Computational Fluid Dynamics on GPUS for Genetic Programming Fitness Evaluation

Jason Normore
Department Of Computer Science
Memorial University
Newfoundland, Canada
jnormore@mun.ca

Simon Harding
Department Of Computer Science
Memorial University
Newfoundland, Canada
simonh@cs.mun.ca

Wolfgang Banzhaf
Department Of Computer Science
Memorial University
Newfoundland, Canada
banzhaf@cs.mun.ca

1. INTRODUCTION

Evolutionary algorithms can be used to generate designs for physical objects. Previous examples include circuits, tables, struts in building construction and aerodynamic shapes. Generally such designs need to be tested in a simulated environment complete with models of the physics. Physical simulations are notoriously computationally expensive - but with recent advances in Graphics Processing Units (GPUs) it is now possible to speed up these simulations (and hence the fitness evaluation) of complicated objects within a complex physical system. In this project we are interested in using evolution to design hydrodynamic shapes (such as wing surfaces) that meet some criteria, e.g. minimize drag and maximize lift. The new contributions in this work are a new developmental GP methodology for producing shape designs and a distributed, GPU Computational Fluid Dynamics (CFD) simulator that is able to efficiently simulate flow around an arbitrary shape. This is important as the shape generation is highly unconstrained - we supply no domain knowledge of what a wing should look like. In this brief overview we do not have space to discuss the GP in detail, and focus on the fitness evaluation by the simulator.

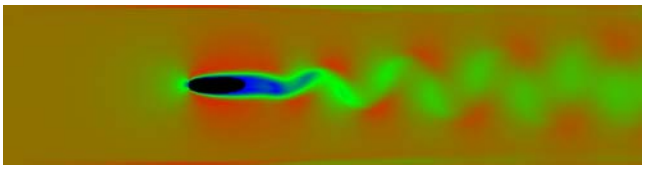

Figure 1: An example CFD simulation result, showing a simple object.

2. ALGORITHMS

Novelty. Although there have been many previous examples of CFD on GPUs, many of these techniques are unable to work with arbitrary shapes. A standard CFD fails to converge, or becomes unstable, when the parameters are incompatible with the shape under test. The solver developed here is also able to simulate a wide range of fluid types, from slow moving, viscous fluids to fast moving, light fluids.

The GPU architecture is exploited by choice of the method for solving linear equations during evaluation. Normally for the transient CFD solution method the conjugate gradient (CG) method would be the best choice for the linear solver since it converges much faster than the alternatives. The CG method is an iterative method, where each iteration involves a matrix-vector multiplication and a vector-vector sum. These operations are costly compared to the successive over-relaxation method (SOR) which is the method used here. This requires only a real value sum of multiplications for each neighboring node for each iteration. Although, the parallel SOR method requires more iterations to converge, it is more efficient on GPUs than the parallel CG method and this out-weighs the efficiency of convergence.

Our implementation is also designed to work with single GPUs, multiple GPUs in one node and multiple nodes. In our typical usage, evaluation of the population is distributed across a network of N computers each with a GPU.

Efficiency. CFD simulations are a cellular automata with a large number of nodes (in this example 2^{18} nodes). Nodes can be processed in parallel, and in a manner highly suited to the GPU architecture. Each node runs the same programs and updates are synchronized. Memory access is inherently coalesced because of the nature of the data representation. Further, the programs are relatively simple and fit within the limited local/shared memory and registers available to each shader processor.

Suitability and GPU-side implementation. All data were stored on the GPU, with minimal amounts of data returned to the host memory. All simulation work is handled on the GPU, with the CPU only monitoring for convergence. The evolutionary algorithm and shape generation is handled on the CPU side, and would be unsuitable for GPU implementation.

Elegance. The evolutionary component of the application is a simple, genetic programming implementation. The CFD simulation uses simple CUDA kernels managed by a higher level *C♯* wrapper. The *C♯* components are responsible for network communication and launching of the CUDA kernels. The distributed computation automatically scales with the availability of nodes, and is able to tolerate individual nodes becoming active/inactive. The system runs on machines in the student labs, and hence there are fluctuations in the availability of the devices.

