8th GRACM International Congress on Computational Mechanics,
Volos, 12 July - 15 July 2015

SHAPE OPTIMIZATION USING THE CONTINUOUS ADJOINT METHOD AND VOLUMETRIC NURBS ON GPUS

Konstantinos T. Tsiakas[1], **Xenofon S. Trompoukis**[2], **Varvara G. Asouti**[3], **Mehdi S. Ghavami Nejad**[4] **and Kyriakos C. Giannakoglou**[5]

[1,2,3,4,5]Parallel CFD & Optimization Unit, School of Mechanical Engineering
National Technical University of Athens
Athens, Greece
e-mail: {tsiakost,xeftro}@gmail.com, {vasouti,mehdi}@mail.ntua.gr, kgianna@central.ntua.gr;
web page: http://velos0.ltt.mech.ntua.gr/research

Keywords: Continuous Adjoint, Computational Fluid Dynamics, NURBS, GPU cluster, Shape Optimization.

Abstract: *This paper is concerned with the development of the continuous adjoint method for aerodynamic shape optimization problems on Graphics Processing Units (GPUs). The primal and adjoint solvers are implemented on parallel GPUs of many computational nodes using CUDA and the MPI protocol. In turbulent flow problems, the differentiation of the turbulence model is included in the adjoint formulation and this improves the accuracy of the computed objective function gradient. The latter are computed with respect to the coordinates of the control points of volumetric NURBS. The same volumetric NURBS are also used to deform the computational mesh during the optimization. Implementation details of the primal and adjoint solvers as well as the parameterization software on GPUs are discussed. The aforementioned tools are used for the shape optimization of a U–bend duct and a linear compressor cascade, aiming at minimum total pressure losses.*

1 INTRODUCTION

The use of gradient–based techniques assisted by the adjoint methods is the most efficcient way to solve large–scale aerodynamic shape optimization problems governed by the flow equations. Compared to methods such as finite differences, the adjoint method computes the gradient of the objective function at cost almost equal to that of the solution of the Navier–Stokes (NS) equations, irrespective of the number of design variables.

In this paper, to further reduce the optimization turnaround time, the solution of both the primal (NS) and adjoint equations is carried out on a many–GPU cluster. In particular, this paper deals with the continuous adjoint method for incompressible flows running on GPUs. For turbulent flows, the differentiation of the Spalart–Allmaras (SA) turbulence model is also considered as presented for the first time in [4], and is herein ported to GPUs.

During the last decade, GPUs became more and more programmable which allowed scientists to use them for their applications. The latter include GPU implementations of CFD codes for both structured [9, 10, 12] and unstructured [11, 5, 13] meshes. Given that memory access/handling plays a significant role in the parallel efficiency of any GPU implementation, the mesh type affects a lot the performance of the GPU–enabled algorithm. GPU–enabled CFD codes for structured meshes may profit from the organized topology and memory access and exhibit high speed–ups. Working with unstructured meshes, the lack of structure in mesh connectivity affects the parallel speed–up, which is also sensitive to the discretization schemes. In general, the cell–centered finite volume scheme can be considered more advantageous in terms of memory access compared to the vertex–centered one. This is because the number of neighbors to each cell barycenter is constant and known a priori. In contrast, in a vertex–centered scheme where the number of the adjacent nodes to any node may vary a lot, a customized memory handling is necessary.

The authors group has addressed various ways to achieve high speed-ups of GPU–enabled codes for unstructured meshes with the vertex–centered finite volume scheme. These include the optimal use of GPU memory, the use of a Mixed Precision Arithmetics (MPA) scheme and different schemes for the computation of numerical fluxes [5, 6].

In this paper, the practicalities of running the primal and adjoint solvers on GPUs are discussed. Applications to the shape optimization of a U–bend duct and a linear compressor cascade aiming at the minimization of total pressure losses are presented.

Although the adjoint method provides all the necessary means for computing sensitivities of the objective function w.r.t. the aerodynamic shape, in order to perform a complete optimization a shape parameterization and a mesh deformation method are needed. In this paper, volumetric NURBS, [16] are used to fulfill these purposes. Unlike direct shape parameterization techniques, volumetric NURBS parameterize a part of the volume domain in which the shape to be optimized is embedded. Concerning the shape of the surface, this approach is equivalent to a free form deformation (FFD) method. However, the use of volumetric NURBS is extended to also deform the computational mesh. This allows for fast and robust mesh deformation within the optimization loop, without the need of using other specific mesh deformation tools (solving PDEs, spring analogy solvers etc). In order to minimize the wall clock time needed for the objective function gradient computation and the mesh deformation, volumetric NURBS are implemented on GPUs.

2 OPTIMIZATION ALGORITHM

The gradient–based optimization algorithm consists of the following steps within each cycle:

Step 1: Solve the primal (flow) equations.
Step 2: Solve the adjoint equations.
Step 3: Compute the objective function gradients.
Step 4: Update the design variables (NURBS control points) by descending using the gradient (steepest descent).
Step 5: Update the computational mesh in accordance/comformity with the already modified NURBS control grid shape.
Step 6: Compute the objective function values. If convergence criteria are not met return to step 1.

The main steps of the optimization algorithm are discussed in the following sections.

