Porous Rock Simulations and Lattice Boltzmann on GPUs [1]

Eirik O. AKSNES [a] and Anne C. ELSTER [a]
[a] *Norwegian University of Science and Technology (NTNU), Trondheim, Norway*

Abstract.
Investigating how fluids flow inside the complicated geometries of porous rocks is an important problem in the petroleum industry. The lattice Boltzmann method (LBM) can be used to calculate porous rockst' permeability. In this paper, we show how to implement this method efficiently on modern GPUs. Both a sequential CPU implementation and a parallelized GPU implementation is developed. Both implementations were tested using three porous data sets with known permeabilities. Our work shows that it is possible to calculate the permeability of porous rocks of simulations sizes up to 368^3, which fit into the 4 GB memory of the NVIDIA Quadro FX 5800 card. Our single floating-point precision simulation resulted in respectbale 0.95-1.59 MLUPS whereas our GPU implentation achieved remarkable 180+ MLUPS for several lattices in the 160^3 to 368^3 range allowing calculations that would take hours on the CPU to be done in minutes on the GPU. Techniques for reducing round-off errors are also discussed and implemented.

Keywords. Lattice Boltzmann Method, Permeability, GPU, Porous Rocks

1. Introduction

To solve the most complex problems in computational fluid dynamics (CFD), powerful computer systems are necessary. State-of-the-art GPUs can provide computing power equal to small supercomputers since a larger portion of their transistors are used for floating-point arithmetic, sicne they have higher memory bandwidth than modern CPUs. In this paper, we investigate the use of GPUs for computing porous rocks' ability to transport fluids (permeability) using the *Lattice Boltzmann Method* (LBM) [10]. This is a very important, but computationally challenging problem for the petroleum industry. major influences on permeability. The LBM has several desirable properties for fluid flows through porous media, particularly the ability to deal with complex irregular flow geometries without significant penalty in speed and efficiency. The LBM is also particularly suitable for GPUs, since it only requires neighbor interactions.

An overview of our research group's GPU programming experiences can be found in [8]. The models analysed in this work may be generated using CT scans [4]. Extended versions of this paper with added color figures, are provided in [2,3].

[1] The authors would like to thank Numerical Rocks AS for their collaborations and access to test samples, Dr. Pablo M. Dupuy, NTNU, for introducing us to the LBM method, and NVIDIA for supporting Dr. Elster and her HPC-lab.

2. The Lattice Boltzmann Method (LBM)

Historically, the LBM is an outcome from the attempts to improve the Lattice Gas Cellular Automata (LGCA). However, the LBM can be derived directly from the Boltzmann equation formulated by Ludwig Boltzmann, [10], uses classical mechanics and statistical physics, to describe the evolution of a particle distribution function. The LBM solves the Boltzmann equation in a fixed lattice. Instead of taking into consideration every individual particle's position and velocity as in classical microscopic models (molecular dynamics), the particle distribution function in the LBM gives the probability of finding a fluid particle located at the location x, with velocity e, at time t [5]. The statistical treatment in the LBM is necessary because of the large number of particles interacting in a fluid [15]. However, it leads to substantial gain in computational efficiency.

In the LBM, fluids flows are simulated by calculating the streaming and collision of particles within the lattices, often together with some boundary conditions that must be fulfilled for each time step. The discrete lattice locations correspond to volume elements that contain a collection of particles, and represents a position in space that holds either fluid or solid. In the streaming phase, particles move to the nearest neighbor along their path of motion, where they collide with other arrived particles. The outcome of the collision is designed to be consistent with the conservation of mass, energy and momentum. After each iteration, only the particle distribution changes, while the particle distribution function in the center of each lattice locations remains unchanged. The underlying lattice must have enough symmetry to ensure isotropy, and typically lattices are D2Q9, D3Q13, D3Q15, and D3Q19, where Da is the number of dimensions and Qb is the number of distinct discrete lattice velocities $\vec{e}_i$. The discrete Boltzmann equation can be written as Eq.(1) [5]:

$$f_i(\vec{x} + \vec{e}_i, t + 1) - f_i(\vec{x}, t) = \Omega \tag{1}$$

where $\vec{e}_i$ are discrete lattice velocities, Ω is the collision operator, and $f_i(\vec{x}, t)$ is the discrete particle distribution function in the i direction. The macroscopic kinematic viscosity of the fluid is $\nu = \frac{2\tau-1}{6}$. Other macroscopic properties of the fluids – such as the mass density $\rho(\vec{x}, t)$, momentum $\rho(\vec{x}, t)u(\vec{x}, t)$ and velocity $\vec{u}(\vec{x}, t)$ of a fluid particle – can be computed from the particle distribution functions, as seen in the first step of the collision phase in Figure 1. More details and references can be found in [2,3].

