

Acceleration of a 2D Euler Flow Solver Using Commodity Graphics Hardware

T. Brandvik and G. Pullan

Whittle Laboratory, Department of Engineering, University of Cambridge

1 JJ Thomson Avenue, Cambridge, CB3 0DY, UK

Abstract

The implementation of a 2D Euler solver on graphics hardware is described. The graphics processing unit is highly parallelised and uses a programming model that is well suited to flow computation. Results for a transonic turbine cascade test-case are presented. For large grids (10^6 nodes) a 40 times speed-up compared to a Fortran implementation on a contemporary CPU is observed.

Keywords: CFD acceleration, GPU

1 Introduction

The compute power of modern graphics processing units (GPUs) is startling, Figure 1. The peak rate of floating point calculations per second (Flops) available on the GPU is an order of magnitude greater than the maximum achievable on a contemporary CPU. This is made possible by constraining the functionality of the GPU and by employing a highly parallelised architecture. The objective of the current work is to utilise this powerful and low cost hardware to perform

computational fluid dynamics (CFD) simulations.

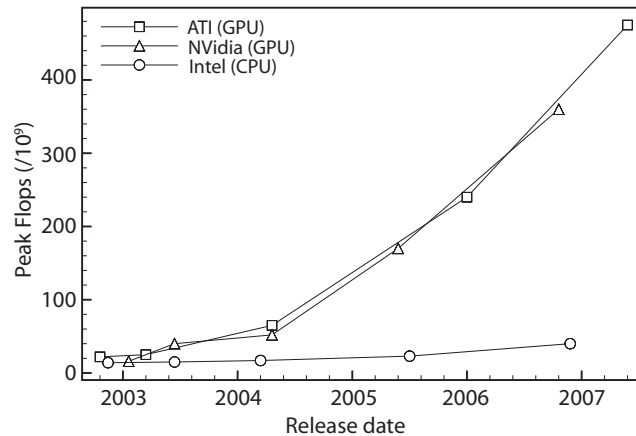


Figure 1: CPU (Intel) and GPU (Nvidia and ATI) floating point performance

There has been little previous work on the acceleration of engineering CFD codes using graphics hardware. The implementation of a 2D incompressible Navier-Stokes solver on a GPU is described by Harris[1]. Here, the emphasis is on producing visually realistic flows. Hagen *et al.*[2] present 2D and 3D Euler solvers with application to two types of spatially periodic flow: 2D shock-bubble interaction and 3D Rayleigh-Taylor instability. The solutions presented are physically sensible, though no comparison with experimental data is made, and speed-ups compared to an equivalent CPU code of 10-20 times are reported.

This paper presents the development of a 2D Euler solver for GPUs. The paper begins by describing current graphics hardware and how GPUs can be programmed. The implementation approach of the solver is discussed and the code is then used to solve transonic flow through a turbine cascade. Finally, the performance of the GPU solver is analysed.

2 Graphics acceleration hardware

2.1 The graphics pipeline

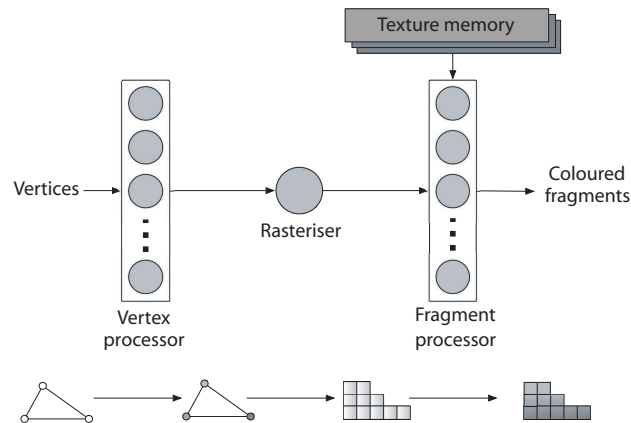


Figure 2: The graphics ‘pipeline’

GPUs are specialised co-processors. To adapt a CPU code to run on the GPU, some knowledge of the functionality of the GPU is required. The GPU is designed to transform a polygon model of a three-dimensional scene into an image rendered on the screen. This process is known as the graphics pipeline, Figure 2. The *vertex processor* receives a list of interconnected vertices (geometric primitives) from the CPU. At this stage, these vertices can be transformed and coloured. The *rasteriser* converts each geometric primitive into a group of *fragments* (pixels which are not yet displayed). Each fragment is then coloured by the *fragment processor* using information from the vertices or from *textures* (artwork) stored in GPU memory. To enable different transformations and shading effects, both the vertex and fragment processors are programmable and operate on 32-bit floating point numbers. The whole process is highly parallelised with many vertex and fragment processors on the GPU. Each geometric primitive can be

sent to a different vertex processor and each fragment to a different fragment processor so that each final pixel is evaluated independently.

