SHAPE OPTIMIZATION USING THE ONE-SHOT ADJOINT TECHNIQUE ON GRAPHICS PROCESSING UNITS

V.G. Asouti, E.A. Kontoleontos, X.S. Trompoukis and K.C. Giannakoglou[1]

[1]National Technical University of Athens, Parallel CFD & Optimization Unit, Athens, Greece
Correspondance: kgianna@central.ntua.gr

Keywords: Graphics Processing Units, Computational Fluid Dynamics, Optimization, Adjoint methods.

Abstract. *This paper presents the implementation of the one-shot technique for the solution of design/shape optimization problems on NVIDIA Graphics Processing Units (GPUs), for incompressible fluid flow problems. One-shot optimization techniques are based on the simultaneous solution of the flow, adjoint and shape correction equations. They are efficient alternatives to standard gradient-based algorithms in which, within each cycle, the flow and the adjoint equations are solved the one after the other, followed by the update of the shape using the computed gradient of the objective function. The aim of this paper is to superimpose the parallel speed-up gained by GPU-enabling the corresponding software to the gain in efficiency offered by the one-shot algorithm. To this end, the experience of the authors' group in porting Navier-Stokes solvers for compressible fluid flows to GPUs is exploited. Key features of the new method are the use of the flow equations for incompressible fluids and the solution of the coupled flow-adjoint equations on the GPU. The programmed software is used for the shape optimization of the tubes of a heat exchanger and an elbow duct.*

1 INTRODUCTION

Gradient-based methods supported by adjoint techniques are frequently used for the design or shape optimization of aerodynamic shapes. The main advantage of the adjoint method is the low cost for computing the gradient of the objective function, which is almost equal to the cost of the flow equations; this cost is independent of the number of the design variables and this is the great advantage of the adjoint method compared to other rival methods, such as finite differences. The adjoint equations in discrete form can be obtained either directly from the discretized flow (state) equations or by deriving and discretizing the adjoint partial differential equations (PDEs). These are referred to as the discrete [1, 2] and the continuous adjoint approach [3, 4, 5, 6], respectively. Each cycle of the optimization, using the standard steepest-descent algorithm, involves the solution of the flow and the adjoint equations (solved in segregated manner, i.e. each system of equations after the other), the computation of the gradient of the objective function and the shape correction based on this gradient. Based on the literature [7, 8], the one-shot optimization technique, which is based on the simultaneous solution of the flow, adjoint and shape correction equations, may reduce the overall computational cost.

This paper deals with the one-shot adjoint technique for incompressible flows with heat transfer effects. The continuous adjoint approach is adopted, by considering variations of the turbulence model variables with respect to the design variables, as originally proposed in [9]. The latter is absolutely necessary for accurate gradients to be computed. Since, nowadays, GPUs are used in scientific computing including CFD [10, 11, 12, 13], the proposed method is ported to NVIDIA GPUs in order to reduce the wall clock time for the solution of the flow and adjoint equations and, as a consequence, the optimization turnaround time.

GPU implementations of CFD algorithms for either structured [11, 12, 14, 15] or unstructured [10, 13, 16, 17] grids can be found in the literature. Those for structured grids profit of the fully organized (structured) memory accesses and lead to high speed-ups, as memory handling is crucial for the performance of any GPU implementation. Regarding GPU-enabled codes for unstructured grids, their parallel performance depends on the spatial discretization scheme. For instance, the cell-centered finite volume scheme is more advantageous compared to the vertex-centered one, as far as memory access is of concern. This is due to the fact that, in a vertex-centered scheme, the number of the adjacent nodes affecting the computation of fluxes crossing the boundaries of any finite volume may vary a lot. In contrast, in a cell-centered scheme, the number of adjacent cell centers is constant (apart from elements in the vicinity of the boundaries).

In this work, the GPU implementation of the one-shot adjoint technique is carried out using the vertex-centered approach. The expected gain in computational cost (compared to the segregated method running on a CPU) is expected to result from (a) the use of the one-shot algorithm which is more efficient than the segregated (standard) one and (b) the software porting to the GPU. Regarding the latter, the authors already have enough experience, for compressible flow predictions and solely for the solution of the flow equations, see [10, 16, 17]. This experience is herein transferred to incompressible flow problems and the one-shot adjoint approach. The

developed GPU-enabled code is used to solve two problems: (a) the shape optimization of a heat tube exchanger for maximum heat transfer and minimum total pressure losses and (b) the design of a duct with 90° turning for minimum total pressure losses. The obtained optimal solutions are the same irrespective of the use of the one-shot or the segregated technique and/or the use of CPUs or GPUs. Here, emphasis is laid to the reduction of the optimization turnaround time.

