Achieving High Speed CFD simulations: Optimization, Parallelization, and FPGA Acceleration for the unstructured DLR TAU Code

E. Andres*, M. Widhalm†, A. Caloto*

National Institute of Aeroespace Technology, Madrid, 28850, Spain

German Aerospace Center, Braunschweig, Niedersachsen, 38108, Germany

Today, large scale parallel simulations are fundamental tools to handle complex problems. The number of processors in current computation platforms has been recently increased and therefore it is necessary to optimize the application performance and to enhance the scalability of massively-parallel systems. In addition, new heterogeneous architectures, combining conventional processors with specific hardware, like FPGAs, to accelerate the most time consuming functions are considered as a strong alternative to boost the performance.

In this paper, the performance of the DLR TAU code is analyzed and optimized. The improvement of the code efficiency is addressed through three key activities: Optimization, parallelization and hardware acceleration. At first, a profiling analysis of the most time-consuming processes of the Reynolds Averaged Navier Stokes flow solver on a three-dimensional unstructured mesh is performed. Then, a study of the code scalability with new partitioning algorithms are tested to show the most suitable partitioning algorithms for the selected applications. Finally, a feasibility study on the application of FPGAs and GPUs for the hardware acceleration of CFD simulations is presented.

I. Introduction

Scientific Computing with its core ingredients, modeling, simulation and optimization, is regarded by many as the third pillar of science, complementary to experiment and theory. In aeronautics engineering, the consistent use of mathematical and computation methods to simulate complex processes has become indispensable to save energy, reduce costs and pollution, and to increase safety. However, the high complexity of some of these processes frequently implies very long computation times. In particular, the analysis of a complete aircraft configuration including all relevant payload elements and flight components, even using a Reynolds-Averaged Navier-Stokes (RANS) modeling, at present still requires a huge computational effort, even using the modern high parallel computational platforms. Thus, reducing the time for aerodynamic analysis is one of the most important challenges of current research in CFD.

An efficient implementation for codes based on unstructured meshes is still a challenging task. In comparison to structured solvers,[1,2] with their lower memory usage and block-structured data, their computational rates and scalability were many years out of reach for any unstructured solvers. Through the use of effective edge-based structures the memory requirements were reduced and by using grid data-reordering techniques for a banded matrix the efficiency was increased remarkably. Especially edge-based structures enabled a homogeneous data-structure independent on the grid volume elements. Additionally, a cache optimization for PCs successfully increased the throughput. With all these improvements, unstructured solvers have matured to be applied for industrial applications.[3–6]

*Research Scientist, Fluid Dynamics Branch, Ctra. de Ajalvir km.4, 28850 Madrid, AIAA Member.

†Research Scientist, Institut of Aerodynamics and Flow Technology, Lilienthalplatz 7, 38108 Braunschweig, AIAA Member.

Apart from these advantages, improvements in the computational speed can also be addressed on many levels. We are proposing three different strategies which are considered from the basic code optimization to parallelisation up to a new hardware architecture.

Code optimization is one of the first objectives in performance improvement, and it is often less considered with respect to other important goals such as stability, maintainability, and portability. This simple level of optimization is beneficial and should always be applied including an efficient implementation, reduction of operations, precomputation of expensive variables and non-redundant interfaces. Code optimization is based on *execution profiling* while performing a time measurement of bottlenecks in the code. Applying optimization strategies over the most time consuming algorithms of the code can provide important reductions of the execution time[7, 8] .

In recent years, much of the attention has been focused on methods for parallel computers to reduce the computation time by taking advantage of concurrent processing of data in different regions of the domain, and to increase the resolution of the model by taking advantage of the larger memory available in parallel computers. To effectively utilize a high parallel computer, it is important that the data has to be distributed over the processors in a balanced manner, so that each processor will complete its work load at approximately the same time to prevent idling of processors.

This distribution must be in the manner that the number of assigned elements to each processor is the same, and the number of adjacent elements assigned to different processors is minimized. The goal of the first condition is to balance the computations among the processors. The goal of the second condition is to minimize the communication resulting from the placement of adjacent elements to different processors.
Efficient partitioning algorithms for highly unstructured graphs are crucial for gaining fast solutions in a wide range of applications areas on high parallel computers, and particularly, in large-scale CFD simulations.[3, 4, 9, 10]
In addition, adaptive scientific computations require that periodic repartitioning, known as load balancing, occur dynamically to maintain load balance. A classic example is the simulation based on adaptive mesh refinement, in which the computational mesh changes between time steps. The difference is often small, but over time, the cumulative change in the mesh becomes significant. An application may therefore periodically re-balance, that is, move data among processors to improve the load balance. This process is known as dynamic load balancing or repartitioning and should be considered in modern applications.

Additionally, a new alternative to boost the performance is to consider new heterogeneous architectures for high performance computing[11–13] combining conventional processors with specific hardware to accelerate the most time consuming functions.
Hardware acceleration gives best results, in terms of overall acceleration and value for money when applied to problems in which:

- Condition A. A great amount of the computational effort is concentrated in a small portion of the whole code, and the computations have to be performed several times for a huge set of data.
- Condition B. The communication and I/O times are small with respect to the total computational cost[14] .

Aerodynamics simulations seem to gather all the desirable characteristics to be executed on a platform with a heterogeneous architecture[13] (specific software - specific hardware accelerator - generic hardware platform):

- The numerical solution of the flow equations is based on a flux interchange between the discrete volumes that represent the physical domain. The numerical flux computation for each discrete volume requires several hundred floating-point operations, and each operation is repeated several million times in a real aeronautical simulation (Condition A).
- On the other hand, the numerical solution of the flow has data locality. This means that the computation to be performed at a certain location of the domain depends only on data that is located in a small neighborhood around it. Communication between the processors is required because the solutions on the subdomains depend on each other, but only on the subdomain boundaries. This is the reason why communication costs are small with respect to the overall cost (Condition B).

II. Brief Introduction to the TAU Code.

The fluid flow over the object of interest is simulated with the TAU Code.[15,16] The unsteady TAU-Code solves the compressible, three-dimensional Reynolds-Averaged Navier-Stokes equations using a finite volume formulation. The TAU-Code is based on a hybrid unstructured-grid approach, which makes use of the advantages of semi-structured prismatic grids in the viscous shear layers near walls, and the flexibility in grid generation offered by tetrahedral grids in the surrounding flow volume. The grids used for flow simulations in this paper were created with the hybrid grid generation software Centaur, developed by Centaur Soft[17] . A dual-grid approach with an edge based data structure is used in order to make the flow solver independent from the cell types used in the initial grid. The TAU-Code consists of several different modules, including:

- The Grid Partitioner, which splits the primary grid into n number of subgrids for n processors.
- The Preprocessor module, which uses the information from the initial grid to create a dual-grid and secondly coarser grids for multi-grid.
- The Solver module, which performs the flow calculations on the dual-grid.
- The Adaptation module, which refines and derefines the computational grid with different indicator functions.
- The Deformation module, which propagates the deformation of surface grid points to the surrounding volume grid.
- The Postprocessing module, which is used to convert result-files to formats usable by popular visualization tools.

Together all modules are available with Python[18] interfaces for computing complex application, e.g. unsteady cases, complete force polar curves or fluid-structure couplings in an automatic framework. Furthermore, it eases the usage on highly massive parallel computers to execute applications.

III. Optimization: Analysis and remodelling of most time consuming algorithms in TAU

In this section, a performance analysis of the most time consuming functions in the flow solver will be outlined. The computations are performed with a Spalart Allmaras one-equation turbulence model[19] with Edwards[20] modification (SAE) and the central spatial discretization scheme with scalar dissipation. The computational grid around a Onera M6[21] Navier Stokes grid has 450.634 points and 1.509.834 elements and contains a prismatic layer around the surface and tetrahedrons in the farfield. The conditions are a free-stream Mach number of 0.8395 and fixed angle of attack at 3.06 degree. For completeness, both available time integration schemes, an explicit Runge-Kutta and a semi-implicit LUSGS method, were profiled to obtain the most time consuming algorithms.

