Submitted to PDPTA'10, Corresponding Author: k.a.hawick@massey.ac.nz

Data-Parallelism and GPUs for Lattice Gas Fluid Simulations

M.G.B. Johnson, D.P. Playne and K.A. Hawick
Institute of Information and Mathematical Sciences
Massey University – Albany, North Shore 102-904, Auckland, New Zealand
Email: mitchelljohnson@ihug.co.nz, d.p.playne@massey.ac.nz, k.a.hawick@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

Abstract

Lattice gas cellular automata (LGCA) models provide a relatively fast means of simulating fluid flow and can give both quantitative and qualitative insights into flow patterns around complex obstacles. Symmetry requirements inherent in the Navier-Stokes equation mandate that lattice-gas approximations to the full field equations be run on triangular lattices in two dimensions and on a 3-D projection of a four dimensional face centred hyper-cubic for three dimensions. Graphics Processing Units (GPUs) offer powerful data-parallel processing capabilities for many simulations as well as the graphics calculations required to simulate them. We describe how GPUs can be used to implement mesh structures for simulating lattice gases. We present performance data on how to optimise data layout in the various levels of localised memory available in modern GPUs and discuss data transfer issues between CPU and GPU and between processing GPU and graphics GPU in a unified simulation platform. We illustrate these ideas with algorithmic fragments in Compute Unified Device Architecture (CUDA) - NVIDIA's GPU programming language.

Keywords: data parallelism; GPU; cellular automata; lattice gas; fluid simulation; triangular lattice; CUDA.

1 Introduction

Computational fluid dynamics is a powerful tool for exploring highly non-linear behaviour in complex fluid systems. Solving the Navier-Stokes equations, even in their simplest form is still a computationally intensive task and requires sophisticated numerical solver techniques. The lattice gas [1] cellular automaton [2] ap-

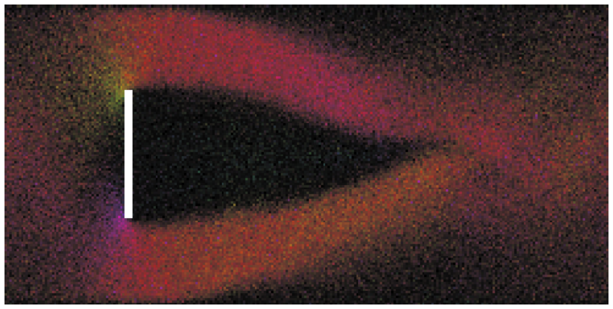

Figure 1: Lattice Gas Cellular Automaton flowing past a plane barrier. System size is $X \times Y$ with rainbow colour wheel cells representing flow direction averaged over M bit cells.

proximation can however be used to explore some qualitative and quantitative aspects of certain fluid flows. LGCA models and various sophisticated refinements such a the Lattice Boltzmann approach [3] can be formulated on more regular structures that is sometimes possible for full-field Navier-Stokes equation formulation. LGCA models are also highly data-parallelisable and offer the potential of simulating large model systems. Many physical systems such as fluid-flow and particularly turbulent fluid flow [4] reveal interesting properties and behaviours on many different length scales. A large simulation that can effectively capture several powers-of-ten of different length scales can therefore provide a useful simulation tool.

Fluid flow simulations are normally based upon a numerical solution or time-integration of the relevant partial differential equations. These are derived from the continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1}$$

which is written here in terms of the fluid density ρ and the time t and position dependent $\mathbf{u}$ velocity, which is a vector field. For our purposes we restrict ourselves to cases of constant density and this "incompressible fluid" case gives: $\nabla \cdot \mathbf{u} = 0$. We use this to obtain a derivation of the Navier-Stokes equation is often expressed in terms of the substantive derivative – the derivative following the fluid motion, which is defined as:

$$\frac{D\mathbf{u}}{Dt} \equiv \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \tag{2}$$

