Aerodynamic Shape Optimization Methods on Multiprocessor Platforms

K.C. Giannakoglou[a] *, I.C. Kampolis[a], P.I.K. Liakopoulos[a], M.K. Karakasis[a], D.I. Papadimitriou[a], T. Zervogiannis[a], V.G. Asouti[a]

[a]National Technical University of Athens (NTUA),
School of Mechanical Engineering, Lab. of Thermal Turbomachines,
Parallel CFD & Optimization Unit, P.O. Box 64069, Athens 157 10, GREECE

Abstract: An overview of modern optimization methods, including Evolutionary Algorithms (EAs) and gradient–based optimization methods adapted for Cluster and Grid Computing is presented. The basic tool is a Hierarchical Distributed Metamodel–Assisted EA supporting Multilevel Evaluation, Multilevel Search and Multilevel Parameterization. In this framework, the adjoint method computes the first and second derivatives of the objective function with respect to the design variables, for use in aerodynamic shape optimization. Such a multi–component, hierarchical and distributed scheme requires particular attention when Cluster or Grid Computing is used and a much more delicate parallelization compared to that of conventional EAs.

1. FROM PARALLEL CFD TO PARALLEL OPTIMIZATION

During the last decade, the key role of CFD-based analysis and optimization methods in aerodynamics has been recognized. Concerning the flow analysis tools, adequate effort has been put to optimally port them on multiprocessor platforms, including cluster and grid computing. Nowadays, there has been a shift of emphasis from parallel analysis to parallel optimization tools, [1]. So, this paper focuses on modern stochastic and deterministic optimization methods as well as hybrid variants of them, implemented on PC clusters and enabled for grid deployment.

The literature on parallel solvers for the Navier–Stokes is extensive. Numerous relevant papers can be found in the proceedings of the past ParCFD and other CFD–oriented conferences. We do not intend to contribute further in pure parallel CFD, so below we will briefly report the features of our parallel flow solver, which is an indispensable part of our aerodynamic shape optimization methods and the basis for the development of adjoint methods. Our solver is a Favre–averaged Navier–Stokes solver for adaptive unstructured grids, based on finite–volumes, with a second–order upwind scheme for the convection terms and parallel isotropic or directional agglomeration multigrid, [2]. For a parallel flow analysis, the grid is partitioned in non-overlapping subdomains, as many as the available processors. The grid (or graph) partitioning tool is based on a single-pass

*e-mail: kgianna@central.ntua.gr

multilevel scheme including EAs, graph coarsening algorithms and heuristics, [3]. This software is as fast as other widely used partitioning tools, [4], and produces optimal partitions with evenly populated subdomains and minimum interfaces. An updated version of the software overcomes the restrictions of recursive bisection and is suitable for use in heterogeneous clusters.

The aforementioned Navier–Stokes solver is used to evaluate candidate solutions in the framework of various shape optimization methods, classified to those processing a single individual at a time and population–based ones. It is evident that the parallelization mode mainly depends upon the number of candidate solutions that can simultaneously and independently be processed. Optimization methods which are of interest here are the *Evolutionary Algorithms* (being by far the most known population–based search method) and *gradient–based methods*. EAs have recently found widespread use in engineering applications. They are intrinsically parallel, giving rise to the so–called Parallel EAs (PEAs). *Metamodel–Assisted EAs* (MAEAs,[5], [6]) and *Hierarchical Distributed MAEAs* (HDMAEAs, [6]) are also amenable to parallelization. We recall that MAEAs are based on the interleaving use of CFD tools and surrogates for the objective and constraint functions, thus, reducing the number of calls to the costly CFD software and generate optimal solutions on an affordable computational budget. On the other hand, this paper is also concerned with *adjoint–based* optimization methods. The adjoint approaches serve to compute the gradient or the Hessian matrix of the objective function for use along with gradient–based search methods, [7], [8]. These approaches can be considered to be either stand–alone optimization methods or ingredients of HDMAEA. Since substantial progress has been made in using hybridized search methods, it is important to conduct a careful examination of their expected performances on multiprocessor platforms.

