A GPU-accelerated compressible RANS solver for Fluid-Structure Interaction simulations in turbomachinery

Andrea Gadda[1]*, Luca Mangani[2], Giulio Romanelli[3], Paolo Mantegazza[4], Ernesto Casartelli[5]

ISROMAC 2016

International Symposium on Transport Phenomena and Dynamics of Rotating Machinery

Hawaii, Honolulu

April 10-15, 2016

Abstract

The typical approach in Computational Fluid Dynamics (CFD) relies on the negligible structural deformation due to aerodynamic loads. However when geometry optimization for performance and power efficiency is the ultimate goal, an accurate Fluid-Structure Interaction (FSI) analysis could improve results accuracy. This is particularly important for turbomachinery applications, where usually only centrifugal and thermal effects are considered to deform the original geometry, through a FEM solver, before computing the pure aerodynamic solution. Even if turbomachinery blades are less susceptible to be deformed by aerodynamic loads than other aeronautical components (e.g. helicopter blades, entire planes) trim solutions could lead to better results. Alongside with results accuracy, computational efficiency is mandatory and today GPUs can be exploited to accelerate simulations. In this article we present and describe a GPU-accelerated FSI solver for turbomachinery applications. The solver is validated with a typical literature case.

Keywords

Fluid-Structure Interaction — GPU computing — Turbomachinery

[1] *Department of Aerospace Sciences and Technologies, Politecnico di Milano, Milano, Italy*

[2] *Department of Fluid Mechanics and Hydraulic Machines, Lucerne University of Applied Sciences and Arts, Luzern, Switzerland*

[3] *Department of Aerospace Sciences and Technologies, Politecnico di Milano, Milano, Italy*

[4] *Department of Aerospace Sciences and Technologies, Politecnico di Milano, Milano, Italy*

[5] *Department of Fluid Mechanics and Hydraulic Machines, Lucerne University of Applied Sciences and Arts, Luzern, Switzerland*

*__Corresponding author__: andrea.gadda@polimi.it

INTRODUCTION

When dealing with aerodynamic components, especially in the turbomachinery field, the usual approach adopted with CFD relies on the negligible structural deformation due to aerodynamic loads. However, if performance and power efficiency are the ultimate goals (e.g. propulsion), a correct prediction of the structure-flow interaction could improve results accuracy. This is particularly important in a typical geometry optimization loop, where hundreds of slightly different geometries have to be analyzed. Aerolastic simulations, i.e trim and flutter analysis, are nowadays a fundamental step in aircrafts [1] and helicopters blades design. In fact, with these aerodynamic components, tip displacements can be easily appreciated as they usually are multiple of the blade thickness. With turbomachinery blades however, deformations due to aerodynamic loads are usually less noticeable. As an example, in our trim simulations of the NASA's Rotor 67 test case, tip displacements are smaller than the blade thickness. Nonetheless, an accurate prediction of this kind of behavior is particularly important due to the presence of the shroud wall near the blade tip. Turbomachinery trim and flutter predictions still represent a challenge due to complex phenomena like rotor-stator interactions, separations and shock waves. The usual time-linearized, frequency-domain strategies can be inadequate when this kind of strong non-linear phenomena occur in the flow, making necessary time-domain simulations [2, 3] or the harmonic balance technique [4]. The scenario is further complicated by the periodic and multi-stage nature typical of axial compressors and turbines. In flutter and aerodynamic damping simulations, a fundamental role is also played by the Inter-Blade Phase Angle (IBPA) [5] when solution spatial periodicity is not related to a single blade anymore. However, in order to save computational time by reducing the computational domain to a single passage, as usually done with periodic boundary conditions, time-delayed boundary conditions [6] can be used.

Besides unsteady analysis, another important aspect, not yet adequately investigated, is represented by the trim analysis, which is fundamental for an accurate steady analysis that aims to consider static blade elasticity for the performance evaluation of turbomachinery. Trim analysis can be used to provide more accurate initial conditions for unsteady simulations or to improve the characteristic curve computations.

With the aim of an optimization process, another requirement of totally different nature arises: computational efficiency. This problem is typically addressed by implementing an efficient implicit formulation [7, 8] or by accelerating explicit formulations with multiple combined convergence acceleration techniques [9]. Nowadays, GPUs provide a cheap and relatively easy way to obtain great SIMD (Single Instruction Multiple Data) floating point performance at the cost of a limited amount of available memory with respect to the usual amount of system RAM [10, 9, 11]. Thus, the idea is to exploit GPUs architecture to accelerate an explicit FSI solver, keeping a low memory usage. Furthermore, the structural FEM model is reduced to a modal representation [12, 1] in order obtain both an accurate and efficient FSI formulation.

The purpose of this article is to describe the architecture

and the validation of a GPU-accelerated Fluid-Structure Interaction (FSI) solver for compressible viscous (using RANS models) flows. A pioneer work describing the impact of static deformation on the turbomachinery performances will be provided. In particular the well known NASA's Rotor 67 case [13, 14] is chosen for the trim analysis. This is a typical axial compressor rotor blade. Effects of trimmed solutions on the most important integral quantities (i.e. massflow, characteristic curves) are investigated and a comparison with pure aerodynamic results is provided. Numerical results are also compared with two sets of experimental data available in literature. The GPU solver is written in OpenCL in order to be compatible with both AMD and NVIDIA GPUs. This choice allows the solver to be natively executed also on CPUs, without source code modifications, improving the solver hardware compatibility and allowing to easily perform comparisons with multi-core CPUs. The proposed approach is validated with typical industrial cases.

1. METHODS

1.1 Aerodynamic numerical formulations

The numerical aerodynamic formulations are chosen with the aim to obtain a computationally efficient general-purpose compressible RANS solver. The purpose of this section is to introduce the aerodynamic numerical formulations implemented in the solver. For a better description of the adopted formulations, the reader is referred to [9].

The Reynolds Averaged Navier-Stokes (RANS) equations governing the dynamics of a compressible, viscous and conductive fluid are written within an Arbitrary Lagrangian Eulerian (ALE) framework in integral conservative form as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathcal{V}} \mathbf{U}\,\mathrm{d}\mathcal{V} + \oint_{\mathcal{S}} \left[\mathbf{f}(\mathbf{U}) - \mathbf{g}(\mathbf{U}) - \mathbf{U}\,\mathbf{w}\right]\cdot\mathbf{n}\,\mathrm{d}\mathcal{S} = \int_{\mathcal{V}} \mathbf{h}(\mathbf{U})\,\mathrm{d}\mathcal{V} \tag{1}$$

where $\mathcal{V}(t)$ is the time-varying spatial domain delimited by the boundary $\mathcal{S}(t)$ with normal unit vector $\mathbf{n}(\mathbf{x}, t)$. In the special case of local velocity of the moving boundaries $\mathbf{w}(\mathbf{x}, t)$ equal to zero, the classical Eulerian description of the continuum is recovered.