Boundary Conditions: The standard boundary condition applied at solid-fluid interfaces is the no-slip boundary condition (a.k.a. bounce-back boundary condition), Eq.(2) [12]:

$$f_i^{in}(\vec{x} + \vec{e}_i, t + 1) = f_i^{out}(\vec{x}, t) = f_i^{in}(\vec{x}, t) \tag{2}$$

Here the particles close to solid boundaries do not move at all, resulting in zero velocity. The particles at the solid-fluid interfaces are reflected [16], as illustrated in Figure 2. Periodic boundary conditions are also common, and allows particles to be circulated within the fluid domain. With the periodic boundary conditions, outgoing particles at the exit boundaries will come back again into the fluid domain through the entry boundaries on the opposed side. For a more comprehensive overview, see [15] and [19].

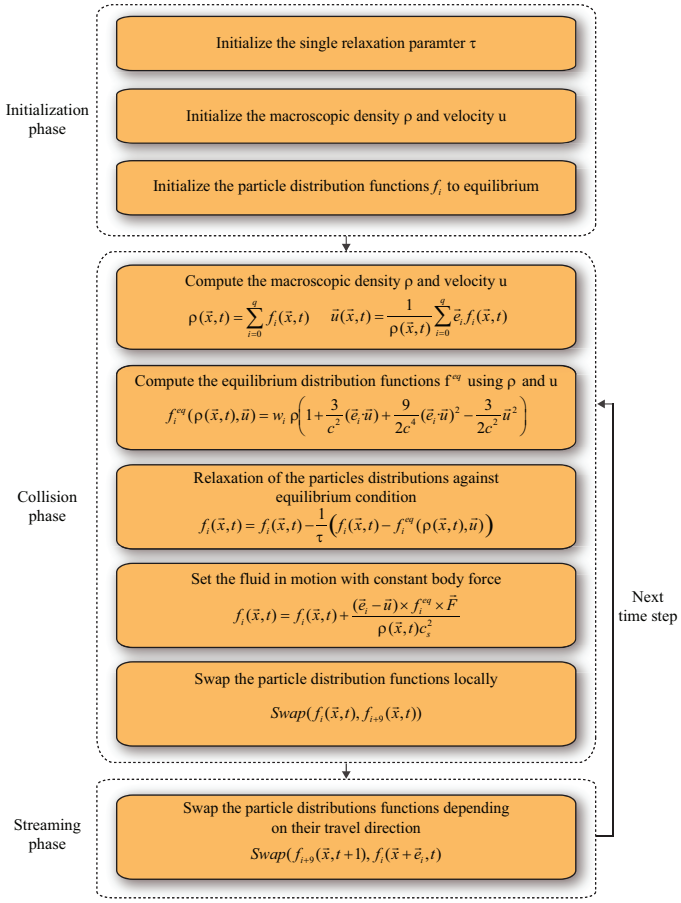

Figure 1. The main phases of the simulation model used. Based on [12].

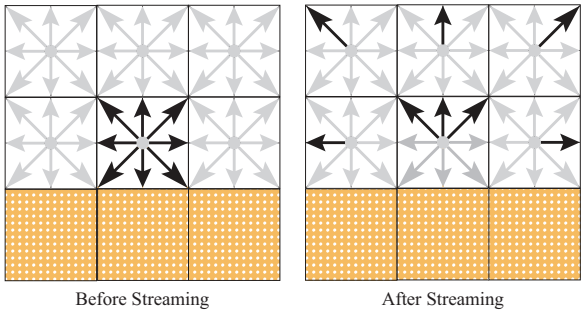

Figure 2. Bounce-back boundary of lattice nodes before (left) and after (right) streaming. Based on [16].

The LBM has been implemented on CPUs for fluid flows through porous media to determine the permeability of porous media [1]. See [18] for a comprehensive overview of efficient CPU implementations of the LBM, in view of the fact that the architecture of the GPU is quite different. See [9,6,17] for recent GPU-based LBMs.