2.1 Flow Model

The flow model comprises the Navier–Stokes equations for incompressible fluid flows. The pseudo-compressibility method introduced by Chorin, [1], is used for satisfying the divergence–free velocity constraint. The mean flow equations read

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

where $U_n = [p \; \upsilon_1 \; \upsilon_2 \; \upsilon_3]^T$ is the vector of the unknowns, with $\upsilon_i, i = 1, 2, 3$ being the velocity components and p being the kinematic pressure. The inviscid and viscous fluxes as well as the stresses are given as

$$f_{nk}^{inv} = \begin{bmatrix} \beta \upsilon_k \\ \upsilon_k \upsilon_1 + p\delta_{1n} \\ \upsilon_k \upsilon_2 + p\delta_{2n} \\ \upsilon_k \upsilon_3 + p\delta_{3n} \end{bmatrix}, \quad f_{nk}^{vis} = \begin{bmatrix} 0 \\ \tau_{1k} \\ \tau_{2k} \\ \tau_{3k} \end{bmatrix}, \quad \tau_{mk} = (\nu + \nu_t) \left(\frac{\partial \upsilon_m}{\partial x_k} + \frac{\partial \upsilon_k}{\partial x_m} \right) \tag{2}$$

where ν and ν_t stand for the kinematic and turbulent viscosity and β is the pseudo-compressibility parameter. The Jacobian matrix A_{nmk} is

$$A_{nmk} = \begin{bmatrix} 0 & \beta\delta_{1k} & \beta\delta_{2k} & \beta\delta_{3k} \\ \delta_{1k} & \upsilon_1 + \upsilon_1\delta_{1k} & \upsilon_1\delta_{2k} & \upsilon_1\delta_{3k} \\ \delta_{2k} & \upsilon_2\delta_{1k} & \upsilon_2 + \upsilon_2\delta_{2k} & \upsilon_2\delta_{3k} \\ \delta_{3k} & \upsilon_3\delta_{1k} & \upsilon_3\delta_{2k} & \upsilon_3 + \upsilon_3\delta_{3k} \end{bmatrix} \tag{3}$$

For turbulent flows, the low–Reynolds number Spalart–Almaras turbulence model [2] is used . The model equation is

$$R_{\tilde{\nu}} = \underbrace{\frac{\partial}{\partial x_k}(\upsilon_k \tilde{\nu})}_{conv(\tilde{\nu})} \underbrace{- \frac{1 + c_{b2}}{\sigma} \frac{\partial}{\partial x_k} \left[(\nu + \nu_t) \frac{\partial \tilde{\nu}}{\partial x_k} \right] + \frac{c_{b2}}{\sigma} (\nu + \tilde{\nu}) \frac{\partial}{\partial x_k} \left(\frac{\partial \tilde{\nu}}{\partial x_k} \right)}_{diff(\tilde{\nu})} \underbrace{- c_{b1} \tilde{S} \tilde{\nu} + c_{w1} f_w \left(\frac{\tilde{\nu}}{\Delta} \right)^2}_{source(\tilde{\nu})} \tag{4}$$

where, $\tilde{\nu}$ is the model unknown linked with ν_t by $\nu_t = \tilde{\nu} f_{v1}$. $\tilde{S}$ is defined as $\tilde{S} = f_{v3}S + \frac{\tilde{\nu}}{k^2\Delta^2}fv2$, S being the vorticity magnitude $S = \left|\epsilon_{klm}\frac{\partial v_m}{\partial x_l}\right|$ and Δ the distance of each node from the nearest wall boundary. The definition of any auxiliary function or quantity can be found in [2].

Along the solid walls, zero velocity components as well as a zero value for $\tilde{\nu}$ are specified; in addition a Neumann boundary condition is used for p. Along the inlet (S_I), fixed values are used for both the velocity components and $\tilde{\nu}$ and a Neumann condition for p; along the outlet (S_O), the p value is defined together with Neumann conditions for the rest of the flow quantities.

2.2 Continuous Adjoint Formulation

The objective function F to be minimized is the volume–averaged total pressure losses between S_I (inlet) and S_O (outlet) boundaries of the computational domain, expressed as

$$F = -\int_{S_I}\left(p + \frac{1}{2}v_m^2\right)v_k n_k dS - \int_{S_O}\left(p + \frac{1}{2}v_m^2\right)v_k n_k dS \tag{5}$$

where n_k are the components of the unit vector normal to the inlet/outlet boundaries and pointing outside of the computational domain. By introducing the adjoint variable field Ψ_n(i.e. the adjoint to the mean-flow variables), $\tilde{\nu}^a$ (i.e. the adjoint to the turbulence model variable $\tilde{\nu}$), the augmented objective functions is defined as

$$F_{aug.} = F + \int_\Omega \Psi_n R_{U_n} d\Omega + \int_\Omega \tilde{\nu}^a R_{\tilde{\nu}} d\Omega \tag{6}$$

Upon convergence of the primal equations $F_{aug} = F$, so computing the sensitivity derivatives ($\frac{\delta F}{\delta b_i}$) of F w.r.t. the design variables b_i is equivalent to computing $\frac{\delta F_{aug.}}{\delta b_i}$. By differentiating eq. 6, we get

$$\frac{\delta F_{aug}}{\delta b_i} = \frac{\delta F}{\delta b_i} + \int_\Omega \frac{\delta}{\delta b_i}(\Psi_n R_{U_n})d\Omega + \int_\Omega \Psi_n R_{U_n}\frac{\delta}{\delta b_i}(d\Omega) + \int_\Omega \frac{\delta}{\delta b_i}(\tilde{\nu}^a R_{\tilde{\nu}})d\Omega + \int_\Omega \tilde{\nu}^a R_{\tilde{\nu}}\frac{\delta}{\delta b_i}(d\Omega) \tag{7}$$

