OPT-*i*
An International Conference on
Engineering and Applied Sciences Optimization
M. Papadrakakis, M.G. Karlaftis, N.D. Lagaros (eds.)
Kos Island, Greece, 4-6, June 2014

THE CONTINUOUS ADJOINT METHOD ON GRAPHICS PROCESSING UNITS FOR COMPRESSIBLE FLOWS

Xenofon S. Trompoukis, Konstantinos T. Tsiakas, Mehdi Ghavami Nejad, Varvara G. Asouti, and Kyriakos C. Giannakoglou

National Technical University of Athens, Parallel CFD & Optimization Unit
Iroon Polytechniou 9, 15780, Athens, Greece
e-mail: (xeftro,tsiakost)@gmail.com, (mehdi,vasouti)@mail.ntua.gr, kgianna@central.ntua.gr

Keywords: Continuous Adjoint Method, Sensitivity Analysis, CFD on Graphics Processing Units

Abstract. *This paper presents the development and application of the continuous adjoint method on Graphics Processing Units (GPUs) for use in aerodynamic shape optimization problems. The techniques used for optimally porting the primal and adjoint solvers on GPUs are described. This paper focuses on the optimal GPU memory access, mixed-precision arithmetics and different scatter-gathering algorithms. The resulting solver is approximately 50 times faster than the equivalent CPU solver. The solver is used in the design/optimization of a 2D airfoil for, maximum lift and minimum drag. A significant decrease in optimization turnaround time is achieved and results are presented. The solver is also used for the computation of the drag sensitivity map on the surface of a transonic wing.*

1 INTRODUCTION

Gradient–based optimization supported by the adjoint method is widely used, for the design–optimization of aerodynamic shapes. The great advantage of the adjoint methods is the low cost of computing the gradients of any objective function, which is almost equal to that of the solution of the flow equations and independent of the number of design variables.

This paper deals with the continuous adjoint method for compressible flows. Over and above the gain achieved by the adjoint method, we are mostly concerned with the reduction of the optimization turnaround time. To this end, the solution of both the state and adjoint equations is carried out on GPU clusters.

During the last ten years, CFD codes, for either structured [6, 3, 5, 8] or unstructured [4, 7, 2, 9] computational grids, have been ported to GPUs. GPUs use their dedicated memories (global, local, constant, texture and shared) with different structure/access patterns than CPUs. Memory access/handling plays a significant role in the parallel efficiency of any GPU implementation. As a consequence, the computational grid type, the flow variable storage pattern, the discretization and solution schemes all affect the performance of a GPU–enabled code. For structured grids, the organized topology of data and the corresponding memory accesses lead to significant speed–ups. Unless special care is taken, this is not necessarily the case of GPU codes for unstructured grids, since their performance depends also on the spatial discretization scheme. In the cell–centered finite volume scheme, the number of neighbors to each cell barycenter, where the flow variables are stored, is small and fixed resulting to better memory accesses. In contrast, a vertex–centered finite volume scheme, where the number of adjacent nodes to any mesh node may vary a lot, asks for a customized memory handling. The authors group has experience in developing GPU–enabled codes for unstructured grids with the vertex–centered finite volume scheme by optimally using all the available GPU memories [7, 2, 12].

This paper deals with the continuous adjoint method for 3D compressible flows, extending the work for incompressible flows presented in [1]. The shape optimization of an isolated airfoil and the computation of sensitivity maps on a transonic wing are presented. In the second case, we do not proceed with the shape optimization since, in this paper, emphasis is laid on how to compute the gradient with low wall clock time cost, rather than the optimization loop itself.

2 THE PRIMAL SYSTEM OF EQUATIONS

The (primal) system of governing PDEs comprises the Navier–Stokes equations for steady–state compressible flows,

$$R_n = \frac{\partial U_n}{\partial t} + \frac{\partial f_{nk}^{inv}}{\partial x_k} - \frac{\partial f_{nk}^{vis}}{\partial x_k} = 0 \tag{1}$$

where $k=1,3,\ n=1,5$ for 3D flows, where $U=[\rho,\rho v_k,E]^T$ is the vector of conservative variables with ρ the density, v_k the velocity components and E the total energy per unit volume. The inviscid and viscous fluxes are given by

$$f_{nk}^{inv} = \begin{bmatrix} \rho v_k \\ \rho v_{n-1} v_k + p\delta_{(n-1)k} \\ v_k(E+p) \end{bmatrix}, \ f_{nk}^{vis} = \begin{bmatrix} 0 \\ \tau_{(n-1)k} \\ v_l \tau_{kl} + q_k \end{bmatrix} \tag{2}$$

where p is the static pressure, $q_k = \kappa \partial T/\partial x_k$ the thermal heat flux components with $\kappa = \frac{C_P}{Pr}\mu$, δ_{ij} the Kronecker symbol and τ_{ij} the viscous stresses, $\tau_{ij} = \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) + \lambda\delta_{ij}\frac{\partial v_k}{\partial x_k}$, with μ

the molecular viscosity and $\lambda = -\frac{2}{3}\mu$. Let Ω be the computational domain. Since external aerodynamic problems are concerned, the boundary S of Ω comprises the solid wall S_w and the far–field boundary S_∞.