The arrays of the conservative variables $\mathbf{U}(\mathbf{x}, t)$, convective fluxes $\mathbf{f}(\mathbf{U})$ and viscous fluxes $\mathbf{g}(\mathbf{U})$ are defined as follows:

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho\,\mathbf{u} \\ \rho\,E^t \end{Bmatrix} \quad \mathbf{f} = \begin{Bmatrix} \rho\,\mathbf{u} \\ \rho\,\mathbf{u}\,\mathbf{u}^{\mathrm{T}} + p\,\mathbf{I} \\ \rho\,E^t\,\mathbf{u} + p\,\mathbf{u} \end{Bmatrix} \quad \mathbf{g} = \begin{Bmatrix} 0 \\ \tau \\ \tau\cdot\mathbf{u} + \mathbf{q} \end{Bmatrix} \tag{2}$$

where $\rho(\mathbf{x}, t)$, $\mathbf{u}(\mathbf{x}, t)$ and $E^t(\mathbf{x}, t)$ are the density, the velocity and the total specific energy. $p(\mathbf{x}, t)$, $\tau(\mathbf{x}, t)$ and $\mathbf{q}(\mathbf{x}, t)$ are the pressure, the viscous stresses tensor and the power exchanged by conduction. The problem is closed when the equation of state for the pressure (Polytropic Ideal Gas model), the constitutive equations for the viscous stresses tensor (Newtonian fluid model) and the power exchanged by conduction (Fourier model) are specified. The source term $\mathbf{h}(\mathbf{U})$ allows for modeling gravitational forces, Coriolis acceleration, porous media.

The source term, together with the ALE framework, is also used when dealing with Multiple Reference of Frame (MRF) simulations, such as for multi-stage compressors and turbines [15]. Given Ω as the rotor row angular velocity, by enforcing the ALE velocity as $\mathbf{w} = \Omega \times \mathbf{r}$ and the source term as $\mathbf{h}(\mathbf{U}) = \{0, -\Omega \times \rho\mathbf{u}, 0\}^T$ it is possible to solve the Navier-Stokes equations (1) for rotor cases without actually moving the mesh.

The solver is capable to handle hybrid unstructured meshes, i.e. meshes where different cells could have different number of faces and where there are no simple index-based strategies to identify cell neighbors. This feature offers significant advantages in terms of handling complex geometries, thanks to numerous unstructured mesh generation tools available. However, this comes at the price of a reduced efficiency in memory access patterns, especially for GPU executions. Therefore the code must be designed to maximize coalesced memory access and minimize branch divergence [10].

Within an unstructured grid framework, let us denote with $\Gamma_i = \{\Gamma_{i1}, \Gamma_{i2}, \ldots, \Gamma_{in}\}$ the set of n interfaces delimiting the i-th finite volume ΔV_i, the FV cell-centered discretized form of the RANS equations corresponds to the following system of ordinary differential equations:

$$\frac{\mathrm{d}(\mathbf{U}_i \Delta V_i)}{\mathrm{d}t} = \mathbf{Q}_i = -\sum_{j=1}^{n} |\Gamma_{ij}|\,(\mathbf{F}_{ij} - \mathbf{G}_{ij}) + \mathbf{H}_i\,\Delta V_i \quad \forall\, i, \tag{3}$$

where $\mathbf{U}_i(t)$ is the vector of unknown variables averaged on the i-th cell ΔV_i and collocated in its center. $\mathbf{F}_{ij}(\mathbf{U}_i, \mathbf{U}_j, \mathbf{W}_{ij})$ and $\mathbf{G}_{ij}(\mathbf{U}_i, \mathbf{U}_j)$ are the convective and viscous numerical fluxes through interface Γ_{ij}. Finally $\mathbf{H}_i(\mathbf{U}_i)$ is the source term evaluated in i-th cell center.

Concerning the space discretization operators, in order to obtain a monotone but sharp solution near shock waves and contemporarily to achieve a second order accuracy in space in smooth flow regions, a combination of the monotone first order accurate Roe's upwind flux and a second order accurate centered flux is used. The blending of the two is automatically controlled by means of the flux limiter by van Leer, requiring the numerical solution on the extended cells i^* and j^* to be available [16]. This operation is performed using a connectivity data structure built in the pre-processing stage. As alternative options AUSM+ [17] and CUSP [18] fluxes are implemented. In this case second order accuracy is achieved by means of min-mod limited solution reconstruction.

The viscous numerical fluxes are assembled using a cell-limited Gauss algorithm for the evaluation of the velocity and temperature gradients. As alternative option a thin layer approximation is also implemented, which is much more efficient since it does not require to assembly the gradient fields but has a limited validity interval. Turbulence is modeled using a 1-equation Spalart-Allmaras model in which the additional transport-diffusion-reaction equation is solved for the new variable $\tilde{\nu}$ [19]. Automatic wall treatment is implemented to

handle meshes at all nondimensional wall distance (y^+), raging from the viscous sublayer to the log layer, thanks to a blended formulation [20, 21] between the solutions provided by the two layers. Alternatively, a simple algebraic Mixing Length (ML) model is also implemented, where the kinematic viscosity $\nu^t = l^2 B$ is a function of the strain rate tensor magnitude B and the length $l = k \min(y, \Delta)$ [22].

Besides the usual inflow, outflow and wall boundary conditions other types required for turbomachinery applications are implemented [9], such as Mixing Plane (MP) [23, 24], AMI periodic boundary conditions, total and mass-flow inlet boundary conditions.

Since the explicit nature of the solver, convergence acceleration techniques like Multi-Grid (MG), Residual Smoothing (RS) and Local Time Stepping (LTS) [25, 26, 9] must be used to damp residuals and to obtain convergence rates that can be comparable with the ones provided by implicit solvers. LTS allows different cells to advance with different time-steps based on their geometrical and flow properties, accelerating in particular the convergence of the bigger cells far from the wall. A simplified MG strategy is implemented, an adaption to unstructured polyhedral grids of the formulation originally proposed by Denton and better explained in [9]. Basically the original mesh is agglomerated in multiple levels of coarser meshes. Transients are dispersed more quickly on the coarser levels, as larger time steps are allowed, while retaining the spacial accuracy of the finest [25, 26, 9]. These techniques accelerate the convergence of steady solutions, but can't be directly used for unsteady simulations since the time (often called "pseudo-time" at this point) has no longer any physical meaning. The problem of performing unsteady simulations by maintaining the convergence boost provided by these techniques is addressed using the Dual Time Stepping (DTS) formulation [27].

1.2 FSI numerical formulations

The following subsections are aimed to introduce the numerical formulations that, alongside with the previously introduced CFD strategies, allow the solver to perform FSI simulations. The reader is also referred to [1, 12] for a more detailed description of the implemented strategies.

1.2.1 Modal reduction of the structural model

With the aim of obtaining a solver capable of flutter and aerodynamic damping computations, the structural model is reduced using the modal approach. Neglecting the structural damping, the free response equation of the structural system can be represented in the matrix notation as follows:

$$\left[\bar{M}\right] \{\ddot{u_s}(t)\} + \left[\bar{K}\right] \{u_s(t)\} = \{0\} \tag{4}$$

where $\{u_s(t)\}$ is the vector of nodal displacements, $\left[\bar{M}\right]$ is the mass matrix in structural nodal coordinates, $\left[\bar{K}\right]$ is the stiffness matrix in structural nodal coordinates. From this equation we can obtain modal shapes and frequencies, basically eigenvectors and eigenvalues of the free deformable system. We can represent the transformation from modal coordinates $\{q(t)\}$ to nodal coordinates as follows:

$$\{u_s(t)\} = [U] \{q(t)\} \tag{5}$$

Matrix $[U]$ represents modal shapes and its transposed is used to obtain modal displacements from structural nodal displacements:

$$\{q(t)\} = [U]^T \{u_s(t)\} \tag{6}$$

This way, using the transformation matrix $[U]$ it is possible to rewrite equation 4 in modal coordinates:

$$[M] \{\ddot{q}(t)\} + [K] \{q(t)\} = \{0\} \tag{7}$$

where $[M]$ is the generalized mass matrix and $[K]$ is the generalized stiffness matrix, the matrices in modal coordinates. Modes can be normalized in different ways, e.g. to obtain a unitary generalized mass, in which case the generalized mass matrix becomes the identity. The main advantage of the modal representation of the structural behavior consists in the possibility to build an accurate and efficient reduced model using just few modes. An important aspect is that when computing modes of rotating components like rotor blades, centrifugal effects must be taken into account. Basically those effects are translated into stiffness contributions by the FEM solver.