2 THE ONE-SHOT ADJOINT ALGORITHM

2.1 Development of the continuous adjoint method

The Navier-Stokes equations for incompressible flows, including the energy equation and the one-equation Spalart-Allmaras turbulence model [18], constitute the system of *state or flow PDEs*

$$R_p = \beta^2 \frac{\partial v_j}{\partial x_j} = 0 \tag{1a}$$

$$R_{v_i} = v_j \frac{\partial v_i}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[(\nu + \nu_t)\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\right] = 0 \tag{1b}$$

$$R_T = \frac{\partial (v_i T)}{\partial x_i} - (a + a_t)\frac{\partial^2 T}{\partial x_i^2} = 0 \tag{1c}$$

$$R_{\tilde{\nu}} = \frac{\partial (v_i \tilde{\nu})}{\partial x_i} - \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\tilde{\nu}}{\sigma}\right)\left(\frac{\partial \tilde{\nu}}{\partial x_j}\right)\right] - \frac{c_{b_2}}{\sigma}\frac{\partial^2 \tilde{\nu}}{\partial x_i^2} - \tilde{\nu} P(\tilde{\nu}) + \tilde{\nu} D(\tilde{\nu}) = 0 \tag{1d}$$

where p, v_i, T denote the static pressure, the velocity components and the static temperature and $\tilde{\nu}$ stands for the Spalart-Allmaras model variable. These quantities constitute the so-called vector of *state or flow variables*. The state equations are solved using the pseudo-compressibility method, by introducing appropriate time-derivatives in all of them and the artificial compressibility coefficient β in (1a). In the above equations, ρ is the constant density, $x_i, i = 1,2$ are the Cartesian coordinates, α, a_t are the bulk and turbulent thermal diffusivities, ν is the bulk viscosity, $\nu_t = f_{v_1}\tilde{\nu}$ is the turbulent viscosity and $P(\tilde{\nu}), D(\tilde{\nu})$ stand for the production and destruction terms respectively, as described in detail in [18]. In incompressible flows, eqs (1a), (1b) and (1d) are solved in a coupled manner and, after these converge to a user-defined convergence level, eqn (1c) is solved separately to provide the temperature field.

For laminar flows with heat transfer, such as the shape optimization of a tube heat exchanger studied below, the solution of the turbulence model equation (1d) is omitted; consequently, $\nu_t = a_t = 0$. For the second examined case which is concerned with the optimization of an elbow duct, where the flow is turbulent without however heat transfer effects, the system of equations (1a), (1b) and (1d) is solved. Note that the development that follows considers the full system of eqs (1).

Based on the continuous adjoint method, to compute the gradient of the objective function F, the augmented function F_{aug} is defined as the sum of F and the field (Ω) integral of the residual of eqs (1) multiplied by the adjoint variables $(q, u_i, T_a, \tilde{\nu}_a)$, $F_{aug} = F + \int_\Omega \left(qR_p + u_i R_{v_i} + T_a R_T + \tilde{\nu}_a R_{\tilde{\nu}}\right) \mathrm{d}\Omega$. Note that q (adjoint pressure) is the adjoint to p, u_i (adjoint velocity) is the adjoint to v_i, T_a (adjoint temperature) is the adjoint to T and $\tilde{\nu}_a$ (adjoint Spalart-Allmaras model variable) is the adjoint to $\tilde{\nu}$. The variation of F_{aug} with respect to the design variables' vector, $\vec{b} \in R^N$, is expressed as follows, [19],

$$\frac{\delta F_{aug}}{\delta \vec{b}} = \frac{\delta F}{\delta \vec{b}} + \int_\Omega \left(q\frac{\delta R_p}{\delta \vec{b}} + u_i \frac{\delta R_{v_i}}{\delta \vec{b}} + T_a \frac{\delta R_T}{\delta \vec{b}} + \tilde{\nu}_a \frac{\delta R_{\tilde{\nu}}}{\delta \vec{b}}\right) \mathrm{d}\Omega + \int_S \left(qR_p + u_i R_{v_i} + T_a R_T + \tilde{\nu}_a R_{\tilde{\nu}}\right)\frac{\delta x_k}{\delta \vec{b}} \mathrm{n_k}\, \mathrm{d}S \tag{2}$$

where S is the boundary of the domain (comprising the inlet S_I, outlet S_O and solid walls S_W) and n_k are the components of the unit normal vector to any boundary segment dS.

The development of the field integrals of eqn (2), based on the Gauss divergence theorem, gives rise to

$$\frac{\delta F_{aug}}{\delta \vec{b}} = \underbrace{\frac{\delta F}{\delta \vec{b}} - \int_{S_w}\left(\nu\frac{\partial u_i}{\partial x_j}\frac{\partial v_i}{\partial x_k} + (a + a_t)\frac{\partial T_a}{\partial x_j}\frac{\partial T}{\partial x_k} + \nu\frac{\partial \tilde{\nu}_a}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_k}\right) n_j \frac{\delta x_k}{\delta \vec{b}}\, dS + \int_\Omega \tilde{\nu}_a \tilde{\nu} C_d \frac{\partial d}{\partial b}\, d\Omega}_{term\ A} + \int_\Omega \left(R_q\frac{\delta p}{\delta \vec{b}} + R_{u_i}\frac{\delta v_i}{\delta \vec{b}} + R_{T_a}\frac{\delta T}{\delta \vec{b}} + R_{\tilde{\nu}_a}\frac{\delta \tilde{\nu}}{\delta \vec{b}}\right) d\Omega + \int_s \left(B_q\frac{\delta p}{\delta \vec{b}} + B_{u_i}\frac{\delta v_i}{\delta \vec{b}} + B_{T_a}\frac{\delta T}{\delta \vec{b}} + B_{\tilde{\nu}_a}\frac{\delta \tilde{\nu}}{\delta \vec{b}}\right) \mathrm{d}S \tag{3}$$