Which leads to the form of the general Navier-Stokes form in D dimensions (typically 2 or 3) as:

$$\frac{D(\rho\mathbf{u})}{Dt} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \eta^2\mathbf{u} + \left(\zeta + \frac{1}{D}\eta\right)\nabla(\nabla\cdot\mathbf{u}) + \mathbf{F} \tag{3}$$

including a general applied force term $\mathbf{F}$ and the bulk viscosity ζ.. In the case of a fluid of constant density ρ this yields the incompressible form:

$$\rho\frac{D\mathbf{u}}{Dt} = -\nabla p + \eta\nabla^2\mathbf{u} + \mathbf{F} \tag{4}$$

It is useful to focus for our purposes on the two dimensional case in simple Cartesian coordinates where $\mathbf{u} \equiv (u, v)$ and:

$$\begin{aligned}
\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \qquad (5)\\
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \frac{F_x}{\rho}\\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + \frac{F_y}{\rho}
\end{aligned}$$

where we have written the kinematic shear viscosity $\nu \equiv \eta/\rho$.

There are various numerical approaches to solving these equations to simulate systems with particular initial flow conditions and boundary conditions. Many authors have reported the use of differing solver methods [5] and detailed numerical approaches [6] that will work well on various parallel architectures. The main point remains however, that this is a highly floating-point intensive calculation and is difficult to deploy on desktop level processing platforms in interactive speed for realistically large model systems.

The Lattice Gas approach to fluid-flow was first developed in earnest in the 1980s [7] with specific results such as the cluster kinetics behaviour reported [8], and was shown to be successful using supercomputers of the late 1980's and early 1990's era [9]. In this paper we discuss the performance of GPUs for a Lattice Gas Cellular Automaton approximation [10] to the incompressible Navier-Stokes partial differential equations [11] that is fast enough to look at complex fluid flow patterns on commodity computing resources in interactive time.

Modern processing devices typically make use of multiple "cores" in a single chip to provide some on-chip parallelism. Present generation CPUs typically offer $\approx 2, 4, 6, 8$ processing cores that fully capable of independent operation. This is useful for many scientific simulation purposes [12]. An alternative approach for highly data-parallel simulations is to deploy devices that have a much larger number (> 100) cores that can operate in closer synchrony and which can yield dramatic performance gains for the right applications [13].

Graphical Processing Units (GPUs) have recently proven highly effective processing performance acceleration tools both for simulation computations [14, 15] as well as for enabling interactive-time quality graphical visualisations of simulated systems [16]. GPUs can in a range of different architectures and have proven to be some of the most effective multi-core processing architectures in recent times. These devices make use of regular and highly structured memory and core layouts and are programmed using a data parallel approach to parallelism. The key to achieving high performance for a given application is to find the right match between the different memory layers and capabilities of the device and the computational data structures required by the application.

The LGCA model is implemented in terms of simple gas particles that flow on the lattice and which can generally be represented by a very small number of bits rather than as by full double precision fields describing velocity, pressure, and so forth. GPU memory at all levels in generally in limited supply - the amount available is highly correlated with the cost of a GPU card. It is therefore important to find ways to optimally compress the LGCA particle bit data to make maximum use of available GPU memory but also to minimise communication overheads between GPU and its control-hosting CPU.

There are various symmetry requirements of the Navier-Stokes equations for fluid flow that mandate use on non-trivial lattice structures for LGCA approximations. It has been shown that for a two dimensional system a simple square lattice has insufficient symmetry and a triangular lattice - with cells having six nearest neighbours, is necessary. Similarly to use LGCA approaches in three dimensions it is necessary to employ a 3-D projection of what is effectively a face-centred four dimensional hyper-cubic lattice structure [17]. These lattices are not trivial to implement and present additional challenges for efficient implementation on GPU memory.

