# Design-Optimization of a Compressor Blading on a GPU Cluster

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

**Abstract** This paper presents the design/optimization of turbomachinery blades using synchronous and asynchronous metamodel–assisted evolutionary algorithms on a GPU cluster. Asynchronous EAs overcome the synchronization barrier at the end of each generation and exploit better all available computational resources. Radial basis function networks are used as on–line trained surrogate evaluation models (metamodels) according to the inexact pre–evaluation (IPE) concept. With the exception of a few initial evaluations, which are based on the exact evaluation tool, each new candidate solution is approximately evaluated using local metamodels and only the most promising among them are, then, re-evaluated using the exact tool. Suggestions about the number of population members to be re–evaluated on the CFD tool, in the framework of the IPE scheme, are provided. The effect of using more than one GPUs to evaluate each candidate solution in the optimization turnaround time is discussed.

## 1 Introduction

Nowadays, evolutionary algorithms (EAs) are successfully applied to many scientific fields including engineering sciences, since they can handle single– or multi–objective, unconstrained or constrained optimization problems by accommodating any evaluation software as a black–box. EAs main disadvantage is related to the great number of evaluations required to reach the optimal solution(s). In engineering problems, based on evaluation software which is computationally demanding (for instance, a CFD code), this noticeably increases the optimization turnaround time.

Konstantinos T. Tsiakas, Xenofon S. Trompoukis, Varvara G. Asouti, Kyriakos C. Giannakoglou
National Technical University of Athens (NTUA), School of Mechanical Engineering, Parallel CFD & Optimization Unit, Lab. of Thermal Turbomachines, Iroon Polytechniou 9, Athens 15780, Greece, e-mail: {tsiakost,xeftro}@gmail.com, vasouti@mail.ntua.gr, kgianna@central.ntua.gr

To decrease the CPU cost and/or the turnaround time of an EA–based optimization, the concurrent evaluation of candidate solutions and/or the implementation of surrogate evaluation models (or metamodels) can be used. Generation–based EAs, to be referred as "synchronous" in this paper, usually implement the master–worker paradigm to concurrently evaluate the generation members. Each population member can optionally be evaluated on many processors, provided that a parallel evaluation software (such as a parallel CFD code, in CFD–based optimization) is available. By overcoming the notion of "generation", the so–called asynchronous EAs (AEAs) have been developed [1, 2]. AEAs may exploit the available computational resources better than a synchronous EA relying upon the master–worker paradigm.

On the other hand, metamodels are tools providing low–cost approximations to the results of costly problem–specific evaluation models. In this work, metamodel–assisted EAs (MAEAs) with on–line trained local metamodels (radial basis function - RBF networks) are used according to the Inexact Pre–Evaluation (IPE) scheme of the candidate solutions. IPE starts after running and archiving a number of individuals using exclusively the problem–specific evaluation model, in order to collect the minimum number of samples to be used to train the metamodels. In the synchronous EA, during the IPE phase [10, 9], all population members of each generation are pre–evaluated on surrogate models built in purpose and only a few top individuals undergo re–evaluation on the exact model. In the asynchronous EAs, [1, 2], instead of generating and evaluating a single new individual every time a CPU becomes idle, a number of trial solutions are generated and pre–evaluated on the metamodel. Then, the best among them, according to the metamodel, is exactly re–evaluated.

In aerodynamic optimization, additional gain is expected from the use of a GPU–enabled Navier–Stokes solver to perform the CFD–based evaluation. Many of the existing CFD solvers running on GPUs use structured grids [15, 4, 11] and thus profit of the aligned access to the GPU memory, leading to high speed–ups. Applications based on unstructured grids are still limited and usually based on cell–centered finite volumes [15]. The GPU–enabled software used in this work solves the Navier–Stokes equations on unstructured grids using the vertex–centered finite volume technique [8, 3, 14]. Though this is the most difficult case regarding GPU memory access, compared to the use of either structured or unstructured grids with cell–centered finite volumes, the optimization of memory access, are discussed in [3, 14], along with the use of mixed precision arithmetics [8] makes the code running 50 times faster on a single GPU than on a single CPU core.

In this paper, synchronous and asynchronous metamodel–assisted EAs (MAEAs and AMAEAs) are used for the design optimization of a compressor blading on a GPU–cluster which consists of four interconnected server blades, with 3 NVIDIA Tesla M2050 each. In this 12 GPU configuration, various parallelization schemes are investigated in order to minimize the optimization turnaround time. These include MAEAs and asynchronous MAEAs (AMAEAs) allowing up to 12 (as many as the available GPUs) concurrent evaluations. Regarding MAEAs, the impact of the number of individuals to be re–evaluated on the CFD model is investigated. An additional investigation of the effect of parallelizing the CFD software on many

GPUs in the optimization turnaround time is also carried out, for both the MAEA and AMAEA.

## 2 The Navier–Stokes Equations Solver – Implementation on many GPUs

A GPU–enabled Navier–Stokes solver [8, 3, 14] is used for steady 3D incompressible flows for the evaluation of candidate solutions. The GPU–solver may use GPUs associated with the same or different computational nodes.