The adjoint field equations (4) and their boundary conditions are derived by forcing all field and boundary

integrals depending on the variations in the state variables $\frac{\delta p}{\delta \vec{b}}, \frac{\delta v_i}{\delta \vec{b}}, \frac{\delta T}{\delta \vec{b}}, \frac{\delta \tilde{\nu}}{\delta \vec{b}}$ to be eliminated from eqn (3). The so-derived field adjoint equations are:

$$R_q = \frac{\partial u_j}{\partial x_j} = 0 \tag{4a}$$

$$R_{u_i} = v_j \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \beta^2 \frac{\partial q}{\partial x_i} + \frac{\partial}{\partial x_j}\left[(\nu + \nu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] + T\frac{\partial T_a}{\partial x_i} = 0 \tag{4b}$$

$$R_{T_a} = v_i \frac{\partial T_a}{\partial x_i} + (a + a_t)\frac{\partial^2 T_a}{\partial x_i^2} = 0 \tag{4c}$$

$$R_{\tilde{\nu}_a} = v_j \frac{\partial \tilde{\nu}_a}{\partial x_j} + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\tilde{\nu}}{\sigma}\right)\left(\frac{\partial \tilde{\nu}_a}{\partial x_j}\right)\right] - \frac{1}{\sigma}\frac{\partial \tilde{\nu}_a}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j} - 2\frac{c_{b_2}}{\sigma}\frac{\partial}{\partial x_j}\left(\tilde{\nu}_a \frac{\partial \tilde{\nu}}{\partial x_j}\right) + \tilde{\nu}_a P(\tilde{\nu}) - \tilde{\nu}_a D(\tilde{\nu}) - \tilde{\nu}_a \tilde{\nu} C_{\tilde{\nu}}$$
$$+ \frac{\delta \nu_t}{\delta \tilde{\nu}}\left(-\frac{\partial u_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \frac{\partial T_\alpha}{\partial x_j}\frac{\partial T}{\partial x_k}\right) = 0 \tag{4d}$$

The corresponding boundary conditions result from the elimination of the last integral (along S) in eqn (3). For instance, at the inlet, $u_i n_i = v_i n_i$, $u_i t_i = 0$, (where t_i are components of the tangent to the boundary unit vector) for the velocity components and zero Dirichlet conditions must be applied for $\tilde{\nu}_a$ and T_a. Zero Dirichlet conditions are also imposed to u_i, $\tilde{\nu}_a$ and T_a and zero Neumann to q along the solid walls. The boundary conditions at the outlet require a much lengthy development which is omitted in the interest of space; the reader may find more about this development in [9].

After numerically solving the discretized adjoint PDEs, eqs (4), with their boundary conditions, the variation in F_{aug} become independent of variations in the state variables; so, the remaining integral terms (marked as *term A* in eqn (3)) yield the gradient of the objective function. Regarding the discretization of the state or adjoint PDEs, these can be handled through similar schemes. However, it is important to note that, even if the energy equation (in system (1)) is decoupled from the remaining ones and can thus be solved separately, the adjoint energy equation, eqn (4c) remains coupled with the other adjoint equations, due to the presence of terms depending on T_a in eqs (4b) and (4d).

2.2 The one-shot technique – Solution algorithm and discretization scheme

The one-shot optimization method, employed in this paper, relies on the coupled solution of the flow and adjoint equations, namely eqs (1) and (4), along with the updating of the shape based on the computed gradient of F. More about the one-shot technique and its effect on a GPU-enabled solver are presented as the paper develops.

In the present work, eqs (1) and (4) are integrated on unstructured grids using the vertex-centered finite volume technique. A typical finite volume (Ω_P), formed around a grid node P, is illustrated in figure 1. Let $\vec{U} = (p, v_i, T, \tilde{\nu}, q, u_i, T_a, \tilde{\nu}_a)$ denote the vector of the flow and adjoint variables.

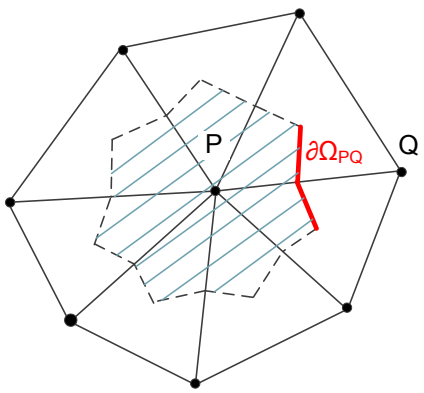

Figure 1: Vertex-centered finite volume Ω_P (hatched area) formed around node P, on an unstructured grid. $\partial\Omega_{PQ}$ denotes the boundary of Ω_P associated with grid edge PQ.