2. PARALLEL EVOLUTIONARY ALGORITHMS (PEAs)–AN OVERVIEW

The recent increase in the use of EAs for engineering optimization problems is attributed to the fact that EAs are general purpose search tools that may handle constrained multi–objective problems by computing the Pareto front of optimal solutions at the cost of a single run. When optimizing computationally demanding problems using EAs, the CPU cost is determined by the required number of evaluations. In the next section, smart ways to reduce the number of costly evaluations within an EA will be presented. Reduction of the wall clock time is also important and one might consider it as well. PEAs achieve this goal on parallel computers, PC clusters or geographically distributed resources, by partitioning the search in simultaneously processed subproblems. PEAs can be classified into *single–population* or *panmictic EAs* and *multi–population EAs*. In a *panmictic EA*, each population member can potentially mate with any other; standard way of parallelization is the concurrent evaluation scheme, with centralized selection and evolution (master–slave model). A *multi–population EA* handles partitioned individual subsets, according to a topology which (often, though not necessarily) maps onto the parallel platform and employs decentralized evolution schemes; these can be further classified to *distributed* and *cellular* EAs, depending on the subpopulations' structure and granularity.

Distributed (DEAs) rely upon a small number of medium–sized population subsets (demes) and allow the regular inter–deme exchange of promising individuals and intra–

deme mating. By changing the inter–deme communication topology (ring, grid, etc.), the migration frequency and rate as well as the selection scheme for emigrants, interesting variants of DEAs can be devised. To increase diversity, different evolution parameters and schemes at each deme can be used (*nonuniform* DEAs). *Cellular* EAs (CEAs) usually associate a single individual with each subpopulation. The inter–deme communication is carried out using overlapping neighborhoods and decentralized selection schemes. The reader is referred to [9], [10] for a detailed overview of PEAs. A list of popular communication tools for PEAs and a discussion on how PEAs' parallel speedup should be defined can be found in [11]. These excellent reviews, however, do not account for what seems to be an indispensable constituent of modern EAs, namely the use of metamodels.

3. METAMODEL–ASSISTED EAs (MAEAs)

Although PEAs are capable of reducing the wall clock time of EAs, there is also a need to reduce the number of calls to the computationally expensive evaluation software (sequential or parallel CFD codes), irrespective of the use of parallel hardware. To achieve this goal, the use of low–cost *surrogate* evaluation models has been proposed. Surrogates, also referred to as *metamodels*, are mathematical approximations (response surfaces, polynomial interpolation, artificial neural networks, etc.) to the discipline–specific analysis models; they can be constructed using existing datasets or knowledge gathered and updated during the evolution. The reader should refer to [12] for general approximation concepts and surrogates and to [5] for the coupling of EAs and metamodels. There is, in fact, a wide range of techniques for using metamodels within EAs, classified to *off-line* and *on–line* trained metamodels, [5].

In EAs assisted by *off-line* trained metamodels, the latter are trained separately from the evolution using a first sampling of the search space, based on the theory of design of experiments. The deviation of the fitness values of the "optimal" solutions resulting by an EA running exclusively on the metamodel from those obtained using the "exact" CFD tool determines the need for updating the metamodel and iterating.

In EAs assisted by *on-line* trained local metamodels, for each newly generated individual a locally valid metamodel needs to be trained on the fly, on a small number of neighboring, previously evaluated individuals. Evaluating the generation members on properly trained metamodels acts as a filter restricting the use of the CFD tool to a small population subset, i.e. the top individuals as pointed by the metamodel. This will be referred to as the Inexact Pre–Evaluation (IPE) technique; IPE has been introduced in [5], [13] and then extended to DEAs, [14], and hierarchical EAs (HEAs), [6]. Note that, the first few generations (usually no more than two or three) refrain from using metamodels and once an adequate piece of knowledge on how to pair design variables and system performances is gathered, the IPE filter is activated.

Let us briefly describe the IPE technique for multi–objective optimization problems. Consider a (μ, λ) EA with μ parents, λ offspring, n design variables and m objectives. The EA handles three population sets, namely the parent set ($\mathcal{P}_{\mu,g}$, $\mu = |\mathcal{P}_{\mu,g}|$), the offspring ($\mathcal{P}_{\lambda,g}$, $\lambda = |\mathcal{P}_{\lambda,g}|$) and the archival one ($\mathcal{P}_{\alpha,g}$) which stores the $\alpha = |\mathcal{P}_{\alpha,g}|$ best individuals found thus far. At generation g, λ local metamodels are trained and, through them, approximate objective function values $\tilde{\mathbf{f}}(\mathbf{x}) \in \mathbb{R}^m$ for all $\mathbf{x} \in \mathcal{P}_{\lambda,g}$ are computed.