### *2.1 The Navier–Stokes Equations*

The pseudo-compressibility approach, introduced by Chorin[5], is used to handle incompressible fluid flows. By introducing the artificial compressibility $\beta$, the mean flow equations with respect to the rotating frame of reference become

$$\mathbf{R}(\mathbf{W}) = \frac{\partial \mathbf{W}}{\partial t} + \frac{\partial \mathbf{F}_j^{inv}}{\partial x_j} - \frac{\partial \mathbf{F}_j^{vis}}{\partial x_j} - \mathbf{S} = 0 \tag{1}$$

where $\mathbf{W} = [p\ w_1\ w_2\ w_3]$ is the vector of the unknowns, $w_i, i{=}1,2,3$ are the relative velocity components and $p$ is the static pressure. $\mathbf{F^{inv}}$, $\mathbf{F^{vis}}$ are the inviscid and viscous fluxes respectively and $\mathbf{S}$ is the vector of source terms containing the Coriolis and centripetal forces,

$$\mathbf{F}_j^{inv} = \begin{bmatrix} \beta w_j \\ w_j w_1 + p\delta_{1j} \\ w_j w_2 + p\delta_{2j} \\ w_j w_3 + p\delta_{3j} \end{bmatrix}, \ \mathbf{F}_j^{vis} = \begin{bmatrix} 0 \\ \tau_{1j} \\ \tau_{2j} \\ \tau_{3j} \end{bmatrix}$$

$$\tau_{ij} = (\nu + \nu_t)\left(\frac{\partial w_i}{\partial x_j} + \frac{\partial w_j}{\partial x_i}\right)$$

$$S_i = 2\varepsilon_{kji} w_k \Omega_j + \varepsilon_{jkl}\varepsilon_{hli}\Omega_h\Omega_j x_k$$

$\Omega_i, i = 1,2,3$ are the components of the angular velocity vectors. The mean–flow equations are coupled with the one–equation low–Reynolds number Spalart–Allmaras [13] turbulence model. The viscosity coefficient is given by $\nu_t = \tilde{\nu} f_{v_1}$, where $\tilde{\nu}$ is the solution variable in the state turbulence equation, $R_{\tilde{\nu}} = 0$, where

$$R_{\tilde{\nu}} = \frac{\partial(w_i\tilde{\nu})}{\partial x_i} - \frac{\partial}{\partial x_i}\left[\left(\nu+\frac{\tilde{\nu}}{\sigma}\right)\frac{\partial\tilde{\nu}}{\partial x_i}\right] - \frac{c_{b_2}}{\sigma}\left(\frac{\partial\tilde{\nu}}{\partial x_i}\right)^2 - \tilde{\nu}P(\tilde{\nu}) + \tilde{\nu}D(\tilde{\nu}) \qquad (2)$$

The production $P(\tilde{\nu})$ and destruction $D(\tilde{\nu})$ terms along with $f_{v_1}$, $f_w$, $\tilde{S}$, and constants $c_{b_1}$, $c_{b_2}$, $c_{w_1}$ and $\sigma$ are all defined in [13].

### 2.2 Boundary Conditions and Discretization

Concerning the boundary conditions, the no–slip condition is applied along the solid walls. At the inlet, the velocity vector profiles are imposed, while at the outlet a fixed mean pressure value is applied. A zero value of $\tilde{\nu}$ is specified along the solid boundaries.

The discretization of the governing PDEs is based on the time–marching technique and the vertex–centered finite volume method. Thus, in each pseudo–time step, equations 1 and 2 are integrated over the finite volumes formed around mesh nodes. The CFD–solver used may handle unstructured/hybrid meshes consisting of tetrahedra, pyramids, prisms and hexahedra.

The inviscid numerical fluxes are computed using the Roe's approximate Riemann solver [12] with second–order accuracy. The stresses on the finite volume faces are computed based on the velocity gradients, the computation of which at the mid-point of each edge (PQ) is given by

$$\left.\frac{\partial w_i}{\partial x_j}\right|_{PQ} = \frac{1}{2}\left[\left.\frac{\partial w_i}{\partial x_j}\right|_P + \left.\frac{\partial w_i}{\partial x_j}\right|_Q\right] - \left[\frac{1}{2}\left[\left.\frac{\partial w_i}{\partial x_j}\right|_P + \left.\frac{\partial w_i}{\partial x_j}\right|_Q\right]n_j - \frac{w_i^Q - w_i^P}{(PQ)}\right]n_j \quad (3)$$

where $n_j$ is the normal to the finite volume interface. Coriolis and centripetal forces are added as source terms.

### 2.3 Numerical Solution

The discretized Navier–Stokes equations are solved iteratively according to the scheme

$$\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\Delta\mathbf{W} = -\mathbf{R}(\mathbf{W}), \quad \mathbf{W}^{k+1} = \mathbf{W}^k + \Delta\mathbf{W} \qquad (4)$$

with $k$ denoting the pseudo-time iteration. The pseudo-time step is calculated locally at each mesh node, based on stability criteria.