The iterative point implicit Jacobi method is employed for updating the $\vec{U}$ values at each one-shot cycle (n), as follows

$$\Delta\vec{U}_P^{n+1,j+1} = (D_P^n)^{-1}\left[-\vec{R}_P^n - \sum_{Q\in nei\,(P)} Z_Q^n \Delta\vec{U}_Q^{n+1,j}\right] \tag{5}$$

where j denotes Jacobi iterations performed within each cycle; the maximum number of Jacobi iterations is defined by the user. $\Delta\vec{U}$ denotes the correction to the flow and adjoint variables, used to update their values $\left(\vec{U}_P^{n+1} = \vec{U}_P^n + \Delta\vec{U}_P^{n+1}\right)$ at the end of each cycle (index n), just before reshaping the body to be designed. D, Z are the diagonal and non-diagonal left-hand-side (l.h.s.) coefficient matrices, respectively. Both result from the first-order discretization of the governing PDEs. Note that all computations are performed with second-order spatial accuracy but this affects only the residual $\vec{R}_P$ of the discretized equations. Ignoring second-order accuracy for the l.h.s. matrices reduces the computational stencil; during the Jacobi iterations, node P is affected only by $nei(P)$ nodes directly linked with it by an edge. Based on the finite-volume technique, $\vec{R}_P$ is formed by accumulating the inviscid and viscous fluxes crossing its $nei(P)$ boundary parts $\partial\Omega_{PQ}$. It should be noted that in vertex-centered approach used herein, $nei(P)$ varies from node to node; thus, different number of fluxes should be computed and summed up per grid node P. This is a key factor for the performance of the GPU implementation. The numerical fluxes are computed based on the 1D Roe approximate Riemann solver [20], whose implementation requires the values of $\vec{U}$ along with their spatial derivatives at nodes P, Q.

3 IMPLEMENTATION ON GPUS

GPU programming issues ensuring that the software developed for the solution of the one-shot adjoint method has maximum parallel efficiency are discussed. The developed code is compared in terms of the optimization turnaround time with the corresponding code running on a CPU.

3.1 The NVIDIA's GTX 285 Graphics Card

The numerical computations to be presented have been carried out on a NVIDIA's Ge-Force GTX 285 graphics card whose peak performance is estimated to 1063 GFLOPs. It is based on the GT200 architecture and comprises 30 Streaming Multiprocessors (SMs) grouped in 10 Texture Processor Clusters (TPCs). Each SM consists of 8 Streaming (Scalar) Processors (SPs), a multithreaded instruction unit, 2 Special Function Units for transcendental operations and 16 KB of on-chip shared memory. Threads, grouped in blocks, are distributed to SMs and then the instruction unit per SM organizes them in groups of 32 (warp) and addresses each thread to a single SP according to the SIMT (Single Instruction Multiple Threads) architecture. Thus, the same code (called kernel), for different data sets, is executed on the SPs of the same SM. Independent SMs may execute different parts of the same code. So the programmer is prompted to split the problem into coarse sub-problems and, then, into finer pieces (assigning for instance each thread to a single grid node) which can be solved in parallel.

Furthermore, threads of the same block can interchange data through the fast shared memory per SM. Synchronization instruction is available as well. The number of threads per block is defined by the programmer restricted by the kernel memory and register requirements (GTX 285 GPU enables 16384 32-bit registers per SM), along with the fact that each block can only be assigned to a single SM and each SM can execute up to 1024 threads.

3.2 The Navier-Stokes equations solver on the GPU

The efficient porting of an in-house compressible Navier-Stokes equations solver employing the vertex-centered finite volume technique on unstructured/hybrid grids, from the CPU to the GPU using the CUDA architecture, has been described in three recent publications by the same group [10, 16, 17]. Both 2D/3D, unsteady/steady computations were carried out and the corresponding speed-up values exceeded 48 for double (DPA) or 70 for single precision arithmetic (SPA) computations, depending on the grid size. In [16], the use of a mixed precision arithmetic (MPA) scheme was proposed as a more efficient alternative to the DPA one, without reducing the prediction accuracy. In MPA, the memory demanding l.h.s. coefficient matrices are stored in single precision variables. This reduces the memory transactions and increases the performance of the GPU-enabled software. Based on [10], speed-up values of the MPA/GPU implementation exceeded 51.

Two different flux computing schemes were employed and compared, in terms of parallel speed-up, in [17]. In the first scheme, the computation and accumulation of fluxes (right-hand-side (r.h.s.) terms, i.e. residuals) and l.h.s. coefficient matrices (D and Z, eqn (5)) were all based on the same kernel for all grid nodes. For each node, by sweeping all edges emanating from it, one-by-one, the corresponding inviscid and viscous fluxes were computed. The cumulative sum of these fluxes yielded the nodal residuals of equations. The same kernel computed also the coefficient matrices D and Z, which are stored by node and edge, respectively. In the second

proposed scheme, two kernels were sequentially executed. Firstly, through an edge-based kernel (i.e. threads were associated with grid edges), fluxes were computed and stored by edge. In this scheme, at the expense of extra GPU memory requirements, each numerical flux associated with a single edge was computed once, as opposed to the first scheme in which all fluxes were computed twice. The edge-based kernel was followed by a node-based (with threads associated with the grid nodes) one, which accumulated fluxes already stored by edge, to compute the r.h.s. and l.h.s. terms for each node. The first scheme, which is more computational intensive but less memory demanding (leading to less and coalesced memory accesses), had better performance and led to higher speed-up values.