For any geometric or flow quantity Φ defined over Ω it can be proved, [3], that

$$\frac{\delta \Phi}{\delta b_i} = \frac{\partial \Phi}{\partial b_i} + \frac{\partial \Phi}{\partial x_l}\frac{\delta x_l}{\delta b_i}\,, \qquad \frac{\delta}{\delta b_i}\left(\frac{\partial \Phi}{\partial x_k}\right) = \frac{\partial}{\partial x_k}\left(\frac{\delta \Phi}{\delta b_i}\right) - \frac{\partial \Phi}{\partial x_l}\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right) \tag{8}$$

Applying the divergence theorem and using eqs. 8, eq. 7 can be expressed in terms of field and boundary integrals containing variations in the flow quantities ($\frac{\delta U_n}{\delta b_i}$, $\frac{\delta \tilde{\nu}}{\delta b_i}$) and variations in geometric quantities ($\frac{\delta x_l}{\delta b_i}$, $\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right)$ and $\frac{\delta}{\delta b_i}(n_k dS)$). Then, the volume integrals containing $\frac{\delta U_n}{\delta b_i}$ and $\frac{\delta \tilde{\nu}}{\delta b_i}$ are eliminated by satisfying the adjoint mean flow PDEs 9 and the adjoint turbulence model PDE 10, namely

$$R_{\Psi_n} = \underbrace{-A_{mnk}\frac{\partial \Psi_m}{\partial x_k}}_{conv(\Psi_n)} \underbrace{- \frac{\partial \phi_{nk}^{vis}}{\partial x_k}}_{diff(\Psi_n)} + \underbrace{T_n^a}_{source1(\tilde{\nu}^a)} = 0 \tag{9}$$

$$\begin{aligned} R_{\tilde{\nu}^a} = & \underbrace{-\, v_k \frac{\partial \tilde{\nu}^a}{\partial x_k}}_{conv(\tilde{\nu}^a)} \underbrace{- \frac{1+c_{b2}}{\sigma}\frac{\partial}{\partial x_k}\left[(\nu+\tilde{\nu})\frac{\partial \tilde{\nu}^a}{\partial x_k}\right] + \frac{c_{b2}}{\sigma}\frac{\partial^2}{\partial x_k^2}[\tilde{\nu}^a(\nu+\tilde{\nu})]}_{diff(\tilde{\nu}^a)} \\ & \underbrace{+\frac{1+c_{b2}}{\sigma}\frac{\partial \tilde{\nu}^a}{\partial x_k}\frac{\partial \tilde{\nu}}{\partial x_k}}_{grad(\tilde{\nu}^a)} \underbrace{+ \frac{\partial \Psi_{m+1}}{\partial x_k}\left(\frac{\partial v_k}{\partial x_m} + \frac{\partial v_m}{\partial x_k}\right)\left(\frac{\partial \nu_t}{\partial \tilde{\nu}} + \frac{\partial \nu_t}{\partial f_{v1}}\frac{\partial f_{v1}}{\partial \chi}\frac{\partial \chi}{\partial \tilde{\nu}}\right)}_{source1(\tilde{\nu}^a)} \\ & \underbrace{-c_{b1}\tilde{\nu}^a\tilde{\nu}\frac{\partial S}{\partial \tilde{\nu}} - c_{b1}\tilde{\nu}^a\tilde{S} + c_{w1}\tilde{\nu}^a\left(\frac{\partial \tilde{\nu}}{\Delta}\right)^2\frac{\partial f_w}{\partial \tilde{\nu}} + 2c_{w1}\tilde{\nu}^a\frac{f_w}{\Delta}}_{source2(\tilde{\nu}^a)} \end{aligned} \tag{10}$$

where

$$\phi_{nk}^{vis} = [0 \quad \tau_{1k}^a \quad \tau_{2k}^a \quad \tau_{3k}^a]^T \qquad \tau_{mk}^a = (\nu + \nu_t)\left(\frac{\partial \Psi_{m+1}}{\partial x_k} + \frac{\partial \Psi_{k+1}}{\partial x_m}\right) \tag{11}$$

and $source1(\tilde{\nu}^a)$ are terms arising from the differentiation of the $conv(\tilde{\nu})$ and the vorticity terms of equation 4. Boundary integrals containing $\frac{\delta U_n}{\delta b_i}$ and $\frac{\delta \tilde{\nu}}{\delta b_i}$ are also eliminated by satisfying apropriate adjoint boundary conditions. The remaining field and boundary integrals contain variations in geometric quantities and, thus, are used to compute the sensitivity derivatives of the objective function. The final expression of the sensitivity derivatives is