Any approximately evaluated offspring $\mathbf{x}$ is assigned a provisional cost value $\phi(\mathbf{x}) = \phi(\tilde{\mathbf{f}}(\mathbf{x}), \{\tilde{\mathbf{f}}(\mathbf{y}) \mid \mathbf{y} \in \mathcal{P}_{\lambda,g} \setminus \{\mathbf{x}\}\}) \in \mathbb{R}$ where ϕ can be any cost assignment function, based on either sharing techniques, [15], or strength criteria, [16]. The subset $\mathcal{P}_{e,g} = \{\mathbf{x}_i,\ i = 1, \lambda_e : \phi(\mathbf{x}_i) < \phi(\mathbf{y}),\ \mathbf{y} \in \mathcal{P}_{\lambda,g} \setminus \mathcal{P}_{e,g}\}$ of the $\lambda_e{<}\lambda$ most "promising" individuals is singled out. For the $\mathcal{P}_{e,g}$ members, the exact objective function values $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$ are computed; this, practically, determines the CPU cost per generation and, consequently, this all we need to parallelize. A provisional Pareto front, $\hat{\mathcal{P}}_{\alpha,g}$, is formed by the non-dominated, exactly evaluated individuals of $\mathcal{P}_{e,g} \cup \mathcal{P}_{\alpha,g}$. A cost value $\phi(\mathbf{x}) = \phi(\hat{\mathbf{f}}(\mathbf{x}), \{\hat{\mathbf{f}}(\mathbf{y}) \mid \mathbf{y} \in \mathcal{P}_{\lambda,g} \cup \mathcal{P}_{\mu,g} \cup \hat{\mathcal{P}}_{\alpha,g} \setminus \{\mathbf{x}\}\})$ is assigned to each individual $\mathbf{x} \in \mathcal{P}_{\lambda,g} \cup \mathcal{P}_{\mu,g} \cup \hat{\mathcal{P}}_{\alpha,g}$, where $\hat{\mathbf{f}}(\mathbf{x}) = \tilde{\mathbf{f}}(\mathbf{x})$ or $\mathbf{f}(\mathbf{x})$. The new set of non-dominated solutions, $\mathcal{P}_{\alpha,g+1}$, is formed by thinning $\hat{\mathcal{P}}_{\alpha,g}$ through distance-based criteria, if its size exceeds a predefined value a_{max}, [16]. The new parental set, $\mathcal{P}_{\mu,g+1}$, is created from $\mathcal{P}_{\lambda,g} \cup \mathcal{P}_{\mu,g}$ by selecting the fittest individuals with certain probabilities. Recombination and mutation are applied to generate $\mathcal{P}_{\lambda,g+1}$. This procedure is repeated until a termination criterion is satisfied. It is important to note that the number λ_e of exact and costly evaluations per generation is neither fixed nor known in advance (only its lower and upper bounds are user–defined). So, care needs to be exercised when parallelizing MAEAs based on the IPE technique (see below).

4. HIERARCHICAL, DISTRIBUTED MAEAs (HDMAEAs)

We next consider the so–called *Hierarchical* and *Distributed* variant of MAEAs (HDMAEAs), proposed in [6]. An HDMAEA is, basically, a multilevel optimization technique, whereby the term multi–level implies that different search methods, different evaluation software and/or different sets of design variables can be employed at each level. Any EA–based level usually employs DEAs (or, in general PEAs) as explained in the previous sections; also, within each deme, the IPE technique with locally trained metamodels is used (MAEAs). Multilevel optimization algorithms can be classified as follows:

(a) *Multilevel Evaluation* where each level is associated with a different evaluation software. The lower levels are responsible for detecting promising solutions at low CPU cost, through less expensive (low–fidelity) problem–specific evaluation models and delivering them to the higher level(s). There, evaluation models of higher fidelity and CPU cost are employed and the immigrated solutions are further refined. HDMAEAs are multilevel evaluation techniques using PEAs at each level.