3. Our Simulation Model

The main phases of our LBM simulation model can be seen in Figure 1. In this model, the collisions of particles are evaluated first, and then particles streams to the lattice neighbors along the discrete lattice velocities. Two types of boundary conditions are implemented: the standard bounce back boundary condition to handle solid-fluid interfaces, and periodic boundary condition to allow fluids to be circulated within the fluid domain. The periodic boundary condition is built into the streaming phase, and the bounce back boundary condition is built into the streaming and collision phase. The different phases of our simulation model accompanied by pseudo-code is described in more detail in [2].

Effcient storage: Our simulation model makes use of the D3Q19 lattice. For every node in the lattice, implementations using the D3Q19 model often store and use 19 values for the particle distribution functions and 19 temporary values for the streaming phase, so that the particle distribution functions are not overwritten during the exchange phase between neighbor lattice nodes. The LBMs using such temporary storage thus require gigabytes of memory for lattices sizes of 256^3 and larger [2] and [3].

Instead of duplicating the particle distribution functions to temporary storage in our streaming phase, we use another approach described by Latt [13] for both implementations. Here the source and destination particle distribution functions are instead swapped between neighbor lattice nodes. This approach reduces the memory requirements by 50 %, compared to using temporary storage.

Permeability calculations: Figure 3 shows the expansion of the simulation model for the calculation of permeability of porous rocks. The permeability is obtained directly from the generated velocity fields of the lattice Boltzmann method, together with using Darcy's law for the flow of fluids through porous media. The fluid flow is driven by some external force in the simulation model, but it could also be driven by pressure on the boundaries. The external force is expected to give the same change in momentum as the true $\frac{\Delta P}{L}$, which is the total pressure drop along the sample length L. Note that in Figure 3, a is the node resolution equivalent to the lattice spacing. Driven by some external force, the permeability is always obtained when the velocity field is at steady state. Our simulations is considered to have converged if the change of the average velocity is less than 10^{-9} between time-steps.

Floating-point precision and round-off errors: In the collision phase, the equilibrium distribution function needs to be computed with a mixture of large and small numbers which may lead to lserious rounding errors [11]. To reduce the rounding error when using single floating-point precision, we used an approach taken from [7]. It has been left out from the previous descriptions, due to readability, but can be found in [2].

4. Implementations and further optimizations

Current NVIDIA GPUs support thousands of threads running concurrently, to hide the latency under uneven workloads in programs. In our GPU implementation, every thread created during execution is responsible for performing the collision and streaming for