$$\begin{aligned}\frac{\delta F}{\delta b_i} =& -\int_\Omega \left(\beta \Psi_1 \frac{\partial \upsilon_k}{\partial x_l} + \Psi_{m+1}\frac{\partial}{\partial x_l}(\upsilon_k \upsilon_m) + \Psi_{k+1}\frac{\partial p}{\partial x_l}\right)\frac{\partial}{\partial x_l}\left(\frac{\delta x_l}{\delta b_i}\right) d\Omega \\ &+ \int_\Omega \left[\Psi_{m+1}\frac{\partial \tau_{km}}{\partial x_l} - \tau^a_{km}\frac{\partial \upsilon_m}{\partial x_l}\right]\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right) d\Omega - \int_\Omega \left(\tilde{\nu}^a \tilde{\nu}\frac{\partial \upsilon_k}{\partial x_l} - \tilde{\nu}^a \upsilon_k \frac{\partial \tilde{\nu}}{\partial x_l}\right)\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right) d\Omega \\ &+ \int_\Omega \left(\frac{1+2c_{b2}}{\sigma}\right)\left(\tilde{\nu}^a \frac{\partial \tilde{\nu}}{\partial x_l}\frac{\partial \tilde{\nu}}{\partial x_k}\right)\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right) d\Omega + \int_\Omega \frac{\tilde{\nu}^a}{\sigma}(\nu+\tilde{\nu})\frac{\partial}{\partial x_l}\left(\frac{\partial \tilde{\nu}}{\partial x_k}\right)\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right) d\Omega \\ &- \int_\Omega \frac{\nu+\tilde{\nu}}{\sigma}\frac{\partial \tilde{\nu}^a}{\partial x_k}\frac{\partial \tilde{\nu}}{\partial x_l}\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right) d\Omega + \int_\Omega c_{w1}\tilde{\nu}^a \left[\left(\frac{\tilde{\nu}}{\Delta}\right)^2 \frac{\partial f_w}{\partial \Delta} - 2 f_w \frac{\tilde{\nu}}{\Delta^2}\right]\frac{\delta \Delta}{\delta b_i} d\Omega \\ &+ \int_\Omega \tilde{\nu}^a \left[\tilde{\nu}\frac{\partial \tilde{S}}{\partial S}\frac{\partial S}{\partial\left(\frac{\partial \upsilon_m}{\partial x_k}\right)}\frac{\partial \upsilon_m}{\partial x_l} - \left(\frac{\tilde{\nu}}{\Delta}\right)^2 \frac{\partial f_w}{\partial\left(\frac{\partial \upsilon_m}{\partial x_k}\right)}\frac{\partial \upsilon_m}{\partial x_l}\right]\frac{\partial}{\partial x_k}\left(\frac{\partial x_l}{\partial b_i}\right) d\Omega \end{aligned} \quad (12)$$

In [15], a way to treat terms containing $\frac{\delta \Delta}{\delta b_i}$ has been proposed by some of the authors, by formulating and solving the adjoint to the eikonal equation for distance computations.

2.3 Discretization And Numerical Solution

The flow and adjoint PDEs are discretized using the vertex–centered finite–volume method on unstructured/hybrid meshes. A finite volume formed around node P is presented in figure 1, for a 2D mesh for the sake of simplicity. In the primal solver, the inviscid numerical fluxes crossing the interface of adjacent finite volumes are computed

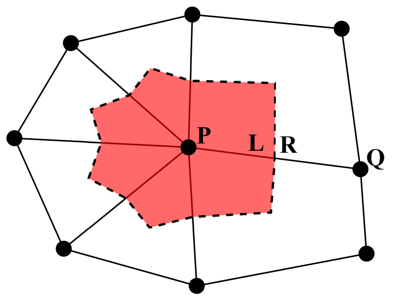


Figure 1: 2D Finite volume formed around node P. At the interface between two mesh nodes, the left L and right R flow variables are extrapolated using nodal (at P and Q) values and spatial gradients.

using the Roe's [8] approximate Riemann solver, with second–order accuracy, as

$$\Phi^{PQ}_n = \frac{1}{2}\left(f^{inv,P}_{nk} + f^{inv,Q}_{nk}\right) n_k - \frac{1}{2}\left|\overline{A}^{PQ}_{nmk} n_k\right| \left(U^R_m - U^L_m\right) \quad (13)$$

where Φ^{PQ} contributes to the residual at node P and $\Phi^{QP} = -\Phi^{PQ}$ for the residual at Q. In eq. 13, n_k is the normal to the interface of the finite volumes formed around nodes P, Q pointing towards Q. $\overline{A}^{PQ}$ is computed using 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 extrapolated at the right and left boundaries of the inteface.

The adjoint inviscid numerical fluxes are computed using non–conservative schemes, namely

$$\begin{aligned}\Phi^{a,PQ}_n &= -\frac{1}{2}A^P_{mnk}\left(\Psi^P_n + \Psi^Q_n\right) n_k - \frac{1}{2}\left|\overline{A}^{PQ}_{mnk} n_k\right| \left(\Psi^R_n - \Psi^L_n\right) \\ \Phi^{a,QP}_n &= \frac{1}{2}A^Q_{mnk}\left(\Psi^P_n + \Psi^Q_n\right) n_k + \frac{1}{2}\left|\overline{A}^{PQ}_{mnk} n_k\right| \left(\Psi^R_n - \Psi^L_n\right)\end{aligned}$$

For the computation of the viscous fluxes, the derivatives of any primal flow or adjoint quantity W on the finite volumes interface (between nodes P and Q) are computed as

$$\left(\frac{\partial W}{\partial x_k}\right)^{PQ} = \overline{\left(\frac{\partial W}{\partial x_k}\right)} - \left[\overline{\left(\frac{\partial W}{\partial x_k}\right)} t_m^{PQ} - \frac{W^Q - W^P}{\sqrt{(x_m^Q - x_m^P)^2}}\right] t_k^{PQ}$$

where

$$t_m^{PQ} = \frac{x_m^Q - x_m^P}{\sqrt{(x_m^Q - x_m^P)^2}}$$
$$\overline{\left(\frac{\partial W}{\partial x_k}\right)} = \frac{1}{2}\left[\left(\frac{\partial W}{\partial x_k}\right)^P + \left(\frac{\partial W}{\partial x_k}\right)^Q\right]$$

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

$$\overline{\frac{\partial R_{U_n}}{\partial U_m}} \Delta U_m = -R_{U_n} \ , \qquad U_m^{j+1} = U_m^j + \Delta U_m$$
$$\overline{\frac{\partial R_{\Psi_n}}{\partial \Psi_m}} \Delta \Psi_m = -R_{\Psi_n} \ , \qquad \Psi_m^{j+1} = \Psi_m^j + \Delta \Psi_m \tag{14}$$

where $\overline{\frac{\partial R_{U_n}}{\partial U_m}}, \overline{\frac{\partial R_{\Psi_n}}{\partial \Psi_m}}$ are approximations of the Jacobians for the primal and adjoint fluxes respectively. Eqs. 14 are solved iteratively (j indicates the iterations) using the point–implicit Jacobi method which can be efficiently parallelized. A similar solution method is used for solving the adjoint and primal turbulence model equations.