3 MATHEMATICAL FORMULATION OF THE CONTINUOUS ADJOINT METHOD

In this paper, the objective function to be minimized is the projection of the aerodynamic force acting on a body along a direction defined by the vector r_k. This is expressed as

$$F = \int\limits_{S_w} p n_k r_k dS - \int\limits_{S_w} \tau_{km} n_m r_k dS \tag{3}$$

where n_m are the components of the unit vector normal to S_w. The body shape S_w depends on the value–set of N design variables b_i, $i \in [1, N]$, through a selected parameterization scheme. The variation in F due to any variation in b_i is expressed as

$$\frac{\delta F}{\delta b_i} = \int\limits_{S_w} \frac{\delta p}{\delta b_i} n_k r_k dS + \int\limits_{S_w} p \frac{\delta}{\delta b_i}(n_k r_k dS) - \int\limits_{S_w} \frac{\delta \tau_{km}}{\delta b_i} n_m r_k dS - \int\limits_{S_w} \tau_{km} \frac{\delta}{\delta b_i}(n_m r_k dS) \tag{4}$$

The formulation of the continuous adjoint method starts by computing the variations of the augmented objective function $F_{aug} = F + \int\limits_{\Omega} \Psi_n R_n d\Omega$ w.r.t b_i, as follows

$$\frac{\delta F_{aug}}{\delta b_i} = \frac{\delta F}{\delta b_i} + \int\limits_{\Omega} \Psi_n \frac{\partial R_n}{\partial b_i} d\Omega + \int\limits_{\Omega} R_n \frac{\partial \Psi_n}{\partial b_i} d\Omega + \int\limits_{S} \Psi_n R_n n_k \frac{\delta x_k}{\delta b_i} dS \tag{5}$$

The last two surface integrals vanish, since upon convergence of the state equations, $R_n = 0$. The first field integral in eq. 5 is written as

$$\int\limits_{\Omega} \Psi_n \frac{\partial R_n}{\partial b_i} d\Omega = \int\limits_{\Omega} \Psi_n \frac{\partial}{\partial x_k} \left(\frac{\partial f_{nk}^{inv}}{\partial b_i} \right) d\Omega - \int\limits_{\Omega} \Psi_n \frac{\partial}{\partial x_k} \left(\frac{\partial f_{nk}^{vis}}{\partial b_i} \right) d\Omega \tag{6}$$

The inviscid part of eq. 6 is further developed as

$$\int\limits_{\Omega} \Psi_n \frac{\partial}{\partial x_k} \left(\frac{\partial f_{nk}^{inv}}{\partial b_i} \right) d\Omega = \int\limits_{S} \Psi_n \frac{\partial f_{nk}^{inv}}{\partial b_i} n_k dS - \int\limits_{\Omega} A_{nmk} \frac{\partial \Psi_n}{\partial x_k} \frac{\partial U_m}{\partial b_i} d\Omega \tag{7}$$

since $f_{nk}^{inv} = A_{nmk} U_m$, with A_{mnk} the flux Jacobians. In eq. 7, the surface integral is decomposed into two integrals, along S_w and S_∞. Since changes in the design variables do not affect S_∞

$$\begin{aligned} \int\limits_{S} \Psi_n \frac{\partial f_{nk}^{inv}}{\partial b_i} n_k dS &= \int\limits_{S_w} \Psi_n \frac{\delta \left(f_{nk}^{inv} n_k dS \right)}{\delta b_i} - \int\limits_{S_w} \Psi_n f_{nk}^{inv} \frac{\delta \left(n_k dS \right)}{\delta b_i} - \int\limits_{S_w} \Psi_n \frac{\partial f_{nk}^{inv}}{\partial x_l} \frac{\delta x_l}{\delta b_i} n_k dS \\ &+ \int\limits_{S_\infty} \Psi_n A_{nmk} n_k \frac{\delta U_m}{\delta b_i} dS \end{aligned} \tag{8}$$

By applying either the no–slip or no–penetration condition (for viscous or inviscid flows, respectively) and setting Ψ_n=0 at the far–field boundaries to eliminate the last term in eq. 8, the above surface integral becomes

$$\int_S \Psi_n \frac{\partial f_{nk}^{inv}}{\partial b_i} n_k dS = \int_{S_w} \Psi_{k+1} n_k \frac{\delta p}{\delta b_i} dS + \int_{S_w} \left(\Psi_{k+1} p - \Psi_n f_{nk}^{inv}\right) \frac{\delta (n_k dS)}{\delta b_i} - \int_{S_w} \Psi_n \frac{\partial f_{nk}^{inv}}{\partial x_l} \frac{\delta x_l}{\delta b_i} n_k dS \qquad (9)$$