(b) *Multilevel Search* where each level is associated with a different search technique. Stochastic methods, such as EAs, are preferably used at the lower levels for the exploration of the design space, leaving the refinement of promising solutions to the gradient–based method at the higher levels. Other combinations of search tools are also possible.

(c) *Multilevel Parameterization* where each level is associated with a different set of design variables. At the lowest level, a subproblem with just a few design variables is solved. At the higher levels, the problem dimension increases. The most detailed problem parameterization is used at the uppermost level.

Given the variety of components involved in a multilevel algorithm (the term HDMAEA will refer to the general class of the multilevel algorithms, since EAs are key search tools) and the availability of parallel CFD evaluation software, parallelization issues of the optimization method as a whole needs to be revisited.

5. ADJOINT–BASED OPTIMIZATION METHODS & PARALLELIZATION

Although gradient–based methods assisted by the adjoint approach are stand-alone optimization methods, they are herein considered one of the components of the aforementioned general optimization framework. They require the knowledge of the first and (depending on the method used) second derivatives of the objective function F with respect to the design variables $b_i,\ i=1,N$. The adjoint method is an efficient strategy to compute ∇F and/or $\nabla^2 F$, [7], [17].

In mathematical optimization, it is known that schemes using only ∇F are not that fast. So, schemes that also use $\nabla^2 F$ (approximate or exact Newton methods) are preferred. However, in aerodynamic optimization, the computation of the exact Hessian matrix is a cumbersome task, so methods approximating the Hessian matrix (the BFGS method, for instance) are alternatively used. We recently put effort on the computation of the exact Hessian matrix with the minimum CPU cost and, thanks to parallelization, minimum wall clock time. Our method, [8], [18], consists of the use of direct differentiation to compute ∇F, followed by the adjoint approach to compute $\nabla^2 F$. The so–called discrete *direct–adjoint* approach is presented below in brief. An augmented functional $\hat{F}$ is first defined using the adjoint variables $\hat{\Psi}_n$, as follows

$$\frac{d^2\hat{F}}{db_i db_j} = \frac{d^2F}{db_i db_j} + \hat{\Psi}_m \frac{d^2R_m}{db_i db_j} \tag{1}$$

where $\frac{d^2F}{db_i db_j} = \frac{d^2F}{db_j db_i}$ and $\frac{d^2R_m}{db_i db_j} = \frac{d^2R_m}{db_j db_i} = 0$. We may also write

$$\frac{d^2\Phi}{db_i db_j} = \frac{\partial^2\Phi}{\partial b_i \partial b_j} + \frac{\partial^2\Phi}{\partial b_i \partial U_k}\frac{dU_k}{db_j} + \frac{\partial^2\Phi}{\partial U_k \partial b_j}\frac{dU_k}{db_i} + \frac{\partial^2\Phi}{\partial U_k \partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j} + \frac{\partial\Phi}{\partial U_k}\frac{d^2U_k}{db_i db_j} \tag{2}$$

where Φ stands for either F or $R_m = 0, m = 1, M$; R_m are the discrete residual equations, arising from some discretization of the flow pde's; M is the product of the number of grid nodes and the number of equations per node. From eq. 1, we obtain

$$\begin{aligned}\frac{d^2\hat{F}}{db_i db_j} = &\frac{\partial^2 F}{\partial b_i \partial b_j} + \hat{\Psi}_n \frac{\partial^2 R_n}{\partial b_i \partial b_j} + \frac{\partial^2 F}{\partial U_k \partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j} + \hat{\Psi}_n \frac{\partial^2 R_n}{\partial U_k \partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j} + \frac{\partial^2 F}{\partial b_i \partial U_k}\frac{dU_k}{db_j} \\ &+\hat{\Psi}_n \frac{\partial^2 R_n}{\partial b_i \partial U_k}\frac{dU_k}{db_j} + \frac{\partial^2 F}{\partial U_k \partial b_j}\frac{dU_k}{db_i} + \hat{\Psi}_n \frac{\partial^2 R_n}{\partial U_k \partial b_j}\frac{dU_k}{db_i} + \left(\frac{\partial F}{\partial U_k} + \hat{\Psi}_n \frac{\partial R_n}{\partial U_k}\right)\frac{d^2U_k}{db_i db_j}\end{aligned} \tag{3}$$