In order to maximize the parallel efficiency of the GPU–enabled solver and reduce the device memory requirements, the solver uses Mixed Precision Arithmetics (MPA) [8], which does not harm the accuracy of the results, due to the delta formulation presented in equation 4. In the proposed MPA scheme, DPA (Double Precision

Arithmetics) is used for computing both the LHS and RHS terms. Then, SPA (Single Precision Arithmetics) is used to store the memory–consuming LHS terms, whereas DPA is used for the RHS (i.e. the residuals) of equation 4.

## *2.4 Implementation on many GPUs*

The flow solver, written in the CUDA programming environment, uses the MPI protocol for the inter–node communications. Each CPU–process is executed on a different computing node and controls the GPUs associated with this node. Generally, the number of CPU–processes is not equal to the number of mesh partitions since each CPU–process controls many on–node GPUs each of which is associated with a single mesh partition. For the communication between the on–node GPUs, event–stream synchronizations have been employed. Besides, GPUs on the same node use the common CPU (pinned) memory for data interchange. In order to increase the parallel efficiency of the CFD solver, data interchange overlap with computations; thus, GPU cores remain active even while data are transferred through the available devices.

In the beginning, the CPU–process with rank 0 reads and broadcasts the data input which include the flow conditions, the nodal coordinates and the connectivity of mesh elements per subdomain. It also constructs the necessary lists of mesh nodes shared by adjacent subdomains, for which data communications are necessary. Then, each CPU–process performs the computation of topology- and mesh related data supporting the finite volume method for the subdomains associated with the GPUs controlled by this CPU–process. The computed data are copied to the GPUs, where the numerical solution of the Navier–Stokes equations takes place through successive GPU kernel launches, data interchange and synchronizations.

As already mentioned, the flow solver uses unstructured grids and the vertex–centered finite volume technique. This is the most difficult case regarding GPU memory access, because (a) the use of unstructured grids leads to “unstructured” memory access due to the random grid element numbering and (b) in the vertex-centered approach the number of neighboring nodes per grid node varies from node to node. On the other hand, in the cell-centered finite volume technique the number of neighboring elements per element is a priori known; for instance, in 3D grids with tetrahedral elements, each internal tetrahedron has 4 adjacent tetrahedra. The optimization of memory access [3, 14] together with the use of mixed precision arithmetics (MPA) [8] makes the flow solver running on a single GPU about 50 times faster than its counterpart running on a single CPU core.

## 3 The EA–based Optimization Platform

The design optimization of the compressor blading presented in this work is based on EAs, using the capabilities of the optimization platform EASY [6] developed by the authors' group. In order to decrease the computational cost of the EA–based optimization, the concurrent evaluation of candidate solutions together with surrogate evaluation models (or metamodels) are implemented. Synchronous and asynchronous metamodel–assisted EAs (MAEAs and AMAEAs) are used to quantify the parallel efficiency from the concurrent evaluation of candidate solutions.

In both MAEA and AMAEA, all evaluated individuals, paired with the corresponding objective function values, are stored in a database (DB). Once a predefined minimum number of DB entries has been archived, the IPE phase (implemented differently in the synchronous and asynchronous mode) starts. On–line trained metamodels are used during the IPE phase. Radial basis function (RBF) networks are trained separately for each new individual on its closest (in terms of Euclidean distances in the normalized design space) DB entries. In constrained problems, such as the one studied herein, candidate solutions violating one or more constraints are penalized using an exponential penalty function.

In the following sections, the basic features of the MAEA and AMAEA, with emphasis on the parallelization model and the IPE implementation, are described.

### *3.1 Metamodel–Assisted EA (MAEA)*

The synchronous or generation–based $(\mu,\lambda)$EA and $(\mu,\lambda)$MAEA, [7]) in each generation handle three populations, namely the parent (with $\mu$ members), the offspring (with $\lambda$ members) and the elite (with $\varepsilon$ members) ones. The $\mu$ parents result from the application of the parent selection operator to the offspring and elite populations of the previous generation. The $\lambda$ offspring are formed from the parents, via the application of evolution operators, such as crossover, mutation, etc., including elitism.

The first few generations are performed as a conventional EA and the MAEA starts once there are "enough" training patterns in the DB. During the IPE phase all population members are pre–evaluated on surrogate models trained on–the–fly and only a few ($\lambda_{IPE} << \lambda$) top population members, i.e. the most promising based on the metamodel, are re–evaluated on the CFD model. The $\lambda_{IPE}$ value may vary between a lower ($\lambda_{IPE,min}$) and an upper ($\lambda_{IPE,max}$) user–defined bound. Initially, only the $\lambda_{IPE,min}$ top individuals are re–evaluated; then, some more, up to $\lambda_{IPE,max}$ in total, may be re–evaluated too, based on a number of criteria.