2.4 Volumetric NURBS Parameterization

The parameterization method used for both the shape to be optimized and the surrounding mesh within the context of this paper is a volumetric NURBS scheme. Volumetric NURBS, are trivariate NURBS defined by a grid of control points and weights (P^{ijk} and w^{ijk} respectively) and three knot vectors (K^ξ,K^η and K^ζ) each one associated with a parametric direction (ξ,η and ζ) and a corresponding degree (p^ξ,p^η,p^ζ). Given a triplet of parametric coordinates, a point's position x_m in the 3D space, is computed as

$$x_m(\xi,\eta,\zeta) = \frac{\sum_i^{N^\xi}\sum_j^{N^\eta}\sum_k^{N^\zeta} \Xi^{i,p^\xi}(\xi) H^{j,p^\eta}(\eta) Z^{k,p^\zeta}(\zeta) P_m^{ijk} w^{ijk}}{\sum_i^{N^\xi}\sum_j^{N^\eta}\sum_k^{N^\zeta} \Xi^{i,p^\xi}(\xi) H^{j,p^\eta}(\eta) Z^{k,p^\zeta}(\zeta) w^{ijk}} \tag{15}$$

where N^r is the number of control points in the r direction and Ξ^{i,p^ξ} is the p^ξ-th degree B-Spline basis function defined on the knot vector K^ξ (H^{j,p^η} and Z^{k,p^ζ} are defined similarly). The B-Spline basis functions w.r.t. the corresponding knot vectors as well as their useful mathematical properties can be found in [7].

Eq. 15 can be differentiated to provide the required geometric sensitivities $\frac{\delta x_l}{\delta b_i}$ and $\frac{\partial}{\partial x_k}\left(\frac{\delta x_l}{\delta b_i}\right)$, required by eq.12. The design variables b_i can be either the coordinates of the control points or the weights. To reduce memory usage by the parameterization scheme, the geometric sensitivities are not stored; instead, there are computed anew whenever needed. However, especially in large meshes, this can be computationally expensive and, in order to reduce the wall clock time needed, their computation is performed on the GPUs.

In order to compute the geometric sensitivities of each mesh node position, its parametric coordinates need to be computed first, using Newton-Raphson iterations. The parametric coordinates of a mesh node are computed at a pre–processing step and remain fixed during the optimization, in which eq. 15 is also the means to adapt the computational mesh to the modified shape.

3 GPU DEPLOYMENT

An in–house CFD solver was used for the solution of the flow and adjoint equations. The solver runs on a cluster of GPUs using the CPU memory of each computational node or the MPI protocol for data interchanges between

GPUs of the same or different nodes, respectively. The GPU cluster used in this paper comprises 4 nodes with 3 NVIDIA Tesla M2050 (Fermi Architecture) each and 2 nodes with 2 NVIDIA K20 (Kepler Architecture) each. The GPU–enabled solver running on a Tesla M2050 is ~ 50 times faster compared to the equivalent CPU–enabled solver on a 2×quad core Intel Xeon E2560 CPU with 12MB cache. K20 GPU is ~ 1.2 times faster than M2050.

The high parallel efficiency of the GPU–enabled solver is strongly related to the GPU–oriented pattern used to access the allocated GPU memory. For instance, large data accesses to the GPU (device) memory are coalesced to 128 Byte memory segments, so as to minimize the device memory bandwidth. Apart from the device memory, GPUs include several other memory types with smaller capacity and different access pattern, namely the constant, texture, shared and local ones. Frequently accessed data, such as the gradients of the flow or adjoint variables, are fetched to textures. Access to the texture memory is performed via the texture cache memory and an appropriate renumbering of mesh nodes is employed in order to minimize cache miss. The low latency shared memory is used for data interchange among the threads of the same block. Finally, the fast constant memory is used for storing constants related to fluid properties and other constant quantities.

Moreover, the GPU solver employs Mixed Precision Arithmetics (MPA), [5], which optimally combines the Double (DPA) and Single Precision Arithmetics (SPA) for the minimization of the device memory transactions, without harming the accuracy of the results. According to MPA, the right-hand-side (r.h.s.) of eqs. 14 is computed and stored using DPA, while the memory consuming left-hand-side (l.h.s.) coefficients are computed using DPA but stored using SPA. MPA minimizes the amount of data transferred between the GPU threads and the device memory, resulting to greater speed-ups compared to the DPA scheme. The truncation error occurring in the storage of the l.h.s. coefficients does not harm the accuracy of the solver since both the primal and adjoint equations are solved in "delta formulation", see equation 14. The use of MPA significantly reduces (by 30% at least) the required GPU memory.