This preliminary step, before considering any partitioning algorithm, is relevant to determine the flow solvers bottlenecks. It is important to be aware of computational routines which might drawback any effort in applying sophisticated grid partitioning and maybe load balancing whenever it hang-up in a numeric routine. A flow solver is composed of very different algorithms either for e.g. residual computation, boundary conditions and time integration. But additionally the management of parameter settings, memory allocation and freeing and creating solution output can become crucial if not considered well. A flow solver profiling determines which algorithms are lacking in their efficiency. Due to the iterative process the most often used algorithms are dependent on the residual computation and time integration. The code computes the complete residual R in split flux functions. Mainly there are routines for the convective and viscous part of the solution system. Furthermore, in case of turbulent flows the turbulent equations with the diffusion flux are added. Due to the properties of this system of equations the computation can be separated in different flux routines. It is desired that each separated flux computation should take the same amount of time per iteration because each subroutine has to evaluate 5 equations for the flow variables which are density, the velocity vector and the pressure. But nevertheless the complexity of each equation can vary tremendous. The turbulence variables are dependent on n number of additional transport equations. Additionally, routines have to be taken into account for computing the eigenvalues of the system of equations, residual smoothing steps and helper functions for conversions between primitive and conservative variables.

The results obtained for the profiling execution of the sequential flow solver using an explicit Runge-Kutta method are displayed in Table 1 and for the semi-implicit LUSGS method in Table 2. The abbreviations

Table 1. Profiling results with an Runge-Kutta method for the Onera M6.

Function*	% of total time
viscous_fluxes_tsl	39.2%
diffusion_fluxes_tsl	10.0%
compute_gradients	6.8%
scaling_factor_dissipation	6.0%
central_inviscid_flux	5.4%
compute_local_eingenvalues	4.7%
smooth_laplace	3.8%
compute_sae_sources	3.5%
additional_state_variables	2.7%
central_turbulent_flux	2.0%
scalar_dissipation	2.0%
convert_consvar_to_primvar	1.6%
compute_laplacian_consvar	1.5%
determine_local_timestepsize	1.5%

Table 2. Profiling results with an LUSGS method for the Onera M6.

Function*	% of total time
viscous_fluxes_tsl	21.6%
additional_state_variables	10.4%
compute_local_eingenvalues	7.9%
compute_gradients	7.4%
scaling_factor_dissipation	6.6%
diffusion_fluxes_tsl	5.6%
implicit_time_integration	4.8%
convert_cons_to_prims	4.7%
central_inviscid_flux	2.9%
linear_solver_lusgs	2.9%
sa_orig_implicit_sources	2.5%
scalar_dissipation	2.2%
smooth_laplace	2.0%
compute_sae_sources	2.0%

used in the Tables 1, 2, 3 and 4 are *tsl* for thin shear layer approximation, *sa* and *sae* for either the Spalart-Allmaras and Spalart-Allmaras with Edwards modification. The most time consuming routine for both time integration schemes is the evaluation of the viscous fluxes for the main equations. Especially for the Runge-Kutta it can become 40% of time per iteration. The second one is either the evaluation of the diffusion fluxes for the Runge-Kutta and the computation of the state variables of each face for the LUSGS scheme. Every other routine is at or below 10 % time per execution and is considered as appropriate during an iteration.

Computing the viscous fluxes with a thin shear layer approximation involves simple but many gradient evaluations for the velocities and temperature. The viscous terms of the main equations involve many CPU operations due to their complexity compared to the convective flux computation. The improvement of this routine was performed in two ways. First, we pre-computed locally constant variables which appeared in the functions very often like differences for velocities or eddy viscosity. This approach becomes important whenever divisions or square root operations are involved. In this particular test-case we saved around one fourth of the time consumption used for the viscous flux computation.

The second step is to pre-compute globally constant variables which are independent of the flow variables such as point distance evaluations from the grid metrics. At this part we are computing once the variables at the initialization step of the flow solver and keeping the data stored during the simulation. Especially the functions which are related to the thin layer approximations benefit from this approach. On the other hand more memory is consumed and it becomes evident if computational time or memory is preferred. The memory increment was about 3% of the total memory used from the code without optimization.

After optimizing the code we obtained Table 3 and 4 for either the Runge-Kutta and LUSGS time integration schemes.

The modifications result in an improvement for the viscous flux and diffusion flux computation over more than half the time consumption as without. The first five main functions are now consuming almost the same amount of time per execution, especially for the LUSGS. The first optimization procedure proposed was introduced in many other routines and it can be seen that the order has changed considerable for the functions.

Improving functions like gradient computation, indicated as *compute_gradient* or state variable computation *additional_state_variables* is much more difficult. These functions are already implemented optimized due to their wide range of application areas. Usually, this should count for each function introduced in any code but developers experience and consciousness have to be aware to follow optimization guidelines strictly.

Table 3. Profiling results after optimization with an Runge-Kutta method for the Onera M6.

Function*	% of total time
viscous_fluxes_tsl	18.1%
compute_gradients	10.6%
scaling_factor_dissipation	9.3%
central_inviscid_flux	8.3%
compute_local_eingenvalues	7.2%
smooth_laplace	5.8%
compute_sae_sources	5.5%
additional_state_variables	4.1%
central_turbulent_flux	3.3%
diffusion_fluxes_tsl	3.1%
scalar_dissipation	3.1%
convert_consvar_to_primvar	2.4%
compute_laplacian_consvar	2.4%
determine_local_timestepsize	1.6%

Table 4. Profiling results after optimization with an LUSGS method for the Onera M6.

Function*	% of total time
additional_state_variables	12.8%
compute_local_eingenvalues	9.7%
scaling_factor_dissipation	9.3%
compute_gradients	9.2%
viscous_fluxes_tsl	8.0%
implicit_time_integration	5.7%
convert_cons_to_prims	5.6%
central_inviscid_flux	3.6%
linear_solver_lusgs	3.6%
sa_orig_implicit_sources	3.2%
scalar_dissipation	2.7%
smooth_laplace	2.5%
compute_sae_sources	2.4%
diffusion_fluxes_tsl	1.4%

Table 5 and Table 6 show the *wall clock time* (WCT) on an i386 32 bit and a x86 64 bit Linux based machine for a Onera M6 wing simulation for either the original and the optimized code using both Runge-Kutta and LUSGS schemes.

Table 5. WCT for the Onera M6 Testcase to steady state for 32 and 64 bit Linux machine (Runge-Kutta method).

Target architecture	Code version	C_L	C_D	C_{MY}	Exec. time	Comp. gain
32 bits	orig	0.26933	0.015774	8.1587	7178	-
	opti	0.26933	0.015774	8.1587	5178	27.9 %
64 bits	orig	0.26932	0.015778	8.1585	4169	-
	opti	0.26932	0.015778	8.1585	3701	11.2 %

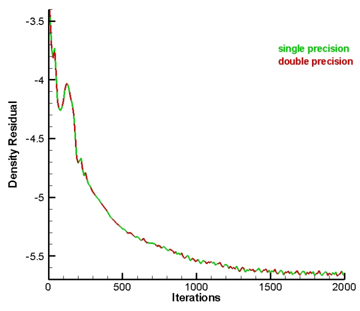
Table 6. WCT for the Onera M6 Testcase to steady state for 32 and 64 bit Linux machine (LUSGS method).

Target architecture	Code version	C_L	C_D	C_{MY}	Exec. time	Comp. gain
32 bits	orig	0.26927	0.015787	8.1569	5657	-
	opti	0.26927	0.015787	8.1569	5487	3.0 %
64 bits	orig	0.26925	0.015791	8.1562	4222	-
	opti	0.26925	0.015791	8.1562	4043	4.2 %

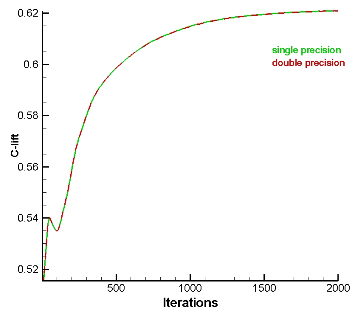
The comparison was made with the same C-compiler version *gcc 4.2.3* using the second optimization level. One thousand iterations were performed for each simulation to ensure either well converged force coefficients like drag and lift and otherwise give substantial time to measure the optimizations during the iterative process apart from additional time used for setup and IO.

Using the explicit Runge-Kutta time integration scheme on a 32 bits system, the WCT is decreased by 27.9 % mainly due to the improvement of the viscous and diffusion flux routines.

