclude that the strong scalability of our code is good, given the fairly small size of the computation domain.

5 Conclusion

In this paper, we describe the implementation of an efficient and scalable LBM solver for GPU clusters. Our code lies upon three main components that were developed for that purpose: a CUDA computation kernel, a set of MPI initialisation routines, and a set of MPI communication routines. The computation kernel's most important feature is the ability to efficiently exchange data in all spatial directions, making possible the use of 3D partitions of the computation domain. The initialisation routines are designed in order to distribute the workload across the cluster in a flexible way, following the specifications contained in a configuration file. The communication routines manage to pass data between sub-domains efficiently, performing reordering and partial propagation. These new components were devised as key parts of the TheLMA framework[10], whose main purpose is to facilitate the development of LBM solvers for the GPU. The obtained performance on rather affordable hardware such as small GPU clusters makes possible to carry out large scale simulations in reasonable time and at moderate cost. We believe these advances will benefit to many potential applications of the LBM. Moreover, we expect our approach to be sufficiently generic to apply to a wide range of stencil computations, and therefore to be suitable for numerous applications that operate on a regular grid.

Although performance and scalability of our solver is good, we believe there is still room for improvement. Possible enhancements include better overlapping between communication and computation, and more efficient communication between sub-domains. As for now, only transactions to the send and read buffers may overlap kernel computations. The communication phase starts once the computation phase is completed. One possible solution to improve overlapping would be to split the sub-domains in seven zones, six external zones, one for each face of the sub-domains, and one internal zone for the remainder. Processing the external zones first would allow the communication phase to start while the internal zone is still being processed.

Regarding ameliorations to the communication phase, we are considering three paths to explore. First of all, we plan to reinvest the concepts presented in [6] and [5] to improve data transfers involving page-locked buffers. Secondly, we intend to evaluate the optimisation proposed by Fan *et al.* in [4], which consists in performing data exchange in several synchronous steps, one for each face of the sub-domains, the data corresponding to the edges being transfered in two steps. Last, following [2], we plan to implement a benchmark program able to search heuristically efficient execution layouts for a given computation domain and to generate automatically the configuration file corresponding to the most efficient one.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Number of GPUs	Nodes × GPUs	Partition type	Domain (weak scalability)	Sub-domains (strong scalability)
4	2 × 2	1 × 2 × 2	128 × 256 × 256	384 × 192 × 192
6	2 × 3	1 × 3 × 2	128 × 384 × 256	384 × 128 × 192
8	4 × 2	2 × 2 × 2	256 × 256 × 256	192 × 192 × 192
12	4 × 3	2 × 3 × 2	256 × 384 × 256	192 × 128 × 192
16	8 × 2	2 × 4 × 2	256 × 512 × 256	192 × 96 × 192
18	6 × 3	2 × 3 × 3	256 × 384 × 384	192 × 128 × 128
24	8 × 3	2 × 4 × 3	256 × 512 × 384	192 × 96 × 128

Table 3: Configuration details for the scaling tests

Number of GPUs	Runtime (s)	Efficiency	Performance (MLUPS)	Perf. per GPU (MLUPS)
4	59.8	100%	1402	350.5
6	64.2	93%	1959	326.6
8	62.7	95%	2676	334.5
12	66.8	90%	3767	313.9
16	71.1	84%	4721	295.1
18	67.0	89%	5634	313.0
24	73.2	82%	6874	286.4

Table 4: Runtime and efficiency for the weak scaling test

Number of GPUs	Runtime (s)	Efficiency	Performance (MLUPS)	Perf. per GPU (MLUPS)
4	335.0	100%	1690	422.6
6	241.9	92%	2341	390.1
8	186.1	90%	3043	380.3
12	134.7	83%	4204	350.3
16	109.9	76%	5152	322.0
18	98.4	76%	5753	319.6
24	80.3	70%	7053	293.9