In this paper we describe various ways of laying out in GPU memory a simple lattice gas cellular automaton model. We summarise the LGCA model in Section 2 and describe some of the GPU architecture in Section 3 and the GPU programming implementation ideas we used in Section 4. We present some selected performance results for various GPU devices in Section 5. We discuss their implications for large scale LGCA simulation on multiple GPU systems and offer some conclusions and areas for further work in Section 6.

2 Lattice Gas Cellular Automata

As previously mentioned the LGCA method provides a efficient way to simulate large scale fluid-flow by moving particles within a triangle mesh (see Figure 2) and applying simple collision rules conserving mass, momentum and energy. A triangular lattice is used to satisfy the required interaction constraints established by the Navier-Stokes equation. The resulting system does not correctly model physics at a microscopic level but produces satisfactory bulk properties at a macroscopic level. Each site has six neighbours that it can interact with based on the collision rules identified by the direction of the velocity vectors within the site (see Figure 3). There are 256 possible particle configurations with applicable rules to be set and stored in a lookup table. Fortunately the rules for the presence or absence of a particle are opposite which allows us to specify only half of the rules and flip the bits on both the input and output to produce the other half based on the status of the particle in question.

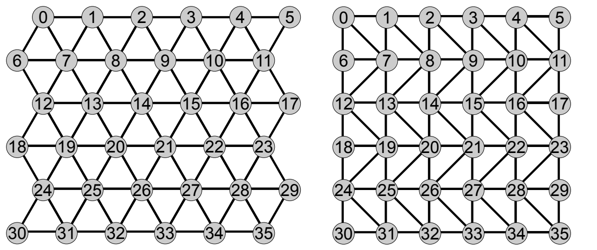

Figure 2: A Triangular mesh as it is used conceptually (left) and the same mesh distorted for ease of storage and computation (right).

Each particle can be represented by one byte, the first six bits from zero to five represent the six velocity vectors. Because mass, momentum and energy are conserved the interaction between the particles is solely based on the collision rules thus allowing the velocity and direction to be identified using only one bit. Bit six represents a rest particle while bit seven, when set allows a particle to act as a barrier that blocks all incoming particles. For example a stationary particle acting as a barrier will be formatted 11000000. To improve memory management we have compacted four lattice sites into one 32bit int allowing the size of the array storing the data to be one quarter of the size it would be if a full int was used for each site.

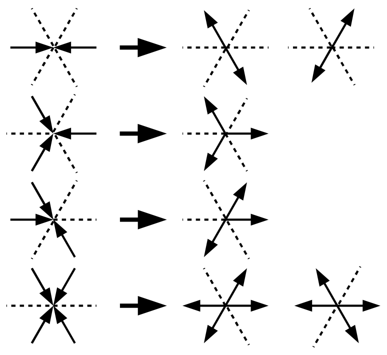

Figure 3: The collision rules of the Lattice Gas Cellular Automata.

The lattice gas model is well suited to parallelisation as the update calculation of each particle is independent of all others in the current time-step and is based purely on the previous arrangement. In the follow section we describe the GPU architecture and several LGCA implementations.

As part of our code debugging and testing we developed a graphical interface implemented in OpenGL and which supports various schematic and bulk system metrics as the flow pattern develops. We implemented a barrier pattern import mechanism that allows arbitrary barrier patterns to be developed using a drawing package and transformed into an appropriate bit-mask on the model triangular lattice. Figure 4 shows the steady-state fluid flow around an embedded barrier made up of some text. This is a somewhat complex flow pattern however and the standard pattern of a plane barrier as shown in Figure 1 was used for the benchmarking data presented in this paper.

3 GPU Architecture Issues

We have made use of NVIDIA's CUDA and GPUs to increase the size of the lattice we can simulate in a reasonable time-step. The computational throughput of GPUs can decrease simulation time allowing for larger and effectively more accurate simulations to be computed. The lattice gas simulation is well suited to parallel computation on GPU architectures as the time-step for each cell in the mesh can be computed independently. However, for this simulation there is some increased complexity as a triangular lattice must be used.

