Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

CFD-based analysis and two-level aerodynamic optimization on graphics processing units

I.C. Kampolis, X.S. Trompoukis, V.G. Asouti, K.C. Giannakoglou *

National Technical University of Athens, Lab. of Thermal Turbomachines, Parallel CFD & Optimization Unit, P.O. Box 64069, Athens 15710, Greece

A R T I C L E I N F O

Article history:
Received 21 May 2009
Received in revised form 24 September 2009
Accepted 3 November 2009
Available online 13 November 2009

Keywords:
Graphics processing units
Computational fluid dynamics
Aerodynamic shape optimization
Evolutionary algorithms

A B S T R A C T

This paper presents the porting of 2D and 3D Navier–Stokes equations solvers for unstructured grids, from the CPU to the graphics processing unit (GPU; NVIDIA's Ge-Force GTX 280 and 285), using the CUDA language. The performance of the GPU implementations, with single, double or mixed precision arithmetic operations, is compared to that of the CPU code.

Issues regarding the optimal handling of the unstructured grid topology on the GPU, particularly for vertex-centered CFD algorithms, are discussed. Restructuring the existing codes was necessary in order to maximize the parallel efficiency of the GPU implementations. The mixed precision implementation, in which the left-hand-side operators are computed with single precision, was shown to bridge the gap between the single and double precision speed-ups. Based on the different speed-ups and prediction accuracy of the aforementioned GPU implementations of the Navier–Stokes equations solver, a hierarchical optimization method which is suitable for GPUs is proposed and demonstrated in inviscid and turbulent 2D flow problems. The search for the optimal solution(s) splits into two levels, both relying upon evolutionary algorithms (EAs) though with different evaluation tools each. The low level EA uses the very fast single precision 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 mixed precision GPU implementation of the same flow solver. Single- and two-objective aerodynamic shape optimization problems are solved using the developed software.

1. Introduction

The continuous increase in performance of GPUs has extended their use from the gaming community to game physics calculations and, lately, to number-crunching applications in the field of computational physics [4,21,42]. Their increased flexibility and performance allowed the implementation of data-parallel algorithms that efficiently unlock their peak computational capabilities by exceeding that of high-end CPUs, combining a great number of floating point operations per second (GFLOPS) and high memory bandwidth. Moreover, their mass production made them a cheaper, viable alternative to conventional high-performance computing systems.

GPUs from both major manufacturers (ATI & NVIDIA) associated with a number of software programming models (Cg [37], Brook [7], Sequoia [15], CTM [1], CUDA [40] to mention a few of them) are available to scientists for making GPUs appropriate to cope with high-performance scientific applications. In this paper, NVIDIA's graphics cards and CUDA are used. CUDA is an extended subset of the C language and is supported by all latest NVIDIA's graphics cards. It uses the GPU as a device suitable for parallel computations with its own random access memory (device/global memory) capable of concurrently executing a large number of threads. The latter are grouped together into blocks; many blocks comprise a grid. Grids of blocks execute the code (referred to as the kernel) on the GPU, using the *single instruction multiple thread* (SIMT) model. Limited shared memory (due to hardware restrictions) as well as synchronization instructions are available to the threads forming the same block.

A wide range of applications which benefit from the excellent parallel performance of GPUs can be found in the literature [42]. However, applications in computational fluid dynamics (CFD) are still limited. In [20], a GPU implementation of the multigrid method for the solution of boundary value heat and fluid flow problems, using the vorticity-streamfunction formulation, was presented. Kruger and Westermann [36] proposed a framework for the implementation of explicit and implicit numerical methods on graphics hardware (ATI 9800 graphics card) and showed applications related to the 2D wave equation and the incompressible Navier–Stokes equations. Bolz et al. [4] describe a conjugate gradient solver for sparse matrices resulting from the use of unstructured grids

* Corresponding author.
E-mail address: kgianna@central.ntua.gr (K.C. Giannakoglou).

0045-7825/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.cma.2009.11.001

and a multigrid solver for structured grids, on NVIDIA's Ge-Force FX cards. Goddeke et al. [19] used many cluster nodes with NVIDIA's Quadro FX4500 PCIe and addressed the issue of limited precision on GPUs (by that period of time) by applying a mixed precision iterative refinement technique. A detailed description of the GPU implementation of a Navier–Stokes flow solver for structured grids was presented in [22]. In [9], this was extended to 3D flows by laying emphasis to realistic and accurate visualization effects for graphics applications. Implementations to compressible fluid flows were firstly presented in [21]. Brandvik and Pullan [5,6] focus on performance comparisons between GPU and CPU codes for the solution of 2D and 3D Euler equations and reported considerable speed-ups using exclusively structured grids. Applications to complex geometries, even for hypersonic flows, can be found in [11]. A first application in 3D unstructured grids for inviscid, compressible flows on NVIDIA Tesla GPU is presented in [8]. Although [8] has certain similarities to the present paper, it is based on the cell-centered whereas this paper on the vertex-centered finite volume technique. As it will be shown as the paper develops, there are important differences between the two schemes as far as porting on GPUs is of concern.