Table 5: Runtime and efficiency for the strong scaling test

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

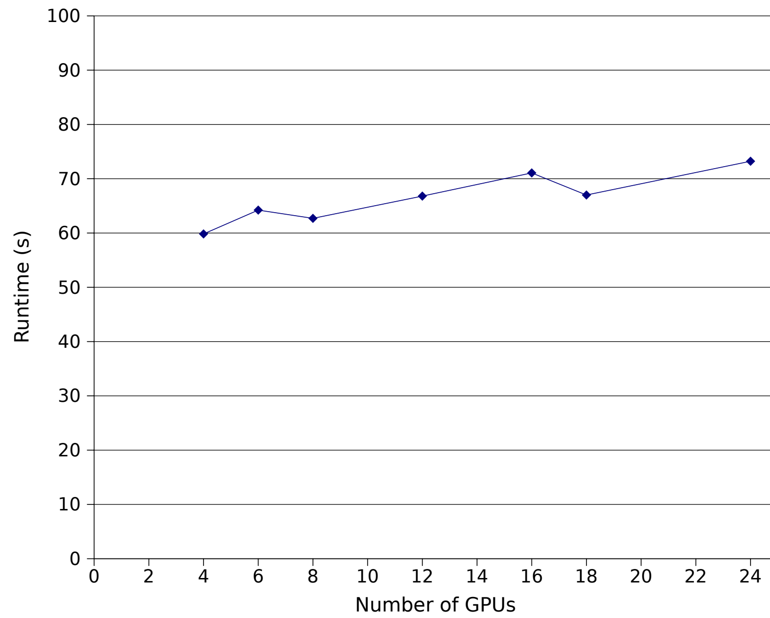


Figure 7: Runtime for the weak scaling test — *Perfect weak scaling would result in an horizontal straight line.*

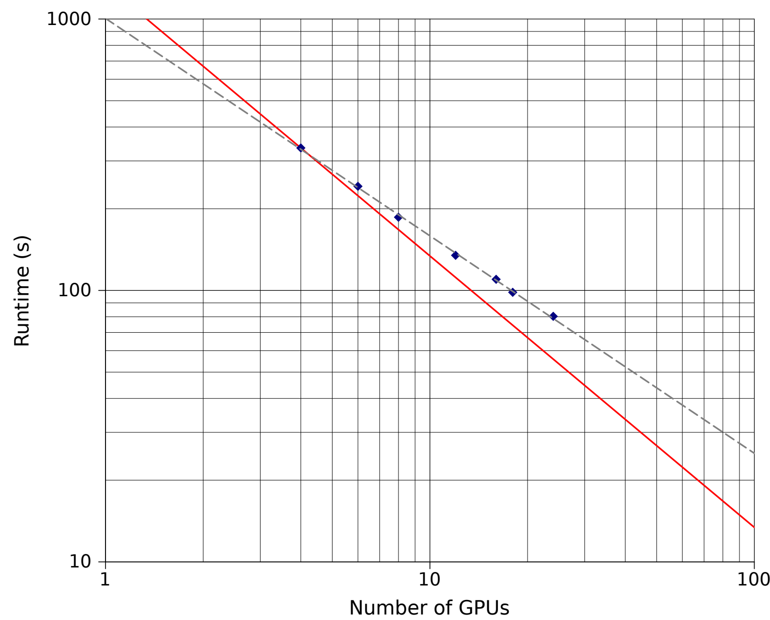


Figure 8: Runtime for the strong scaling test — *Perfect strong scaling is indicated by the solid red line*

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Acknowledgments

The authors wish to thank the INRIA PlaFRIM team for allowing us to test our executables on the Mirage GPU cluster.

References

[1] S. Albensoeder and H. C. Kuhlmann. Accurate three-dimensional lid-driven cavity flow. *Journal of Computational Physics*, 206(2):536–558, 2005.

[2] R. Clint Whaley, A. Petitet, and J.J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1):3–35, 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 of the Royal Society A*, 360:437–451, 2002.

[4] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. GPU cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, pages 47–58. IEEE, 2004.

[5] P. Geoffray, C. Pham, and B. Tourancheau. A Software Suite for High-Performance Communications on Clusters of SMPs. *Cluster Computing*, 5(4):353–363, 2002.

[6] P. Geoffray, L. Prylli, and B. Tourancheau. BIP-SMP: High performance message passing over a cluster of commodity SMPs. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, pages 20–38. ACM, 1999.

[7] 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):6811–6817, 1997.

[8] W. Li, X. Wei, and A. Kaufman. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19(7):444–456, 2003.

[9] NVIDIA. *Compute Unified Device Architecture Programming Guide version 4.0*, 2011.

[10] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. Thermal LBM on Many-core Architectures. Available on www.thelma-project.info, 2010.

[11] 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*, 61(12):3628–3638, 2011.