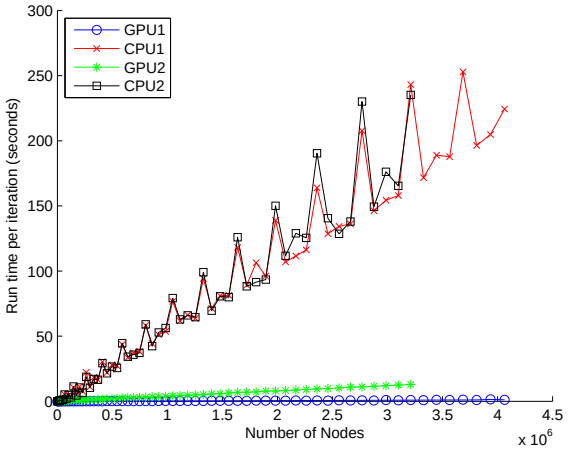

Figure 2: Run times per iteration vs. number of nodes in system

Portability. The core of the CFD software itself is written in CUDA. This has been tested on various nVidia cards and both on Windows and Linux. The host application is cross platform. The CFD software could be replaced by other implementations, e.g. OpenCl or CPU without affecting other parts of the application. The level of detail of the CFD simulation can be tailored to fit any GPU architecture. In future, we intend to use newer generation hardware to evaluate very high definition grids.

3. SPEED

Speed-up. Two different test setups were used to measure the effective speedup and scalability of a single CFD evaluation on two GPUs, GPU1 and GPU2 respectively. GPU2 is a relatively old and low-end onboard GPU (nVidia GeForce 8200), and GPU1 is a higher-end GPU (nVidia GeForce 8800). The CPUs used in the tests were similar, AMD Athlon 4850e at 2.5GHz and AMD Athlon 3200 at 2.0GHz. The results are shown in Fig. 2. The CPU code is written in $C\sharp$, which reduces the efficiency of this code if compared to a natively compiled code.

Resources. The core of the CFD code runs on 9 separate CUDA kernels. According to the CUDA Occupancy Calculator, the occupancy of these kernels range from 33% to 100%, with an average of 74%. The kernel that is called more than 95% of the time (the linear solver) has an occupancy of 100%. The limiting factor of the occupancy of the 33% kernels (3 of them) is the number of registers required. The limiting factor of the most used kernel is shared memory required per call, which increases with the number of threads per block.

Scalability. For each test setup the run times and speedups were tested for an increasing number of nodes (in the CFD system) to measure and compare the scalability of the GPU implementation to the CPU implementation. The results are shown in Fig. 2.

Communications overhead per task is approximately 0.6 seconds. This includes sending the mesh and evaluation configuration to each node and returning the fitness measure and a visualization data for each evaluation. Communications are all handled by .net's serialization, and the transfer data are compressed XML files. Only a small amount of data is therefore transmitted over the network, and network utilization is minimal (less than 1% of the 1Gb/s connection from the master node).

Preparing an individual mesh from the genotype is handled by the master node. This process is done asynchronously with the other tasks (using multiple cores) and takes negligible time. Hence, the system speed up scales linearly with the number of evaluation nodes.

4. EVOLUTIONARY COMPUTATION

Utility. These results show that using EC with highly complex fitness functions is practically achievable. Problems that require CFD simulations for fitness evaluation can be handled using low cost, readily available hardware and with minimum programming effort.

Practicality. We obtain up to a 20 times speed up per evaluation on GPU2 and up to about 400 times speed up per evaluation on GPU1. To tackle the design problem, we expect that we will need millions of evaluations. On a single CPU, to perform 1,000,000 evaluations would take for a 1024x512 system about 18 seconds per iteration, and there are about 1000 iterations per evaluation, which would be about 5 minutes per evaluation and about 10 years to solve the problem. Even with a cluster of CPUs, for example 50, this would still take about 70 days, which is impractical. Using a single GPU, this can be reduced about 1.4 years. With a cluster of GPUs this value is reduced by the number of GPUs in the cluster, for example with 50 GPUs this will take about 11 hours, which makes the task achievable.

Science. Human competitive results are becoming increasingly common in EC. The techniques proposed here opens up another potential avenue of challenges that can be tackled by automated design.

Previous approaches to evolving shapes (such as wing designs) are highly constrained by both the types of shapes that fast running simulators would allow. The system implemented here, is able to handle arbitrary designs and still obtain rapid results.

5. CONCLUSIONS

We implemented a CFD solver on GPUs for the purpose of doing shape design optimization using a GP methodology on an unconstrained problem shape. We found that the GPU CFD approach improves the efficiency enough to make the problem solvable in a practical amount of time, compared with the same CPU approach.

6. ACKNOWLEDGMENTS

WB acknowledges ACENET postdoctoral support for SH.