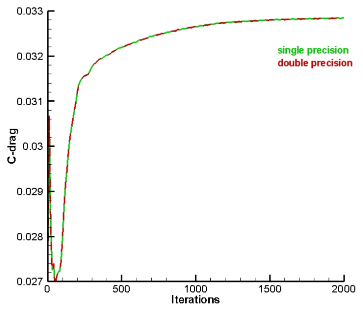
In order to extend this section about code optimization, further consideration can be applied to the precision requirement for CFD computations which is affected directly by the number of bits representing each operand in the code. Related to this issue, we have made some analysis about how the selection of a floating point format (single 32 bits or double 64 bits) affects the integral coefficients of lift (C_L) and drag (C_D). Figure 1 shows the convergence history and the global force coefficients for the DLR F6[22,23] configuration. The DLR-F6, see Figure 2, is a simplified wing-fuselage geometry which has been used in the past for validation of CFD codes at the second[24] and third[25] AIAA sponsored Drag Prediction Workshops. The computational grid around the F6 Navier-Stokes grid has 5.8 mill. points and 16.1 mill. elements. The flow conditions are a free-stream Mach number of 0.749 and a fixed angle of attack at 1 degree with a Reynolds number of $3 * 10^6$. Figure 1 does not show any significant differences in the solution, only



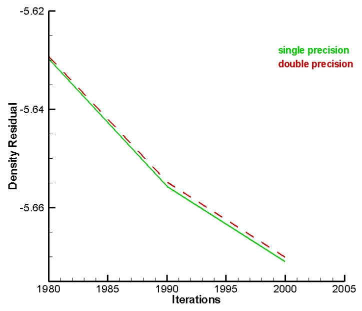
(a) Residual



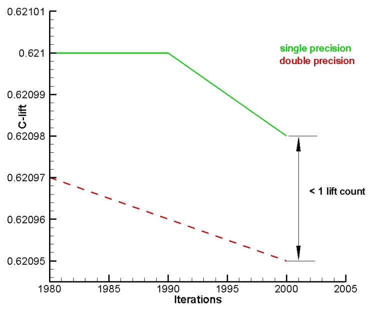
(b) C_L



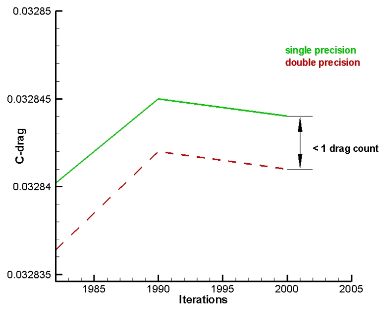
(c) C_D



(d) Residual



(e) C_L



(f) C_D

Figure 1. DLR F6 viscous flow precision study. Complete convergence view on top and closer view on bottom for the density residual and global force coefficients.

the fifth decimal digit (in drag and lift coefficients) is affected by the numerical error due to the operand representation. This comparison has shown that single or double precision computation has to be further considered which is mainly driven by the accuracy requested from CFD engineers, usually less than one drag or lift count and will have mainly an important impact for any hardware-software platform like FPGAs.

IV. Parallelization: Solver scalability using different partitioning algorithms

The second point of activity for improving the application performance is an efficient parallelization. For parallel computing on large unstructured grids domain decomposition is a powerful concept: Given a number P of processors, the whole computational grid is decomposed into P subgrids. Each of the P processors computes on one of the subgrids. Usually communication between the processors is required because the solutions on the subgrids depend on each other. The solver operates in different ways on two in principle different kinds of data types:

- Operations that compute point variables directly from one or more point variables. For these operations no data of other points is required to compute on one point.
- Operations that need data of neighbouring points to compute the result for one point. For these operations connectivity information is required. The main kind of connectivity data used in the solver are the edges. Other connectivity data is given by the boundary faces where a near point is connected to a boundary point. More connectivity data is defined by the grid to grid connections of the different grid levels from the multigrid algorithm. As much as possible of these operations should be performed on one subdomain of the grid without communication.

In this section, we first introduce the current partitioning status of the TAU-Code, then we explore advanced partitioning algorithms and finally we apply them on complex industrial applications.

IV.A. Current partitioning status of the DLR TAU Code

Parallelization in the DLR TAU Code is based on domain decomposition and the message passing concept using MPI. For parallel computations the grids are partitioned in the requested number of domains at the beginning of the flow simulation. Up to now, a simple bisecting algorithm (referred here as Geometric) is employed. The load balancing is performed on point weights which are adjusted for the needs of the solver, which is the most time consuming part of the simulation system.

This partitioner computes the edge cuts according to coordinates. If two partitions have to be computed it is compared if the partitioning at x = const (x is the position on half the way between x-max and x-min) requires less cuts of edges than parallelization cut at y = const or z = const. The best of the three cuts is used. If three domains have to be computed the partitioning is performed by dividing first in 2 subdomains weighted with 1/3 and 2/3. The second subdomain is then divided again in 2 partitions with equal weights. All other numbers of subdomains are computed using the same algorithm recursively; e.g. 7 subdomains are obtained by dividing first in 2 domains weighted with 3/7 and 4/7. The first is then partitioned in 3; the second is two times divided in 2 partitions[15] . The load balancing is performed on point weights based on the amount of edges which finish on each point. These weights try to represent the computation cost of each point in the flow solver. However, the communication cost is not represented in the algorithm, and it is not possible to establish several point weights to deal with different aspects.

For testing purposes, we used the DLR F6 configuration. The mesh decomposition into different domains achieved with the geometric partitioning algorithm is shown in Figure 2 and 3.

Using this domain decomposition, the parallel scalability and speedup of the TAU flow solver for the F6

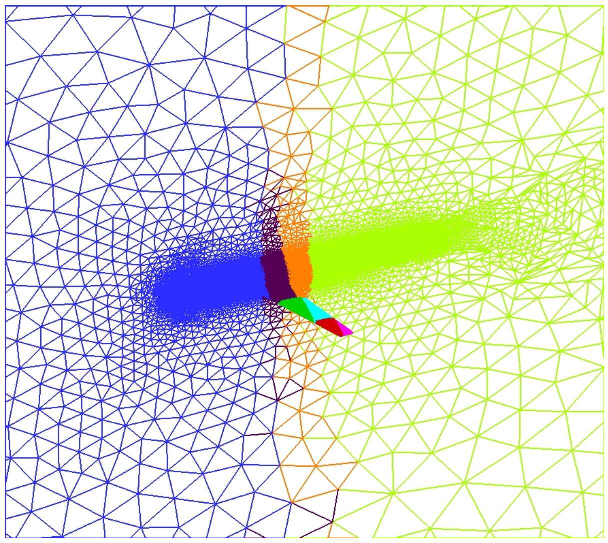


Figure 2. Volume view of the partitioned F6 grid into 8 domains.

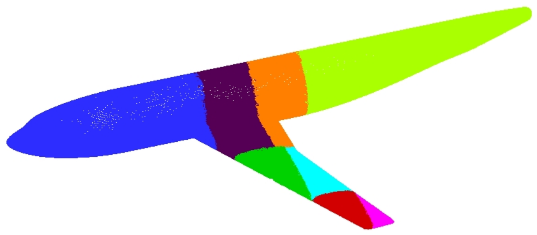
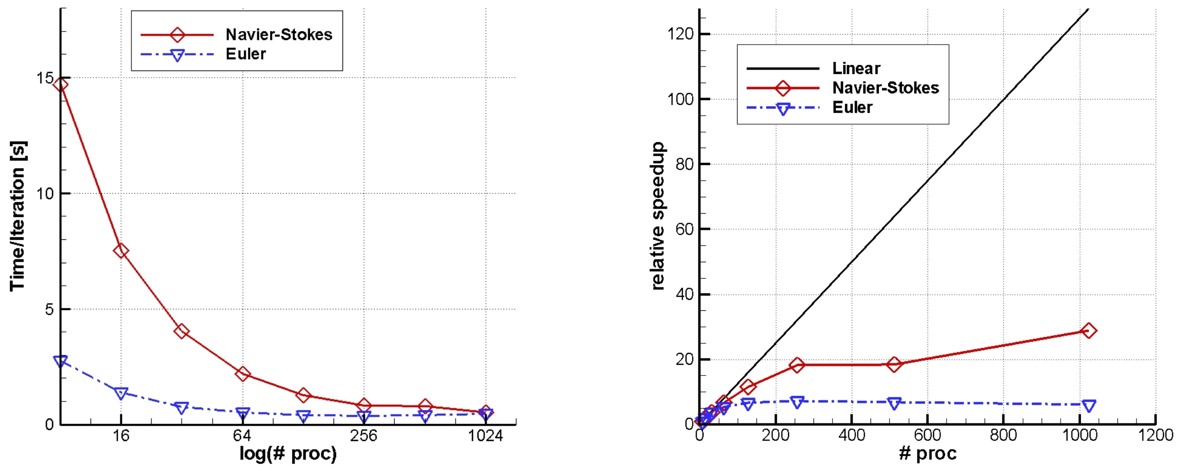


Figure 3. Surface view of the partitioned F6 grid into 8 domains.

configuration, either for an Euler grid with about 1 mill. points (tetrahedrons only) and a Navier Stokes grid with about 6 mill. points (prismatic boundary layer and tetrahedrons) can be observed in Figure 4. The speedup values are computed using as the starting point the execution time for 8 processors. In our cases we consider a saturated speedup whenever the value of it from one computation (using n processors) to another computation (using 2n processors) is less than 1.5 which means that at least half of the new computational resources will be used making effective operations instead of waiting for communication. The speedup of the Euler test-case saturates when using more than 64 processors which implies that the acceptable minimum of points per domain should be more than around 16.000 points. The Navier-Stokes test-case becomes inefficient using more than 256 processors, so the number of points per processor should be higher than approximately 23.500 points.