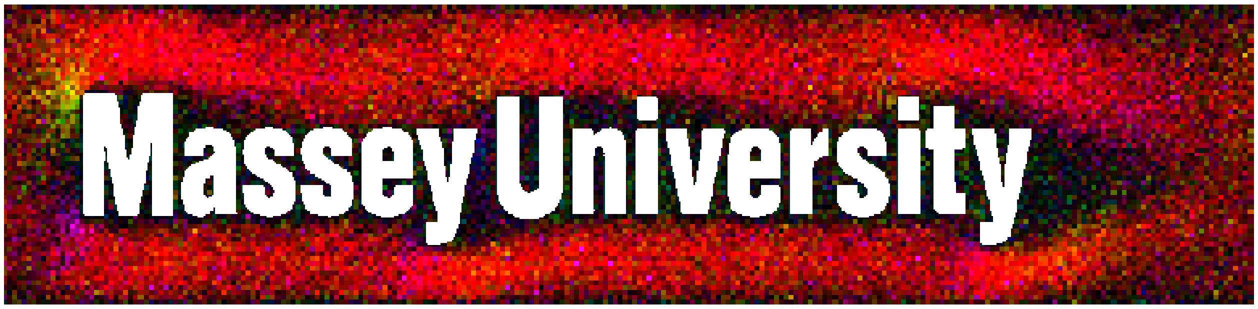

Figure 4: The Lattice Gas Cellular Automata simulating fluid flow past the "Massey University barrier". The cells are coloured using the HSV colour model with hue defined by the direction of the flow and the brightness determined by the velocity.

To properly explain how the simulation can be optimised for GPU architectures and memory optimisation, we provide a brief overview of GPU architectures.

NVIDIA CUDA-enabled GPUs all have similar architectures. In general GPUs are designed to manage, schedule and execute large numbers of threads (thousands or millions of threads are common), as this thread management is performed in hardware it presents little overhead. These threads are separated into blocks (each block can contain up to 512 threads) which are executed on the GPU's Multi-Processors (MPs). Each MP contains eight Scalar Processors (SPs) which execute individual threads. As there is no automatic caching on GPUs (at least the current generation GTX 200 series), GPUs contain several types of explicitly accessed memory types. These are as follows:

Global memory is the largest (and slowest) type of memory. All data input from the CPU or to be copied out to the CPU must be placed in this global memory. Thread access time to global memory can be reduced by the use of coalescing. When sequential threads access sequential values from global memory the transactions can be coalesced into a single memory access.

Shared memory is an 16KB area of memory on the MPs, threads in the same thread block can share data with this memory type. This can be very useful as it can reduce the number of required accesses to global memory.

Texture memory is a cached method for accessing global memory. Arrays can be bound to a 1D, 2D or 3D texture to tell CUDA to use the texture cache. This is useful when threads in the same block access values in the same spatial locality.

Constant memory is another cached method of accessing global memory. However, this cache is designed to allow threads to access the same value in memory at the same time.

When implementing any program on a GPU, carefully consideration of memory type and access patterns is necessary to make full use of the card's computation capabilities.

4 LGCA on the GPU

To simulate the LGCA on the GPU we must first determine how to decompose the simulation among individual threads. The task of updating each cell in the triangular mesh can be performed independently, we create one thread for each cell. Each thread should read in the value of the cell and the necessary neighbouring cells from memory, apply the appropriate collision rules and write the new value to an output array. At this point is should be noted that each cell in the mesh does in fact contain four sites packed into a single integer. To determine the optimal memory configuration we have implemented six kernels which use different types of CUDA memory for both the mesh and the rule lookup table.

4.1 Memory for triangular mesh

We have used two types of GPU memory for accessing the triangular mesh - Global and Texture. In previous neighbour-based, square lattice GPU simulations it was found that texture memory provided the best performance [14]. However, we have tested both types as the access patterns are different for triangular lattices. The first three kernels (A,B,C) read the cell and neighbouring values directly from global memory whereas the last three kernels (D,E,F) make use of the texture cache when accessing these values.