The parallelization of EA and MAEA is based on the master–worker paradigm where the master assigns the $\lambda$ evaluations to the available GPUs ($N_{GPU}$). The number of CPU processes ($N_{CPU}$) is equal to $N_{GPU}$ if the evaluation of a candidate solution is assigned to a single GPU or $N_{CPU} < N_{GPU}$ if assigned to more than one GPUs, as described in section 2.4. In the general case where $\lambda < N_{GPU}$, the first $N_{GPU}$ evalu-

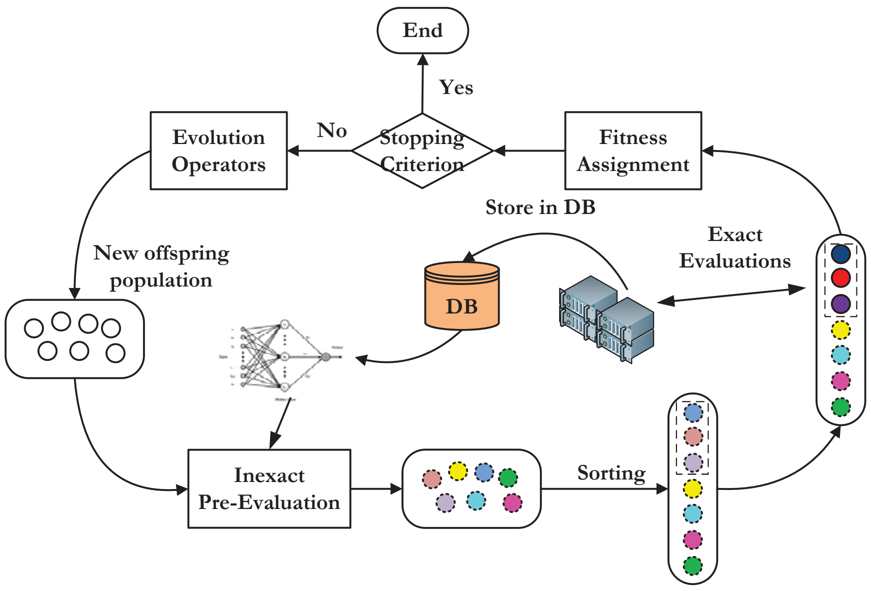

**Fig. 1** Schematic representation of the IPE phase in a (synchronous) MAEA.

ations are assigned to the $N_{GPU}$ devices. The remaining evaluations within the same generation are then assigned to GPUs that become idle anew. The master waits for all GPUs to complete their evaluations before proceeding to the next generation. In the case of a MAEA, the master undertakes the IPE and assigns the $\lambda_{IPE}$ evaluations to the available GPUs.

## *3.2 Asynchronous Metamodel–Assisted EA (AMAEA)*

In the asynchronous EA (AEA) and AMAEA, [1, 2], the population members are associated with the nodes of a supporting mesh which is periodic along its opposite sides. The mesh is subdivided into demes of six nodes each, namely a pole, which acts as the deme's front–end where the best individual of the deme is stored, and five evaluation agents, figure 2. Demes interact through the shared supporting grid nodes. The application of the evolution operators is restricted within each deme.

Asynchronous EAs overcome the notion of generation and better exploit the available computational resources, in comparison to the master–worker paradigm. In particular, the optimization starts by randomly generating individuals and assigning their evaluation to the available GPUs. Upon completion of the evaluation of any individual, the corresponding GPU becomes idle. Instantaneously, a new individual to undergo evaluation is generated through intra– and inter–deme operations. An intra–deme operation, based on dominance criteria, decides whether the just

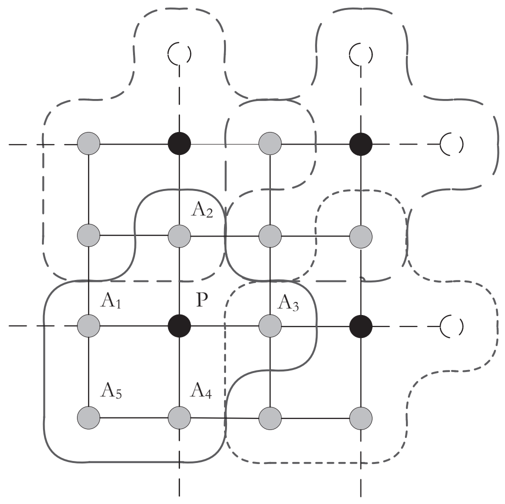

**Fig. 2** Topology of a $4 \times 4$ supporting mesh of an asynchronous EA or MAEA. For the deme associated with pole $P$, agents $A_1$ to $A_4$ are shared with its four neighbouring demes whereas agent $A_5$ is the only non–shared one.

evaluated individual must displace the corresponding pole(s) storing the best so–far computed solution on the deme. Then, an inter–deme operation selects, based on priority criteria, the next agent to undergo evaluation, [1].