(a) Parallel scability. (b) Parallel speedup.

Figure 4. Parallel performance of the TAU code over the number of processors.

IV.B. Exploring different partitioning algorithms for the DLR TAU Code

To extend the parallelization study, additional partitioning algorithms are included into the TAU Code using the package ZOLTAN.[9,26] We present a brief overview of the algorithms which will be used in our investigation.

IV.B.1. Recursive Coordinate Bisection (RCB)

RCB was first proposed as a static load-balancing algorithm by Berger and Bokhari[27] , but is attractive as a dynamic load-balancing algorithm because it implicitly produces incremental partitions. In RCB, the computational domain is first divided into two regions by a cutting plane orthogonal to one of the coordinate axes so that half the work load is in each of the sub-regions. The splitting direction is determined by computing in which coordinate direction the set of objects is most elongated, based upon the geometric locations of the objects. The sub-regions are then further divided by recursive application of the same splitting algorithm until the number of sub-regions equals the number of processors. Although this algorithm was first devised to cut into a number of sets which is a power of two, the set sizes in a particular cut needn't be equal. By adjusting the partition sizes appropriately, any number of equally-sized sets can be created. If the parallel machine has processors with different speeds, sets with nonuniform sizes can also be easily generated.

IV.B.2. Recursive Inertial Bisection (RIB)

RIB was proposed as a load-balancing algorithm by Williams[28] and later studied by Taylor and Nour-Omid,[29] but its origin is unclear. RIB is similar to RCB. It divides the domain based on the location of the objects being partitioned by use of cutting planes. In RIB, the computational domain is first divided into two regions by a cutting plane orthogonal to the longest direction of the domain so that half the work load is in each of the sub-regions. The sub-regions are then further divided by recursive application of the same splitting algorithm until the number of sub-regions equals the number of processors.

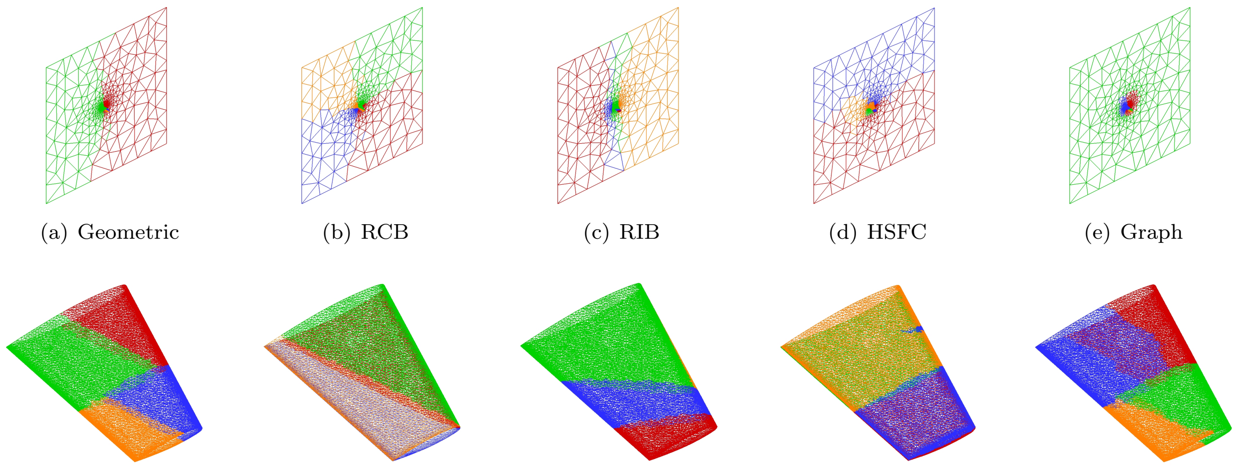
IV.B.3. Hilbert Space-Filling Curve (HSFC)

The Inverse Hilbert Space-Filling Curve functions map a point in one, two or three dimensions into the interval [0,1]. The Hilbert functions that map [0, 1] to normal spatial coordinates are also provided in the ZOLTAN library. (The one-dimensional inverse Hilbert curve is defined here as the identity function, $f(x) = x$ for all x.) The HSFC partitioning algorithm seeks to divide [0,1] into P intervals each containing the same weight of objects associated to these intervals by their inverse Hilbert coordinates. N bins are created (where N > P) to partition [0,1]. The weights in each bin are summed across all processors. A greedy algorithm sums the bins (from left to right) placing a cut when the desired weight for current partition interval is achieved. This process is repeated as needed to improve partitioning tolerance by a technique that maintains the same total number of bins but refines the bins previously containing a cut.

IV.B.4. Graph partitioning

Graph partitioning is a difficult, long-standing computational problem. It has applications to VLSI (Very Large Scale Integration) design, sparse matrix-vector multiplication, and parallelizing scientific algorithms. The general k-way partitioning problem is described by a graph $G(V, E, W_V, W_E)$ where W_V and W_E are vertex and edge weights respectively. The output of partitioning G consists of subsets of vertices, $V_1, V_2, ...V_k$ where $V_i \bigcap V_j = \Phi$. The goal is to balance the sum of vertex weights for each V_i, and minimize the sum of edge weights whose incident vertices belong to different partitions. Graph partitioning can be used to successfully compute high quality partitions by first modeling the mesh by a graph, and then partitioning it into equal parts.

Figure 5 shows the volumetric and surface mesh decomposition into four domains achieved with the Geometric, RCB, RIB, HSFC and the Graph partitioning algorithms. From the top row in Figure 5 looking at the symmetry plane the extensions of the partitioned grid into the volume can be seen.



(a) Geometric (b) RCB (c) RIB (d) HSFC (e) Graph

(f) Geometric (g) RCB (h) RIB (i) HSFC (j) Graph

Figure 5. Volume view (top) and surface view (bottom) of the partitioned Onera M6 grid into 4 domains.

IV.C. Runtime Performance Analysis for different partitioning algorithms.

A comparison between the different partitioning algorithms is performed using complex aircraft configurations to gather experience for industrial applications. We have chosen a high-lift wing-body configuration which was investigated in the European project HiRett[30,31] , see Figure 6 , and a high-lift wing-body-pylon-nacelle configuration with fully operable engines, an ALVAST[32,33] configuration, see Figure 7. Both grids contain a structured prismatic boundary layer region for a viscous flow simulation. The computational grid around the HiRett configuration, Figure 6 has about 13.6 mill. points with approximately 35.2 mill volume elements. The grid around the ALVAST configuration, Figure 7 contains about 13.2 mill. points with approximately 45.3 mill volume elements. Both configurations, HiRett and ALVAST, have deployed slats and flaps and are validation test cases for take-off conditions at low speed.

The main amount of the execution time in a CFD computation is spent in the flow solver. In the TAU-Code a pre-processing step is necessary before running any flow simulation. The pre-processor generates the median dual mesh and metric information, like face normals from the primary grid. The data is then stored in an edge based structure which is used for evaluating the fluxes independently of the different primary volumes used. However, the pre-processing time needs only one to double of the time of one flow solver

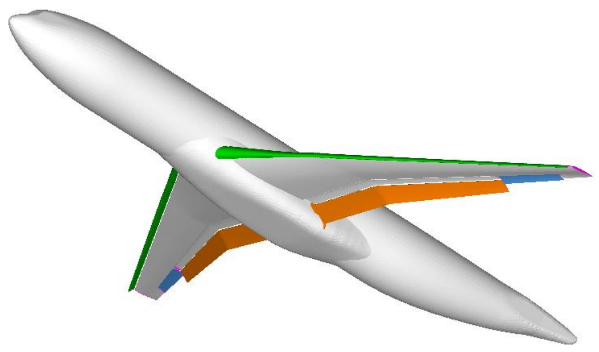


Figure 6. HiRett wing-body configuration with high-lift devices and ailerons.