When the structural system is loaded with aerodynamic forces, equation 7 can be rewritten adding the generalized aerodynamic forces vector to the right hand side:

$$[M] \{\ddot{q}(t)\} + [K] \{q(t)\} = \{Q(t)\} \tag{8}$$

Matrix $[U]^T$ can be used to obtain generalized aerodynamic forces from aerodynamic forces known on structural nodes:

$$\{Q(t)\} = [U]^T \{f_s(t)\} \tag{9}$$

Thus, thanks to relations 5, 9 and equation 8, knowing aerodynamic forces on structural nodes allows the solver to compute displacements on structural nodes.

1.2.2 Aeroelastic Interface

Here we briefly describe the adopted aeroelastic interface. For a more detailed description, the reader is referred to [12]. Usually aerodynamic and structural grids are independent. This means that aerodynamic nodes $\{x_a\}$ are usually different from structural nodes $\{x_s\}$. Thus we need a way to project aerodynamic loads from aerodynamic nodes ($\{f_a(t)\}$) to structural nodes ($\{f_s(t)\}$) in order to compute structural displacements, and a way to project structural displacements from structural nodes $\{u_s(t)\}$ to aerodynamic nodes $\{u_a(t)\}$ in order to provide the FSI solver the updated geometry from which re-compute the flow.

The aeroelastic interface can be represented by means of a linear operator $[\mathcal{I}]$:

$$\{u_a(t)\} = [\mathcal{I}] \{u_s(t)\} \tag{10}$$

As explained in [12], it is fundamental for the aeroelastic interface the conservation of momentum and energy exchanged between the aerodynamic and the structural subsystems. This means that the virtual work made by the aerodynamic forces $\{f_a(t)\}$ for the structural displacements interpolated on the aerodynamic nodes $\{u_a(t)\}$ must be equivalent to the virtual work made by the aerodynamic forces interpolated on structural nodes $\{f_s(t)\}$ for the structural displacements $\{u_s(t)\}$. The consequence of this requirement is that the transpose of the aeroelastic interface operator can be used to obtain the aerodynamic forces on structural nodes from aerodynamic forces on aerodynamic nodes:

$$\{f_s(t)\} = [\mathcal{I}]^T \{f_a(t)\} \tag{11}$$

The interface matrix can be computed using different strategies. The approach here used consists of an interpolation scheme based on a Moving Least Squares (MLS) technique and the use of Radial Basis Functions (RBF). When using RBF, the influence between the structural and aerodynamic nodes is weighted using functions that depends only by their distance:

$$\phi(\mathbf{x_s}, \mathbf{x_a}) = \phi(||\mathbf{x_s} - \mathbf{x_a}||) \tag{12}$$

where $\mathbf{x_s}$ is a structural point and $\mathbf{x_a}$ is an aerodynamic point. Different kind of functions can be used, allowing to adjust the smoothness of the interpolation.

It must be noted that the aeroelastic interface binds only the aerodynamic and structural nodes that reside on the surface of the deformable geometry. Thus from the displacements of the structural nodes it is possible to compute only the displacements of the aerodynamic nodes on the moving walls. Furthermore, the aeroelastic interface is computed only once and can be used for all the subsequent FSI simulations like trim, aerodynamic damping and flutter analysis.

The aeroelastic interface matrix is stored and reused each time the aerodynamic loads and the structural displacements have to be exchanged between the aerodynamic and structural meshes. Storing this matrix on the GPU memory would lead to an unacceptable memory overhead. Thus, the interface matrix is stored in the system RAM and computations are performed by the CPU. Although the GPU execution is temporarily stopped to wait the CPU in this phase, the procedure is still computationally efficient. In fact the data exchanged between the CPU and the GPU is limited to the aerodynamic loads and structural displacements of wall nodes. Furthermore the computational time required by this procedure is basically negligible with respect to the time required by the GPU for the convergence of the aerodynamic solution over the updated aerodynamic mesh geometry, thanks also to the modal representation of the structural model that reduces the total number of structural d.o.f. This is due to the fact that few thousands aerodynamic iterations are needed to reach aerodynamic residuals convergence between two geometry updates. This is true both for steady trim analysis and unsteady analysis using the DTS technique.

1.2.3 Internal nodes displacements

Once the aerodynamic wall nodes displacements are obtained, we need to compute the displacements of the aerodynamic mesh internal nodes of the fluid domain. This problem can be addressed using different strategies such as laplacian smoothing. This phase during a typical FSI simulation could easily need to be repeated hundreds or thousands of times, thus a computational efficient algorithm is required. It must be also kept in mind the reduced amount of memory available on GPUs: an explicit algorithm is preferred over a strategy that involves the solution of a system of equations. In particular an Inverse Distance Weighting (IDW) algorithm is adopted for this purpose.

Basically, the displacements $\left\{u_a^k\right\}$ of an internal aerodynamic node k are computed weighting the displacements of wall nodes $\left\{u_a^i\right\}$ through a function of the inverse of the distance between node k and wall node i:

$$\left\{u_a^k\right\} = \frac{\sum_{i=0}^{N_{wall}} W_{k-i} \ \left\{u_a^i\right\}}{\sum_{i=0}^{N_{wall}} W_{k-i}} \tag{13}$$

W_{k-i} is the weighting function and can be expressed as follows:

$$W_{k-i} = \frac{1}{|\mathbf{x}_k - \mathbf{x}_i|^p} \tag{14}$$

The exponent n can be changed to adjust the smoothness of the results. Usually $p = 2$ or $p = 3$ provides good results.

IDW weights are constants during the FSI simulation, so they could be stored in a matrix that can be accessed every time the aerodynamic mesh has to be updated. However, storing this matrix would require a large memory overhead. Since the limited amount of GPU memory, IDW weights are recomputed every time they are needed by GPU cores instead of being stored. This would lead to large computational overhead in a typical CPU architecture composed by few cores. However this is not a problem in a typical GPU architecture where several floating point computations can be performed while waiting for data to be recovered from GPU memory. Furthermore, the computational time spent to update the mesh with this algorithm is basically negligible with respect to the time required by the explicit aerodynamic iterations to reach convergence over the updated mesh. This is true both for trim simulations and FSI simulation where the mesh deformation is enforced (aerodynamic damping and flutter analysis).

After this phase, the aerodynamic mesh is changed. The connectivity between elements is preserved, however mesh metrics have to be recomputed from the new positions of the aerodynamic points. This can be efficiently done by the GPU since recomputing the metrics of millions of cells can be concurrently done by the thousands of available GPU cores.