While the spatial caching of texture memory is advantageous for accessing neighbouring values, it does require a additional memory copy from the output array into the texture-bound array. This overhead must be weighed against the performance benefits the spatial caching provides.

4.2 Memory for Rule Look-up Table

The other consideration that should be made is how the threads should access the collision rule look-up table. In this case we use three types of memory - Global, Shared and Constant. In kernels A&D, the threads read the necessary values from the rule look-up table (LUT) directly out of global memory. As these accesses will be uncoalesced in most cases, we expect this to be the slowest implementation. Kernels B&E still read the look-up table from global memory but instead of each thread reading the value it requires, each thread reads one value and stores it in shared memory. The threads can then access the required look-up table values from the faster shared memory. Finally kernels C&F use the constant cache for accessing the LUT, this should provide some benefit over the global memory method as it will allow multiple threads in the same block to access the same value at the same time.

5 Performance Results

To test the performance of the LGCA implementations, we have executed the six kernels (A-F) on four different NVIDIA GeForce 200 series graphics cards. The cards tested are the GeForce 210, GT 220, GTX 260+ (factory overlocked to 666MHz) and a GTX 295. The GPU implementations are also compared to two CPU simulations of the LGCA in Java and C++ running on an Intel Core i7 920 2.66GHz with 6 GB of DDR3-1600 memory and on an Intel Core 2 Quad 2.66 GHz with 8GB of DDR2-800 memory. First we compare the different GPU kernels.

To determine the fastest kernel, we have tested all six on our fastest GPU (the GTX 260+) computing fluid flow around a simple barrier (see Figure 1). These kernels computed the simulation for 1024 steps for field lengths of N={256-8192}. These performance results can be seen in Figure 5.

The two kernels that used Global memory for the rule look-up table (kernels A and D) clearly performed the worst. The results for the other four kernels where very close, with kernel B (using Global memory for the lattice and Shared memory for the rule look-up table) providing the best results. Interestingly the kernels that use texture memory for accessing the lattice (kernels D, E and F) performed worse than their Global memory counterparts. These findings were unexpected as in previous, similar rectilinear simulations texture memory provided an important performance gain [14].

We believe that the reason for this different performance is the different memory access patterns of the triangular lattice. Because the cells on odd and even rows have different neighbour access patterns, the texture memory cache cannot be used to full efficiency. The added time of the extra memory copy from the output array to the texture-bound input array outweighs the caching benefits. It should also be noted that texture memory caching is also less important on GeForce 200 series cards due to the improved coalescing of global memory transactions.

We have also compared the best CUDA implementation (kernel B) on several GeForce 200 series cards and also to several CPU implementations. The CPU simulations have been implemented in Java and C++ and have been tested on an Core 2 Quad and a Core i7 920. The results comparing these various CPUs and GPUs are presented in Table 1.

Table 1: Performance results for the different implementations of the LGCA simulating 1024 time-steps on a 4096x4096 triangular lattice. All CUDA implementations use kernel B (Global memory for lattice and Shared memory for look-up table).

Implementation	Time (seconds)	Million hits per second
Java (i7 920)	73.6 ± 1.3	233 ± 4
C++ (Core 2 Quad)	101.3 ± 0.7	169 ± 1
C++ (i7 920)	51.7 ± 0.6	332 ± 4
GeForce 210	46.6 ± 3.4	369 ± 27
GT 220	11.1 ± 0.04	1,548 ± 6
GTX 260+	2.8 ± 0.005	6,136 ± 11
GTX 295	2.9 ± 0.005	5,924 ± 10

The performance for the GPUs is very positive (the GTX 260+ provides a 18x speedup over the i7), yet the speed-up factors are not as high as seen in previous square-lattice simulations [14]. The performance between the GTX 260+ and the GTX 295 are very similar (note that only one of the GPUs in the GTX 295 was used) with the GTX 260+ performing slightly faster. As seen previously [13] the higher clock speed of the GTX 260+ (666MHz vs 576 MHz) has more impact the the slighter higher number of cores in the GTX 295 GPU (240 vs 216).