The viscous integral in eq. 6 is written as

$$-\int_\Omega \Psi_n \frac{\partial}{\partial x_k}\left(\frac{\partial f_{nk}^{vis}}{\partial b_i}\right) d\Omega = -\int_S \Psi_n \frac{\partial f_{nk}^{vis}}{\partial b_i} n_k dS + \int_\Omega \frac{\partial \Psi_n}{\partial x_k} \frac{\partial f_{nk}^{vis}}{\partial b_i} d\Omega \qquad (10)$$

where

$$\int_\Omega \frac{\partial \Psi_n}{\partial x_k} \frac{\partial f_{nk}^{vis}}{\partial b_i} d\Omega = \int_\Omega \frac{\partial \tau_{mk}}{\partial b_i}\left(\frac{\partial \Psi_{m+1}}{\partial x_k} + v_m \frac{\partial \Psi_5}{\partial x_k}\right) d\Omega + \int_\Omega \frac{\partial \Psi_5}{\partial x_k} \tau_{km} \frac{\partial v_m}{\partial b_i} d\Omega + \int_\Omega \frac{\partial \Psi_5}{\partial x_k} \frac{\partial q_k}{\partial b_i} d\Omega \qquad (11)$$

If μ is a constant fluid property, $\frac{\partial \mu}{\partial b_i} = 0$, through the Green–Gauss theorem and the no–slip condition we get

$$\int_\Omega \frac{\partial \tau_{mk}}{\partial b_i}\left(\frac{\partial \Psi_{m+1}}{\partial x_k} + v_m \frac{\partial \Psi_5}{\partial x_k}\right) d\Omega = -\int_\Omega \frac{\partial \tau_{km}^{adj}}{\partial x_k} \frac{\partial v_m}{\partial b_i} d\Omega - \int_{S_w} \tau_{km}^{adj} \frac{\partial v_m}{\partial x_l} \frac{\delta x_l}{\delta b_i} n_k dS \qquad (12)$$

where

$$\tau_{km}^{adj} = \mu\left(\frac{\partial \Psi_{m+1}}{\partial x_k} + \frac{\partial \Psi_{k+1}}{\partial x_m} + v_m \frac{\partial \Psi_5}{\partial x_k} + v_k \frac{\partial \Psi_5}{\partial x_m}\right) + \lambda \delta_{km}\left(\frac{\partial \Psi_{l+1}}{\partial x_l} + v_l \frac{\partial \Psi_5}{\partial x_l}\right) \qquad (13)$$

are the so–called adjoint stresses. In eq. 12 it was assumed that the gradient of the adjoint variable along S_∞ is negligible. The last integral in eq. 11 is written as

$$\int_\Omega \frac{\partial \Psi_5}{\partial x_k} \frac{\partial q_k}{\partial b_i} d\Omega = -\int_\Omega \frac{\partial}{\partial x_k}\left(\kappa \frac{\partial \Psi_5}{\partial x_k}\right) \frac{\partial T}{\partial b_i} d\Omega + \int_{S_w} \kappa \frac{\partial \Psi_5}{\partial x_k} n_k \frac{\partial T}{\partial b_i} dS \qquad (14)$$

where, also, the gradients of the adjoint variables along S_∞ are neglected. Based on the perfect gas state equation, the field integral on the r.h.s member of eq. 14 becomes

$$-\int_\Omega \frac{\partial}{\partial x_k}\left(\kappa \frac{\partial \Psi_5}{\partial x_k}\right) \frac{\partial T}{\partial b_i} d\Omega = -\int_\Omega \frac{\partial}{\partial x_k}\left(\kappa \frac{\partial \Psi_5}{\partial x_k}\right)\left(\frac{T}{p}\frac{\partial p}{\partial b_i} - \frac{T}{\rho}\frac{\partial \rho}{\partial b_i}\right) d\Omega \qquad (15)$$

The surface integral in eq. 10 is decomposed into two integrals over S_w and S_∞ and since $\Psi_n = 0$ along S_∞, applying the no–slip condition and assuming the solid wall to be adiabatic, this takes the form

$$-\int_S \Psi_n \frac{\partial f_{nk}^{vis}}{\partial b_i} n_k dS = -\int_{S_w} \Psi_{m+1} \frac{\delta \tau_{km}}{\delta b_i} n_k dS + \int_{S_w} \Psi_{m+1} \frac{\partial \tau_{km}}{\partial x_l} \frac{\delta x_l}{\delta b_i} n_k dS + \int_{S_w} \Psi_5 \left(\frac{\partial v_m}{\partial x_l} \tau_{km} + \frac{\partial q_k}{\partial x_l}\right) \frac{\delta x_l}{\delta b_i} n_k dS + \int_{S_w} \Psi_5 q_k \frac{\delta (n_k dS)}{\delta b_i} \qquad (16)$$

Finally, the surface integral in eq. 14, can be expressed as

$$\int_{S_w} \kappa \frac{\partial \Psi_5}{\partial x_k} n_k \frac{\partial T}{\partial b_i} dS = \int_{S_w} \kappa \frac{\partial \Psi_5}{\partial x_k} n_k \frac{\delta T}{\delta b_i} dS - \int_{S_w} \kappa \frac{\partial \Psi_5}{\partial x_k} n_k \frac{\partial T}{\partial x_l} \frac{\delta x_l}{\delta b_i} dS \tag{17}$$