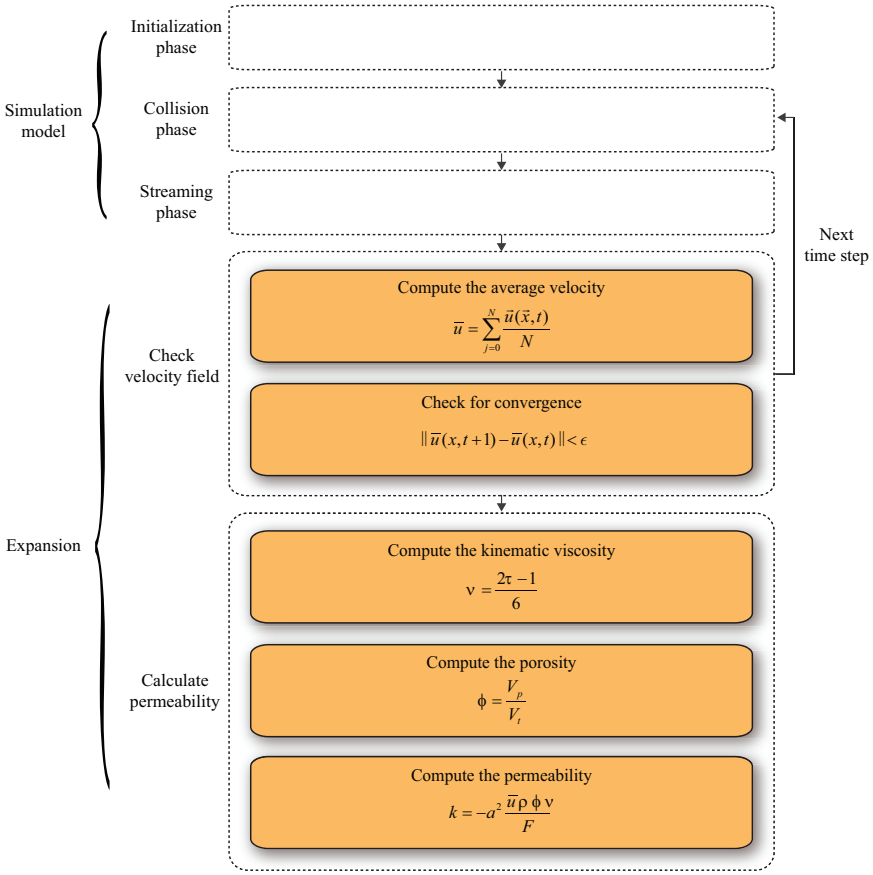

Figure 3. Expansion of the simulation model for permeability calculations.

a single lattice node. To get high utilization of global memory bandwidth, the access pattern to the global memory must be correctly aligned to achieve coalescing.

A structure-of-arrays is useful to achieve coalescing. Threads access the arrays in contiguous memory segments to obtain coalescing. This enables efficient reading and writing of particle distribution functions. Each array contains one discrete direction of the particle distributions functions. Arrays in the structure are three-dimensional, allocated as contiguous memory on the device using cudaMalloc3D. This function takes the width, height, and depth of simulations as input, and pads the allocation to meet the alignment requirements to achieve coalescing. The function returns a pitch (or stride), which is the width in bytes of the allocation. Two-dimensional grids were necessary in order to simulate large lattices, due to NVIDIA's restriction of the maximum number of threads blocks being 65535 in one direction of the grid on our test bench. Note thateven simulation sizes of 512^3 without using temporary storage and with single floating-point precision would alone result in a memory consumption of 9.5 Gigabyte!

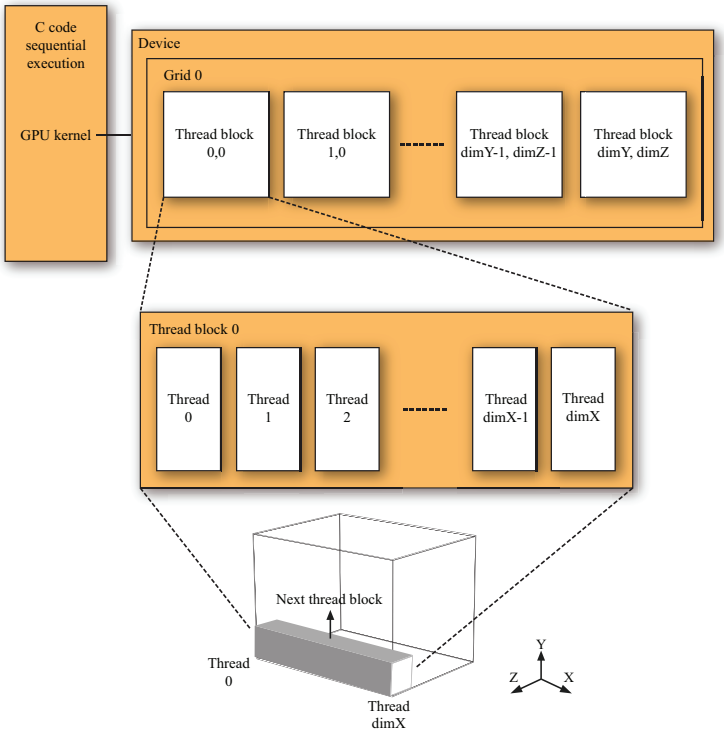

Figure 4. The configurations of grids and thread blocks in kernels.

5. Benchmarks

Our test-bed, the NVIDIA Quadro FX 5800 GPU has 4 GiB of memory. provides high memory bandwidth and 4 GiB of memory. Performance is, like [6], measured in MLUPS (million lattice nodes updates per second).

In **Poiseuille Flows** the analytical solutions of the velocity profile are known. To validate the numerical correctness and exactness of our two implementations, the numerical velocity profile of fluid flow between two parallel plates with lattice dimension 32^3 were compared to known analytical solutions. Two types of boundary conditions were used: bounce back boundaries along the two parallel plates and periodic boundaries in the x, y, and z direction for conservation of fluid particles. The values of the single relaxation parameter and the external force used in the calculations of the permeability of the three datasets used were τ=0.65 and F_x=0.00001 with $F_y = F_z = 0$. Both single and double precision Poiseuille Flow tests were performed. In Figure 5, the solid lines show the analytical solution, and the circles are the numerical results obtained from the Poiseuille Flow simulations. The measured deviations between the numerical and analytical solutions were only $1.680030e-006$ for both CPU and GPU 32, and $1.622238e-006$ for both CPU and GPU 64 [3].