In AMAEAs, the metamodels are activated only after completing and archiving a user–defined minimum number of exact evaluations. From this point on (figure 3), for each idle processor, $N_{IPE}$ trial individuals are instantaneously generated by the evolution operators applied within the corresponding deme. For each one of them, a local metamodel is trained and an approximate ("inexact") fitness value is computed. Then, the "best" among the $N_{IPE}$ individuals, according to the metamodel, is the one to undergo re–evaluation by the problem–specific (CFD) tool.

## 4 Design–Optimization of a Compressor Blade

This section presents the optimization of a peripheral compressor cascade. The existing (reference) compressor comprises 12 blades and operates at 1300 rpm. Air in axial direction enters at velocity of 15.28 m/s. The blades are mounted on the hub, with a hub–to–tip radius ratio equal to 0.6 and form a 0.005 m clearance with the stationary shroud. In this paper, the blade is redesigned for minimum viscous losses defined as the averaged relative total pressure difference between the rotor inlet and outlet ($\Delta P_{tR}$). A constraint to maintain the operating point of the reference configuration is imposed in terms of the mass averaged difference of the total pressure between the cascade outlet and inlet ($\Delta P_t$).

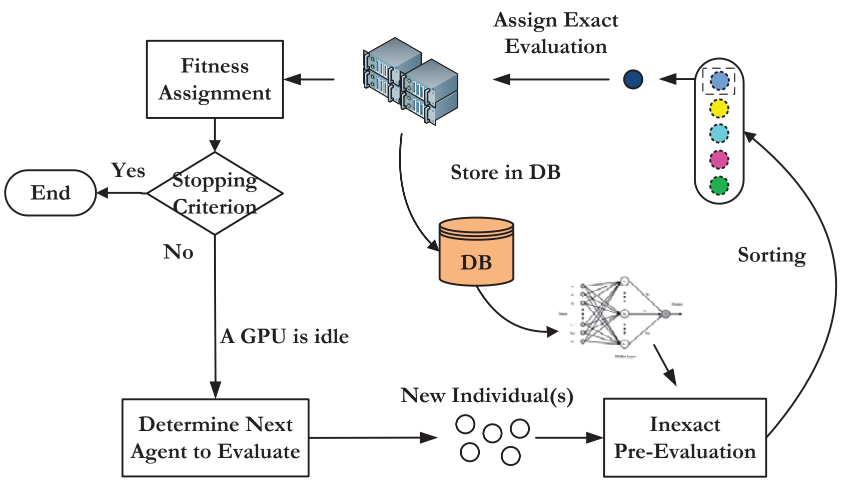

**Fig. 3** Schematic representation of the IPE phase as implemented by an AMAEA.

## *4.1 Parameterization and Grid Generation*

The blade shape is defined by superimposing parameterized thickness profiles on a parameterized mean–camber surface. Bezier control polygons are used to specify the distributions of all geometrical quantities. The design variables, 38 in total, are the coordinates of the control points of Bezier curves used to:

1. generate the meridional projection of the leading (LE) and trailing (TE) edges, as well as the hub and shroud generatrices (figure 4),
2. parameterize the spanwise distributions of (a) the mean–camber surface angles at the LE and TE, (b) the circumferential position of the blade LE and TE and (c) the mean–camber surface curvature,
3. parameterize the non-dimensional thickness profiles at a number of spanwise positions (figure 5, left) and
4. dimensionalize the thickness distribution along the spanwise direction (figure 5, right).

For the evaluation of each candidate solution, a 3D unstructured mesh with about 400000 nodes is generated using an in–house grid–generation software. The final grid comprises hexahedra and prisms over the blade surface, prisms over the casing and tip region, tetrahedra at the inner part of the domain and a zone of pyramids at the interface between hexahedra and tetrahedra. The CFD evaluation relies on the solver described in section 2.

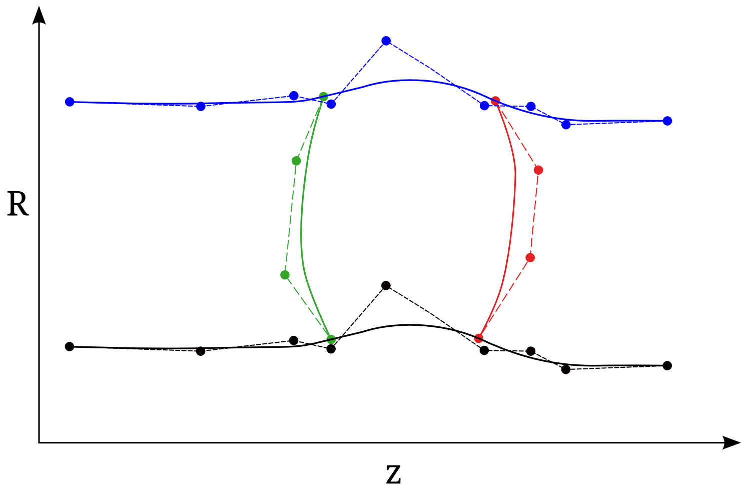

**Fig. 4** Parameterization of the hub and shroud generatrices and the meridional projection of the leading and trailing edges. Curves and their corresponding Bezier polygons are shown.