2.2 General purpose GPU programming

As the non-graphics community has become aware that GPUs are fast, programmable and can handle floating point numbers, general purpose GPU (GPGPU) applications have begun to be developed. In general, the fragment processor is targetted because this supports reading from memory. At this point we note that any CPU code which involves the repeated application of the same function to the elements of a large array of data can be re-coded for the GPU and may demonstrate performance benefits. CFD is, therefore, an ideal application for the GPU.

There are a number of options open to the CFD developer who wishes to convince the GPU that it is rendering the latest computer game when it is actually solving the Navier-Stokes equations. For simplicity, a high level language, BrookGPU[3], was chosen for the present work. BrookGPU is the graphics hardware development of the Brook language ([4]) for ‘streaming’ architectures. Streaming is an approach for producing highly parallel computers which are easy to scale to an arbitrary number of processors. As compared to conventional languages, the functionality of a streaming program is highly constrained and centres around the use of *streams* and *kernels*. A stream is a collection of data records similar to a traditional array. A kernel is a function that is implicitly called for each record in an input stream to produce a new record in an output stream. A kernel has access to read only memory (gather streams) but the result of a kernel operation can only depend on the current record in the input stream - no dependency

on neighbouring records, for example, is permitted. Brook has previously been used to code a 2D Euler solver for the Merrimac streaming machine ([5]), but no results from the hardware are shown and the code is not aimed at GPUs. It is evident that the streaming paradigm is well suited to GPUs where: kernel=fragment program, input stream=fragment, gather stream=texture memory.

3 2D Euler implementation

The equations to be solved are the Euler equations for compressible inviscid flow in integral conservative form:

$$\int_{\text{vol}} \frac{\partial}{\partial t} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{bmatrix} d\text{vol} = \oint_A \begin{bmatrix} \rho u \\ p + \rho u^2 \\ \rho uv \\ \rho u h_0 \end{bmatrix} dA_x + \oint_A \begin{bmatrix} \rho v \\ \rho uv \\ p + \rho v^2 \\ \rho v h_0 \end{bmatrix} dA_y \quad (1)$$

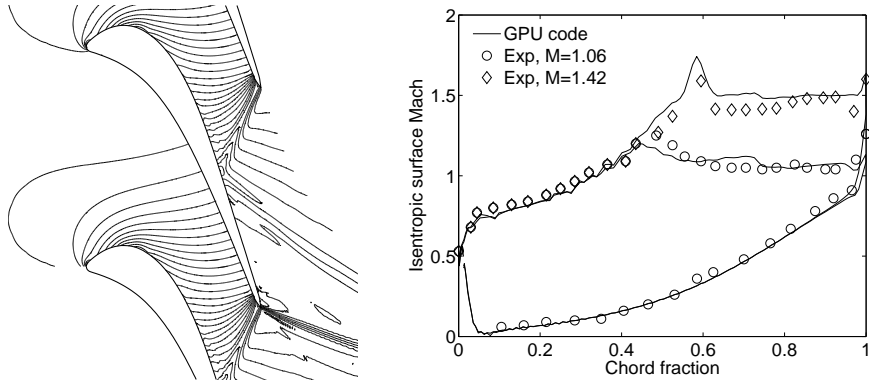
where ρ is density, (u, v) is the velocity vector, p is pressure, e is the specific total energy and h_0 is the specific stagnation enthalpy. In Equation (1), the integrals are over a volume vol with surface A , (A_x, A_y) is the inward facing surface area vector. The problem is discretised using a finite volume approach with vertex storage in a structured grid of quadrilateral elements. The spatial derivatives are evaluated using a centred, second order, scheme and the temporal ones to first order using the Scree scheme (Denton[6]). Second order smoothing, spatially varying time-steps and negative feedback (to damp out large residuals) are employed but no multigrid method

is implemented. The authors emphasise that this numerical approach is not novel and is based on the widely used methods of Denton[7, 8].

The implementation is split between the CPU and the GPU. All pre- and post-processing is done on the CPU, leaving only the computation itself to be performed on the GPU. For example, the CPU constructs the grid and evaluates the face areas and cell volumes. The initial guess of the flowfield is also done on the CPU. Each time-step of the computation then involves a series of kernels on the GPU which: evaluate the cell face fluxes; sum the fluxes into the cell; calculate the change in properties at each node; smooth the variables; and apply boundary conditions. Each kernel operates on all the nodes, i.e. no distinction is made between boundary nodes and interior nodes. This can cause difficulties if an efficient code is to be obtained. For example, the change in a flow property at a node is formed by averaging the flux sums of the adjacent cells; four cells surround an interior node, but only two at a boundary node. This problem is overcome using *dependent texturing*. The indices of the cells required to update a node are pre-computed on the CPU and loaded into GPU texture memory. For a given node, the kernel obtains the indices required and then looks up the relevant flux sums which are stored in a separate GPU texture. This avoids branching within the kernel.