In view of the above, in the present paper, a solver for the 2D steady state, Navier–Stokes equations for compressible fluids on unstructured grids with triangular elements, coupled with a one-equation turbulence model, is firstly presented; the programming details for porting the code to the GPU are discussed. The first part of this study focuses on both single (*SPA*) and double (*DPA*) precision arithmetic GPU implementations, in order to quantify their speed-ups compared to the CFD code running on the CPU. A mixed precision arithmetic (*MPA*) implementation, which uses *SPA* for the left-hand-side (*l.h.s.*) part of the discretized equations and *DPA* for the right-hand-side terms (*r.h.s.*, i.e. the residuals), is devised. Below, the three GPU implementations with *SPA*, *MPA* and *DPA* will be referred to as GPU_{SP}, GPU_{MP} and GPU_{DP}, respectively. The solver running on the CPU implements *DPA*; the speed-ups of all GPU-enabled variants are measured with respect to the same CPU code. The proposed GPU_{MP} leads to high speed-ups without damaging the prediction accuracy of the GPU_{DP}. A 3D Euler equations GPU-enabled solver (using *SPA*, *MPA* and *DPA*, too) for unstructured grids with tetrahedral elements is also presented and assessed, in terms of speed-ups, with the corresponding CPU code.

The second part of this paper deals with aerodynamic shape optimization, by exploiting the CFD software variants which have already been ported on GPUs. In particular, the GPU-enabled flow solvers are used as analysis tools in the context of a hierarchical EA, to efficiently solve aerodynamic shape optimization problems. In such a method, the search for the optimal solution(s) is considerably accelerated by the combined use of (a) a low cost, low fidelity flow solver (GPU_{SP}) which undertakes the exploration of the design space on the low level and (b) a high fidelity flow solver (GPU_{MP}), with higher cost, which is mainly used to refine the most promising solutions by injecting them into the high level search algorithm. The present applications include single- and multi-objective optimizations of a compressor cascade and an isolated airfoil.

A final statement should be made. Nowadays, "personal supercomputers", i.e. clusters of CPUs and GPUs, allow a significant amount of performance at a reasonable cost-price. The present CFD software may certainly run on these clusters, after defining an appropriate number of grid subsets exchanging data for physically coincident grid nodes which reside on different grid subsets. However, this is beyond the scope of this paper. The reason is simple: since the GPU code is used to support an EA, where candidate solutions can be simultaneously and independently evaluated and due to the limited number of available GPUs, it was decided that each CFD code runs on a single GPU. We thus minimize the communication overhead and applications are limited by the available memory.

2. 2D/3D CFD Implementations on GPUs

This section presents the porting of the 2D and 3D Navier–Stokes equations solvers to the GPU. The basic features of the time-marching Navier–Stokes flow solver (CPU implementation) are first presented. Programming issues for the GPU implementations are then discussed by taking into consideration the specific characteristics/architecture of the available NVIDIA graphics cards. The section concludes with performance comparisons between CPU and GPU for 2D and 3D, inviscid and/or turbulent steady-state flows around an isolated airfoil, an aircraft and in a compressor cascade.

2.1. The Navier–Stokes equations solver – implementation on CPUs

The Navier–Stokes equations for a compressible fluid are written in vector form as

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{f}_i^{inv}}{\partial x_i} - \frac{\partial \vec{f}_i^{vis}}{\partial x_i} = 0, \tag{1}$$

with $i = 1, 2$ in 2D ($i = 1, 2, 3$ in 3D) and x_i the cartesian coordinates. $\vec{U} = [\varrho, \varrho\vec{u}, E]^T$ is the vector of conservative variables. The inviscid ($\vec{f}_i^{inv}$) and viscous ($\vec{f}_i^{vis}$) fluxes are given by

$$\vec{f}_i^{inv} = \begin{bmatrix} \varrho u_i \\ \varrho u_i \vec{u} + p\vec{\delta}_i \\ u_i(E+p) \end{bmatrix}, \quad \vec{f}_i^{vis} = \begin{bmatrix} 0 \\ \vec{\tau}_i \\ u_j \tau_{ij} + q_i \end{bmatrix}, \tag{2}$$

where ϱ is the density, $\vec{u}$ is the velocity vector, $\vec{\tau}_i = [\tau_{i1}, \tau_{i2}]^T$ ($\vec{\tau}_i = [\tau_{i1}, \tau_{i2}, \tau_{i3}]^T$ in 3D) are the viscous stresses, $\vec{\delta}_i = [\delta_{i1}, \delta_{i2}]^T$ ($\vec{\delta}_i = [\delta_{i1}, \delta_{i2}, \delta_{i3}]^T$ in 3D) the Kronecker symbols and $q_i = k\frac{\partial T}{\partial x_i}$ the thermal flux components.

The solution of Eqs. (1), on unstructured grids with triangular (2D) or tetrahedral (3D) elements, is based on the vertex-centered finite volume technique and a second-order upwind scheme for the inviscid fluxes. On a 2D grid, Fig. 1 shows the finite volume Ω_P defined around node P; in 3D cases, finite volumes are defined in a similar way. The integration of the governing equations over Ω_P gives

$$\frac{\Omega_P}{\Delta t_P} \Delta \vec{U}_P + \sum_{Q \in nei(P)} \left[\vec{\Phi}_{PQ}^{inv} - \vec{\Phi}_{PQ}^{vis} \right] (ab) = 0, \tag{3}$$

where Δt_P is the local pseudo-time step and $\vec{\Phi}_{PQ}^{inv}, \vec{\Phi}_{PQ}^{vis}$ are the inviscid and viscous numerical fluxes across ab, i.e. the interface of Ω_P and Ω_Q.

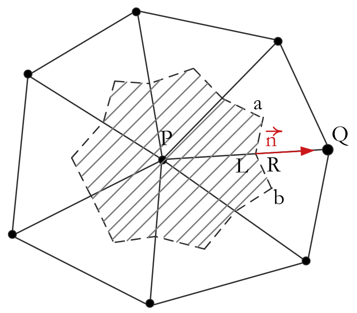

Fig. 1. Vertex-centered finite volume Ω_P (hatched area) defined around node P of an unstructured grid with triangular elements.