Recall that the present paper is dealing with incompressible flows and extends previous findings to the numerical solution of the adjoint equations, over and above the flow equations, in the one-shot fashion in specific.

3.3 The one-shot optimization algorithm on the GPU

In any GPU-enabled software, a sequence of routines which concurrently process data on the GPU are called and synchronized by the CPU. The developed software for the GPU is schematically illustrated in figure 2. The CPU reads the input data (such as the coordinates and the IDs of the unstructured grid nodes, the flow boundary conditions, etc.) and computes the necessary pieces of information related to the grid topology and connectivity as well as geometrical quantities related to the integration of the governing equations over the finite-volumes centered at grid vertices. These are communicated to the GPU, where the Navier-Stokes, adjoint and shape correction equations are solved. The proposed algorithm includes a first phase during which the flow and, then, the adjoint equations are solved in a segregated manner, up to a user-defined tolerance for the corresponding residuals. In the second phase, the one-shot algorithm is applied; during this phase, within each cycle, one pseudo-time step of both the flow and adjoint equations, followed by the shape correction, is performed.

Each one-shot cycle involves the following steps:

1. Computation/accumulation of the numerical fluxes (for both the Navier-Stokes and adjoint equations) crossing the finite volume boundaries, to come up with the nodal residuals and the l.h.s. coefficient matrices; this corresponds to the first flux computing scheme described in 3.2.
2. Updating the flow/adjoint field by performing a user-defined number of Jacobi sub-iterations.
3. Computation of the gradient of F with respect to each design variable. Each gradient component is assigned to a different SM, increasing thus, even more, the parallel efficiency.
4. Updating the design variables' values based on the so-computed gradient and steepest descent.
5. Reshaping the part of the domain boundary which is controlled by the design variables.
6. Deforming the existing grid and adapting it to the new boundaries. Since, in the one-shot algorithm, each cycle yields small changes in the designed shape, simple analytical functions are used for the movement of grid nodes, [21]. Through the movement of nodes, the need for remeshing the computational domain is avoided.
7. Computation of the updated geometrical data for the deformed grid.

3.3 A few comments on programming issues

A distinguishing feature of the developed GPU-enabled software is the use of the vertex-centered technique for the integration of the Navier-Stokes and adjoint equations on unstructured grids. This approach calls for delicate memory handling in order to maximize the parallel efficiency of the code due to the random numbering of the grid nodes and the variable number of surrounding nodes per node. Efficient ways of accessing the memory demanding l.h.s. coefficient matrices (D, Z) of the discretized equations (5) can be found in [10, 16, 17]. According to these, the 16 threads of a half-warp, executed concurrently, access a sequence of data forming a 128 Byte segment of the GPU global memory, maximizing the memory bandwidth. Besides, for the less memory demanding r.h.s. terms (nodal residuals) the fast GPU texture memory is employed to hide the high latency of the GPU global memory. Moreover, in this paper, a grid vertex ID renumbering technique is used to further reduce the time needed for read/write operations from/to the GPU global memory. According to this technique, threads running in parallel access nearby GPU global memory spaces.

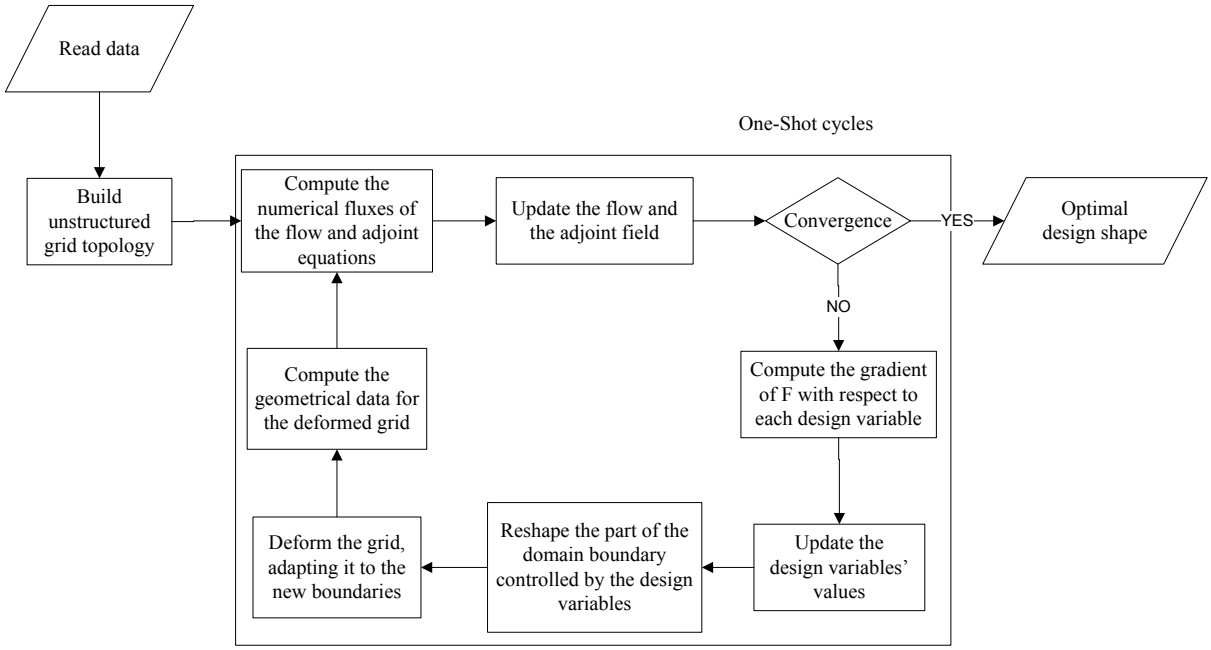

Figure 2: Flowchart of the GPU one-shot implementation. The first phase, in which the flow and adjoint equations are solved the one after the other up to a user-defined accuracy, is omitted.