4 Results

Results from the VKI McDonald transonic turbine test case ([9]) are presented in Figure 3. The GPU results were found to be identical to those from a Fortran implementation so only the former are presented. The grid used employed 208×208 points. The discrepancies with the measured



(a) Isobars at $M_2 = 1.42$ (b) Surface Mach number distributions
 Figure 3: VKI ‘McDonald’ transonic turbine test case

data are greatest on the late suction surface for the exit Mach number $M_2 = 1.42$ case. The calculated data in this region is very sensitive to any smearing of the shock emanating from the pressure-side trailing edge of the adjacent blade, Figure 3(a).

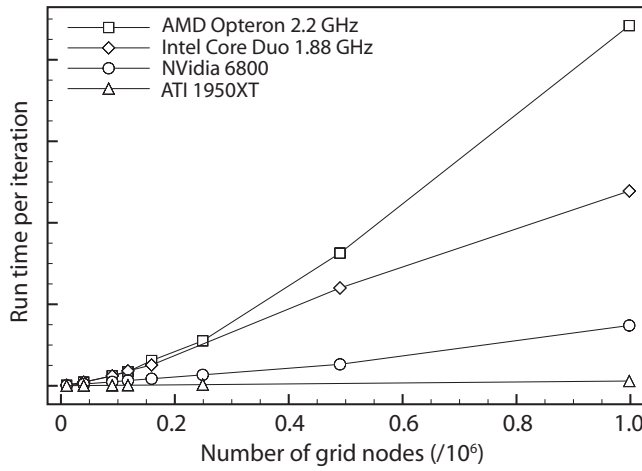


Figure 4: Run times for different CPUs and GPUs

The speed of the code was evaluated on four different CPUs and GPUs: from 2004, the AMD Opteron 248 CPU and Nvidia 6800GT GPU; from 2006 the Intel Core 2 Duo 1.88GHz CPU (single core used) and ATI 1950XT GPU. The CPU code is an efficient, optimise compiled,

Fortran program. The time taken to complete a fixed number of time steps was recorded for different grid sizes and the results are shown in Figure 4. The most recent GPU is seen to outperform its contemporary CPU by a factor of 40 for the largest grid sizes. The run times for the GPU solver were seen to increase approximately linearly with grid size, whereas, on the CPU, the time taken per node per step was found to increase for large grids. The latter observation is thought due to a reduction in the efficiency of cache usage as the memory requirements of the solver are increased.

5 Conclusions

The GPU is well suited to CFD simulations. Although the solver needs to be re-coded for the GPU, the large speed-ups that can be obtained ($40\times$ demonstrated here) make this task worthwhile. The authors believe that this type of many-core architecture, be it on the GPU or future incarnations of the CPU, is likely to be the future of scientific computation.

References

- [1] **Harris, M. J.** *GPU Gems*, chapter Fast Fluid Dynamics Simulation on the GPU. Addison-Wesley, 2004.
- [2] **Hagen, T. R., Lie, K.-A., and Natvig, J. R.** Solving the Euler Equations on Graphics Processing Units. *Comp. Sci. - ICCS*, 2006. pp. 220–227.

- [3] **Buck, I., Foley, T., Horn, D., Sugerman, J., Kayvon, F., Houston, M., and Hanrahan, P.** Brook for GPUs: Stream Computing on Graphics Hardware. In *ACM SIGGRAPH 2004*. New York, NY, 2004 pp. 777–786.
- [4] **Dally, W. J., Hanrahan, P., Erez, M., Knight, T. J., Labonte, F., Jayasena, N., Kapasi, U. J., Das, A., Gummaraju, J., and Buck, I.** Merrimac: Supercomputing with Streams. In *ACM SIGGRAPH 2003*. Phoenix, 2003 .
- [5] **Fatica, M., Jameson, A., and Alonso, J.** Stream-FLO: an Euler Solver for Streaming Architectures. *AIAA 2004-1090*, 2004.
- [6] **Denton, J. D.** The Effects of Lean and Sweep on Transonic Fan Performance. *TASK Quart.*, 2002. pp. 7–23.
- [7] **Denton, J. D.** An Improved Time Marching Method for Turbomachinery Flows. *ASME 82-GT-239*, 1982.
- [8] **Denton, J. D.** The Calculation of Three Dimensional Viscous Flow Through Multistage Turbomachines. *ASME J. Turbomachinery*, 1992. 114(1).
- [9] **Sieverding, C.** Base Pressure in Supersonic Flow. In *VKI LS: Transonic Flows in Turbomachinery*. 1976 .

Figure captions

Figure 1: CPU (Intel) and GPU (Nvidia and ATI) floating point performance

Figure 2: The graphics 'pipeline'

Figure 3: VKI 'McDonald' transonic turbine test case

Figure 4: Run times for different CPUs and GPUs