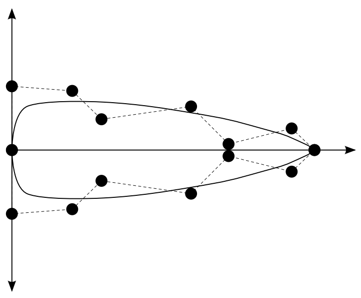
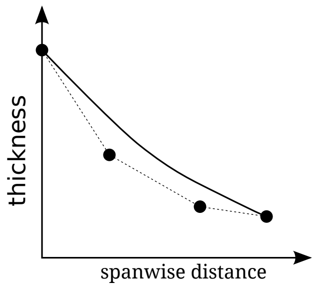

**Fig. 5** Parameterization of a non-dimensional thickness profile, at certain spanwise positions (left) and the spanwise maximum thickness distribution (right).

### *4.2 Optimization Results and Discussion*

This case is studied using both MAEA and AMAEA with 12 concurrent evaluations, i.e. with $N_{CPU} = N_{GPU} = 12$, i.e. each evaluation is assigned to one GPU. For the MAEA $\lambda = 48$, $\mu = 16$ were used. The offspring population size ($\lambda$) was selected to be a multiple of $N_{GPU}$ in order to get the "maximum" parallel efficiency from the master–worker model. In the AMAEA, a $8 \times 8$ supporting mesh (2) is used, which corresponds to 16 poles and 48 evaluation agents.

For both synchronous and asynchronous variants, 27 to 40 training patterns are used for each RBF network and the IPE started once the first 80 (non–penalized, i.e. feasible) individuals were evaluated and stored in the DB. Using the MAEA, two different optimization runs were carried out. In the first one, $\lambda_{IPE} = 12$ members are selected to be re–evaluated on the CFD code whereas, in the second one, $\lambda_{IPE}$

value was allowed to vary between $\lambda_{IPE,min}=5$ and $\lambda_{IPE,min}=8$. This range was decided so as to have some idle GPUs and measure the effect on the optimization turnaround time. For the AMAEA, $N_{IPE}=8$ trial individuals are generated and pre–evaluated before assigning the "best" of them to the idle *GPU* for evaluation on the flow solver.

The convergence histories in terms of computational CPU cost units of the three optimization runs are presented in figure 6. The AMAEA performs better for the same computational cost. It is interesting to comment on the differences between the two MAEA runs. The use of variable $\lambda_{IPE}$ value is beneficiary for the optimization algorithm (though some GPUs remain idle during the re–evaluation) compared to the $\lambda_{IPE}=N_{GPU}$ option.

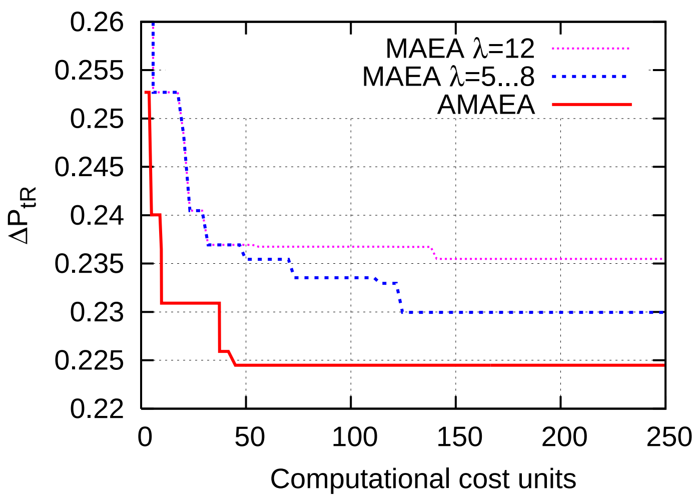

**Fig. 6** Comparison of the convergence history of MAEA and AMAEA.

Comparison of the relative total pressure losses between the reference and the optimal blading at two transverse cross–sections at axial positions $z=0.59c_{ax}$ and $z=1.80c_{ax}$ (the z–origin is at the LE and $c_{ax}$ is the axial chord) are shown in figures 7 and 8. It can be seen that, in the optimal blade, the high pressure losses region, corresponding to the tip clearance vortex, is minimized.

The corresponding objective function and constraint values are compared in table 1.

