NATIONAL TECHNICAL UNIVERSITY OF ATHENS
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines

PhD Thesis

NUMERICAL SOLUTION OF AERODYNAMIC-AEROELASTIC PROBLEMS ON GRAPHICS PROCESSING UNITS

Xenofon S. Trompoukis

Athens 2012

Supervisor: **Kyriakos C. Giannakoglou,** Professor NTUA

Abstract

The main objective of this PhD thesis was the development of software for the solution of aerodynamic and aeroelastic problems, running on modern Graphics Processing Units (GPUs). On GPUs, the analysis and design of aerodynamic shapes can be done with substantial reduction in the computational cost, compared to the corresponding software running on Central Processing Units (CPUs).

Over the last 10 years, GPUs were evolved into many-core shared memory parallel systems. Their computational capabilities exceed those of modern CPUs by more than one order of magnitude. The GPUs, used in this PhD thesis, are based on the CUDA architecture developed by NVIDIA. Regarding hardware, the GeForce 280, 285 and Tesla M2050 cards were used. The Tesla M2050 graphics cards, in particular, are part of a GPU-cluster, possessed by the PCopt/LTT, which comprises four interconnected blade servers with three Tesla M2050 each. So, for the solution of large scale aerodynamic and aeroelastic problems, a cluster with twelve Tesla M2050 units was used.

In this PhD thesis, GPU-solvers for the 2D steady/unsteady Navier-Stokes equations for compressible fluids and the 3D steady/unsteady Euler equations for compressible fluids have been developed. Both GPU-solvers use dynamic unstructured/hybrid grids which are able to deform, following the deflection/deformation of the body surface. The grid quality is ensured by the spring analogy method. For turbulent flows, the one-equation turbulence model of Spalart-Allmaras is used to effect closure. The integration of the flow equations on the unstructured grids is carried out using the finite volume method, with vertex-centered storage. The GPU-enabled software reproduces the results of the corresponding CPU solver, which has been thoroughly validated in previous PhD theses at PCopt/LTT. Therefore, emphasis was exclusively laid on the reduction of the computational cost.

The major drawback of programmable GPUs, with respect to nowadays CPUs, is their limited cache memory. Cache memories are low-latency memories, used for temporary data storage. The greater the size of the available cache memories, the

faster a program runs. Thus, the parallel efficiency of any GPU-code is strongly related with memory handling. From the memory handling point of view, the vertex-centered approach of the finite volume technique is the worst case, compared either to the use of structured grids or even unstructured grid with the cell-centered finite volume approach. This is due to the lack of structure in the numbering of the grid nodes and, compared to cell-centered schemes, the variable number of neighbors per grid node.

To program CFD codes on GPUs required rewriting the existing CPU-solver (programmed in FORTRAN90) in the programming language supported by CUDA (C++ with some extensions for the efficient control of the GPU in that period of time). Even if the firstly developed GPU-code could solve the flow equations faster than the corresponding CPU-software, its parallel speed-up was not as high as expected. In order to increase the parallel efficiency of the GPU-solver, the following actions were taken:

- Different programming approaches for the computation of the flow fluxes were programmed and assessed in terms of parallel speed-up.
- Since GPUs contain extra memories, other than the main (device or global) one, the programming technique was reconsidered so as to maximize their exploitation. These memories are the texture, constant, local and shared ones. The cached texture, cached constant and non-cached local memory spaces reside in device memory. On the other hand, the shared memory is distributed to the GPU multiprocessors and is used for interchanging data between the processes executed in parallel on the same multiprocessor. The efficient use of all GPU memories increased the parallel speed-up of the GPU-solver. Among other:
  (a) It was ensured that concurrently running processes access nearby (or even consecutive) global memory spaces.
  (b) CPU and GPU processes may run in parallel.
  (c) The computational grid was divided into subdomains and the nodes of each subdomain were renumbered and sorted, based on the number of their neighbors, in ascending order.
- A mixed precision arithmetic (MPA) solver has been proposed. Since the discretized flow equations are solved for the flow variables' corrections, a less accurate numerical scheme for the computation of the l.h.s. coefficient matrices and an accurate one for the r.h.s. terms (i.e. the residuals) can be used. In the proposed MPA code variant, the l.h.s. coefficient matrices are stored into single precision (SPA) variables whereas the residuals are stored into double precision variables (DPA). This reduces the total number of global memory accesses and, thus, increases the parallel efficiency of the GPU-code. Besides, MPA does not harm the accuracy of the solution since the residuals are computed with double precision.

These actions called for the complete restructuring of the initial GPU-code. The final GPU-code solves the flow equations about 20 times faster compared to the initial one. In order to measure the parallel speed-up of the programmed GPU-enabled code(s), a number of inviscid and turbulent flow problems around an airfoil and a wing were solved. In these cases, unstructured grids (consisting of triangular or tetrahedral elements for 2D or 3D cases respectively) with different numbers of grid nodes were used, on the GTX 280, 285 and Tesla M2050 graphics cards. It was shown that the flow model (inviscid of turbulent), the grid size and the compute capabilities of the GPUs affect the parallel speed-up of the GPU-solver. As a consequence, the examined problems give different speed-ups. Large scale problems take more advantage of the GPUs, as larger grids achieve higher speed-ups. For 2D turbulent flows of compressible fluids, the highest speed-up is about 60x, 90x and 110x, using a Tesla M2050, compared to a single core of a modern CPU, using double, mixed or simple precision arithmetic, respectively. The same computations on a GTX 285 yield about 30% less speed-up. The highest recorded speed-ups for the solution of the 3D Euler equations is 40x, 55x and 90x, on Tesla M2050, with DPA, MPA and SPA, respectively. The corresponding speed-ups for GTX285 are about 25% less. For the measurement of the aforementioned speed-ups, the CPU-solver used DPA and was executed on a single core of an Intel Xeon CPU, 2.00GHz, 4096 MB cache memory.