In the interest of space, we will refrain from presenting the lengthy final expression of eq. 5 after considering the previous development on a term–by–term basis. Next step is to make this expression independent from variations in the flow variables. At first, the field integrals in this expression which depend on variations $\delta U_m/\delta b_i$ are eliminated by satisfying the field adjoint equations

$$\frac{\partial \Psi_n}{\partial t} - A_{nmk} \frac{\partial \Psi_n}{\partial x_k} - K_k \frac{\partial V_k}{\partial U_m} = 0 \tag{18}$$

where V_k are the primitive flow variables and

$$K_n = \begin{bmatrix} -\frac{T}{\rho} \frac{\partial}{\partial x_k} \left(\kappa \frac{\partial \Psi_5}{\partial x_k} \right) \\ \frac{\partial \tau^{adj}_{(n-1)m}}{\partial x_m} - \tau_{(n-1)m} \frac{\partial \Psi_5}{\partial x_m} \\ \frac{T}{p} \frac{\partial}{\partial x_k} \left(\kappa \frac{\partial \Psi_5}{\partial x_k} \right) \end{bmatrix} \tag{19}$$

In inviscid flows, in order to eliminate the surface integrals of pressure variations over the solid walls in eqs. 4, 9, the equivalent to the no–penetration condition for the primal velocity, namely $\Psi_{m+1} n_m = -r_m n_m$ should be applied. The normal adjoint velocity becomes equal to zero only if $\vec{n}$ is normal to $\vec{r}$. For viscous flows, the equivalent to the no–slip condition should be imposed for the adjoint velocity along S_∞, namely, $\Psi_{m+1} = -r_m$, in order to eliminate the surface integrals consisting variations in stresses over the wall in eqs. 4, 16. These boundary conditions for the adjoint velocity over the wall eliminate also the surface integrals including variations in pressure over the wall in eqs. 4, 9. For the elimination the S_w integral with variations in temperature in eq. 17, a zero Neumann condition is applied for the adjoint energy.

Finally, the remaining surface integrals give the sensitivity derivatives,

$$\begin{aligned} \frac{\delta F_{aug}}{\delta b_i} &= \int_{S_w} p \frac{\delta}{\delta b_i} (n_k r_k dS) - \int_{S_w} \tau_{km} \frac{\delta}{\delta b_i} (n_m r_k dS) + \int_{S_w} \Psi_5 q_k \frac{\delta (n_k dS)}{\delta b_i} \\ &- \int_{S_w} \Psi_n \frac{\partial f^{inv}_{nk}}{\partial x_l} \frac{\delta x_l}{\delta b_i} n_k dS + \int_{S_w} \left(\Psi_{k+1} p - \Psi_n f^{inv}_{nk} \right) \frac{\delta (n_k dS)}{\delta b_i} \\ &+ \int_{S_w} \left[\left(-\tau^{adj}_{km} + \Psi_5 \tau_{km} \right) \frac{\partial v_m}{\partial x_l} + \Psi_5 \frac{\partial q_k}{\partial x_l} + \Psi_{m+1} \frac{\partial \tau_{km}}{\partial x_l} \right] \frac{\delta x_l}{\delta b_i} n_k dS \end{aligned} \tag{20}$$

4 DISCRETIZATION AND NUMERICAL SOLUTION

The primal and adjoint equations (eqs. 1 and 18) are discretized using the vertex–centered finite–volume method on unstructured/hybrid meshes. A finite volume formed around node P is presented in fig. 1, for a 2D mesh for the sake of simplicity. In the primal solver, inviscid numerical fluxes crossing the interface of adjacent finite volumes are computed using the Roe's [11] approximate Riemann solver, with second–order accuracy, as

$$\Phi^{PQ}_n = \frac{1}{2} \left(\underline{A}^P_{nm} U^P_m + \underline{A}^Q_{nm} U^Q_m \right) - \frac{1}{2} \left| \tilde{\underline{A}}^{PQ}_{nm} \right| \left(U^R_m - U^L_m \right) \tag{21}$$

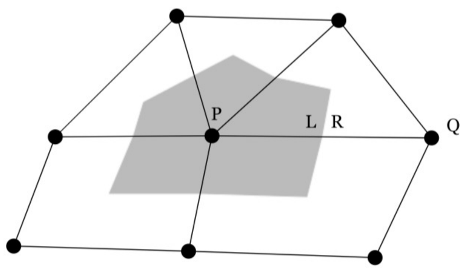

Figure 1: The finite volume formed around node P. At the interface between two mesh nodes the left L and right R flow variables are extrapolated using nodal (at P and Q) values and spatial gradients.

where, Φ^{PQ} is used for the computation of the residual at node P and $\Phi^{QP} = -\Phi^{PQ}$ is used for the residual at Q. In eq. 21, $\underline{A}_{nm} = A_{nmk}n_k$ where n_k is the normal to the interface of the finite volumes formed around nodes P, Q and points towards Q. $\underline{\tilde{A}}^{PQ}$ is computed based on the Roe-averaged flow variables, U^P, U^Q are the flow variables stored at P, Q respectively and U^R, U^L are the flow variables on the right and left boundaries of the finite volume, obtained by extrapolating U^Q, U^P respectively.

