Introduction

The Lattice Boltzmann Method (LBM) is a technique for simulating complex fluid systems. Fluid systems are used within many industries to transmit signals and power using a network of tanks, pipes, values, pumps and other flow devices. Example applications include industrial processing, vehicular control and medical appliances.

It is important that customers have a systematic method of mathematically modelling different types of fluid systems for safe and reliable operation. This can be achieved, but typically at significant computational cost. The general field of Computational Fluid Dynamics (CFD) is one of the most demanding branches of high-performance computing (HPC). There is constant demand for cheaper, faster CFD computing platforms.

This paper describes how it is possible to dramatically accelerate the LBM technique on an energy-efficient FPGA-based platform using OpenCL – the open standard for parallel programming.

Techniques

Traditional CFD methods solve the conservation equations for mass, energy, etc, whereas the LBM model uses particles to propagate these quantities. To simulate every particle in a system would be impossible, hence the LBM technique using particle densities confined to a discrete lattice to simulate particle interactions.

The LBM technique is split into two stages, collision and streaming steps. The collision stage looks to balance the particle distributions. There are various techniques for finding an equilibrium, some more accurate than others. The operator used here is the Bhatnagar-Gross-Krook (BGK) operator.

Lattice

Different lattice topologies are possible for different dimensions and algorithm approaches. A popular way to classify lattices is the DnQm scheme, where n stands for the number of dimensions and m the lattice velocity distributions. The lattice used in this white paper is a D2Q9 lattice illustrated in Figure 1.

![Figure 1: D2Q9 Lattice](image)
LBM Maths

The following equation is the BGK operator applied to the 9 lattice points contributing to the current lattice point ...

$$f_i^{eq} = \omega_i \rho \left( 1 + 3 \frac{e_i \cdot u}{c^2} + 9 \left( \frac{e_i \cdot u}{2c^2} \right)^2 - 3 \left( \frac{e_i \cdot u}{2c^2} \right)^2 \right)$$

Equation 1: D2Q9 equilibrium distribution function

where the weights \( \omega_i \) are 4/9 for rest particles \((i = 0)\), 1/9 for \(i = 1,2,3,4\) and \(1/36\) for \(i = 5,6,7,8\). \(C\) is the basic speed of the lattice. \(U\) is the velocity and \(e_i\) is the direction vector for the 9 possible directions.

Once the new distributions are calculated they must be distributed to neighboring lattice points. This is the streaming step.

![Streaming stage diagram](image)

**Figure 2: Streaming stage**

Particle distributions are swapped between lattice points along the 8 non-zero direction vectors.

Implementations

The lattice Boltzmann code is a memory bound problem. For the D2Q9 lattice 9 floating point numbers must be read and updated for every lattice during the collision phase. Here data is read in a linear fashion, however the propagate stage must implement some out of order memory accesses to swap data between adjacent lattice points.

For a GPU implementation it is the global memory access that ultimately limits the performance of the lattice Boltzmann code. FPGAs however offer an alternative approach that removes this memory bottleneck and provides almost unlimited scalability.

Lattice Boltzmann FPGA OpenCL

Typical OpenCL Lattice Boltzmann implementations work by creating hundreds of threads all working in parallel which is ultimately limited to the global memory bandwidth available. The Altera OpenCL compiler offers an alternative OpenCL programming model that creates one or more pipelined kernels, where parallelism comes from the complexity of the pipeline. The more complex the pipeline, the more floating logic is performed in parallel.

FPGAs have significant local memory resource than can be configured in many different ways, from large single buffers to hundreds of small buffers. This flexibility allows the Altera OpenCL compiler (AOC) to create memory topologies specifically targeted at the algorithm that needs accelerating. The consequence of this is to significantly reduce the global memory bandwidth requirements of the algorithm.

The collision stage of the LB algorithm accesses global memory in a linear fashion and needs no optimizations. However, the propagate stage requires data from the neighboring lattice points. This can be optimized by using a cached copy of the output in what is referred to as a **sliding window**. A sliding window approach allows data to be read linearly and buffered in local memory, from which data can be read as often as required.

![Sliding window diagram](image)

**Figure 3: Sliding window**

For LBM the sliding window allows the previously calculated rows to be stored in local memory allowing the streaming stage to be combined with the collision stage. The entire algorithm can then be pipelined to generate a result every clock cycle. What’s more multiple pipelines can be cascaded to gather with global memory access only required for input into the first stage and the output from the final stage. The number of Lattice points calculated per second therefore increases linearly per pipeline stage with no increase in global memory requirements.
FPGA Acceleration of Lattice Boltzmann using OpenCL

White Paper

Table 1 lists the performance of the BGK D2Q9 algorithm for various technologies. The Arria 10 performance figures are extrapolated from the PCIe385n_d5 performance and take into consideration the extra resource and global memory bandwidth available for these products.

There are 106 floating point calculations required per LUT. This makes the sustained floating point performance equivalent to 106 multiplied by the LUTs/sec.

Performance

The pipelining allows the performance to be linearly improved with each new pipeline stage until resource on the FPGA is exhausted. Four such pipelines fit into a PCIe385n_d5 part.

Table 1

<table>
<thead>
<tr>
<th>Device</th>
<th>Pipeline stages or Threads</th>
<th>MLUTs/ Sec</th>
<th>GFlops/ Sec</th>
<th>MLUT/sec/ Watt</th>
<th>GFlops/sec/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon E5-2430L</td>
<td>1</td>
<td>4.5</td>
<td>0.45</td>
<td>0.05</td>
<td>0.005</td>
</tr>
<tr>
<td>HD7970 (GPGPU)</td>
<td>2048</td>
<td>480</td>
<td>51</td>
<td>2.59</td>
<td>0.28</td>
</tr>
<tr>
<td>PCIe385n_d5</td>
<td>4</td>
<td>818</td>
<td>86</td>
<td>21.5</td>
<td>2.26</td>
</tr>
</tbody>
</table>

Figure 4: Multiple time steps implemented in a pipeline

Figure 5: PCIe385n_d5 and Server

Figure 6: Performance, MLUTs/Sec
Power

When measuring HPC performance it is important to consider the power footprint of different technologies. Table 1 also shows the performance per/watt for the 3 technologies available to study.

![Performance, MLUTs/Sec/Watt](image)

**Figure 7**: Performance, MLUTs/Sec/Watt

### Results

The following images show the output of the FPGA implementation. Yellow depicts the areas of fastest flow, whilst the black areas are slowest.

![Flow through a slit](image)

**Figure 8**: Flow through a slit

![Turbulent flow around a sphere](image)

**Figure 9**: Turbulent flow around a sphere

![Flow through a porous object](image)

**Figure 10**: Flow through a porous object

### D3Q19

The implementation described here can also be applied to a 3D lattice. In this case the sliding window stores planes rather than lines of lattice data. This requires more internal memory and limits the plane size and therefore the cross section of the volume than can be calculated using this approach. The depth of the lattice is however unlimited.

### Conclusion

Using the OpenCL tool flow, it was possible to achieve significant acceleration of a well-known HPC problem using FPGA technology in only a few days of coding. By abstracting the complexities of FPGA interfaces and hardware description languages, OpenCL massively increases productivity without significantly sacrificing design performance. This allows developers to quickly verify the suitability of FPGA acceleration without committing to months/years of design effort.

To learn more about the advantages of FPGA-based acceleration, please visit nallatech.com