The computation of the flow/adjoint residuals and l.h.s. coefficients, requires the computation and accumulation of the numerical primal/adjoint fluxes and their Jacobians. This can be done using either the so–called Single or Two–Kernel scheme, [6]). In the Single–Kernel scheme, each GPU thread, associated with a mesh node, computes and accumulates the numerical fluxes/Jacobians to form the mesh node residuals and l.h.s. coefficients by looping over all the finite volume boundaries formed around the mesh node. This implies that the numerical fluxes and Jacobians are computed twice per volume interface. This is avoided by using the Two–Kernel scheme where two kernels are launched for the computation of the mesh node residuals and l.h.s. coefficients. The first kernel associates each GPU thread with a single mesh edge (i.e. interface between the finite volumes formed around two adjacent mesh nodes). The numerical fluxes/Jacobians are computed once per finite volume interface by the first kernel and are accumulated by the second one which associates GPU threads with mesh nodes. However, in the Two–Kernel scheme, extra memory space is required for storing temporarily the Jacobians computed by the scheme's first kernel. So, in case the l.h.s. coefficients are not constant, the Two–Kernel scheme is rather prohibitive for large scale applications due to the limited GPU memory capacity. For this reason, the primal solver uses the One–Kernel scheme. On the other hand, the adjoint l.h.s. coefficients depend only on the flow solution field and are constant during the iterative solution of the adjoint equations. Thus, the Single–Kernel scheme is used for the computation of the adjoint l.h.s. coefficients once per optimization cycle, before iteratively solving the adjoint equations, and the Two–Kernel scheme for the residuals at each pseudo–time iteration [14].

4 APPLICATIONS

The continuous adjoint formulation on GPUs described in the previous sections was applied to the shape optimization of a U–bend duct and a compressor cascade.

The first case is concerned with the shape optimization of a linear compressor cascade for minimum total pressure losses. The blade is built based on the NACA6512 airfoil with a stagger angle of 30°. The flow is turbulent with $Re = 4.28 \times 10^5$. The velocity angle at inflow is $56.2°$. Modifications in the airfoil shape are controlled by the control grid of 11×11 points shown in figure 2(b), which also affects part of the computational mesh. Cubic basis functions and uniform knot vectors are used in both parametric directions.

The computational mesh consists of 50662 nodes and 40052 elements. The latter is generated as a 2D triangular mesh using the advancing front technique and superimposed to quadrilateral layers close to the airfoil. Given that the GPU solver handles 3D unstructured/hybrid meshes, the 2D mesh is extruded towards the z–direction. The resulting 3D mesh comprises hexahedral elements close to the blade boundary for better capturing viscous phenomena, while the rest of the domain is filled with prismatic elements. Since, in this case, we are interested in simulating only 2D effects symmetry conditions are applied in the two iso–z planes.

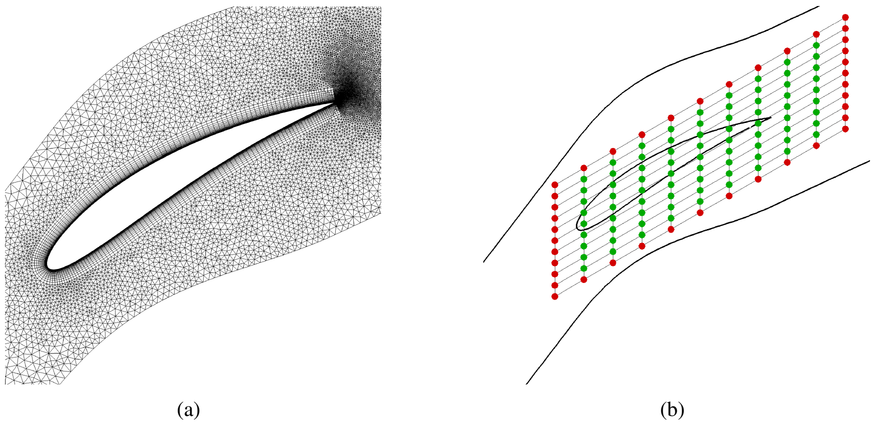


Figure 2: Shape optimization of a linear compressor cascade. (a) Computational mesh close to the blade. (b) Definition of control points for parameterizing the compressor blade. Points in red are kept fixed while the green points are allowed to move in both x and y directions.

The optimization runs for 9 cycles on a single NVIDIA K20 GPU. The average wall clock time for a single solution of the flow PDEs, converged to machine accuracy, is about 1.5min, while for the adjoint field is 1.10min resulting in $\sim$ 3min/cycle. The overall optimization requires $\sim$ 27 min. The optimization convergence history is shown in figure 3. The primal velocity fields along with the total pressure contours around the initial and optimized geometry are presented in figure 4. It is clear that the flow separation zone on the rear part of the suction side is reduced and this leads to lower total pressure losses.