4 APPLICATIONS

4.1 Design of a tube heat exchanger

The first problem is concerned with the shape optimization of the tubes of a staggered tube bank heat exchanger, for minimum volume-averaged total pressure losses f_1 and maximum heat exchange f_2 between the inlet (S_I) to and the outlet (S_o) from the flow domain, as

$$f_1 = -\int_{S_{I,O}} \frac{1}{\rho}\left(p + \frac{1}{2}\rho v^2\right) v_i \, n_i dS$$

$$f_2 = -\int_{S_{I,O}} T \, dS$$

Here, n_i correspond to the components of the normal to the boundary vector and v is the velocity vector norm. In this case, the objective function F is defined by concatenating the two objectives f_i into a single scalar function, $F = \omega_1 f_1 + \omega_2 f_2$, where $\omega_1 = 0.9, \omega_2 = 0.1$ are chosen as weighting factors. In industrial and power engineering applications, heat exchangers with banks of tubes are widely used as evaporators or cooling heat exchangers. The tube length in the longitudinal direction is much larger than its width, making this 2D study physically consistent with the flow over the mid-span plane of heat exchangers. The heat exchanger and the boundaries of the 2D computational domain are shown in figure 3. The computational domain is periodic along its sidewalls and contains only four tubes. The outlet boundary is extended several chord-lengths downstream the last tubes. The fluid enters the domain with temperature 293K, the flow Reynolds number based on the vertical distance h ($h = 2c$, where $c = 1$ is the chord of the tube, figure 3) is equal to 160, while, high temperature fluid flows inside the tubes in order to ensure constant wall temperature 353K. The tube shape is symmetric along the horizontal axis and is parameterized using Bezier-Bernstein polynomials, with 8 control points on each side. Three of them were allowed to vary both on the chord and the normal-to-the-chord direction summing up to 12 design variables in total. All tube cross sections are identical and located in predefined positions. The computational grid is formed by generating structured-like layers of triangles around each tube and, then, by filling in the remaining domain with triangular elements using the advancing front technique.

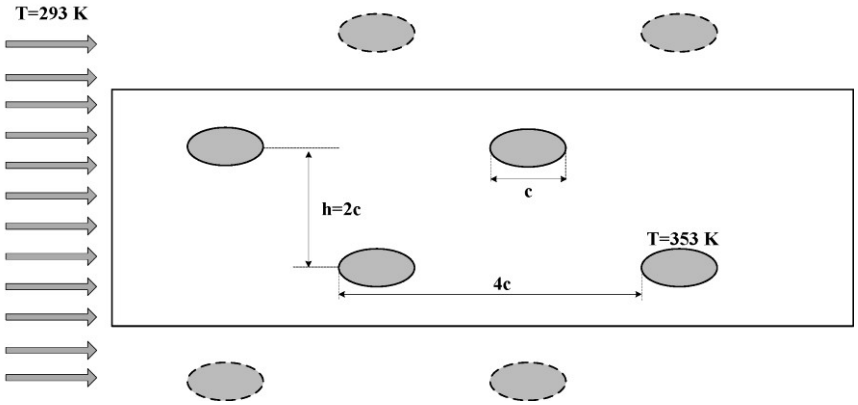

Figure 3: Schematic representation of the computational domain of a staggered tube bank heat exchanger.

Figure 4 compares the convergence histories of the one-shot and the segregated technique, both running on the GPU, in terms of wall clock time. The first one-shot cycle, where the flow and adjoint equations are separately solved up to a user-defined accuracy (exactly as in the first cycle of the segregated technique), is omitted for both techniques. It is clear that the one-shot technique (on the GPU) reaches the optimal shape about 1.8 times faster than the segregated technique (on the GPU). The parallel speed-up, computed vis-à-vis to the CPU, for an unstructured grid with ~80K nodes, which proved to be adequate for grid-independent computations, is approximately 40x. A GTX 285 graphics card and a single core of a 2xQuad Core Intel Xeon CPU (2.00 GHz, 4096 KB cache size) have been employed for the aforementioned runs. For the compilation of the CPU code the GNU Fortran compiler version 4.1.2 was used, with full optimization options. The initial and the optimal tube shapes are shown in figure 5; either the segregated or the one-shot algorithm give the same optimal shapes.