Simulation Size Restrictions: The most memory demanding parts of the implementations is the structure-of-arrays used to store the particle distribution functions, together with the array used to store the porous rocks models. The GPU used has 16384 registers

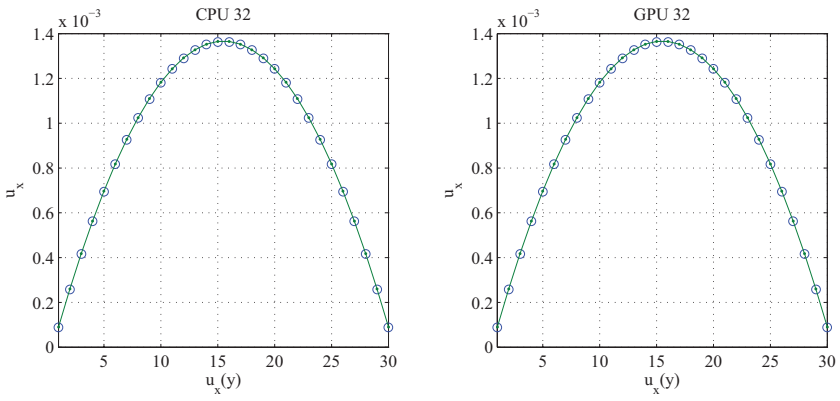

Figure 5. Poiseuille Flows: Comparison of numerical and analytical velocity profiles.

and 16 KB shared memory available per multiprocessor. Threads of all thread blocks running on a multiprocessor must share these registers and shared memory during execution. Kernels will fail to launch if threads uses more registers or shared memory than available per multiprocessor [14]. We found that the number of available registers per thread varies from 64 registers for blocksizes of 256, to only 21 registers available for blocksizes of 768. For more details, see [3].

Performance Measurements: In order to measure the performance of our CPU and GPU implementation, cubic lattice sizes ranging from 8^3 up to 368^3 were used. The cubic lattices were filled only with fluid elements, so that no extra work was required for solid-fluid interfaces. Figure 6 shows arithmetic means over 25 iterations.

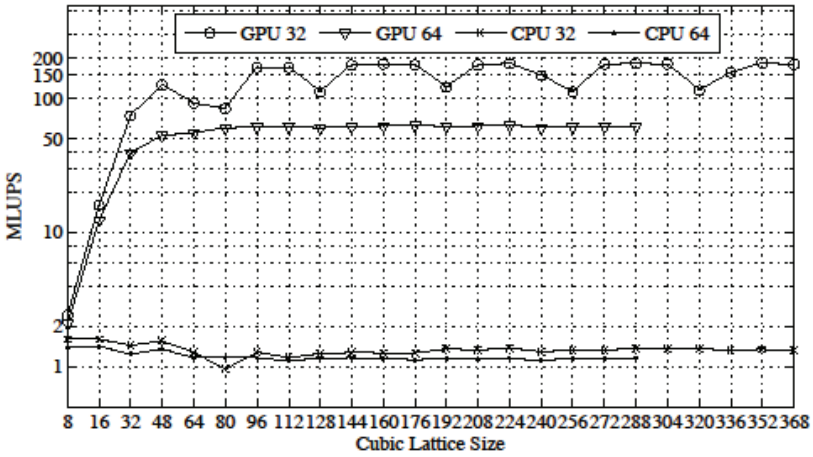

Figure 6. Performance results in MLUPS.

Here our GPU implementation clearly outperforms the CPU implementation. Using single floating-point precision, CPU 32 and GPU 32 achieved the maximum performance equal to 1.59 MLUPS and 184.30 MLUPS. CPU 64 and GPU 64 achieved maximum

performance equal to 1.40 MLUPS and 63.35 MLUPS. Highest performance of the CPU 32 and CPU 64 was with lattice sizes smaller than 64^3 and 48^3, since these lattices fit into cache memory. The performance difference between the GPU 32 and GPU 64 is because our NVIDIA GPUs have more single precision cores than double precision cores, and because the GPU 32 and GPU 64 have some differences in occupancy. More details regarding our occupancy calculations and experiments can be found in [2,3]. The turbulent performance of the GPU 32 is caused by the changes in occupancy of the collide kernel, due to changes in thread block sizes.