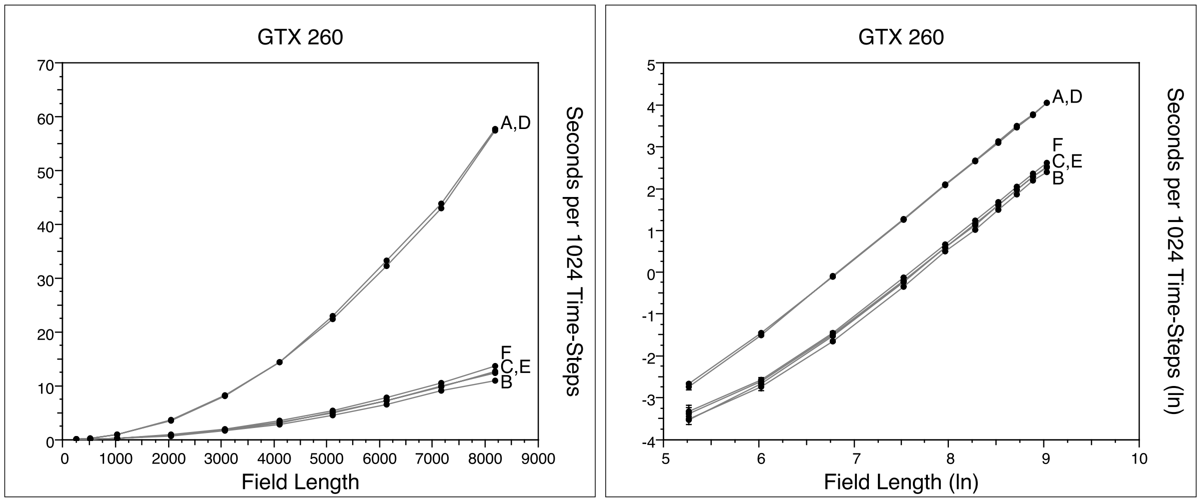

Figure 5: Performance results of the size LGCA kernels (A-F) on a GTX 260+. Results shown in normal scale (left) and ln-ln scale (right).

The performance results of the i7 920 is particularly impressive compared to the Core 2 Quad. Both processors have the same clock-speed yet the i7 can compute the simulation almost twice as fast. We attribute this performance improvement to the processor design and also the memory speed. The Core 2 Quad is using DDR2-800 memory while the i7 has DDR3-1600. This increased memory frequency also contributes to the increased performance.

6 Discussion and Conclusions

We have described how a simple lattice gas cellular automaton can be implemented on a two-dimensional triangular lattice, where each node has six nearest neighbours. We have implemented and measured the performance of this in *Java* and in *C* for CPU platforms and using *CUDA* for various NVIDIA GPUs with selected combinations of cores and memory configurations.

We find that our *CUDA* version scales well with the number of cores in the GPU architecture. However we also find that while the GPU is capable of significantly accelerating the performance of the algorithm, the triangular lattice requires some unexpected choices for GPU memory usage to attain optimal speed. Normal rectilinear simulations that exploit data-parallelism of the GPU performs best using **Texture** memory for lattice access whereas we find the triangular lattice of the LGCA simulation performs best with simple **Global** memory.

This appears to be due to alternating access patterns required for the triangular mesh. We believe the differing access patterns restrict the GPU from making full use of the GPU texture cache.

We implemented an interactive-time graphical rendering of the simulated system with OpenGL graphics code and this has proved invaluable for debugging purposes. The performance analysis has been done without graphics overheads. An area for future work is to combine the power of multiple GPUs to accelerate both the graphics **and** the simulation itself.