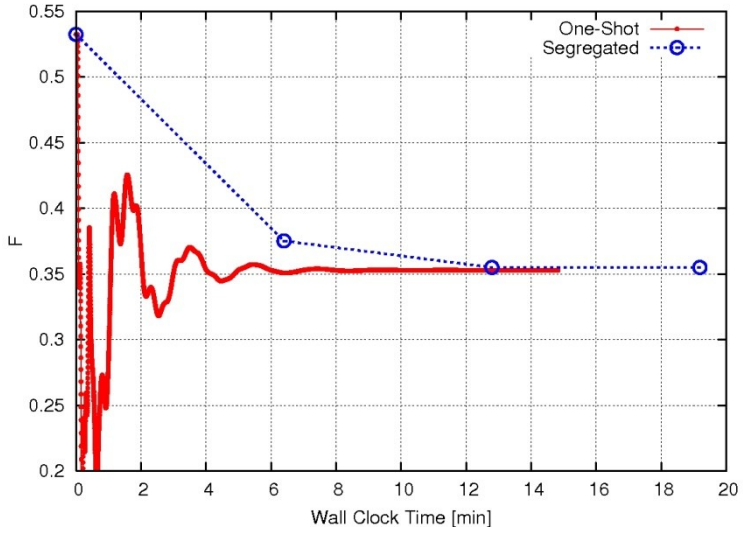

Figure 4: Comparison of the wall clock time required for the tube heat exchanger's design using the segregated and the one-shot technique on GPUs.

(a) $f_1 = 0.89, f_2 = 35.23$ (b) $f_1 = 0.70, f_2 = 34.10$

Figure 5: The initial (a) and the optimal (b) tube shape obtained by the one-shot optimization.

4.2 Design of a duct with 90° flow turning

The second case deals with the design of an elbow duct, turning the flow by 90° which is designed for minimum volume-averaged total pressure losses ($F = f_1$). The duct solid walls were parameterized using two Bezier-Bernstein polynomials, with 7 control points each. The domain inlet and outlet boundaries are located far upstream and far downstream the elbow starting and ending cross-sections. As before, the computational grid is formed by generating structured-like layers of triangles around the wall boundaries and filling in the remaining domain with triangular elements using the advancing front technique.

The convergence histories of the one-shot and the segregated techniques on GPUs, as a function of wall clock time are presented in figure 6. As before, the very first (segregated) cycle is not plotted for both methods. One-shot reaches the optimal shape ~1.7 times faster than the segregated technique. This gain is superimposed to the speed-up obtained (34x for a grid of ~40K nodes, employing the same computational platforms mentioned in 4.1) by using GPUs. The initial and the optimal elbow and the velocity field of the optimal solution are shown in figure 7.

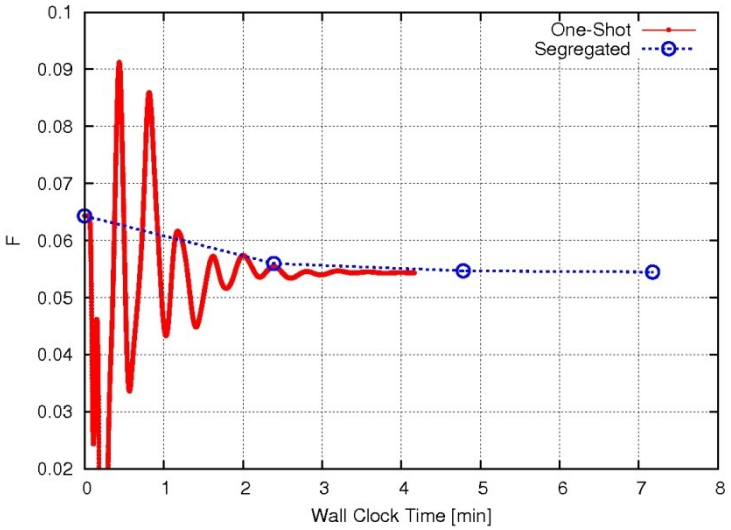

Figure 5: Comparison of the wall clock time required for the design of the elbow using the segregated and the one-shot technique on GPUs.

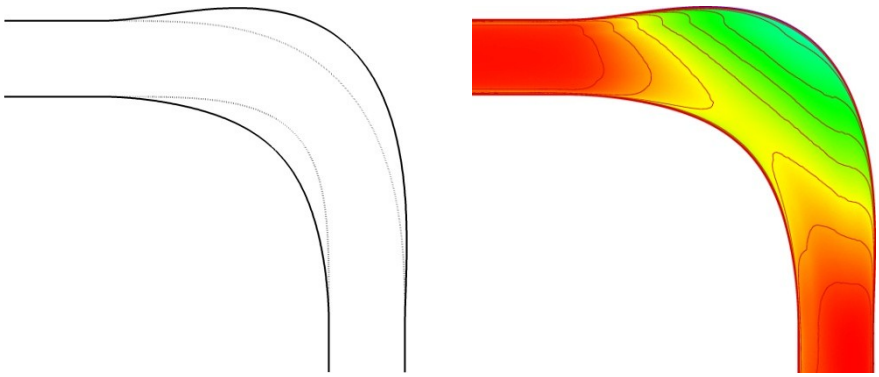

Figure 7: The initial (dotted line) and the optimal (bold line) elbow shape (left) and the velocity field of the optimal solution (right).