6. Porous Rock Measurements

In order to evaluate our implementations ability to calculate the permeability of porous rocks, three porous datasets with known permeability provided by Numerical Rocks AS were used. Porosity that reflects only the interconnected pore spaces within the three porous datasets calculated by Numerical Rocks AS was also used. The values of the single relaxation parameter and the external force used in the calculations of the permeability of the three datasets were $\tau = 0.65$, $F_x = 0.00001$ and $F_y = F_z = 0.0$

In our simulations, the configurations of the boundaries parallel to the flow direction were made solid, and with bounce back boundary conditions. The entry and exit boundaries were given periodic boundary conditions. There were also 3 empty layers of void space added at both the entry and exit boundaries. Simulations were run until velocity fields reached a steady state, before calculating the permeability of the three porous datasets. Due to space considerations, we include only the results from our Fontainebleau, the most realistic and interesting dataset here. The results from the two simplest datasets, symmetrical cube and square cube, can be found in [3].

The *Fontainebleau* dataset has a lattice size of 300^3, with the known permeability equal to 1300 mD. The dataset's porosity is 16 %. Note that the dataset is too large to allocate with double precision on our NVIDIA Quadro FX5800 GPU. Table 1 shows the results of our performance measurements and computed permeability. GPU 32 was the fastest of the implementations with the total simulation time equal to 38.0 seconds. All of the implementations calculated the permeability within deviation, with the relative error equal to 4.0% and the absolute error equal to 53. GPU 32 obtained the highest average performance equal to 58.81 MLUPS.

Table 1. Fontainebleau performance and computed permeability results.

Implementation	Average MLUPS	Maximum MLUPS	Total Time	Number Of Iterations	Permeability Obtained
CPU 32	1.03	1.04	2152 s	445	1247.80 mD
GPU 32	58.81	59.15	38.0 s	445	1247.81 mD
CPU 64	0.94	0.94	2375.4 s	445	1247.80 mD

Figure 7 shows the first 4 iteration of fluid flow inside our Fontainbleau. See [3] and [2] for more detailed results.

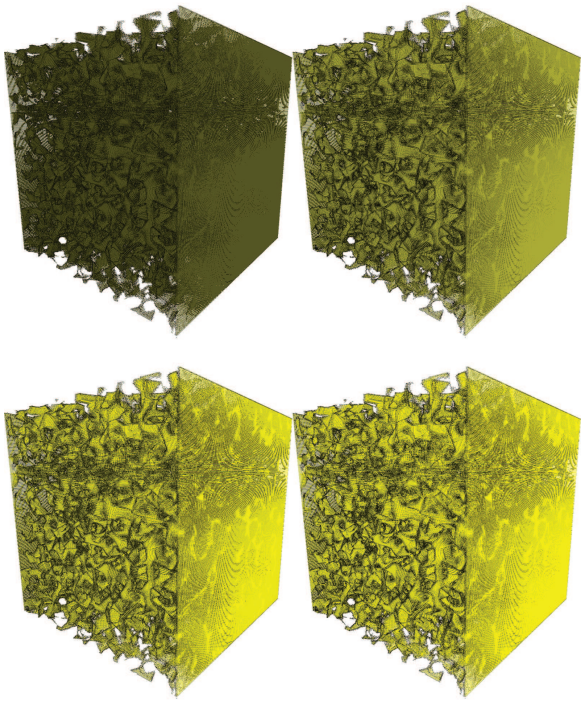

Figure 7. The first 4 iterations of fluid flow inside Fontainebleau.

7. Conclusions and future work

Modern *Graphics Processing Units* (GPUs) are used to accelerate a wide range of scientific applications which earlier required large clusters of workstations or large expensive supercomputers. Since it would be very valuable for the petroleum industry to analyze petrophysical properties of porous rocks, such as the porosity and permeability, through computer simulations, the goal of this work was to see if such simulations could benefit from GPU acceleration. LBM was used to estimate porous rock's ability to transmit fluids. In order to better analyze our results, both parallel CPU and GPU implementations of the LBM were developed and benchmarked using three porous datasets provided by Numerical Rocks AS where the permeability of each dataset was known. This also allowed us to evaluate the accuracy of our results.