1.3 Trim analysis

The aim of this section is to briefly describe the strategy used to perform trim analyses. The FSI algorithms and strategies previously introduced are combined together to perform this

numerical simulation. The main differences between a trim analysis and an aerodynamic damping or flutter analysis are basically two. First of all, a trim analysis is basically a steady analysis while aerodynamic damping and flutter analyses require unsteady solutions. Furthermore, in aerodynamic damping and flutter analyses the structural displacements given by modal shapes are used to enforce the deformation on the structure. Conversely, in trim analysis, aerodynamic loads are responsible for the structural deformation, thus the steady solution of system 8 is required. Moreover, trim analysis is performed using an iterative procedure were aerodynamic loads are used to deform the structure. The new obtained shape is then used to recompute the aerodynamic solution. The procedure is repeated until modal and aerodynamic residuals reach convergence.

Before starting the trim analysis two pre-processing steps are required. In particular the modal representation of the structural model of the deformable geometry is needed. This is done only once before any FSI simulation, i.e. trim, aerodynamic damping or flutter analysis. Modal shapes alongside with generalized masses, stiffness and frequencies can be obtained using several available commercial and free FEM solvers like NASTRAN or Code_Aster [28]. Different kind of FEM elements can be used, e.g. beam, shell/plate, solid elements, in a trade-off between accuracy and computational efficiency. It is worth to remind that since we are studying rotating blades, centrifugal effects must be taken into account in the modal analysis. FEM solvers can easily account for centrifugal effects as stiffness contributions. Another fundamental step is performing the computation of the aeroelastic interface matrix. When dealing with several thousands of aerodynamic and structural wall nodes, the assembly of this matrix can be a computationally intensive job, requiring minutes on a modern multi-core CPU. However, it is needed to compute the aeroelastic interface matrix only once, as it can be saved on disk and re-loaded every successive FSI simulation.

After the modal representation of the structural behavior and the aeroelastic interface matrix computation, it is possible to start the trim analysis for the entire rotor characteristic curve. First of all the pure aerodynamic solution at choking condition is computed, providing a guess solution to start the entire trim analysis. This is done using the explicit time-stepping procedure with the help of convergence acceleration techniques since we are searching a steady solution. At this point, aerodynamic loads over the aerodynamic mesh wall boundary points $\{f_a\}$ are obtained. Using relation 11, thus using the aeroelastic interface matrix, aerodynamic loads are projected over the structural mesh wall points of the deformable blade in order to obtain $\{f_s\}$. Since the structural model is represented by modal properties, generalized aerodynamic forces have to be computed. Relation 9 allows the solver to obtain forces in modal coordinates $\{Q\}$ from forces in structural nodal coordinates $\{f_s\}$. Now it is possible to compute the response, in term of modal displacements $\{q\}$, of the structure due to aerodynamic loads. Basically we need the steady version of the forced system 8:

$$[K]\{q\} = \{Q\} \tag{15}$$

Since usually just few modes are sufficient for an accurate representation of the structural behavior, the solution of system 15 requires very small computational times. Once modal displacements $\{q\}$ are obtained, in the next step the structural nodal displacements $\{u_s\}$ are computed using relation 5. Using again the aeroelastic interface matrix and relation 10, aerodynamic mesh wall nodes displacements $\{u_a\}$ are recovered from structural mesh wall nodes displacements $\{u_s\}$. Now it is possible to compute the aerodynamic mesh internal nodes displacements using the IDW strategy and recompute the aerodynamic mesh metrics. With the updated aerodynamic mesh a new steady solution is computed. Convergence is reached when aerodynamic residuals, generalized aerodynamic loads and modal displacements residuals are under a user-specified tolerance.

For the Rotor 67 test case about 500 explicit aerodynamic iterations are performed between two mesh updates and about 15-20 mesh updates are required for the aeroelastic convergence of one characteristic curve point. It must be underlined that thanks to the efficient IDW algorithm and the reduced amount of data transferred between the CPU and the GPU at each mesh update, the computational time required by the mesh update is basically negligible with respect to the time required by explicit aerodynamic iterations.

1.4 Hardware and software aspects

Since the solver is written in OpenCL, its source code is composed by two sets of files: one for the "host" and one for the "device". The host is basically the CPU that starts the solver and enqueues work for the device. The host code is written in standard C/C++ language. The device is the GPU or the CPU that is actually used to perform the CFD/FSI computations. The device code is composed by functions called "kernels" written in OpenCL C, a language similar to the standard C99 but with some restrictions. Functions and types of the OpenCL API are used in the host source code in order to organize the work (kernels executions) that is sent to the device and to exchange "buffers" (basically memory arrays) between the host and the device. Most CPU and GPU vendors, e.g Intel, AMD, NVIDIA, have their own OpenCL-compliant implementations, allowing the solver to be executed natively on a wide range of devices [11, 9]. During kernel executions, the whole work is split in chunks, called "work-items", that are executed in parallel by GPU or CPU cores. Since nowadays GPUs have thousands of cores, their architectures are well suited for the data parallelism where the same operation have to be performed on a large dataset. This is basically what is needed for an explicit solver. It must be noted however, that if different "work-items" have to perform different computations (branch divergence), a performance loss is inevitable with GPUs. Thus, device code must be opportunely optimized for the GPU architecture. Another typical aspect of GPUs is the memory optimized for sequential access. Again, kernels must

be optimized to reduce performance loss given by a typical hybrid unstructured mesh addressing. CPUs, nonetheless, are less susceptible to this kind of problems thanks to their MIMD (Multiple Instructions Multiple Data) architectures and bigger caches. Furthermore, GPUs main limitation is the reduced amount of "global memory", basically the amount of memory available to store buffers. A continuous data flow between the CPU and the GPU during the simulation would not be a good strategy since the PCI-Express bus that connects them would be a bottleneck. With these restrictions in mind, the explicit formulation is the better choice since the matrix storage is avoided. Despite these types of GPU problems and limitations, a typical 400 USD gaming GPUs (e.g AMD 290X) can provide over 5 TFLOPS of single precision performance, an order of magnitude more than a CPU of the same price level (e.g. Intel i7 3930K). Since the main advantages in using GPUs to perform numerical computations can be achieved with single precision, all the equations in the solver are opportunely kept dimensionless to reduce numerical problems. Finally it is important to notice that with the ability to perform computations in single precision floating point format and having low memory requirements, the solver is compatible with both HPC (High Performance Computing) GPUs and cheap gaming GPUs. HPC GPUs typically are an order of magnitude more expensive than gaming GPUs but exhibit more memory, ECC memory feature and higher double precision performance. However, HPC GPUs usually have about the same single precision peak computational power as gaming GPUs. All the simulations performed in this article were carried out using cheap gaming GPUs and single precision. Results were also reproduced with double precision and using an ECC memory-compliant GPU (Tesla C1060) but appreciable differences were not found [9].

The solver uses the OpenFOAM framework to carry out pre-processing and post-processing phases. In particular mesh loading, metrics computations, domain decomposition for multi-GPU executions, AMI (Arbitrary Mesh Interface) weights and addressing computations, and monitoring relevant performance metrics (i.e. mass-flow, pressure ratio) are computed with the help of the OpenFOAM API. The device code that performs the actual CFD/FSI computation is however written from scratch with OpenCL API and OpenCL C. Thanks to the OpenFOAM framework and an the opportunely tuned GPU code, the solver is compatible and computationally efficient with hybrid unstructured meshes [9].

2. RESULTS AND DISCUSSION

2.1 NASA's Rotor 67 trim

Here we describe the results of the trim analysis of the well known NASA's Rotor 67 case. Rotor 67 represents a typical fan blade geometry, here modeled as an isolated rotor, for which two sets of experimental data are available (one taken in 1984 and one lately in 2004). For this transonic axial fan test case, a number of CFD performance predictions have been published [13, 14]. The fan angular velocity adopted for the tests is 16043 rpm.