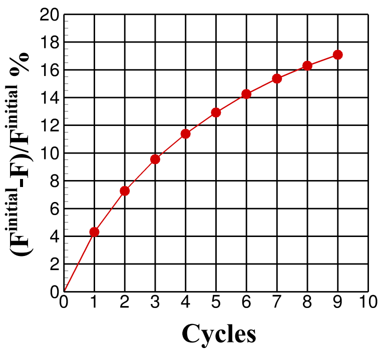
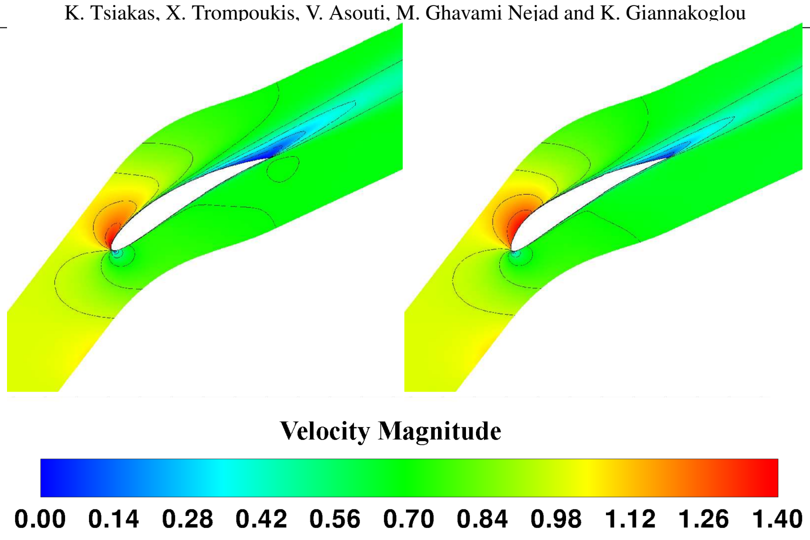


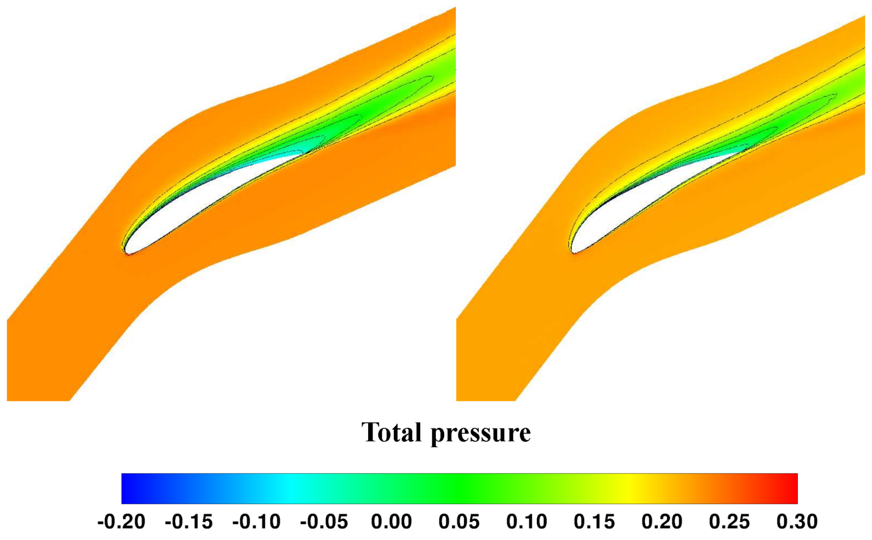
Figure 3: Convergence history of the optimization. The horizontal axis corresponds to optimization cycles, while the vertical axis displays the $\%$ improvement of the objective function value.

.

The second case is concerned with the optimization of a 3D U–bend for minumum total pressure losses. The design variables are the control point coordinates shown in figure 7. The control points form a $7 \times 8 \times 5$ grid and cubic basis functions are used along all parametric directions. Points marked in red are kept fixed so as to ensure C^1 continuity across the interface between the deformed and undeformed regions. The remaining points are allowed to move along the x,y and z axes resulting to $(5 \times 8 \times 5) \times 3 = 600$ design variables overall. The optimization history is shown in figure 5(a). The solution of the flow PDEs requires about 4.3min while the adjoint ones 2.5min resulting in $\sim$ 7.8min/cycle and a total of 1 hour overall for the optimization procedure (9 cycles). The optimized duct shape, computed after 9 cycles, compared to the initial one is presented in figure 5(b). The corresponding contol points are shown in figure 7. In the optimized geometry, the separation area at the exit of the U–bend is practically eliminated. This can be seen in figure 6, where the velocity contours are plotted on a plane half–width of the duct.



(a)



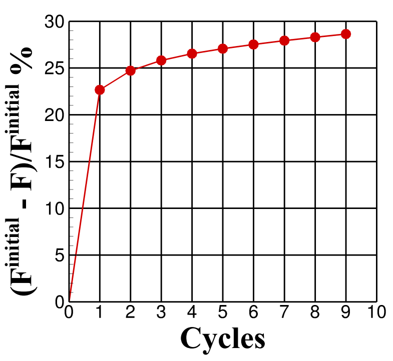
(b)

Figure 4: Shape optimization of a linear compressor cascade. Primal velocity magnitude (a) and total pressure fields (b) for the initial (left) and the optimized (right) shape.

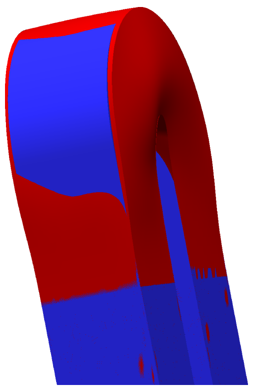
5 CONCLUSIONS

This paper presented the use of the continuous adjoint method for the minimization of total pressure losses in incompressible internal flows. The shapes were parameterized using a volumetric NURBS method, which was also used for deforming the computational mesh around the parameterized shape. The more computationally intensive tasks of the optimization process, such as the solution of the flow and adjoint equations and the computation of the objective function gradient, were implemented on GPUs so as to reduce the optimization wall–clock time. In addition, the volumetric NURBS parameterization scheme introduced additional flexibility to the optimization process to handle arbitrarily shaped geometries. The application of the developed software in a 3D U–bend duct case and a 2D linear compressor cascade airfoil, significantly reduced the total pressure losses with minimum computational effort. In Practicaly the optimization of the 2D case takes $\sim$ 27min while an hour is sufficient for a 3D case.