The adjoint inviscid numerical fluxes are computed as

$$\Phi_m^{adj,PQ} = -\frac{1}{2}\underline{A}_{nm}^P\left(\Psi_n^P + \Psi_n^Q\right) - \frac{1}{2}\left|\underline{\tilde{A}}_{nm}^{PQ}\right|\left(\Psi_n^R - \Psi_n^L\right) \tag{22}$$

$$\Phi_m^{adj,QP} = \frac{1}{2}\underline{A}_{nm}^Q\left(\Psi_n^P + \Psi_n^Q\right) + \frac{1}{2}\left|\underline{\tilde{A}}_{nm}^{PQ}\right|\left(\Psi_n^R - \Psi_n^L\right) \tag{23}$$

Both the primal and adjoint equations are solved iteratively for the corrections of the primal and adjoint variables (ΔU and $\Delta\Psi$) respectively, as

$$\begin{aligned} \frac{\partial R_n}{\partial U_m}\Delta U_m &= -R_n \qquad U_m^{n+1} = U_m^n + \Delta U_m \\ \frac{\partial \mathcal{R}_m}{\partial \Psi_n}\Delta \Psi_n &= -\mathcal{R}_m \qquad \Psi_n^{n+1} = \Psi_n^n + \Delta \Psi_n \end{aligned} \tag{24}$$

Eqs. 24 are solved using the point-implicit Jacobi method which can be efficiently parallelized.

5 IMPLEMENTATION ON GPUS

An in–house GPU–enabled CFD solver was used for the solution of the primal and adjoint equations. The solver may run on many GPUs using either the shared on–board memory for data transactions among GPUs of the same computational node or the MPI protocol for the communication of the GPUs on different computational nodes.

In the cases presented in this paper, the speed–up achieved using the GPU–enabled solver on a single NVIDIA Tesla M2050 instead of the corresponding CPU–enabled solver variant running on a single core of an Intel Xeon CPU E5620 at 2.40GHz, exceeds $50\times$ and $60\times$ for the solution of the primal and the adjoint equations respectively. The next paragraphs present some key programming features of the GPU solver.

5.1 GPU memory handling

The parallel efficiency of any GPU–enabled software is highly related to GPU memory handling issues. Programmable GPUs implement several memory types, such as the global (or device), constant, texture, shared and local memories. Among them, the global memory is the one with the largest capacity. Memories of GPUs based on the Fermi architecture (such as the Tesla M2050 model used), are cached, in contrast to previous architectures, where only the constant and texture memories are cached.

In order to maximize the parallel efficiency of the GPU solver, the access to the aforementioned memory spaces adheres to different patterns. For instance, even if GPUs based on the Fermi architecture are used, large data accesses to the global GPU memory are coalesced to 128 Byte memory segments, so as to minimize the global memory bandwidth. In order to better understand how important the latter is, it worths noting that the time needed for the execution of almost 27 arithmetic operations is nearly the same with a single cache miss global memory transaction. In addition, frequently and randomly accessed data, such as the gradients of the primal or adjoint variables, are fetched to textures. Access to the texture memory is performed via the texture cache memory. Renumbering of mesh nodes is employed to minimize cache miss memory transactions. Moreover, threads of the same block interchange data through the low latency shared memory. The fast constant memory is used for the storage of gas constants, etc.

5.2 Mixed Precision Arithmetics

The GPU solver employs Mixed Precision Arithmetics (MPA), [7], for the minimization of the global memory transactions without harming the accuracy of the results. According to MPA, the right-hand-side (r.h.s.) of eq. 24 is calculated and stored using DPA, while the memory consuming left-hand-side (l.h.s.) coefficients are calculated using DPA but stored using SPA. MPA minimizes the data capacity transferred between GPU threads and the global memory, resulting to great speed-ups.

The truncation error occurred in storing the l.h.s. coefficients does not harm the accuracy of the solver since both the primal and adjoint equations are solved in “delta form’. Practically, the use of MPA significantly reduces (over 30%) the required GPU memory.

5.3 Computation of fluxes

For the computation of the primal/adjoint residuals and l.h.s. matrix coefficients, the accumulation of the numerical primal/adjoint fluxes crossing the finite volume interfaces and their Jacobians are needed. For the CPU code, this can be done by looping over the mesh edges, computing the associated numerical fluxes and their Jacobians and contributing to the residuals and l.h.s. coefficients of the edge nodes.

However, this procedure is not acceptable in a multi-threaded code (like a GPU code) as it leads to memory conflicts since independent threads may write to the same memory space. Thus, a different approach must be followed.

The simplest, though inefficient, approach is to use atomic operations which block threads attempting to write simultaneously to the same memory space. Though this approach exhibits minimal complexity, is rather inefficient.

An efficient approach is to associate each GPU thread with a mesh node (One–Kernel scheme). Then, each thread loops over the edges emanating from the same node, computes and accumulates the numerical fluxes and the corresponding Jacobians. Using this approach, memory conflicts are avoided and no thread synchronization is required. However, fluxes and Jacobians are computed twice.