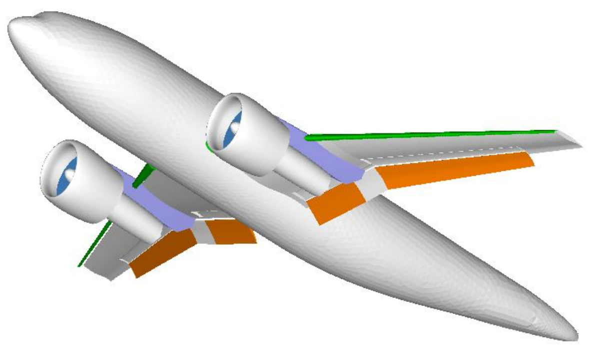


Figure 7. ALVAST High-Lift configuration with fully operable turbines.

iteration, so it is usually a negligible amount in comparison to the flow solver execution time in which we are focusing on.

As the pre-processing step is necessary, we also studied how the domain decomposition affects the execution time. Based on the tests we performed, it can be stated that there is an improvement when using Graph instead of Geometric partitioning algorithms especially with a high number of processors.

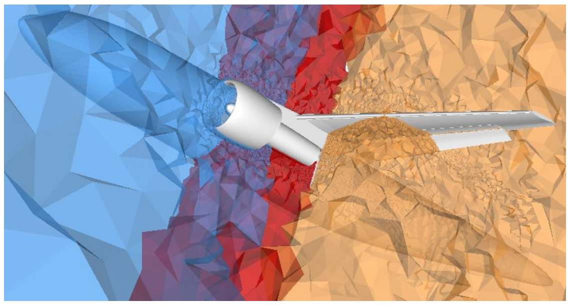


Figure 8. Volume view of the partitioned ALVAST grid into 8 domains using a Geometric partitioner.

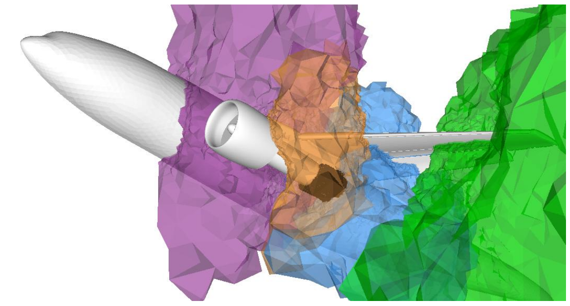


Figure 9. Volume view of the partitioned ALVAST grid into 8 domains using a Graph partitioner.

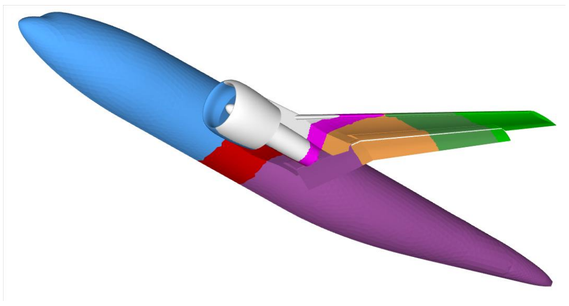


Figure 10. Surface view of the partitioned ALVAST grid into 8 domains using Geometric partitioner.

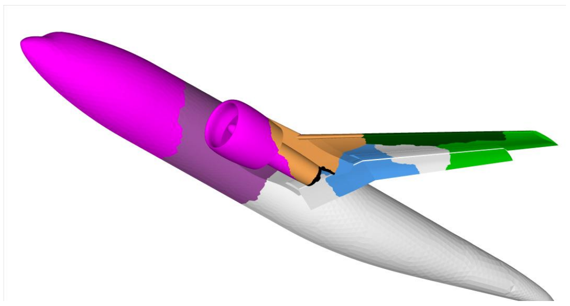


Figure 11. Surface view of the partitioned ALVAST grid into 8 domains using a Graph partitioner.

Figure 8 and 9 show the differences between the volumetric mesh decomposition for the ALVAST configuration using both Geometric and Graph partitioning. The surface mesh decomposition can be seen in Figure 10 and 11. The critical part in the ALVAST grid is the decomposition of the inflow and outflow of the engines. To keep the physical behavior of fan turbines effective, a mass flow coupling is needed which introduces an increased communication. Focusing on this subject, it can be observed for low domains that the Geometric partitioner divides the inflow and outflow area into different domains while the Graph one keeps the inflow in one domain and the outflow entirely in one domain.

Regarding to the flow solver, the computations were performed with a Spalart Allmaras one-equation turbulence model with Edwards modification (SAE) and the central spatial discretization scheme with scalar dissipation and a $4w$ multigrid cycle. The flow conditions of the considered configurations are displayed in Table 7 .

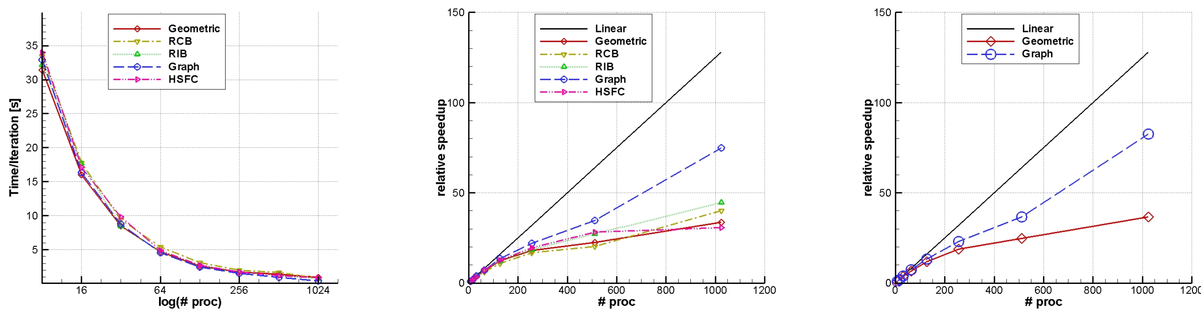
Table 7. Flow conditions for ALVAST and HiRett configurations.

	Mach number	Reynolds number	Angle of Attack []
HiRett	0.2	$25 * 10^6$	19.0
ALVAST	0.182	$1.66 * 10^6$	4.3

While both configurations are created for take-off conditions the purpose of each was very different as may be seen from the flow conditions. The HiRett model simulates an airplane shortly before reaching the maximum angle of attack in authentic flow conditions and the ALVAST model was used in a wind tunnel experiment with a substantial reduced Reynolds number.

Since the solver appears to react different to the many possible time and spatial integration schemes and unfortunately testing all these possibilities would overcome a certain expense, therefore we will concentrate on an explicit Runge-Kutta and a semi-implicit LUSGS scheme. The first association for the explicit time integration is a common scalability over a wide range of processors using the same CFL (Courant-Friedrich-Levy) number. In comparison, the semi-implicit time integration performs for the flow field implicit but treats the boundary conditions explicitly. In relation to a sequential computation where the percentage of boundary points to flow field points usually ranges between 5 to 10 percent, this relation will vary enormous for a high number of processes. While some grid domains may have only flow field points and can be integrated entirely implicitly others may contain many boundary points and stability problems can appear.

The approach for speedup was done in the same manner as in section IV.A where the minimum size of eight grid partitions were restricted by the amount of memory for the used Linux cluster and was taken as the first point for the linear speedup. The results in Figure 12 show that improvements in the scalability (left) and the solver speedup using a Runge-Kutta (middle) and a LUSGS (right) time integration scheme can be obtained through Graph partitioning algorithms.