[12] 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 *Lecture Notes in Computer Science 6449, High Performance Computing for Computational Science, VECPAR 2010 Revised Selected Papers*, pages 151–161. Springer, 2011.

[13] 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, 2011.

[14] F. Song, S. Tomov, and J. Dongarra. Efficient Support for Matrix Computations on Heterogeneous Multi-core and Multi-GPU Architectures. Technical Report UT-CS-11-668, University of Tennessee, 2011.

[15] 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.

[16] X. Wang and T. Aoki. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. *Parallel Computing*, 37(9):521–535, 2011.

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés

FOLIO ADMINISTRATIF

THÈSE SOUTENUE DEVANT L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

NOM : OBRECHT DATE de SOUTENANCE : 11 décembre 2012
(avec précision du nom de jeune fille, le cas échéant)
Prénoms : Christian, Charles

TITRE : High Performance Lattice Boltzmann Solvers on Massively Parallel Architectures with Applications to Building Aeraulics

NATURE : Doctorat Numéro d'ordre : 2012ISAL0134

École doctorale : MEGA (mécanique, énergétique, génie civil, acoustique)

Spécialité : Génie civil

RÉSUMÉ :

Avec l'émergence des bâtiments à haute efficacité énergétique, il est devenu indispensable de pouvoir prédire de manière fiable le comportement énergétique des bâtiments. Or, à l'heure actuelle, la prise en compte des effets thermo-aérauliques dans les modèles se cantonne le plus souvent à l'utilisation d'approches simplifiées voire empiriques qui ne sauraient atteindre la précision requise. Le recours à la simulation numérique des écoulements semble donc incontournable, mais il est limité par un coût calculatoire généralement prohibitif. L'utilisation conjointe d'approches innovantes telle que la méthode de Boltzmann sur gaz réseau (LBM) et d'outils de calcul massivement parallèles comme les processeurs graphiques (GPU) pourrait permettre de s'affranchir de ces limites. Le présent travail de recherche s'attache à en explorer les potentialités.

La méthode de Boltzmann sur gaz réseau, qui repose sur une forme discrétisée de l'équation de Boltzmann, est une approche explicite qui jouit de nombreuses qualités : précision, stabilité, prise en compte de géométries complexes, etc. Elle constitue donc une alternative intéressante à la résolution directe des équations de Navier-Stokes par une méthode numérique classique. De par ses caractéristiques algorithmiques, elle se révèle bien adaptée au calcul parallèle. L'utilisation de processeurs graphiques pour mener des calculs généralistes est de plus en plus répandue dans le domaine du calcul intensif. Ces processeurs à l'architecture massivement parallèle offrent des performances inégalées à ce jour pour un coût relativement modéré. Néanmoins, nombre de contraintes matérielles en rendent la programmation complexe et les gains en termes de performances dépendent fortement de la nature de l'algorithme considéré. Dans le cas de la LBM, les implantations GPU affichent couramment des performances supérieures de deux ordres de grandeur à celle d'une implantation CPU séquentielle faiblement optimisée.

Le présent mémoire de thèse est constitué d'un ensemble de neuf articles de revues internationales et d'actes de conférences internationales (le dernier étant en cours d'évaluation). Dans ces travaux sont abordés les problématiques liées tant à l'implantation mono-GPU de la LBM et à l'optimisation des accès en mémoire, qu'aux implantations multi-GPU et à la modélisation des communications inter-GPU et inter-nœuds. En complément, sont détaillées diverses extensions à la LBM indispensables pour envisager une utilisation en thermo-aéraulique des bâtiments. Les cas d'études utilisés pour la validation des codes permettent de juger du fort potentiel de cette approche en pratique.

MOTS-CLEFS : calcul intensif, méthode de Boltzmann sur gaz réseau, processeurs graphiques, aéraulique des bâtiments.

Laboratoire(s) de recherche : CETHIL (Centre de Thermique de Lyon)

Directeur de thèse : Jean-Jacques ROUX

Président de jury : Christian INARD

Composition du jury : Jean-Luc HUBERT, Christian INARD, Manfred KRAFCZYK, Frédéric KUZNIK, Jean ROMAN, Jean-Jacques ROUX, Bernard TOURANCHEAU

Cette thèse est accessible à l'adresse : http://theses.insa-lyon.fr/publication/2012ISAL0134/these.pdf
© [C. Obrecht], [2012], INSA de Lyon, tous droits réservés