## *4.3 Concurrent evaluations using CFD on many GPUs*

Aiming at further minimizing the optimization turnaround time, apart from the "smart" use of metamodels and the concurrent evaluations discussed thus far, the

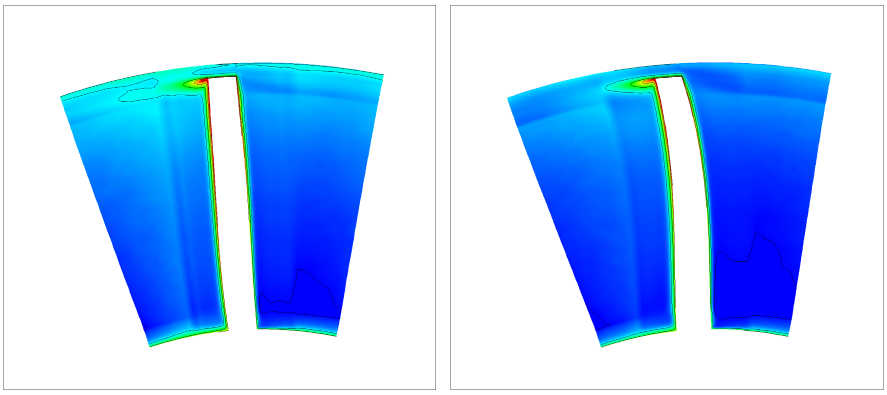

**Fig. 7** Comparison of the relative total pressure between the reference (left) and the optimal (right) cascade at a transverse cross–section located at axial position $z = 0.59c_{ax}$. The $P_{tR}$ field at $z = 0.59c_{ax}$ subtracted from the average inlet $P_{tR}$ value is shown.

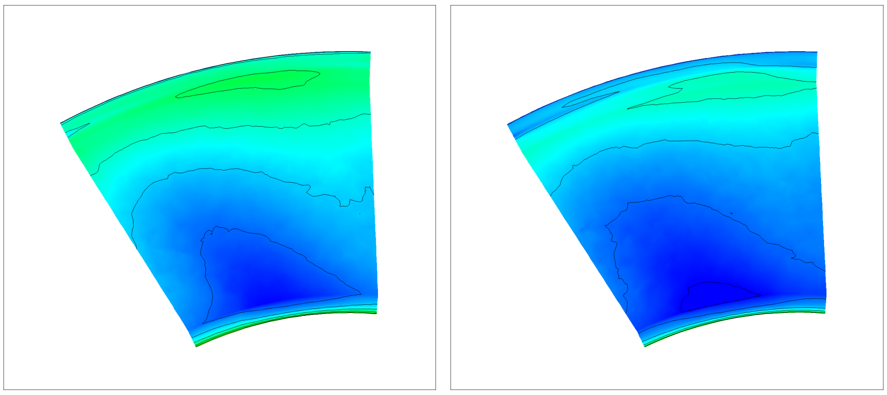

**Fig. 8** Comparison of the relative total pressure between the reference (left) and the optimal (right) cascade at a transverse cross–section located at axial position $z = 1.80c_{ax}$. The $P_{tR}$ field at $z = 1.80c_{ax}$ subtracted from the average inlet $P_{tR}$ value is shown.

**Table 1** Comparison of the objective function and constraint values between the reference and the optimized blade.

| | | Reference blade | Optimized blade |
|---|---|---|---|
| objective: | Losses ($\Delta P_{tR}$) | 0.329 | 0.224 |
| constraint: | $\Delta P_t$ | 0.53 | 0.54 |

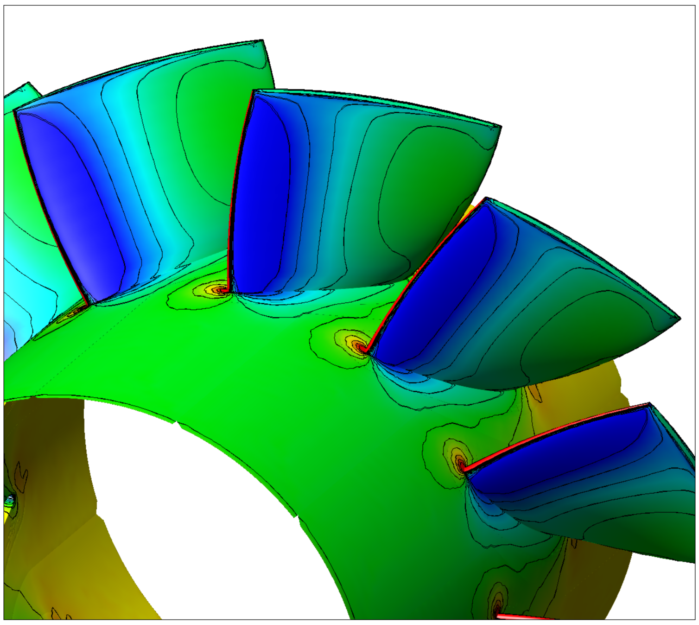

**Fig. 9** Pressure distribution on the optimal geometry obtained by the AMAEA–based optimization.

parallelization of the evaluation software on many GPUs is also considered. In this paper, for parallelizing the evaluation software, one should take into account the search method (synchronous or asynchronous), the possible use of IPE and the computational system in hand (4 computational nodes with 3 Tesla M2050 GPUs on each node).

If an evaluation can be carried out on a single GPU (depending on the computational domain size and the memory of each GPU) the use of asynchronous search with IPE is recommended, as described in the previous section.