(a) Runge-Kutta (b) Runge-Kutta (c) LUSGS

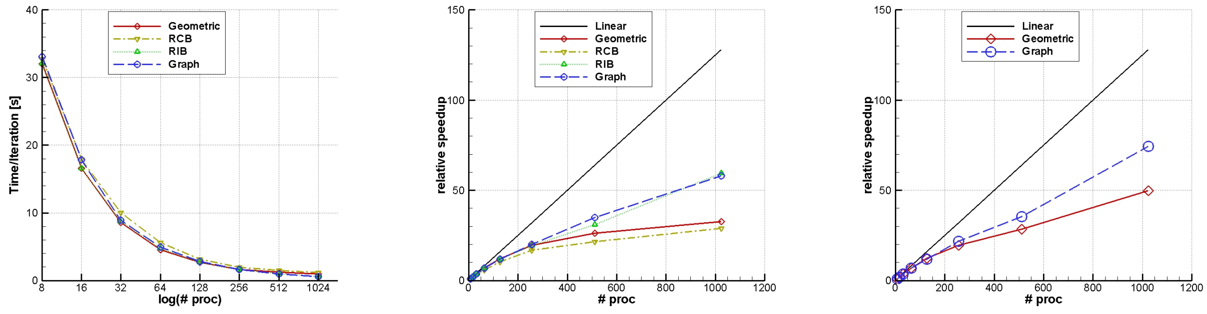
Figure 12. Parallel efficiency (left) and speedup (middle/right) of the TAU Solver using different partitioning algorithms for the HiRett test-case.

Table 8. Solver execution time per iteration using different partitioning algorithms for HiRett with Runge-Kutta and LUSGS.

Partitioner	Time integration	8	16	32	64	128	256	512	1024
Geometric	RK	31.42	16.03	8.49	4.68	2.57	1.74	1.39	0.93
	LUSGS	30.15	15.09	8.22	4.58	2.54	1.6	1.22	0.81
RCB	RK	33.45	17.82	9.4	5.4	3.12	1.98	1.65	0.83
RIB	RK	32.11	17.64	8.46	4.74	2.64	1.73	1.18	0.72
HSFC	RK	34.01	17.13	9.8	4.87	2.69	1.74	1.2	0.85
Graph	RK	32.91	16.37	8.76	4.56	2.45	1.51	0.95	0.44
	LUSGS	31.89	15.95	8.4	4.41	2.39	1.39	0.87	0.39

Analyzing Table 8, we can see for the first time a surprising trend. Below 128 processors the time per iteration for the Geometric partitioner is smaller in comparison to the Graph partitioner while it becomes, as expected, vice versa from 128 up to 1024 processors. One explanation may be because of dividing the domains by elements for the Geometric one and by edges for the Graph one. Imagine a homogeneous grid with hexahedrons and the smallest partition would include one central hexahedron and four neighboring elements on each side which will contain 24 points in the domain. The smallest partition for the Graph will result in only 11 edges which introduces more communication whenever there is a small number of partitions. For a high number of domains firstly this relation becomes more equal due to contributions of boundaries in each domain for the Geometric one (for boundaries it will result in about 20 points) and secondly to the smaller hull, compare Figure 8 and Figure 9 of each domain for the Graph partitioner which decreases the communication load.

The performance results for the ALVAST case, Figure 13, show again improvements in the scalability (left) and the solver speedup (middle) using a Graph instead of Geometric partitioner for Runge-Kutta (middle) and LUSGS (left).



(a) Runge-Kutta (b) Runge-Kutta (c) LUSGS

Figure 13. Parallel efficiency (left) and speedup (middle/right) of the TAU Solver using different partitioning algorithms for the ALVAST test-case.

Table 9. Solver execution time per iteration using different partitioning algorithms for ALVAST with Runge-Kutta and LUSGS.

Partitioner	Time integration	8	16	32	64	128	256	512	1024
Geometric	RK	32	16.56	8.57	4.56	2.68	1.63	1.22	0.98
	LUSGS	31.59	16.16	8.45	4.58	2.59	1.62	1.11	0.63
RCB	RK	32.73	17.9	10.05	5.6	3.17	1.94	1.53	1.14
RIB	RK	32.08	16.61	8.74	4.71	2.7	1.66	1.04	0.54
Graph	RK	33.07	17.83	8.92	4.91	2.81	1.65	0.95	0.57
	LUSGS	32.67	17.51	8.77	4.77	2.78	1.51	0.92	0.44

In comparison to the HiRett case Figure 12 the speedup of the ALVAST case, Figure 13 shows a visible lack of the speedup due to the additional communication for the engine mass coupling. Concerning the time used in parallelization mode there are two main parts which rely firstly on setting up the communication between two processors and the time used for transferring an amount of data. These two times may vary enormous between less to many partitions. Partitions with a very low number of points per partition spend a high amount of time for setting up the complex socket communication. Engine mass coupling introduces in the simple way only one integral value or in a more complex way a partial mass flow over the inflow and outflow faces. This very small amount of data which have to be sent over domains is predominately driven by setting up the additional communication for a high number of processors.

The maximum allowable CFL number for the explicit three stages Runge-Kutta scheme with a $4w$ multigrid cycle remained unchanged throughout all simulations for both cases. Unlike the semi-explicit LUSGS scheme for the ALVAST computations for 512 and 1024 partitions. The CFL number had to be decreased by half of the CFL number as used for 8 domains to maintain a converging simulation.

From the execution times obtained in Table 8 and Table 9 we can see that even slight improvements per iteration could provide a significant enhancement in case of complete simulations over a huge number of processors.

Finally, the convergence history obtained for the HiRett case, using 8 and 64 processors is seen in Figure 14. Independently of the number of processors the density residual and integral force coefficients

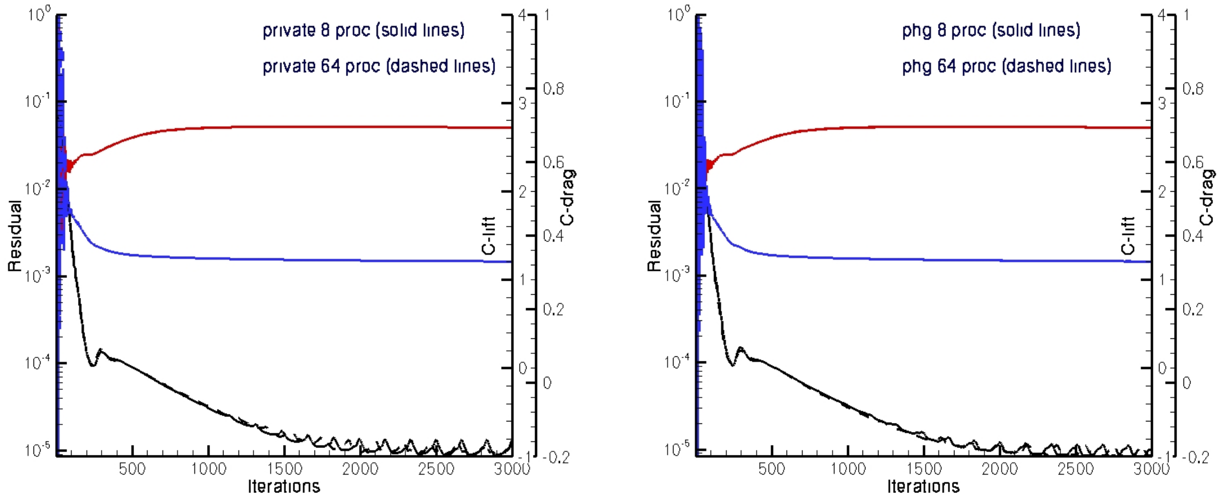


Figure 14. Solution comparison using Geometric and Graph for 8 (left) and 64 processors (right)

should converge to the same values theoretically when using the same flow parameters for the simulation. The lift and drag coefficients are the same which is reassuring but the density residual exhibits a slightly non matching behavior between different processors. Flow phenomena, like separation or shocks are very sensitive with respect to slight but unforeseen algorithmic changes on partitioned boundaries.

The advanced partitioning algorithms pay off for their efficiency whenever the computational work, which is related to the number of points per domain, is higher than the communication load. As expected, the Graph partitioner shows even for a high number of processors a speedup closer to the linear speedup while the Geometric one saturates very soon at around 200 for HiRett and around 250 processors for ALVAST.

All of the new partitioning algorithms used with TAU have some important features that can be adapted for complex applications. RCB, RIB and Graph permit dynamic load balancing and Graph permits also multiconstraint partitioning to partition a graph in the presence of multiple balancing constraints. The idea is that each vertex has a vector of weights of size m associated with it, and the objective of the partitioning algorithm is to minimize the edgecut subject to the constraints that each one of the m weights is equally distributed among the domains.
This feature is important when the architecture of the computation platform is heterogeneous, as occurs in hardware acceleration and will be subject of further investigations.