The last term is eliminated by satisfying the adjoint equation

$$\frac{\partial F}{\partial U_k} + \hat{\Psi}_n \frac{\partial R_n}{\partial U_k} = 0 \tag{4}$$

and the remaining terms in eq. 3 give the expression for the Hessian matrix elements. However, to implement the derived formula, we need to have expressions for $\frac{dU_n}{db_j}$. These can be computed by solving the following N equations (*direct differentiation*)

$$\frac{dR_m}{db_i} = \frac{\partial R_m}{\partial b_i} + \frac{\partial R_m}{\partial U_k}\frac{dU_k}{db_i} = 0 \tag{5}$$

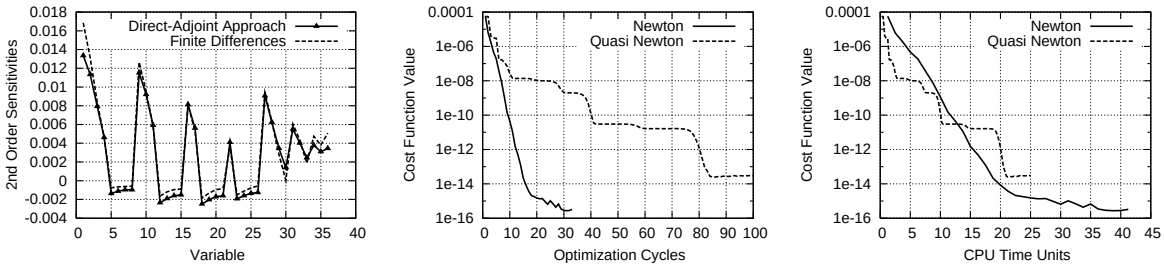

Figure 1. Optimization of a compressor cascade using exact– and quasi–Newton method, based on parallel adjoint and direct solvers (eight design variables). Left: Hessian matrix values computed using the *direct–adjoint* approach and finite differences; only the lower triangular part of the symmetric Hessian matrix is shown. Mid: Convergence of the exact– and quasi–Newton (BFGS) methods, in terms of optimization cycles; the computational load per cycle of the former is 10 times higher. Right: Same comparison in terms of (theoretical) wall clock time on an eight–processor cluster; the flow and the adjoint solvers were parallelized using subdomaining (100% parallel efficiency is assumed) whereas the eight direct differentiations were run in parallel, on one processor each.

for $\frac{dU_k}{db_i}$. To summarize, the total CPU cost of the Newton method is equal to that of N system solutions to compute $\frac{dU_n}{db_j}$, eqs. 5, plus one to solve the adjoint equation, eq. 4 and the solution of the flow equations ($N+2$ system solutions, SSs, per step, in total). In contrast, BFGS costs 2 SSs per descent step. The application of the method (as stand alone optimization tool) on the inverse design of a compressor cascade is shown and discussed in fig. 1.

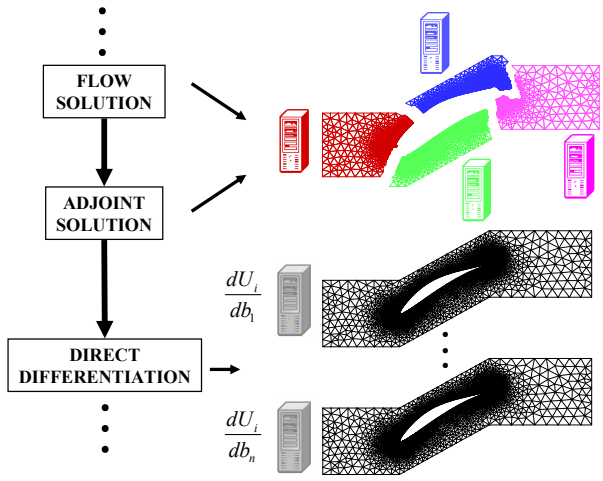

Figure 2: Parallelization of a Newton cycle based on the direct-adjoint method.