An alternative approach is to to use a Two–Kernel scheme, [2]. The first kernel associates each GPU thread with a mesh edge and computes the numerical fluxes and Jacobians, which are temporarily stored in the global memory. Then, a second kernel is launched, where each GPU thread is associated with a single mesh node. Threads loop over the edges emanating from the associated node and accumulate the already computed numerical fluxes and Jacobians. By doing so, memory conflicts are avoided and fluxes are computed once. On the other hand, especially in the accumulation of the Jacobians in order to form the l.h.s. coefficients, non–coalesced large global memory accesses are required. Moreover, extra memory space is required for storing the edge-based information computed by the first kernel. This makes the use of the Two–Kernel scheme rather prohibitive for large scale applications due to the limited GPU memory capacity.

Comparing the aforementioned schemes, the One-Kernel represents the best compromise between speed-up and memory consumption and is, thus, used in the primal solver. The l.h.s. of the discretized adjoint equations involves only the primal variables, which remain constant during the adjoint solution, and thus the l.h.s. coefficients need to be computed only once before the iterative solution of the adjoint equations. For this reason, the accumulation of non–coalesced large data corresponding to the Jacobians is avoided during the iterative solution of the adjoint equations. Thus, the One–Kernel scheme is used for the computation of the adjoint l.h.s. matrix coefficients and the Two–Kernel scheme for the less memory consuming residuals at each pseudo-time iteration. It should be noted that the adjoint residuals computed by the first kernel of the Two–Kernel scheme, are stored at the same memory location with the extrapolated to the finite–volume interfaces adjoint variables. These extrapolated adjoint quantities are required for the computation of the adjoint fluxes with second order accuracy, but there is no need to keep them stored afterwards. The aforementioned memory positions are fetched to textures in order to reduce the memory transaction time while accumulating the adjoint fluxes in the second kernel of the Two–Kernel scheme.

6 APPLICATIONS

The continuous adjoint method on GPUs described in the previous sections was applied to the shape optimization of an isolated airfoil and used to compute the surface derivatives of a transonic wing. Recall that, the main goal of this paper is to present “optimal” techniques for making the adjoint solver run on GPUs rather than demonstrating the accuracy of the adjoint solver, which was the subject of previous publications, [10].

6.1 Shape optimization of a 2D airfoil

The first application is concerned with the shape optimization of an isolated airfoil for maximum lift and minimum drag. The two performance metrics (C_L and C_D) are combined using weighting factors into a single objective function F,

$$F = -W_L C_L \; + \; W_D C_D \tag{25}$$

where $W_L\!=\!0.1$, $W_D\!=\!1.0$. The initial geometry is a NACA0012 airfoil parameterized using Bézier–Bernstein polynomials with 8 control points per airfoil side. Five (out of the 8) control points per airfoil side were allowed to vary along the normal to the chord direction, summing up to 10 design variables in total. Leading and trailing edge control points and those next to the leading edge control point are kept fixed. Fig. 2 presents the initial shape of the airfoil surrounded by the initial positions of the Bézier control points. The filled circles and squares correspond to control points which are allowed to move normal to the airfoil chord.

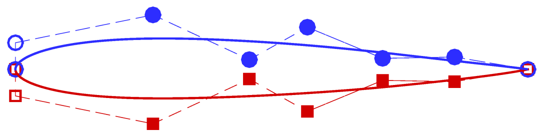

Figure 2: Shape optimization of an isolated airfoil. Initial geometry parameterized using Bézier–Bernstein polynomials.

After the completion of each optimization cycle, the airfoil mesh nodes are updated based on the computed sensitivity derivatives and the rest of the mesh nodes are deformed accordingly. Each mesh node is associated with the deformation of the closest airfoil node scaled by a factor depending on the distance of the mesh node from its closest wall node. The flow conditions are: $M_\infty = 0.3$, $\alpha_\infty = 2^o$ and $Re_c = 1000$. The starting hybrid 2D computational mesh consists of 56270 nodes, 101859 triangular and 5090 quadrilateral elements. Since a solver for 3D flows is used this mesh is extruded to the third direction by a single cell width.

Comparison of the computed sensitivity derivatives between the adjoint method and finite differences for the initial geometry is shown in fig. 3. The first 5 sensitivities correspond to the design variables controlling the pressure side. The last 5 sensitivities correspond to the suction side. The computed sensitivity derivatives are very close to those computed using finite differences.

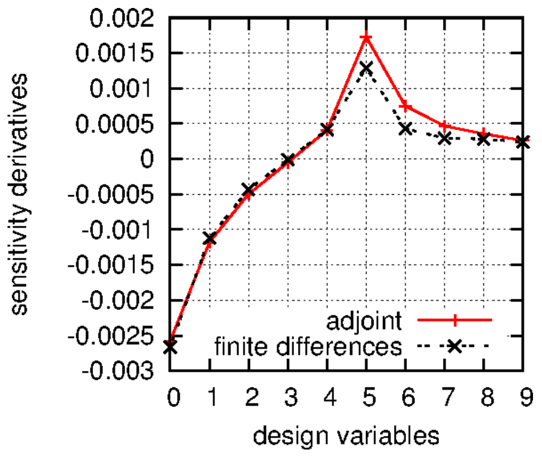

Figure 3: Shape optimization of an isolated airfoil. Comparison between sensitivity derivatives computed using the GPU–enabled adjoint solver and finite differences for the initial geometry.