We also plan to extend this work to use suitable hypercubic face-centred cubic lattices to yield an effective 3-d simulation system with appropriate symmetries. We expect this will also scale well on the GPU architecture although will quite likely also present some unexpected memory choices due to the symmetry constraints.

References

[1] O.Penrose, A.Buhagiar: Kinetics of nucleation in a lattice gas model: Microscopic theory and simulation compared. J.Stat.Phys **30** (1983) 219–241

[2] Dab, D., Boon, J.P., Li, Y.X.: Lattice-Gas automata for coupled reaction diffusion equations. Phys.Rev.Lett. **66** (1991) 2535–2538

[3] R.Benzi, S.Succi, M.Vergassola: The lattice boltzmann equation: theory and applications. Rome Preprint ROM 2F-92-10 (1992)

[4] Advisory Group for Aerospace Research and Development: Special Course on Modern Theoretical and Experimental approaches to Turbulent Flow Structure and its Modelling, 7 Rue Ancelle, 92200 Neuilly Sur

Seine, France, Advisory Group for Aerospace Research and Development, NATO (1987) AGARD Report No 755.

[5] Kalland, K.M.: A Navier-Stokes Solver for Single and Two-Phase Flow. Master's thesis, Faculty of Mathematics and Natural Sciences, University of Oslo (2008)

[6] Simon, H., Golub, G.H., Huang, L.C., Tang, W.P.: A Fast Poisson Solver for the Finite Difference Solution of the Incompressible Navier-Stokes Equations. SIAM Journal on Scientific Computing **19** (1998) 1606–1624

[7] d'Humieres, D., Lallemand, P.: Lattice gas automata for fluid mechanics. Physica A: Stat. Mech and its Applications **140** (1986) 326–335

[8] Toral, R., Marro, J.: Cluster kinetics in the lattice gas model: The Becker-Doring type of equations. J. Phys. C: Solid State Physics **20** (1987) 2491–2500

[9] Wylie, B.J.: Application of Two-Dimensional Cellular Automaton Lattice-Gas Models to the Simulation of Hydrodynamics. PhD thesis, Physics Department, Edinburgh University (1990)

[10] Gould, H., Tobochnik, J., Christian, W.: An Introduction to Computer Simulation Methods. 3rd edn. Addison-Wesley (2006) ISBN: 0-8053-7758-1.

[11] Boghosian, B.M.: A survey of techniques for simulating partial differential equations with lattice gases. TMC Technical Note CA89-1, Thinking Machines Corporation (1989)

[12] Hawick, K.A., Leist, A., Playne, D.P.: Mixing Multi-Core CPUs and GPUs for Scientific Simulation Software. Technical Report CSTN-091, Computer Science, Massey University (2009)

[13] Playne, D.P., Johnson, M.G.B., Hawick, K.A.: Benchmarking GPU Devices with N-Body Simulations. In: Proc. 2009 International Conference on Computer Design (CDES 09) July, Las Vegas, USA. Number CSTN-077 (2009)

[14] Leist, A., Playne, D., Hawick, K.: Exploiting Graphical Processing Units for Data-Parallel Scientific Applications. Concurrency and Computation: Practice and Experience **21** (2009) 2400–2437 CSTN-065.

[15] Hawick, K., Leist, A., Playne, D.: Damaged Lattice and Small-World Spin Model Simulations using Monte-Carlo Cluster Algorithms, CUDA and GPUs. Technical Report CSTN-093, Computer Science, Massey University (2009)

[16] Leist, A., Playne, D., Hawick, K.: Visualising Spins and Clusters in Regular and Small-World Ising Models with GPUs. Technical Report CSTN-108, Computer Science, Massey University (2009)

[17] d'Humieres, D., Lallemand, P., Frisch, U.: Lattice gas models for 3d hydrodynamics. Europhys. Lett. **2** (1986) 291–297