For what concerns the aerodynamic mesh, the details for the hub and the blade are shown in figure 1. The adopted unstructured grid is composed by about $1.1 \cdot 10^6$ hexahedral cells and wall regions are discretized with an average $y^+ \simeq 0.5$. Respectively, total and average pressure boundary conditions are imposed at inlet and outlet. The adopted turbulence model for aerodynamic and aeroelastic simulations over the entire characteristic curves is Spalart-Allmaras. However, a comparison between Spalart-Allmaras and Mixing Length turbulence models is also provided to check the mass flow convergence at choking conditions.

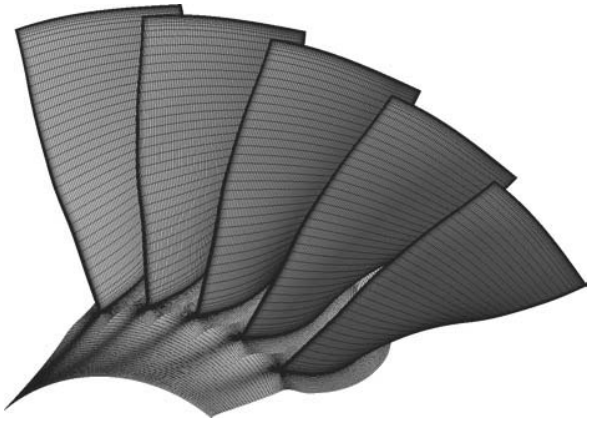

Figure 1. Detail of the computational mesh for Rotor 67 test case.

The structural mesh is instead composed by about $8 \cdot 10^4$ tetrahedrons (solid elements). Only the blade was modeled with FEM elements while the hub is supposed to be non deformable. The modal analysis was performed with the Code_Aster FEM solver. Just 3 modes are sufficient for an accurate representation for trim purposes. However, 4 modes are used in this simulation to check modal convergence. Using 5 or more modes leads basically to negligible differences in the results. Table 1 shows modes frequencies and describes their shapes.

Table 1. Rotor 67 modes frequencies and description.

Mode	Frequency (Hz)	Description
1	760.3	1st flexural
2	2174	2nd flexural
3	3146	1st torsional
4	4920	Mixed

With the aim of the characteristic curves computation, let us first discuss the choking point. The solver needs approximately $50 \cdot 10^3$ explicit iteration to reach pure aerodynamic steady solution for this particular point. Using an AMD 290X GPU the required time is about 25 min. Figures 2 show the residual and mass flow values during convergence at choking point using Mixing Length and Spalart-Allmaras turbulence models. From this solution, the trim analysis is started and another 10min of wall time are required by the same GPU

to reach aeroelastic convergence. Experimental value of the mass flow is 1.589 kg/s. The pure aerodynamic solution and the aeroelastic solution converges to a mass flow of 1.570 kg/s with an error of 1.76%.

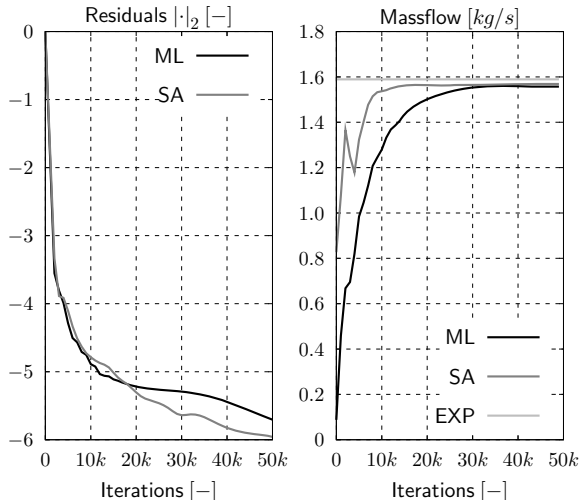

Figure 2. Residuals and mass-flow convergence history for Rotor 67 test case. Comparison between Mixing Legnth (ML), Spalart-Allmaras (SA) turbulence models and experimental data (EXP).

Figure 3 and 4 shows the displacement field and a detail of the blade edge of the deformed configuration obtained with the trim solution at the choking point. It is possible to see from the magnitude (colors from blue to red) that the deformation of the blade is mainly flexural, basically the shape of the first mode, with smaller influence from higher modes.

Figure 3. Blade displacements field at choking conditions, colors from blue to red represent displacements magnitude.

In figures 5 and 6 the efficiency and pressure ratio characteristic curves as a function of non-dimensional mass flow are compared with the two sets of experimental results. It is possible to see that both the aerodynamic and aeroelastic solutions are in good agreement with the experimental points.

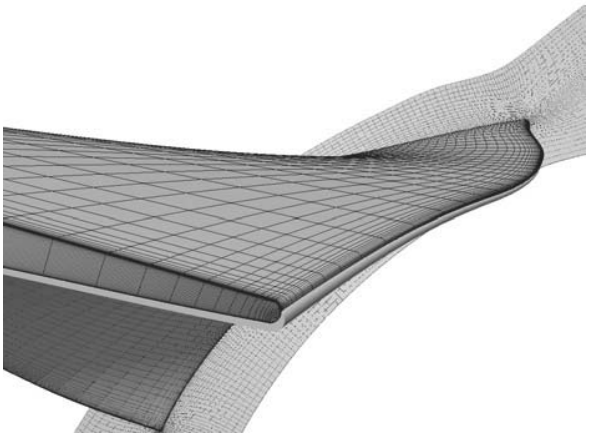

Figure 4. Detail of blade deformation. Grey surface: original non deformed blade. Surface with edges: deformed blade at choking.

In particular, both numerical curves are in better agreement with the 2004 data set for what concerns the peak efficiency point while are significantly lower than 1984 measurements. It must be stressed however the measurement uncertainty of the 1989 data set. It is possible to see that the trim solution basically provides the same results as the aerodynamic solution for the efficiency curve, and a slightly higher curve for the pressure ratio, especially near the efficiency peak region. Generally, however, the differences between the aerodynamic and the aeroelastic solution are small, due to the high stiffness of the blade.

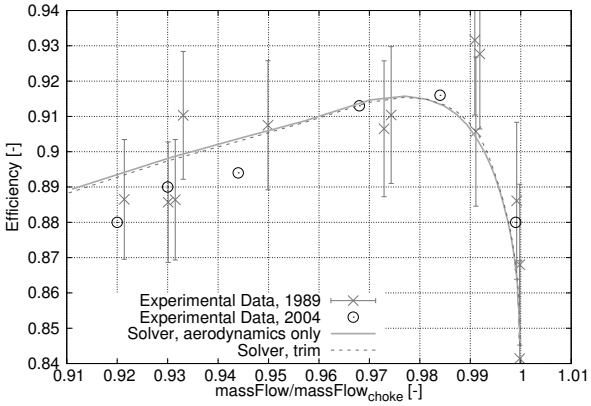

Figure 5. Efficiency as a function of mass flow for the Rotor 67 test case.