C_L increased about two times, from 0.120 in the initial geometry to 0.264 in the optimized one. Moreover, the drag coefficient C_D decreased from 0.118 to 0.102. Fig. 4 presents the Mach number and the adjoint velocity fields computed around the initial and optimized airfoils. The optimized airfoil is cambered to increase C_L, being much thinner than the initial one to decrease C_D.

In the presented optimization, a single NVIDIA Tesla M2050 was used. This GPU is based on the Fermi architecture and has 3 GB RAM with maximum bandwidth 148 GB/sec. The peak

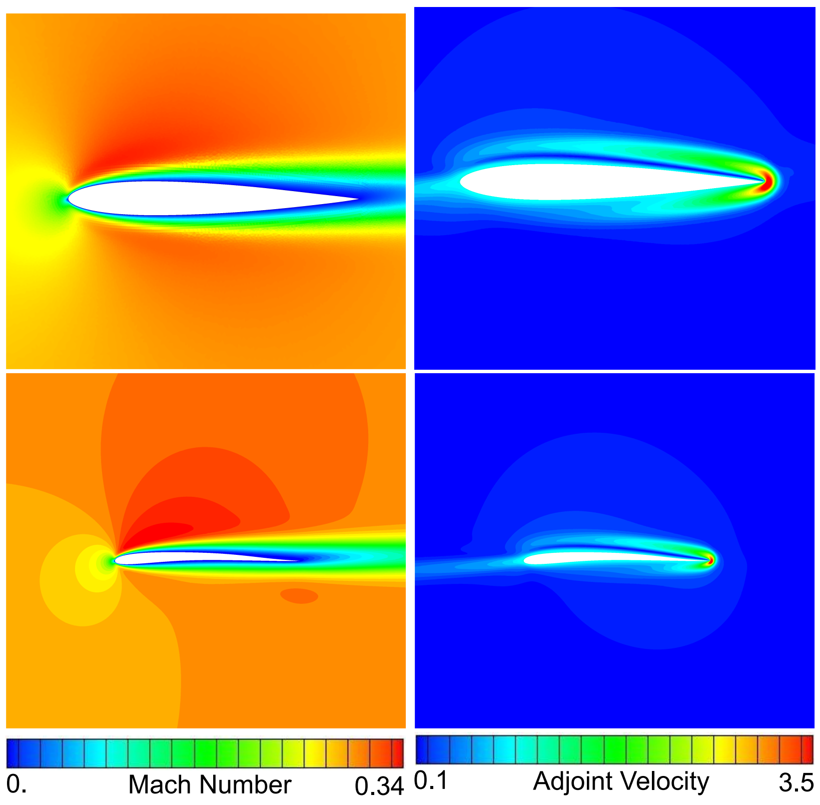

Figure 4: Shape optimization of an isolated airfoil. Computed Mach number (left) and adjoint velocity (right) field for the initial (top) and optimized (bottom) configurations.

double precision floating point performance is 515 GFLOPS. In each optimization cycle, the solution of the primal and adjoint equations on a Tesla M2050 took merely around 40 secs. So, overall (i.e. for 30 optimization cycles) the time needed for the solution of the primal and adjoint equations were $30 \times 40 = 1200$ secs or 20 minutes. In the aforementioned time measurements, the time needed for the deformation of the mesh and the re–computation/update of the data which are related with the mesh geometry, after the end of each optimization cycle, is not taken into account. This part of the code is the only one which runs on the CPU. So, overall the optimization wall–clock time was around 25 minutes. The use of a single GPU instead of a single core of the dual–quad–core Intel Xeon CPU (with 2.40 GHz and 12288 KB of cache memory) for the solution of the primal and adjoint equations speed–ups the optimization about 45 times. This means that the same optimization with the primal/adjoint equations solver running on a single CPU, it would take more than 15 hours.

6.2 Sensitivity analysis of a Transonic Wing

The second case deals with the sensitivity analysis of the ONERA M6 wing for drag minimization. The flow is considered to be inviscid, with $M_\infty = 0.84$ and $\alpha_\infty = 4^o$. Fig. 5 presents iso–lines of the Mach number and the adjoint velocity, while fig. 6 shows the computed sensitivity map over the wing for drag minimization. Areas with red color should be moved inwards whereas blue areas should be moved outwards in order to reduce drag. This computation was carried out on NVIDIA Tesla M2050 which speeds–up the solution of the primal equations of about 47 times and the adjoint ones about 52 times compared to a single CPU core (same as in the previous case).