Working with either discrete or continuous adjoint approaches and any parallel flow solver (such as the aforementioned one, [2]), the parallelization of the adjoint solver for ∇F is based on subdomaining in a similar manner to that used for the associated flow solver. At this point and in view of what follows, this will be considered as fine–grained parallelization. What is new and interesting is the parallelization of the proposed adjoint method for the computation of $\nabla^2 F$. Among the $N+2$ SSs needed, N of them corresponding to the computation of $\frac{dU_k}{db_i}$ (direct differentiation, eqs. 5) may run concurrently, due to the independence of the N equations (coarse–grained parallelization; needless to say that subdomaining can additionally be used if more processors are available). The remaining two system solutions (for the adjoint and flow equations) are fine–grained par-

allelized using subdomaining. The process is illustrated in fig 2.

6. PARALLELIZATION OF HDMAEA

6.1. HDMAEAs and Cluster Computing

In small commodity clusters, one may use a customized configuration for each application; this case is of no interest in this paper. In larger clusters or HPC systems, the presence of a local resource manager that keeps an updated list of the available processors along with their characteristics, becomes necessary. Using these pieces of information, the *Local Resource Management Software* (LRMS) allows the optimal use of the cluster, by assigning each evaluation request to the appropriate computational resource(s) according to user–defined specifications (i.e. RAM, CPU etc.).

The deployment of the optimization software within a cluster requires linking it to the LRMS. In this respect, DEAs or HDMAEAs use a central evaluation manager allowing concurrent job submissions from any deme. The evaluation manager executes in a separate thread inside the optimization algorithm process and receives evaluation requests from all demes. These requests are submitted to the centralized LRMS *first–in–first–out* queue and each new job is assigned to the first node that becomes available and meets the requirements set by the user. *Condor* middleware was chosen as LRMS, [19]. The *DRMAA* library was used to interface the optimization software with *Condor*, by communicating arguments related to file transfer, I/O redirection, requirements for the execution host etc.

In a distributed optimization scheme (DEA , HDMAEA), upon the end of an evaluation, the evaluation manager contacts the corresponding deme and updates the objectives' and constraints' values of the evaluated individual. Therefore, all evaluation specific details are hidden from the EA core. A unique database (per evaluation software) for all demes is maintained to prevent the re-evaluation of previously evaluated candidate solutions.

Let us lend some more insight into the role of the evaluation manager by comparing evaluation requests launched by a conventional EA and the HDMAEA. In a conventional (μ, λ) EA, λ evaluations must be carried out per generation; they all have the same hardware requirements and, theoretically, the "same" computational cost; this is in favor of parallel efficiency. In contrast, maintaining a high parallel efficiency in hierarchical optimization (*Multilevel Evaluation* mode) is not straightforward, due to the use of evaluation tools with different computational requirements. For instance, in aerodynamic optimization, a Navier–Stokes solver with wall functions on relatively coarse grids can be used at the low level (for the cheap exploration of the search space) and a much more accurate but costly flow analysis tool with a better turbulence model on very fine grids at the high level. The latter has increased memory requirements. In such a case, given the maximum amount of memory available per computational node, a larger number of subdomains may be needed. In heterogeneous clusters (with different CPU and RAM per node), a different number of processors might be required for the same job, depending on the availability of processors by the time each job is launched. In the case of synchronous inter–level communication, a level may suspend the execution temporarily, until the adjacent level reaches the generation for which migration is scheduled and this usually affects the assignment of pending evaluations. Also, the IPE technique in a MAEA changes dynamically

the number of "exact" evaluations to a small percentage of the population (as previously explained, the percentage λ_e/λ is not constant but changes according to the population dynamics), limiting thus the number of simultaneous evaluation jobs and causing some processors to become idle. For all these reasons, the use of an evaluation manager in HDMAEAs is crucial. Note that any call to the adjoint code within a HDMAEA (*Multilevel Search* mode) is considered as an "equivalent" evaluation request.

6.2. Grid Enabled HDMAEAs

The modular structure of the HDMAEA and the utilization of an evaluation manager allows its deployment in the Grid, leading to a Grid Enabled HDMAEA (GEHDMAEA). Grid deployment requires a link between the cluster and the grid level. A middleware which is suitable for this purpose must: (a) match grid resources to job–specific requirements, (b) connect the LRMS to the grid framework to allow job submission using security certificates for user authentication and (c) allow grid resource discovery by collecting information regarding resource types and availabilities within the clusters. In the developed method, this is accomplished via *Globus Toolkit 4* (*GT*4, [20]).