Our development efforts showed that it is possible to simulate fluid flow through the complicated geometries of porous rocks with high performance on modern GPUs. Our GPU implementations clearly outperformed our CPU implementation, in both single and double floating-point precision. Both implementations achieved their highest performances when using single floating-point precision, resulting in their maximum performance equal to 1.59 MLUPS and 184.30 MLUPS, respectively for datsets of size 8^3 by 352^3, where MLUPS is the measurement of million lattice nodes updates per second (indicating the number of lattice nodes that is updated in one second). Suggestions for improving and extending our results inculde:

1. Maximum simulation lattice size in the simulations is limited by the 32-bit architecture of current GPUs which limits memory to 4 GiB. One way to compensate for this is to use multiple GPUs.
2. Use of grid refinement for the improved analysis of the fluid flow inside the narrow pore geometry of the porous rocks.
3. Storing only fluid elements, this will reduce memory usage, since the porous rocks of interest often have small pore geometries.
4. Extenting the LBM to perform multiphase fluid dynamics.

Given the importance for the petroleum industry of getting good fluid simulations of porous rocks, we expect that this will continue to be a great area of research. Flows through porous materials are also be of interest to other fields, including medicine.

Since our ParCo 2009 presentation, NVIDIA annouced in November 2009 their new Fermi GPU architecture that should also be investigated when it becomes available.

References

[1] Urpo Aaltosalmi. Fluid Flows In Porous Media With The Lattice-Botlzmann Method, 2005.

[2] Eirik Ola Aksnes. Simulation of Fluid Flow Through Porous Rocks on Modern GPUs , July 2009. Masters thesis, NTNU, Norway.

[3] Eirik Ola Aksnes and Anne C. Elster. Fluid Flows Through Porous Rocks Using Lattice Boltzmann on Modern GPUs , December 2009. IDI Tech report no. 03-10, ISSN 1503-416X, NTNU, Norway.

[4] Eirik Ola Aksnes and Henrik Hesland. GPU Techniques for Porous Rock Visualization, January 2009. Masters project, IDI Tech report no. 02-10, ISSN 1503-416X, Norwegian University of Science and Technology.

[5] Usman R. Alim, Alireza Entezari, and Torsten Möller. The Lattice-Boltzmann Method on Optimal Sampling Lattices. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):630–641, 2009.

[6] Peter Bailey, Joe Myre, Stuart D. C. Walsh, David J. Lilja, and Martin O. Saar. Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors. 2008.

[7] Bastien Chopard. How to improve the accuracy of Lattice Boltzmann calculations.

[8] Anne C. Elster. Gpu computing: History and recent challenges. in this ParCo2009 Proceedings.

[9] J Habich. Performance Evaluation of Numeric Compute Kernels on nVIDIA GPUs, June 2008. Friedrich-Alexander-Universität.

[10] Xiaoyi He and Li-Shi Luo. Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Phys. Rev. E*, 56(6):6811–6817, Dec 1997.

[11] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[12] C. Korner, T. Pohl, U. Rude, N. Thurey, and T. Zeiser. Parallel Lattice Boltzmann Methods for CFD Applications. *Numerical Solution of Partial Differential Equations on Parallel Computers*, 51:439–465, 2006.

[13] Jonas Latt. Technical report: How to implement your DdQq dynamics with only q variables per node (instead of 2q), 2007.

[14] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture. Programming Guide. V2.0*, 2008.

[15] Michael C. Sukop and Daniel T. Thorne Jr. *Lattice Boltzmann Modeling, An Introduction for Geoscientists and Engineers*. Springer, Berlin, Heidelberg, 2007.

[16] Nils Thurey. A single-phase free-surface Lattice Boltzmann Method, 2002. Friedrich-Alexander-Universität.

[17] J Tolke. Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA. *Computing and Visualization in Science*, July 2008.

[18] G. Weillein, T. Zeiser, G. Hager, and S. Donath.

[19] Dieter A. Wolf-Gladrow. *Lattice-Gas, Cellular Automata and Lattice Boltzmann Models, An Introduction*. Lecture Notes in Mathematics. Springer, Heidelberg, Berlin, 2000.