V. Acceleration: A Feasibility Study on the Application of FPGAs and GPUs for the Hardware Acceleration of CFD Simulations

The aeronautic industry requirements in the near future will not be accomplished by the usual evolution of current processors and architectures. Indeed, it is estimated that to obtain "engineering-accuracy" predictions of surface pressures, heat transfer rates, and overall forces on asymmetric and unsteady turbulence configurations, the grid size will have to be increased to a minimum of 50-80 million nodes. This means that the simulation of a complete aircraft configuration will require several days to obtain solutions in a high performance cluster with hundreds of processors, so an exponential increment of the computational requirements is estimated.
Therefore, the improvements expected at optimization and parallelization levels are not enough, and it will be necessary to introduce new concepts in typical simulation platforms in order to satisfy these demands

and to face more complex challenges.

As hardware acceleration is a very innovative task, in this chapter, some on going activities and feasibility studies in several projects will be introduced.

Two alternatives to speedup the performance of the cluster of PCs are quickly emerging due to recent technological advances.

The first one is based on Graphics Processing Units (GPU)[34–36] as computing resources that complement the processors in the PCs. These general-purpose devices have evolved to constitute powerful pipelined vector processors capable of delivering very high throughputs in optimal conditions at a relatively low cost. The effective improvements that can be obtained from this technology in CFD simulations remain to be demonstrated and will be partially analyzed in this section.

The second alternative is based on reconfigurable hardware, in particular, Field Programmable Gate Arrays (FPGA) as computing resources in accelerator boards connected to the PCs of the cluster. This technology uses a completely different approach to the problem. Instead of general-purpose resources, FPGAs provide application-specific data paths that are optimized to compute the critical parts of the CFD algorithms. The optimization includes the possibility of both, space and time parallelism so the implemented circuits are capable of delivering throughputs well above those provided by general-purpose processors. The overall idea with FPGAs is to introduce a new level of parallelism in the system architecture. While the PCs in the cluster implement coarse-grained parallelism resulting from partitioning the dataset, the specialized hardware in the accelerators implements fine-grain parallelism at the operation level, accelerating the set of operations assigned to each local processor (parallelism in the hardware functional units that provide several results at the same time)[13] .

One main idea is that all the computational nodes in the cluster are FPGA-powered, so load balance has to be taken into account. This means that the subdomain decomposition process has to assign to the FPGA-powered computational nodes an adequate workload in order to support efficient synchronization in the overall system. Due to the heterogeneous nature of the architectures existing in a cluster platform, including differences in performance between the general-purpose CPUs, special considerations about global synchronization have to be explored. The partitioning algorithm has to allow a multiconstraint partitioning with at least 2 weights per mesh node, one related to each level of paralellism. The new graph partitioning algorithm that has been linked to be used in TAU, permits also multiconstraint partitioning to partition a graph in the presence of multiple balancing constraints. Each vertex can have a vector of weights of size m associated with it, and the objective of the partitioning algorithm is to minimize the edgecut subject to the constraints that each one of the m weights is equally distributed among the domains. It is assumed that at least two weights should be considered (for the fine and coarse level of parallelism) in the case of homogeneity between the type of processors and HW accelerators.

Of course, porting a critical part of an algorithm to an FPGA implies a complete design cycle, but once completed, the design is ready to be loaded at any time in any FPGA. Since FPGAs are reconfigurable, different designs can be loaded at different times depending on the application needs, making FPGAs very interesting devices to act as coprocessors in computation intensive applications.

FPGAs also have some limitations. In particular, there is a limit on the amount of computations that can be performed in an FPGA at maximum throughput, depending on the available resources in the FPGA. When this limit is exceeded, resources must be shared among computations and performance reduces. Also, the communications interface between the PC memory and the FPGA is a major concern as the available bandwidth can limit the effective global performance. From these considerations, it can be observed that FPGAs are best adapted to applications where most computational effort concentrates on a small portion of the code which is repeatedly executed for a large dataset. In addition, the input/output (I/O) communications load has to be small when compared to the computational load, to avoid saturating the PC-FPGA interface. CFD codes seem to fullfill these requirements and, therefore, appear as good candidates to benefit from FPGA-based accelerators.

In this section, a preliminary performance analysis is carried out for different scenarios, based on detailed error analyses and FPGA syntheses. CPU-, GPU-, and FPGA-based system performances are drawn and compared to establish the feasibility of FPGA-based acceleration.

V.A. The architecture: A cluster of PCs with accelerator boards

This subsection is a short review of the state of the art of the specific technologies involved in the architecture. First, the families of high-performance FPGAs available in the market are presented. Second, the interface between the PC and the FPGA is shown, with special focus on the PCI Express (PCI-e) interface which appears as one of the most promising alternatives. Finally, the main characteristics of today's GPUs are briefly introduced.

V.A.1. Field Programmable Gate Arrays (FPGAs)

Field Programmable Gate Arrays (FPGAs) are reconfigurable devices that, over the years, have gained the reputation of being the most used devices for complex system prototyping. With the addition of high-speed I/O and highly-efficient embedded processing and memory blocks, FPGAs are also gaining acceptance as relatively low-cost devices in production systems where high-performance, and not only flexibility, is required. Among the FPGA manufacturers, Xilinx Incorporated and Altera Corporation comprise the fabrication of more than 90% of such devices. These two big companies offer different types of FPGAs, but in both cases they distinguish between high-performance and low-cost families of FPGAs. Table 10 summarizes the characteristics of the most advanced families of FPGAs from these companies. The columns show the total amounts of user I/O pins, configurable logic cells, and dedicated-memory bits.

Table 10. Summary of characteristics of the most recent families of FPGAs.

Manufacturer	Family	I/O pins	Logic cells ($x10^3$)	Memory (Mb)
XILINX	Virtex-5 LX	400-1200	30.7-331.7	Up to 10.3
XILINX	Virtex-5 LXT	172-960	19.9-331.7	Up to 11.6
XILINX	Virtex-5 SXT	360-640	34.8-94.2	Up to 8.8
ALTERA	Stratix III-L	288-1104	47.5-338	Up to 16.3
ALTERA	Stratix III-E	288-960	47.5-254.4	Up to 14.7

V.A.2. High-speed interfaces

Complex systems are always demanding higher communication bandwidths. In order to cope with these demands, modern standards have successively increased the transmission frequency and/or the number of wires of the interfaces to provide greater capacities. Table 11 shows some of the fastest and most relevant interfaces proposed in these years, and their effective capacity.

Table 11. Transmission capacity of current high-speed interfaces.

Interface	Capacity (Gbps)
PCI Express (8 lane)	16.000
AGP 8x	17.066
PCI-X DDR	17.066
RapidIO (16 lane)	20.000
HyperTransport (1 GHz, 16-pair)	32.000
PCI Express (16 lane)	32.000
AGP 8x 64-bit	34.133
PCI Express (32 lane)	64.000
PCI Express 2.0 (32 lane)	128.000
HyperTransport (2.8 GHz, 32-pair)	179.200

PCI-e is one of the most suitable interfaces in the architecture for CFD simulation. The main reasons for this are its flexibility of configuration and its capacity, together with the availability of FPGA development boards with PCI-e support from both major manufacturers, Xilinx and Altera. However, some FPGA-based coprocessor solutions using HyperTransport have recently appeared in the market. DRC Computer offers its family of RPUs (Reconfigurable Processor Units), based on Virtex-4 devices, that connects to AMD Opteron systems through an 16-pair HyperTransport interface at 400 MHz (maximum 12.8 Gbps). Other solution is provided by XtremeData Inc. and its family of in-socket accelerators based on Altera's

Stratix-II FPGAs. These coprocessors plug directly into specific AMD sockets and connect through a 16-pair HyperTransport bus at 800MHz (maximum 25.6 Gbps). Other alternative is provided by SGI, that offers its RASC (Reconfigurable Application Specific Computing) technology, based on Virtex-4 FPGAs and its proprietary NUMAlink 4 interface, which is specially targeted for shared memory multiprocessing, supporting 51.2 Gbps.

V.A.3. Graphics Processor Units (GPU)

Traditionally, GPUs have been used as rendering units and their range of application has been limited to graphics processing. Mainly, GPUs have provided approximations to physical models (water caustics, light reflection, shadows, etc.) to give a more realistic feel to video games. In modern graphics cards, GPUs have evolved to become a cost effective computation workhorse. Multiple independent pipelines and vector arithmetic make GPUs a powerful parallel processing platform. Recent additions of full programmability and IEEE-standard floating point arithmetic finally open the field to general purpose intensive computations in the so-called GPGPUs. However, GPUs are not suited for all kinds of general purpose applications due to its lack of support for double-precision data types (64-bit GPU devices providing one third of the performance of current 32-bit GPUs have been announced). Another possible drawback is that the rapid evolution of these devices which lacks architectural continuity (algorithms which are optimized for some GPU architecture may perform poorly in a different GPU architecture).