For the purpose of comparison, an Euler solver using either structured or unstructured grids and the cell-centered approach was also programmed. Moreover, the present PhD thesis also contributed to the efficient porting of the adjoint equation solvers, from the CPU to the GPU, so as to reduce the cost of gradient-based optimization processes.

As far as the aerodynamic analysis is of concern, the GPU-solver was used for the study (a) of the dynamic interaction between the boundary layer and shock waves (buffeting) on the OAT15A airfoil, and (b) the impact of a synthetic jet in the reduction of the separation region downstream of a bump geometry. Moreover, the GPU-solver is used for the prediction of inviscid flow around a civil aircraft and inside a hypersonic compressor cascade. For these problems, the speed-up ranges from 40x to 45x using a single GTX 285 graphic card vis-à-vis to a single core of a modern CPU. It is worth noting that these GPUs have been plugged in some old personal computers. Using the Tesla M2050 graphics cards, the speed-up increased by ~20%, compared to GTX 285.

Having developed efficient GPU flow solvers, the next step was to use them in evolutionary algorithm (EA) based optimization (EASY software). The optimization of the steady suction control parameters, for the minimization of the length of the separation region downstream a fixed bump geometry inside a duct, was carried out on a number of GPUs (one GPU per evaluation). Thus, the gain from using GPUs, instead of CPUs, was superposed to the gain from the parallel evaluation of the population members.

Based on the different speed-ups and prediction accuracy of the SPA and MPA GPU implementations of the Navier-Stokes equations solver, a hierarchical optimization method which is suitable for GPUs is proposed and demonstrated in turbulent 2D flow problems. The search for the optimal solution(s) splits into two EA-based levels, with different evaluation tools each. The low level EA uses the very fast SPA GPU implementation with relaxed convergence criteria for the inexpensive evaluation of candidate solutions. Promising solutions are regularly broadcast to the high level EA which uses the MPA GPU implementation of the same flow solver. Single- and two-objective aerodynamic shape optimization problems are solved using the developed software. Namely, the design of an isolated airfoil for maximum lift and minimum drag coefficient and the design of a compressor cascade airfoil for minimum pressure losses were carried out using the aforementioned two-level EA. The gain from using the hierarchical EA instead of a conventional (single-level) EA was superposed to the gain from using GPUs instead of CPUs.

As far as aeroelastic problems are of concern, the GPU-solver was validated in a pitching airfoil case and used for the prediction of the flutter boundaries of the airfoil, with two or three degrees of freedom. In these cases, the motion of the airfoil is constrained by a linear (plunging) and a torsional (pitching) spring. In the three-degree of freedom airfoil case, the motion of the airfoil trailing edge (flap) is constrained by means of an additional torsional spring. The numerical results obtained by using the developed GPU-solver are in good agreement with published numerical results using different CFD tools.

Due to memory limitations, a single GPU cannot be used for solving large scale aerodynamic/aeroelastic problems. For large scale problems, GPU clusters are employed. In the present thesis, a single CPU-thread was used to control the available GPUs per computational node. The communication between the GPUs of the interconnected nodes is based on the MPI parallel protocol. The parallel GPU-solver was used for the analysis of a Blended-Wing-Body (BWB) civil aircraft. This type of aircrafts are non-commercial having many aerodynamic benefits (higher ratio lift to drag, lower fuel consumption per passenger and mile, reduced noise production, etc) compared to commercial aircrafts with equivalent specifications. The corresponding studies were carried out in the context of a European project (Active Control for Flexible 2020 Aircraft, ACFA 2020), funded by the European Union. In this computationally demanding application, the GPUs were used for the reduction of the wall clock time; the use of three Tesla M2050 on the same computational node was about six times faster than 64 CPU-cores. Steady and unsteady inviscid flows around the BWB aircraft are presented; in the unsteady cases, a control surface or the aircraft was forced to oscillate/deform periodically.

In conclusion, in this PhD thesis, GPUs were used for the solution of aerodynamic and aeroelastic problems using unstructured computational grids. Speed-ups from the use of GPUs ranging between 40x and 110x are reported. This substantial decrease in computational cost affects positively both the aerodynamic/aeroelastic analysis of aerodynamic bodies and the optimization-design via EAs, where a large number of evaluations must be performed. Since, larger grids achieve higher speed-ups, large scale problems take more advantage of the GPUs. This fact extends the range of industrial applications that can be carried out using CFD analysis tools. Finally, it should be noted that the proposed programming techniques can easily be implemented in any other GPU-enabled software solving general purpose PDEs on unstructured grids.