In this chapter, we will be taken into the broad field of the Lattice Boltzmann method using the CUDA platform. We will explore this theme following a specific order structured around focus areas. Firstly, we will look into Computational Fluid Dynamics and the major approaches to it. Then we will be given an overview on the Lattice Boltzmann method. Afterwards we will discuss how it is possible to employ parallelisation techniques to computations. Finally we will look into the previous efforts employed in this area.

**An overview of Computational Fluid Dynamics -** <http://www.yumpu.com/en/document/view/9231796/an-overview-of-computational-fluid-dynamics-mbr-train>

Computational Fluid Dynamics comes from the need to model fluid flows and associated processes. As such, it is “the science of determining a solution to fluid flow through space and time”. The models needed to calculate the fluid computations include:

* Flow geometry
* Differential (Governing) equations – These describe the physics and chemistry of the flow
* Boundary and initial conditions
* Structured mesh of points

As such, CFD can be used as a design and troubleshooting tool, as well as making the process dynamics easier to understand. It is used extensively by scientists and researchers, but it also has innumerable applications in the industry. Below we can find a few examples of it’s use:

MACROSCOPIC

In this approach, the fluid can be seen as a collection of a huge number of particles. To solve these governing equations, one needs to apply conservation of energy, mass and momentum. But since these equations are difficult to solve, finite schemes, boundary and initial conditions are used to convert these equations into a system of algebraic equations. These equations can then be solved iteratively until the solutions converges.

Microscopic

If we consider the fluid to be represented by individual particles then we’ll fall under the microscopic approach. In this approach, there is no definition of temperature or viscosity and collision between particles needs to be considered. Thus one needs to solve the differential equation of Newton’s second law /cite{lbm\_springer}. Hence, the location and velocity of each particle needs to be taken into account.

We can easily see that this approach becomes unfeasible for normal fluid sizes as the number of equations needed to be solved grows to the order of billions (consider that one mole of water contains more than $6 \times 10^23$ molecules.

**Lattice Boltzmann Method** – A.A. Mohamad

The Lattice Boltzmann method is a mesoscopic scale approach to CFD. It is used to describe a fluid based on probabilities using the Maxwell-Boltzmann equation in the fluid’s equilibrium state. In this method, we do not consider the individual characteristics of each particle. By grouping particles together in a D2Q9 (nodes containing 9 particles for 2D problems) or in D3Q19 (nodes containing 19 particles for 3D problems), we can analyse the collection’s behaviour as a whole.

This way we can enjoy the advantages of both the macro and microscale approach with affordable computer resources. Since communications between nodes is very small, LBM also offers the possibility of employing parallel computing to achieve the solution in even faster times.

This thesis will continue the work that has been continuously expanded since some years ago. As such, the work from four Cranfield students needs to be analysed.

Tamás Józsa, together with Máté Szőke, adapted two different inhouse C and C++ code into one single C code.\cite{jozsa\_thesis}\cite{szoke\_thesis}. Józsa then parallelised the critical parts of the C code using CUDA and ran tests on the Fermi GPU Cluster from Cranfield, achieving a three times speedup in general, with a peak of 15 times speedup.

**Parallelization of lattice Boltzmann method**

**using CUDA platform -** Tamás István Józsa

The author proposes a way to parallelize the Lattice Boltzmann method using GPGPU programming. To this end, he starts by investigating the previous work done in the area, then he introduces us to the topics that will be mentioned such as CFD and HPC. Afterwards we are given an introduction to how the method works and how he will adapt the existing in-house code to C and how he will parallelise the critical parts. The GPU (Fermi GPU Cluster from Cranfield) of the cluster was almost three times faster in general.

**Efficient Implementation of a 2D Lattice Boltzmann Solver Using Modern Parallelisation Techniques -** Máté Tibor Szőke

The author proposes a different parallelisation approach using Unified Parallel C, which is a Partitioned Global Address Space language. This means that it’s possible to use shared memory to compute the solution. However, the author verified that a local memory-based approach (like MPI) provided the best results. He also compared the results obtained with the one’s obtained by Józsa on the CUDA approach. They found that to achieve the same speed-up as that of a single GPU card, one needs an entire workstation (16 threads in the case of Astral).

**Optimisation of 2D lattice Boltzmann method using CUDA -** Ádám Koleszár

The author continues the work of the previous thesis and aims to further optimise the parallel version of the LBM method using CUDA. He does an excellent job, resulting in a 10 times faster execution than the previous 2D parallel solver, which means that the new optimised code is 30 times faster than the original, in-house, serial solver.

**MACIEJ**

Finally, Maciej Kubat proposed a new version of the LBM solver. He converts the 2D parallel solver to a 3D parallel solver. Going from 2D to 3D means that (almost) every piece of code needs to be remodelled, from data containers to logic cycles.

After first trying for a direct adaptation, he found that his code was too slow to produce meaningful solutions. After optimising his own code he was able to reach an almost one hundred times speedup.

However, Kubat states that a lot can be done to improve his code, from boundary conditions to code readability and maintainability.

**High performance computing**

As humanity evolves, so too does our desire for expanding previous unobtainable goals. As computer technology kept progressing further and further, we soon realised that some problems simply took too many resources to be completed. But what if instead of focusing on computational power we focused our efforts in splitting the workload?

High performance computing (HPC) comes from the harnessing computer power to deliver a much higher performance that one could obtain from a typical computer. To this end, we can talk of HPC as being a collection of computer clusters. This way, it’s possible to use the various clusters as one big machine and split the workload between them. Problems that could take weeks, months or even years can be solved in minutes or hours under these powerful devices.

**GPGPU**

General Purpose computing on Graphical Processing Units stands for the use of Graphics Processing unit (GPU) to perform computations on applications normally performed by the CPU. One of the main advantages of using this approach is the amount of cores that a single GPU has. While a typical desktop CPU has up to 4 cores, a GPU can have thousands of cores, allowing users to take advantage of its massively parallel architecture.

CUDA was then developed by NVIDIA. CUDA code allows programmers to take advantage of GPUs by employing a unified shader pipeline under the familiar C language \cite{cuda\_by\_example}. Users were no longer required to have specific knowledge of OpenGL or DirectX and could now perform general computations (rather than graphic-specific computations) whilst benefiting from the massive computational powered offered by GPUs.