5 CONCLUSIONS - DISCUSSION

This paper demonstrated the advantage of using the one-shot adjoint technique for solving design-optimization problems on GPUs. Compared to previous works by the same group related to the porting of CFD solvers to NVIDIA's GPUs, there are three major extensions and novelties, namely: (a) the solution of the incompressible, rather than compressible, fluid flow equations, (b) the solution of the adjoint to the flow equations, corresponding to an objective function that includes viscous losses and heat transfer effects and (c) the coupling of the solution scheme used for the system of flow and adjoint equations, which is usually referred

to as the one-shot technique. Considering the "usual" parallel speed-up gained from porting and running the CFD software on the GPU, the additional implementation of the one-shot technique almost divides the computational cost by 2.

The parallel speed-up obtained in this paper (for the incompressible equations and the coupled system of flow and adjoint equations) appears to be smaller than that obtained in [10, 16, 17] (in which the reader may find a parametric study regarding the effect of the grid size on the parallel speed-up). This is related to the size of computational grids. For instance, by generating and using a very dense unstructured grid of about 180K nodes in the tube heat exchanger case (though such grid is much finer than needed for this problem) the achieved speed-up reaches 48x.

AKNOWLEDGMENT

The first author was supported by a Post-Doc scholarship from the Greek State Scholarships Foundation. The second author was supported by the Center for Renewable Energy Systems and Saving. The third author was supported by a scholarship from the Public Benefit Foundation Alexandros S. Onassis.

REFERENCES

[1] Elliot, J. and Peraire, J. (1996), ''Aerodynamic design using unstructured meshes", *27th Fluid Dynamics Conference*, New Orleans, LA, USA, AIAA Paper 1996-1941.

[2] Duta, M. and Giles, M. and Campobasso, M. (2002), ''The harmonic adjoint approach to unsteady turbomachinery design", *International Journal for Numerical Methods in Fluids*, Vol. 40, pp. 323-332.

[3] Papadimitriou D., Giannakoglou K. (2008), "Aerodynamic shape optimization using adjoint and direct approaches", *Archives of Computational Methods in Engineering*, Vol. 4, pp. 447-488.

[4] Papadimitriou D., Giannakoglou K. (2009), ''The continuous direct-adjoint approach for second order sensitivities in viscous aerodynamic inverse design problems", *Computers & Fluids*, Vol. 38, pp. 1539-1548.

[5] Jameson, A. (1988), ''Aerodynamic design via control theory", *Journal of Scientific Computing*, Vol. 3, pp. 233-260.

[6] Anderson W., Venkatakrishnan V. (1997), "Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation", *35th Aerospace Sciences Meeting and Exhibit,* Reno, NV, USA, AIAA Paper 1997-643.

[7] Hazra S., Schulz V., Brezillon J. and Gauger N. (2005), "Aerodynamic shape optimization using simultaneous pseudo-time stepping", *Journal of Computational Physics*, Vol. 204, pp. 46-64.

[8] Hazra S., Jameson A. (2007), "One-shot pseudo-time method for aerodynamic shape optimization using the Navier-Stokes equations", *45th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, AIAA Paper 2007-1470.

[9] Zymaris A., Papadimitriou D., Giannakoglou K. and Othmer C. (2009), "Continuous adjoint approach to the Spalart-Allmaras turbulence model for incompressible flows", *Computers & Fluids*, Vol. 38, pp. 1528-1538.

[10] Trompoukis X., Asouti V., Kampolis I. and Giannakoglou K., (2011), *CUDA implementation of Vertex-Centered, Finite Volume CFD methods on Unstructured Grids with Flow Control Applications*, GPU Computing Gems Vol. 2, Morgan Kaufmann.

[11] Hagen T., Lie K. and Natvig J. (2006), "Solving the Euler equations on Graphics Processing Units", *Computational Science – ICCS*, Vol. 3994, pp. 220-227.

[12] Bradvik T. and Pullan G. (2008), "Acceleration of a 3D Euler solver using commodity graphics hardware", *46th Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, AIAA Paper 2008-607.

[13] Corrigan A, Camelli F, Löhner R, Wallin J. (2009), "Running unstructured grid based CFD solvers on modern graphics hardware", *19th AIAA Computational Fluid Dynamics*, San Antonio, TX, USA, AIAA Paper 2009-4001.

[14] Elsen E., LeGresley P. and Darve E. (2008), "Large calculation of the flow over a hypersonic vehicle using a GPU", *Journal of Computational Physics*, Vol. 227, pp. 10148-10161.

[15] Cohen J. and Molemaker M. (2009), "A fast double precision CFD code using CUDA", *Proceedings of Parallel CFD 2009*, Moffett Field, California.

[16] Kampolis I., Trompoukis X., Asouti V. and Giannakoglou K. (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.

[17] Asouti V., Trompoukis X., Kampolis I. and Giannakoglou K. (2011), "Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units", *International Journal for Numerical Methods in Fluids,* to appear.

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

[19] Papadimitriou D., Giannakoglou K. (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.
[20] Roe, P.L. (1981), "Approximate Riemann solvers, parameter vectors, and difference schemes", *Journal of Computational Physics*, Vol. 43(2), pp. 357–372.
[21] Zhao Y., Forhad A. (2003) "A general method for simulation of fluid flows with moving and compliant boundaries on unstructured grids", *Computer Methods in applied Mechanics and Engineering*, Vol. 192, pp. 4439–4466.