Figure 7 shows the displacements ($\{q\}$) of the four considered modes during convergence over the trimmed characteristic curve. It is possible to see that the curves are made by multiple steps, representing the convergence over different points of the characteristic curves, starting from the choking point and moving to the stall region. From modal displacements and modal stiffness it is possible to compute the elastic energy contribution of each mode. Basically the main contribution to the total elastic energy is given by the first mode, while the second, the third and the fourth mode contributes are at least

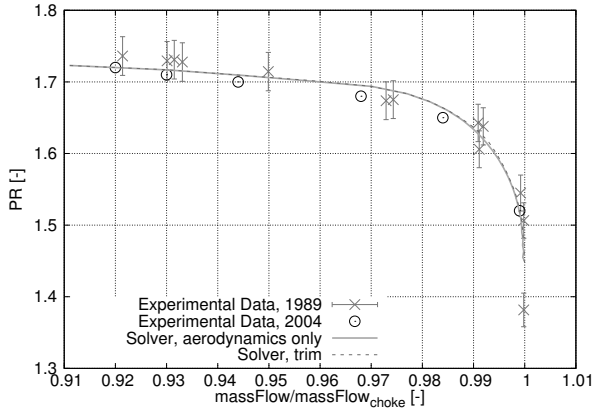

Figure 6. Pressure ratio as a function of mass flow for the Rotor 67 test case.

two order of magnitude lower. Using just 3 modes basically provides the same results. In particular, we can see that mode 4 displacements are basically 0 over the entire characteristic simulation. Using 5 or more modes with higher stiffness leads to further negligible contributions to the results.

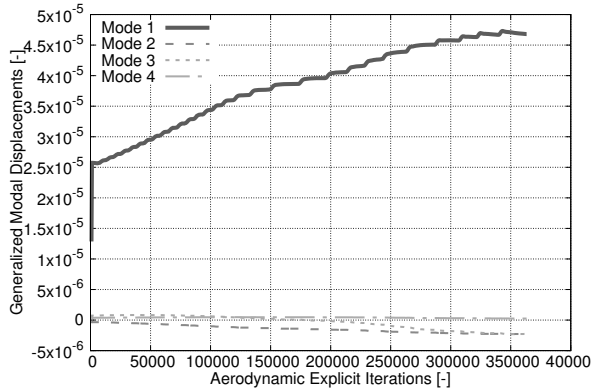

Figure 7. Modal displacements of the considered modes for the Rotor 67 trim analysis

To give a more complete view to the trim analysis using modal representation of the structural behavior, figure 8 shows generalized aerodynamic forces ($\{Q\}$) of the four considered modes.

Figures 9 and 10 shows the detail of efficiency and pressure ratio characteristic curves for the Rotor 67 near the peak efficiency point. It is possibile to see that when using different number of modes the curves are slightly different. However, using 3 or 4 modes the results are basically identical.

Figures 11 and 12 show the relative Mach number contours predicted at the peak-efficiency point. A comparison with experimental data is also provided at 10% and 30% span from the shroud wall. It is possible to see that both the pure aerodynamic and the aeroelastic solutions are in agreement with the measurements, in particular for what concerns the shock position in figure 11. The differences in shocks locations between the aerodynamic and trim solutions are very small,

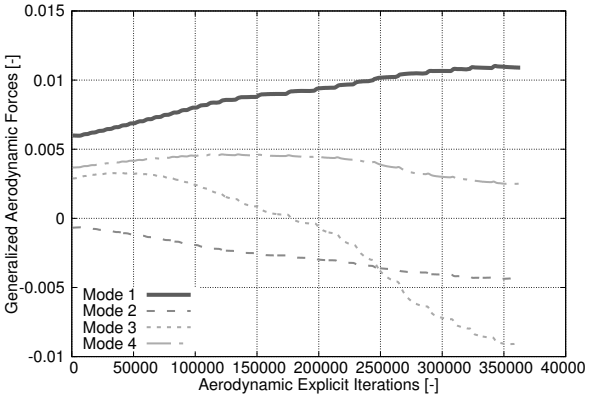

Figure 8. Generalized aerodynamic forces of the considered modes for the Rotor 67 trim analysis.

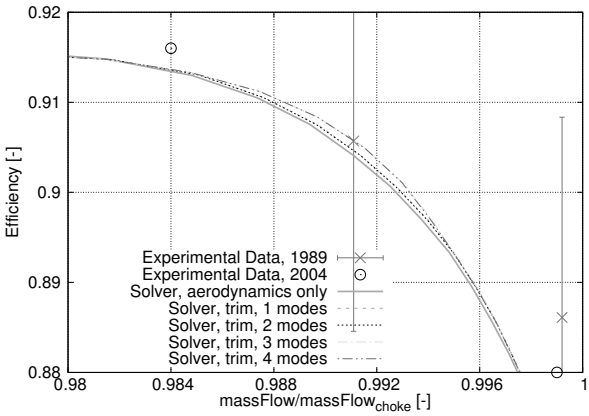

Figure 9. Efficiency as a function of mass flow for the Rotor 67 test case, detail near peak efficiency using different number of modes.

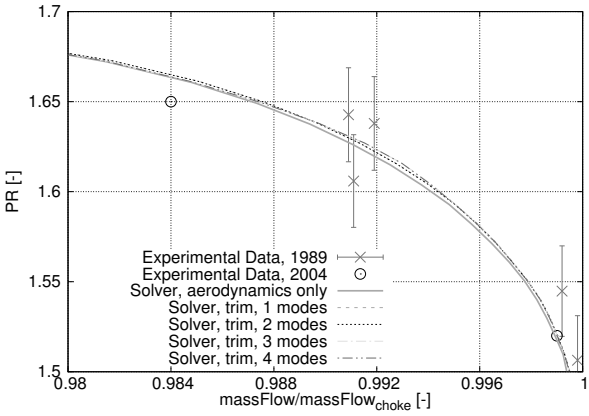

Figure 10. Pressure ratio as a function of mass flow for the Rotor 67 test case, detail near peak efficiency using different number of modes.

basically negligible.

In order to highlight the importance of a trim analysis during the design and optimization phases of a rotor blade let us try to see the effects given by changing the blade stiffness. Since the limited amount of space available we will not perform

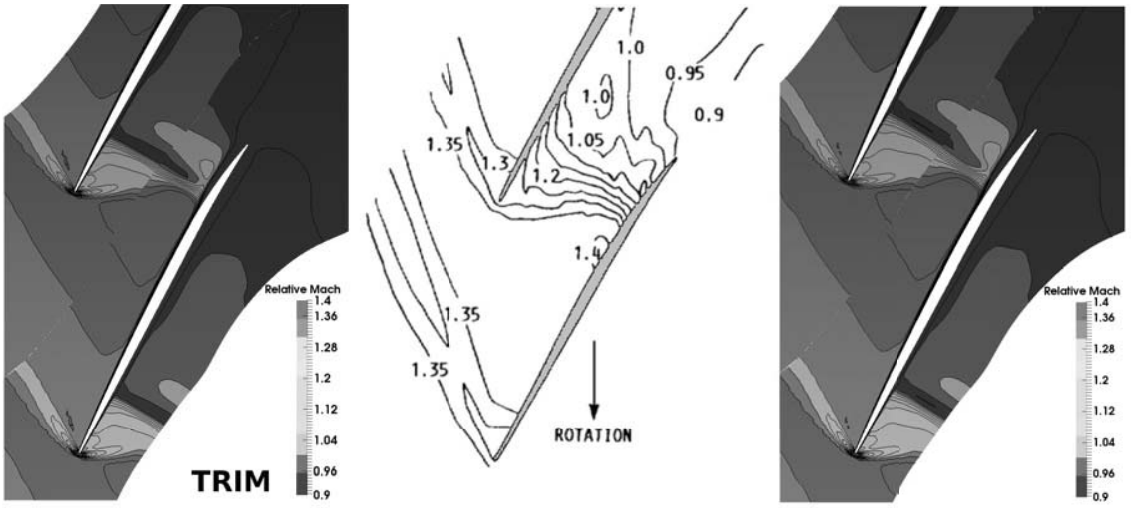

Figure 11. Comparison of relative Mach number contours with experiments at peak efficiency, 10% span from shroud. From left to right: numerical trim solution, experimental data, numerical aerodynamic solution.