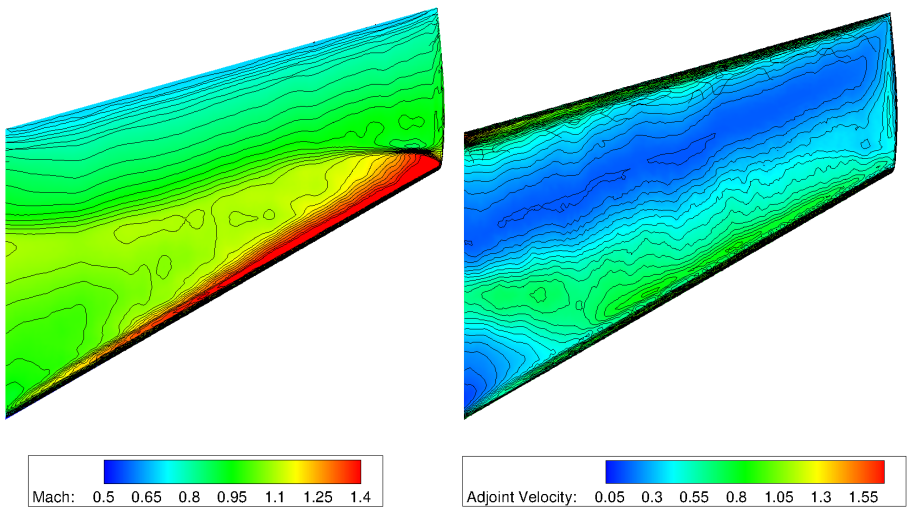

Figure 5: Transonic wing drag reduction. Primal and adjoint fields.

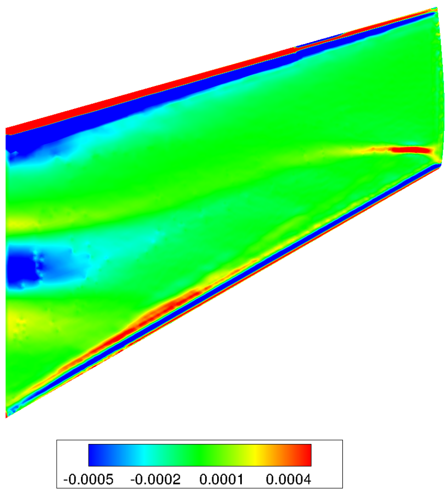

Figure 6: Transonic wing drag reduction. Sensitivity maps plotted over the surface of the wing.

7 CONCLUSIONS

This paper presented the continuous adjoint method for compressible flows on GPUs. This paper is dealing with 3D compressible flows and the adjoint method was used not only for design–optimization assisted by a gradient–based method but, also, for the computation of sensitivity maps. Emphasis was laid on the speed–up of the GPU–enabled software for both the primal and the adjoint equations solvers and, thus, the reduction of the optimization turnaround time. In particular, the GPU solvers presented are about 50 times faster than the corresponding ones running on a single CPU core. Applications in the computation of sensitivity derivatives on a transonic wing and the shape optimization of an airfoil. For the latter, thanks to the use of GPUs the optimization is completed within a few minutes instead of hours on a single CPU core.

ACKNOWLEDGMENTS

The third author acknowledges the support from the People Programme (ITN Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 317006 (AboutFLOW project).

REFERENCES

[1] V.G. Asouti, E.A. Kontoleontos, X.S. Trompoukis, and K.C. Giannakoglou. Shape optimization using the one-shot adjoint technique on graphics processing units. In *7th GRACM International Congress on Computational Mechanics*, Athens, June 30-July 2 2011.

[2] V.G. Asouti, X.S. Trompoukis, I.C. Kampolis, and K.C. Giannakoglou. Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units. *International Journal for Numerical Methods in Fluids*, 67(2):232–246, 2011.

[3] T. Brandvik and G. Pullan. Acceleration of a 3D Euler solver using commodity graphics hardware. AIAA Paper 2008–607, 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada,, January 2008.

[4] A. Corrigan, F. Camelli, R. Löhner, and J. Wallin. Running unstructured grid based CFD solvers on modern graphics hardware. AIAA Paper 2009–4001, 19th AIAA Computational Fluid Dynamics, San Antonio, Texas,, June 2009.

[5] E. Elsen, P. LeGresley, and E. Darve. Large calculation of the flow over a hypersonic vehicle using a GPU. *Journal of Computational Physics*, 227(24):10148–10161, 2008.

[6] T.R. Hagen, K.A. Lie, and J.R. Natvig. Solving the Euler equations on graphics processing units. *Computational Science - ICCS*, 3994:220–227, 2006.

[7] I.C. Kampolis, X.S. Trompoukis, V.G Asouti, and K.C. Giannakoglou. CFD-based analysis and two-level aerodynamic optimization on graphics processing units. *Computer Methods in Applied Mechanics and Engineering*, 199(9-12):712–722, 2010.

[8] M. Lefebvre, P. Guillen, J.-M. Le Gouez, and C. Basdevant. Optimizing 2d and 3d structured euler CFD solvers on graphical processing units. *Computers & Fluids*, 70:136 – 147, 2012.

[9] G. Oyarzun, R. Borrell, A. Gorobets, O. Lehmkuhl, and A. Oliva. Direct numerical simulation of incompressible flows on unstructured meshes using hybrid cpu/gpu supercomputers. *Procedia Engineering*, 61:87 – 93, 2013.

[10] D.I. Papadimitriou and K.C. Giannakoglou. A continuous adjoint method with objective function derivatives based on boundary integrals for inviscid and viscous flows. *Computers & Fluids*, 36(2):325–341, 2007.

[11] P. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.

[12] X.S. Trompoukis, V.G. Asouti, I.C. Kampolis, and K.C. Giannakoglou. *CUDA implementation of Vertex-Centered, Finite Volume CFD methods on Unstructured Grids with Flow Control Applications*, chapter 17. Morgan Kaufmann, 2011.