ACKNOWLEDGMENTS The fourth author acknowledges the support, as an Early Stage Researcher, 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).

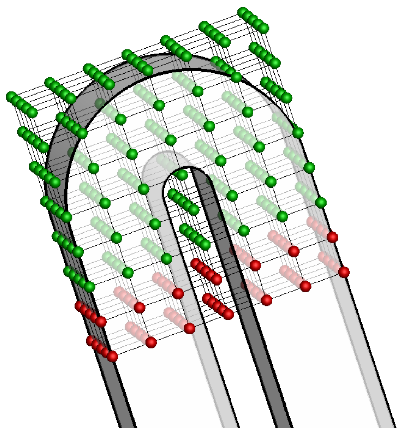


(a)

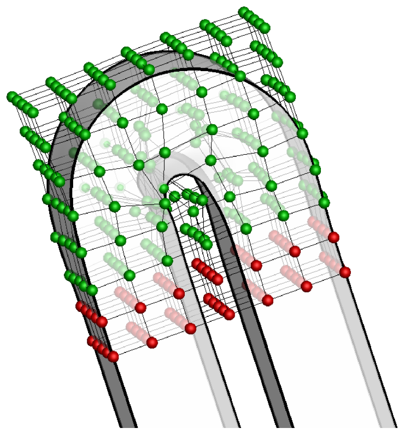


(b)

Figure 5: Shape optimization of a U–bend duct.(a) Optimization convergence history.(b) Initial (blue) and optimized (red) geometry.

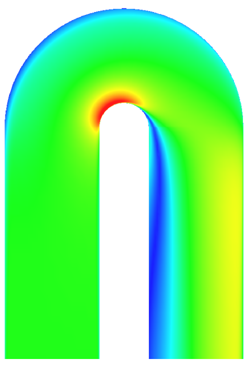


(a)

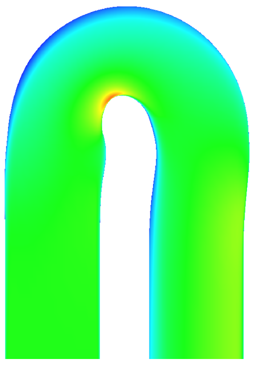


(b)

Figure 6: Shape optimization of a U–bend duct. NURBS control points on the initial (a) and optimized (b) geometries. Points marked in red remain fixed during the optimization.



(a)



(b)

Figure 7: Shape optimization of a U–bend duct. Velocity contours on a half–width plane on the initial (a) and optimized (b) geometries.

References

[1] Chorin A. (1967), ”A numerical method for solving incompressible viscous flow problems”, *Journal of Computational Physics*, Vol. 2, pp. 12-26.

[2] Spalart P., Allmaras S. (1994), ”A one-equation turbulence model for aerodynamic flows”, *La Recherche Aerospatiale*, Vol. 1, pp. 5-21.

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

[4] Zymaris A.S., Papadimitriou D.I., Giannakoglou K.C., Othmer, C. (2009), ”Continuous Adjoint Approach to the Spalart-Allmaras Turbulence Model for Incompressible Flows, *Computers & Fluids*, Vol. 38 , pp. 1528-1538.

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

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

[7] Piegel, L., Tiller, W. (1997), *The NURBS book,* Springer-Verlag, New York.

[8] Roe P. (1981),”Approximate Riemann solvers, parameter vectors, and difference schemes”, *Journal of Computational Physics*, Vol. 43, pp. 357-372.

[9] Hagen T.R., Lie K.A., Natvig J.R.(2006),”Solving the Euler equations on graphics processing units”,*Computational Science - ICCS*, Vol. 3994, pp. 220-227.

[10] Brandvik T., Pullan G. (2008), ”Acceleration of a 3D Euler solver using commodity graphics hardware”, *46th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, USA.

[11] Corrigan A., Camelli F.F., Löhner R., Wallin J. (2011), ”Running unstructured grid-based CFD solvers on modern graphics hardware”, *International Journal for Numerical Methods in Fluids*, Vol. 66, pp. 221-229.

[12] Lefebvre M., Guillen P., Le Gouez J.-M., Basdevant C. (2012), ”Optimizing 2D and 3D structured Euler CFD solvers on Graphical Processing Units”,*Computers & Fluids*, Vol, 70, pp. 136-147.

[13] Oyarzun G., Borrell R., Gorobets A., Lehmkuhl O., Oliva A. (2013), ”Direct Numerical Simulation of Incompressible Flows on Unstructured Meshes Using Hybrid CPU/GPU Supercomputers”, *Procedia Engineering*, Vol. 61, pp. 87-93.

[14] Trompoukis X.S., Tsiakas K.T, Ghavami Nejad M., Asouti V.G. and Giannakoglou K.C. (2014), ”The Continuous Adjoint Method on Graphics Processing Units for Compressible Flows”, *OPT-i, International Conference on Engineering and Applied Sciences Optimization*, Kos, Greece.

[15] Papoutsis–Kiachagias E.M, Giannakoglou K.C (2015), ”Continuous Adjoint Methods for Turbulent Flows, Applied to Shape and Topology Optimization: Industrial Applications”, *Archives of Computational Methods in Engineering, to appear*

[16] Martin M.J., Andres E., Lozano C., Valero E. (2014), ”Volumetric b-splines shape parametrization for aerodynamic shape design”, *Aerospace Science and Technology*, Vol. 37, pp. 26-36.