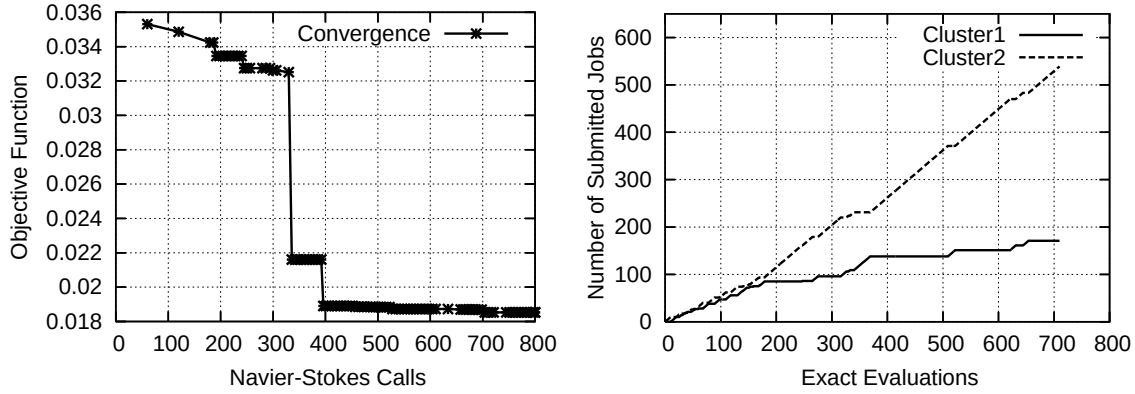

Figure 3. Optimization of an axial compressor stator using the Grid Enabled HDMAEA. Left: Convergence history. Right: History of the number of submitted jobs to each cluster.

The execution of a job through the *Globus Toolkit* starts with the verification of user's credentials before submitting a job to a grid resource. A user may submit a job to a cluster in two different ways. The submission can be made through either the front-end of the cluster and, then, passed on to the LRMS, or through a gateway service by submitting a job from a host that has *GT*4 installed to a remote grid resource. For the second option, the user's credentials are also transmitted for authentication to the remote resource (via *GT*4) before the job is accepted for execution.

All grid resources that have *GT*4 installed publish information about their characteristics which can be accessed using the *http* protocol. The discovery and monitoring of resources via a tool that probes all resources for their characteristics by accessing the corresponding information service of the *GT*4, i.e. a *Meta–Scheduler*, is needed. The *Meta–Scheduler* unifies the grid resources under a common queue and associates each job submitted to the queue with the first available grid resource satisfying the requirements

set by the user. Herein, *GridWay*, [21], was used as *Meta–Scheduler*. *GridWay* has an API that enables GEHDMAEA to forward all job requests from the evaluation server to the common grid queue. *GridWay* collects information regarding the available resources from the information services of the $GT4$ installed on each resource and keeps the complete list available to all authorized users. When submitting jobs to *GridWay*, each job is forwarded from the global grid queue to the $GT4$ of the matched grid resource and, then, to the LRMS, if this resource is a cluster. The LRMS (*Condor*) performs the final matchmaking between the job and the available resources within the cluster, followed by the job execution.

The optimization process in a grid environment is carried out the same way as within a cluster. For illustrative purposes we present a design problem of an axial compressor stator with minimum total pressure loss coefficient ω, at predefined flow conditions (more details on this case are beyond the scope of this paper). A number of constraints were imposed regarding the minimum allowed blade thickness at certain chordwise locations as well as the flow exit angle (by, consequently, controlling the flow turning). The convergence history of the optimization process is presented in fig. 3. The grid environment in which the optimization software was deployed consists of two clusters. The description of the clusters along with the number of jobs submitted, during the airfoil design, per cluster is given below. The number of submitted jobs to each cluster are shown in fig. 3 (right).

	Host types	Number of Hosts	Jobs Submitted
Cluster1	PIII and P4 up to 2.4GHz	14	154
Cluster2	P4 and Pentium D up to 3.4GHz	30	550

7. ACKNOWLEDGEMENTS

Parts of this synthetic work and development during the last years in the lab were supported by the $PENED01$ Program (Measure 8.3 of the Operational Program Competitiveness) under project number $01ED131$ and by the Operational Program for Educational and Vocational Training II (EPEAEK II) (Program PYTHAGORAS II); in both, funding was 75% by the European Commission and 25% by National Resources. Some of the PhD students were supported by grants from the National Scholarship Foundation.