Should the computational domain be partitioned and run on many GPUs, things become more complicated. As described in section 2.4, running the Navier–Stokes solver on many GPUs can be done using a single CPU thread for the GPUs on the same node, MPI for GPUs on different nodes or a combination of both if more than 3 GPUs are involved. So, one should also select the "best" configuration, i.e. which GPUs (devices) from which node should undertake each evaluation.

Now, assume that a computational domain is partitioned into two subdomains. In such a case, 6 concurrent evaluations should be carried out. The CFD software may run on two GPUs of either the same node or different nodes. Using a single CPU thread to manage more than one GPUs on the same node, the parallel speed–up is greater than that of using MPI for GPUs on different nodes. So, for the asynchronous search, assigning as many evaluations as possible to pairs of GPUs belonging to the same node (4 in the GPU cluster under consideration) is the best practice in terms of the optimization turnaround time. For the synchronous search, if the offspring population is a multiple of 6 (as many as the concurrent evaluations), all combinations will practically lead to the same optimization turnaround time, since the synchronization barrier is determined by the slowest evaluation.

If 3 GPUs are required at minimum for each evaluation, then assigning 4 concurrent evaluations on the 3 GPUs of each node is the optimal choice for both synchronous and asynchronous search on the available GPU cluster.

## 5 Conclusions

In this paper, metamodel–assisted evolutionary algorithms, synchronous (MAEA) and asychronous (AMAEA), are used in combination with a parallel GPU–enabled CFD solver, in order to reduce the overall optimization time of a low speed compressor blading. The overall turnaround time is shown to be greatly reduced by (a) appropriately tuning the IPE scheme, (b) using asynchronous search in order to minimize the idle time, (c) using GPUs for the exact evaluation tool (CFD) instead of CPUs, and (d) "smartly" distributing the concurrent evaluations on the available GPUs of the cluster, especially in cases where the CFD solver is obliged to run on more than one GPUs. Thus, by appropriately combining all the above techniques and features, the use of evolutionary algorithms for aerodynamic optimization problems, can be made very appealing, in terms of optimization turnaround time, even in large scale industrial applications.

## References

1. V.G. Asouti and K.C. Giannakoglou. Aerodynamic optimization using a parallel asynchronous evolutionary algorithm controlled by strongly interacting demes. *Engineering Optimization*, 41(3):241–257, 2009.
2. V.G. Asouti, I.C. Kampolis, and K.C. Giannakoglou. A grid-enabled asynchronous metamodel-assisted evolutionary algorithm for aerodynamic optimization. *Genetic Programming and Evolvable Machines (SI:Parallel and Distributed Evolutionary Algorithms, Part One)*, 10(3):373–389, 2009.
3. V.G. Asouti, X.S. Trompoukis, I.C. Kampolis, and K.C. Giannakoglou. Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units. *International Journal for Numerical Methods in Fluids*, 67(2):232–246, 2011.
4. T. Brandvik and G. Pullan. An accelerated 3D Navier–Stokes solver for flows in turbomachines. *Journal of Turbomachinery*, 133(2):619–629, 2011.
5. A. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, August 1967.
6. K.C. Giannakoglou. The EASY (Evolutionary Algorithms SYstem) software, http://velos0.ltt.mech.ntua.gr/EASY., 2008.
7. I.C. Kampolis and K.C. Giannakoglou. A multilevel approach to single- and multiobjective aerodynamic optimization. *Computer Methods in Applied Mechanics and Engineering*, 197(33-40):2963–2975, 2008.

8. I.C. Kampolis, X.S. Trompoukis, V.G Asouti, and K.C. Giannakoglou. CFD-based analysis and two-level aerodynamic optimization on graphics processing units. *Computer Methods in Applied Mechanics and Engineering*, 199(9-12):712–722, 2010.
9. M.K. Karakasis and K.C. Giannakoglou. On the use of metamodel-assisted, multi-objective evolutionary algorithms. *Engineering Optimization*, 38(8):941–957, 2006.
10. M.K. Karakasis, A.P. Giotis, and K.C. Giannakoglou. Inexact information aided, low-cost, distributed genetic algorithms for aerodynamic shape optimization. *International Journal for Numerical Methods in Fluids*, 43(10-11):1149–1166, 2003.
11. A. Khajeh-Saeed and J. Blair Perot. Direct numerical simulation of turbulence using GPU accelerated supercomputers. *Journal of Computational Physics*, 235:241–257, 2013.
12. P. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.
13. P. Spalart and S. Allmaras. A one–equation turbulence model for aerodynamic flows. *La Recherche Aérospatiale*, 1:5–21, 1994.
14. X.S. Trompoukis, V.G. Asouti, I.C. Kampolis, and K.C. Giannakoglou. *CUDA implementation of Vertex-Centered, Finite Volume CFD methods on Unstructured Grids with Flow Control Applications*, chapter 17. Morgan Kaufmann, 2011.
15. B. Tutkun and F. Edis. A GPU application for high-order compact finite difference scheme. In 22nd International Conference on Parallel CFD 2010, Kaohsiung, Taiwan, May 17–21 2010.