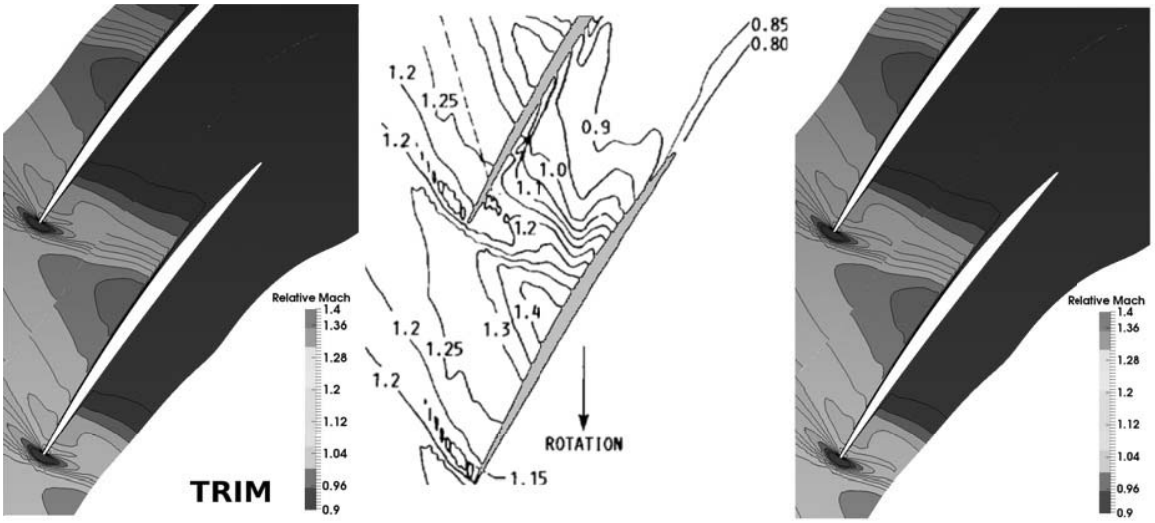

Figure 12. Comparison of relative Mach number contours predicted with experiments at peak efficiency, 30% span from shroud. From left to right: numerical trim solution, experimental data, numerical aerodynamic solution.

a detailed quantitative analysis but just a qualitative study of the changing in the blade tip displacements obtained through a stiffness reduction of the blade. As an example we can reduce the first mode (flexural shape) stiffness of about ten times and see the blade shape at choking conditions. Figure 13 shows the detail of the tip displacements. It is easy to see the differences from figure 4: the blade tip is now characterized by bigger tip displacements. It is straightforward that one of the possible aims of a rotor trim analysis would be an accurate study of the interaction between the blade tip and the shroud wall.

2.2 GPU acceleration

Here we briefly present the GPU speedup results for the Rotor 67 test case. For more detailed benchmarks regarding the GPU solver, the reader is referred to [9, 11]. All the convergence acceleration techniques (MG, RS, LTS) were active during these benchmarks in order to honestly represent a true simulation. The adopted cost metrics is the elapsed time per iteration per cell and the comparison involves CPUs in single thread execution, CPUs with multi-threading and GPUs. Thanks to OpenCL, the solver can be natively executed by several different CPUs and GPUs architectures without modifications. Benchmarks were carried out with single precision to fully exploit peak computational power provided by the adopted devices. Furthermore, when available, the compiler auto-vectorization is used to exploit SSE/AVX capabilities of the modern multi-core CPUs. Table 2 shows the CPUs used for the comparison, basically an entry level CPU (AMD Phenom II X4 840) and an high end CPU (Intel i7 3930k). Table 3 shows instead the GPUs used for the comparison, raging from

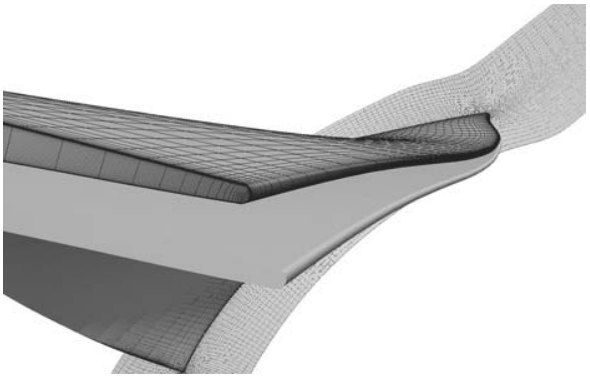

Figure 13. Mesh deformation at choking point using 1/10 the stiffness of the first mode.

an entry level gaming GPU (NVIDIA GTX 650) to an high end gaming GPU (AMD 290X). It is worth remembering that gaming GPUs are here preferred thanks to their high single precision performances provided at relatively low cost with respect to typical HPC GPUs like NVIDIA Tesla or AMD FirePro series.

Tables 2 and 3 also show prices and theoretical single-precision peak performance of the adopted devices for a more honest comparison. For the AMD CPU, however, data regarding the theoretical peak performances where not found on the official website (although it can be estimated to be about 30-50 GFLOPS).

Table 4 shows the results for the time per iteration per cells for each available device. From these results it is possible to build table 5. In this table, at each row, CPUs in single and multi-thread execution timings are taken as reference for the speedup value computation. As an example, on the last row, we can see that the AMD 290X is 13.06 times faster than the Intel i7 3930k CPU using all the 6 cores. Considering that this GPU costs about 400 USD and the CPU costs about 500 USD, it is clear the advantage to perform a simulation using the GPU. Even on a possible low-end workstation composed by the AMD Phenom II X4 840 CPU and the NVIDIA GTX 650 GPU we can see that using the GPU allows the solver to complete simulations about 16 times faster. Furthermore it is possible to see that the cheapest GPU (GTX 650) is about 3.5 times faster than the most expensive CPU (i7 3930k) used for these test, but costs only 1/5 of the price. Alongside with the price, it is possible to consider the TDP (Thermal Design Power), that gives an idea of the power consumption. For instance, the Intel CPU and the NVIDIA GTX 660 GPU have the same TDP but the GPU is about 6 times faster than the CPU. This also underlines the power consumption advantages of using GPUs for numerical simulations when the particular problem allows an efficient execution on these kinds of devices, a non secondary aspect in clusters and supercomputers.

3. CONCLUSIONS

In this paper we have described the architecture and implementation of a GPU-accelerated explicit density-based aeroelastic RANS solver for turbomachinery applications. The solver is based on the OpenFOAM framework but the code that actually performs CFD/FSI computations is written from scratch with OpenCL API and OpenCL C in order to be compatible with a wide range of multi-core CPUs and GPUs. The solver was validated using a well known turbomachinery case, the NASA's Rotor 67 axial fan. For this particular case two experimental data sets are available. A comparison between the pure aerodynamic solution and the trim solution for the entire characteristic curve was provided in order to asses the possible advantages given by simulating the interaction between the flow and the blade structure. However, as the results suggest, for this particular case the differences between the aerodynamic and aeroelastic solution are very small. This is basically due to the fact that with respect to other typical aeronautical cases (e.g. helicopter blades and planes) these fan blades are very stiff. Thus, blades displacements are very small in all the considered operating conditions and their effects on the flow are basically negligible.