REFERENCES

1. D. Lim, Y-S. Ong, Y. Jin, B. Sendhoff, B-S. Lee, Efficient Hierarchical Parallel Genetic Algorithms using Grid Computing, J. Fut. Gener. Comput. Syst., 23(4):658–670, 2007.
2. N. Lambropoulos, D. Koubogiannis, K. Giannakoglou, Acceleration of a Navier-Stokes Equation Solver for Unstructured Grids using Agglomeration Multigrid and Parallel Processing, Comp. Meth. Appl. Mech. Eng., 193:781-803, 2004.
3. A. Giotis, K. Giannakoglou, An Unstructured Grid Partitioning Method Based on Genetic Algorithms, Advances in Engineering Software, 29(2):129-138, 1998.
4. G. Karypis, V. Kumar, Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM J. Scientific Computing, 20(1):359-392, 1998.
5. K. Giannakoglou, Design of Optimal Aerodynamic Shapes using Stochastic Optimiza-

tion Methods and Computational Intelligence, Int. Review J. Progress in Aerospace Sciences, 38:43-76, 2002.
6. M. Karakasis, D. Koubogiannis, K. Giannakoglou, Hierarchical Distributed Evolutionary Algorithms in Shape Optimization, Int. J. Num. Meth. Fluids, 53:455-469, 2007.
7. D. Papadimitriou, K. Giannakoglou, A Continuous Adjoint Method with Objective Function Derivatives Based on Boundary Integrals for Inviscid and Viscous Flows, Computers & Fluids, 36:325-341, 2007.
8. D. Papadimitriou, K. Giannakoglou, Direct, Adjoint and Mixed Approaches for the Computation of Hessian in Airfoil Design Problems, Int. J. Num. Meth. Fluids, to appear.
9. E. Cantu-Paz, A Survey of Parallel Genetic Algorithms", Calculateurs Paralleles, Reseaux et Systemes Repartis, 10(2):141-171, 1998.
10. M. Nowostawski, R. Poli, Parallel Genetic Algorithm Taxonomy, Proc. 3rd Int. Conf. on Knowledge-based Intelligent Information Engineering Systems KES'99:88-92, IEEE, 1999.
11. E. Alba, M. Tomassini, Parallelism and Evolutionary Algorithms, IEEE Trans. Evol. Comp., 6(5), Oct. 2002.
12. A. Keane, P. Nair, *Computational Approaches for Aerospace Design. The Pursuit of Excellence*, John Wiley & Sons, Ltd, 2005.
13. K. Giannakoglou, A. Giotis, M. Karakasis, Low–Cost Genetic Optimization based on Inexact Pre-evaluations and the Sensitivity Analysis of Design Parameters, Inverse Problems in Engineering, 9:389-412, 2001.
14. M. Karakasis, K. Giannakoglou, Inexact Information aided, Low-Cost, Distributed Genetic Algorithms for Aerodynamic Shape Optimization, Int. J. Num. Meth. Fluids, 43(10-11):1149-1166, 2003.
15. N. Srinivas, K. Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, Evolutionary Computation, 2(3):221–248, 1995.
16. E. Zitzler, M. Laumans, L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Report, Swiss Federal Institute of Technology (ETH), Computer Engineering and Communication Networks Lab., May 2001.
17. D. Papadimitriou, K. Giannakoglou, Total Pressure Losses Minimization in Turbomachinery Cascades, Using a New Continuous Adjoint Formulation, Journal of Power and Energy (Special Issue on Turbomachinery), to appear, 2007.
18. K. Giannakoglou, D. Papadimitriou, Adjoint Methods for gradient- and Hessian-based Aerodynamic Shape Optimization, EUROGEN 2007, Jyvaskyla, June 11-13, 2007.
19. D. Thain and T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurrency - Practice and Experience, 17(2-4):323-356, 2005.
20. I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, International Conference on Network and Parallel Computing, LNCS: 2-1317(2-4), Springer-Verlag, 2006.
21. E. Huedo, R. Montero, I. Llorente, A Framework for Adaptive Execution on Grids, Journal of Software - Practice and Experience, 34:631-651, 2004.