Alongside with the aeroelastic investigation, a comparison between the time per iteration per cell given by different CPUs and GPUs was provided, highlighting the advantages given by GPU executions in term of wall time and costs.

REFERENCES

[1] G. Romanelli. *Computational Aeroservoelasticity of Free-Flying Deformable Aircraft*. PhD thesis, Politecnico di Milano, Italy, 2012.

[2] Yun Zheng and Yang Hui. Coupled fluid-structure flutter analysis of a transonic fan. *Chinese Journal of Aeronautics*, 24(3):258–264, 2011.

[3] Volker Carstens, Ralf Kemme, and Stefan Schmitt. Coupled simulation of flow-structure interaction in turbomachinery. *Aerospace Science and Technology*, 7(4):298–306, 2003.

[4] Earl H Dowell and Kenneth C Hall. Modeling of fluid-structure interaction. *Annual Review of Fluid Mechanics*, 33(1):445–490, 2001.

[5] N. Donini. Aeroelasticity of turbomachines linearized flutter analysis. Master's thesis, Politecnico di Milano, Italy, 2012.

[6] G Romanelli, L Mangani, and E Casartelli. Implementation of a cfd-based aeroelastic analysis toolbox for turbomachinery applications. In *ASME Turbo Expo 2014: Turbine Technical Conference and Exposition*, pages V02BT45A010–V02BT45A010. American Society of Mechanical Engineers, 2014.

[7] L. Mangani, M. Buchmayr, and M. Darwish. Implementation and Evaluation of a Fully Coupled Solver in Openfoam: Steady State Incompressible Turbulent Flows. *Numer. Heat Transfer B, in print*, 2014.

[8] L. Mangani, M. Buchmayr, and M. Darwish. Implementation and Evaluation of a Fully Coupled Solver in Openfoam: Steady State Incompressible Turbulent Flows

Table 2. CPU used for the benchmark.

Vendor	Model	Cores	SP Performance	Frequency	Price	TDP	OpenCL runtime
AMD	Phenom II X4 840	4	N/A	3.2GHz	~ 100 USD	95 W	AMD APP SDK
Intel	i7 3930k	6	~ 170 GFLOPS	3.2GHz	~ 500 USD	130 W	Intel OpenCL Runtime

Table 3. GPU used for the benchmark.

Vendor	Model	Global Memory	Cores	SP Performance	Frequency	Price	TDP	OpenCL runtime
NVIDIA	GTX 650	1 GB	384	~ 800 GFLOPS	928 MHz	~ 100 USD	110 W	CUDA SDK
NVIDIA	GTX 660	2 GB	960	~ 1800 GFLOPS	980 MHz	~ 200 USD	130 W	CUDA SDK
AMD	290X	4 GB	2816	~ 5600 GFLOPS	1040 MHz	~ 400 USD	290 W	AMD APP SDK

Table 4. Time per iteration per cell.

Phenom 1-core	Phenom 4-core	i7 1-core	i7 6-core	GTX 650	GTX 660	290X
5.99e-6	1.74e-6	1.82e-6	3.67e-7	1.04e-7	6.37e-8	2.81e-8

Table 5. Relative speedups.

Phenom 1-thread	Phenom 4-threads	i7 1-thread	i7 6-threads	GTX 650	GTX 660	290X
1.00x	3.44x	3.29x	16.32x	57.6x	94.04x	213.17x
0.29x	**1.00x**	0.96x	4.74x	16.26x	27.32x	61.92x
0.30x	1.04x	**1.00x**	4.96x	17.5x	28.57x	64.77x
0.06x	0.21x	0.20x	**1.00x**	3.53x	5.76x	13.06x

in Rotational Reference Frames. *Numer. Heat Transfer B, in print*, 2014.

[9] G Romanelli, L Mangani, E Casartelli, A Gadda, and M Favale. Implementation of explicit density-based unstructured cfd solver for turbomachinery applications on graphical processing units. In *ASME Turbo Expo 2015: Turbine Technical Conference and Exposition*, pages V02BT39A034–V02BT39A034. American Society of Mechanical Engineers, 2015.

[10] M. Favale and A. Gadda. A gpu parallelized two fields full potential formulation for real gases. Master's thesis, Politecnico di Milano, 2013.

[11] Luca Mangani, Giulio Romanelli, Andrea Gadda, and Ernesto Casartelli. Comparison of acceleration techniques on cfd open-source software for aerospace applications. In *22nd AIAA Computational Fluid Dynamics Conference*, page 3059, 2015.

[12] Giulio Romanelli, Michele Castellani, Paolo Mantegazza, and Sergio Ricci. Coupled csd/cfd non-linear aeroelastic trim of free-flying flexible aircraft. In *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2012.

[13] Andrea Arnone. Viscous analysis of three-dimensional rotor flow using a multigrid method. *Journal of turbomachinery*, 116(3):435–445, 1994.

[14] A. D. Grosvenor. Rans prediction of transonic compressive rotor performance near stall. *ASME Paper*, (GT2007-27691), 2007.

[15] Robert T Biedron and James L Thomas. Recent enhancements to the FUN3D flow solver for moving-mesh applications. *AIAA Paper*, 1360:2009, 2009.

[16] G. Romanelli, E. Serioli, and P. Mantegazza. A Free Approach to Modern Computational Aeroelasticity. *48-th AIAA Aerospace Sciences Meeting*, 2009.

[17] Meng-Sing Liou. A sequel to ausm: Ausm+. *Journal of computational Physics*, 129(2):364–382, 1996.

[18] Shigefumi Tatsumi, Luigi Martinelli, and Antony Jameson. Flux-limited schemes for the compressible navier-stokes equations. *AIAA journal*, 33(2):252–261, 1995.

[19] P. R Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA*, (92-0439), 1992.

[20] M Popovac and K Hanjalic. Compound wall treatment for RANS computation of complex turbulent flows/heat transfer. *Flow, turbulence, combustion*, 78(2), 2007.

[21] Georgi Kalitzin, Gorazd Medic, Gianluca Iaccarino, and Paul Durbin. Near-wall behavior of RANS turbulence

models and implications for wall functions. *Journal of Computational Physics*, 204(1):265–291, 2005.

[22] R. V. Chima, P. W. Giel, and R. J. Boyle. An algebraic turbulence model for three-dimensional viscous flows. *31-st Aerospace Sciences Meeting and Exhibit*, 1993.

[23] Lucian Hanimann, Luca Mangani, Ernesto Casartelli, Thomas Mokulys, and Sebastiano Mauri. Development of a novel mixing plane interface using a fully implicit averaging for stage analysis. *Journal of Turbomachinery*, 136(8), 2014.

[24] L.S. Dzung. Konsistente mittelwerte in der theorie der turbomaschinen fuer kompressible medien. *Brown-Boveri Mitteilungen*, 1971.

[25] Tobias Brandvik and Graham Pullan. An accelerated 3d navier–stokes solver for flows in turbomachines. *Journal of Turbomachinery*, 133(2), 2011.

[26] Christian Klostermeier. *Investigation into the capability of large eddy simulation for turbomachinery design*. PhD thesis, University of Cambridge, 2008.

[27] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*. Oxford University Press, 2001.

[28] Code aster website. `http://www.code-aster.org/`.