Although other big manufacturers have announced GPGPU products, nVidia is the major manufacturer today in the market. Table 12 summarizes the characteristics of some of the most relevant nVidia GPUs. It can be observed that multiple clocks are used in the units.

Table 12. Main characteristics of GPU devices from nVidia

GPU	Clock Core (MHz)	Clock Shader (Mhz)	Memory Size (MB)
GeForce 8400 GS	450	900	256
GeForce 8800 GT	600	1,500	512
GeForce 8800 GTX	575	1,350	768
GeForce 8800 Ultra	612	1,500	768
Tesla C870	575	1,350	1,500
Tesla D870	2 * 575	2 * 1,350	3,000
Tesla S870	4 * 575	4 * 1,350	6,000

V.B. Preliminary performance analysis

The main goal of this study, apart from establishing the technological feasibility of FPGA or GPU -based hardware accelerators for CFD computing, is to determine the speedups that can be achieved. In this sense, several issues need to be addressed. First, an error analysis is carried out to determine the data representation formats to be used in the FPGAs. Then, a performance study is conducted for different FPGA and GPU configurations in order to determine the effective performances that can be achieved for Euler's algorithm in systems with limited bandwidth in the coprocessor interface are analyzed[37] .

The reference for performance evaluation are the execution times of the software implementation of Euler's algorithms running in a PC. They are measured in a PC with an AMD Athlon dual core processor 4800+ at 2.51 GHz and 1 GB of RAM. Computation times are measured as true CPU times (i.e. using C functions of the system library) in order to avoid the bias produced by other system processes in stopwatch measurements. GPU performace values to be used as reference are also drawn, either measured using an nVidia GeForce 8400 GS, or estimated for other GPUs from these measurements. In the case of FPGAs, a Virtex V LX110T has been selected.

V.B.1. Error analysis

FPGA performance depends on the latencies of the internal functional units, which, in turn, depend on the word-lengths assigned to their operands in the quantization design phase. For this reason, it is necessary to determine the error introduced by different uniform word-lengths, and to compare it to the overall precision requirement. Error analysis is performed through simulations of the algorithm specification modified for hardware implementation. Errors are measured in terms of the standard deviation σ_e (i.e. the square root of the variance) of the differences between the values obtained using constrained data types and reference values. In particular, measured errors are provided as $\log_{10}(\sigma_e)$ so they directly relate to the fractional decimal digit affected by the error.

A first set of simulations is run in order to determine the errors introduced by word-length constraints only in Roe's computation. The objective is to determine the range of precisions that can be expected from specific data types in fixed-point and floating-point formats. A trace-based simulation approach is followed, where input reference traces of the conservative variables in each iteration are used to obtain output traces of the computed fluxes. A grid with 10^3 cells is simulated during 10^3 iterations for a total of 10^6 Roe's computations. Tables 13 and 14 present the results obtained for the three output components, f1, f2, and f3 (inviscid fluxes). In 14 , word-lengths are described in terms of the bits of the mantissa and the exponent of the floating-point format (i.e. the type 18/6 uses 24 total bits).

Table 13. Approximate errors expressed as $\log_1 0(\sigma_e)$ for different fixed-point data types.

Output variable	Word-length (bits)							
	8	16	24	32	40	48	56	64
f1	-2.62	-4.93	-7.36	-9.86	-12.26	-14.67	-17.07	-19.47
f2	-2.32	-4.71	-7.09	-9.53	-11.93	-14.33	-16.76	-19.16
f3	-1.93	-4.26	-6.64	-9.17	-11.56	-13.97	-16.35	-18.76

Table 14. Approximate errors expressed as $\log_1 0(\sigma_e)$ for different floating-point data types

Output variable	Mantissa/exponent lengths (bits)							
	18/6	20/4	24/8	28/4	32/8	40/8	46/10	53/11
f1	-5.45	-6.04	-7.22	-8.40	-9.61	-12.02	-13.82	-15.97
f2	-5.14	-5.74	-6.90	-8.09	-9.29	-11.70	-13.51	-15.64
f3	-4.75	-5.35	-6.49	-7.67	-8.88	-11.29	-13.09	-15.23

In particular, 8- and 16-bit fixed-point types produce errors that show in the third (or second in f3) and fifth digits of the computed fluxes, respectively, so they are discarded. Also, word-lengths above 40 bits provide too much precision (errors show below the 12th digit) which is considered unnecessary, so they are also discarded from further study. Results for the floating-point format show that, not only type 18/6 and types with total word-length above 48 bits are also discarded for the reasons above, but, mainly, that this format generates larger errors than the fixed-point format for the same total bits. In fact, it produces errors comparable to those produced by the fixed-point types with the word-length of the floating- point mantissa. In other words, floating-point errors reduce if more bits are assigned to the mantissa thus reducing the exponent bits.

As a conclusion, simulations indicate that a minimum fixed-point word-length of around 24 bits should be used for the hardware implementation. Similarly, a minimum of around 20/4 mantissa/exponent bits should be used in floating-point types.

V.B.2. FPGA synthesis results

Table 15 summarizes the reported synthesis values of the fixed-point and floating-point designs.

Table 15. Design estimates for maximum throughput.

Data format	Word-length	Bit slices	DSP cores	Max Frequency
Fixed	24	17,108	21	290.2
Fixed	32	38,740	22	265.4
Fixed	40	49,676	44	219.7
Floating	24/8	69,772	0	263.8
Floating	24/8	56,960	42	249.3
Floating	24/8	62,860	64	244.1

The advantages of using shorter word-lengths are clearly shown. Not only higher processing speeds are obtained (the 24-bit fixed-point design can perform $290 * 10^6$ flux computations per second), but reduced I/O rates are required for such maximum speeds. Floating-point designs almost perform as fast for equivalent word-lengths (24/8-bit vs 32-bit).

Fixed point format provides improved performance and reduced area but however, floating point format is not discarded because it provides feasible design (using available Xilinx IP cores) and can be required in case of large dynamic range (in Navier Stokes).

V.B.3. Effective performance analysis

Performance has been analyzed using the previous FPGA values together with reference values measured for CPU and GPU devices. The performance analysis for the module is shown in Table 16.

Table 16. Speedups obtained for Euler's algorithm.

Processor	Format	Bits	Cores	I/O (Gbps)	Speedup			
					x8	x16	x32	Max
Athlon X2 4800+	Floating-point	53/11	-	0.6	1	1	1	1
GeForce 8800 GT	Floating-point	24/8	-	11.1	34	34	34	34
GeForce 8800 Ultra	Floating-point	24/8	-	12.9	40	40	40	40
Tesla S870	Floating-point	24/8	-	48.5	50	99	150	150
Virtex-5 LX30T	Fixed-point	24	1	20.9	66	86	86	86
Virtex-5 LX110T	Floating-point	24/8	1	25.3	50	79	79	79
Virtex-5 LX110T	Fixed-point	40	1	26.4	40	65	65	65
Virtex-5 LX110T	Fixed-point	32	1	25.5	50	79	79	79
Virtex-5 LX110T	Fixed-point	24	3	62.7	66	132	259	259
Virtex-5 LX330T	Floating-point	24/8	3	75.9	50	99	199	236
Virtex-5 LX330T	Fixed-point	40	4	105.6	40	79	159	262
Virtex-5 LX330T	Fixed-point	32	5	127.5	50	99	198	395
Virtex-5 LX330T	Fixed-point	24	6	188.1	66	132	265	778

The acceleration with specific hardware is feasible today and generates even better expectations for the future. Expected interface developments in the near future could multiply the current FPGA figures by a factor of four. Ultimately, the processing limitations of the accelerators for Euler rate are much higher, with maximum GPGPU speedups of 150, and maximum FPGA speedups above 770.

VI. Conclusions

The improvement of the code efficiency has been addressed through optimization by applying different tuning strategies to reduce the execution time of the unstructured DLR TAU solver. Significant computational gains (around 11% for RK and 4% for LUSGS for a 64 bits machine) have been achieved and it makes clear the necessity of code profiling and optimization involving multidisciplinary knowledge to reduce the execution time and memory consumption. Regarding with value precision, a study of single 32 bits and double 64 bits precision has shown an important improvement (specially in memory consumption) using a reduced precision if there are not significant differences in the solution.

Additionally, the analysis of different algorithms for parallelization to make a more efficient grid partitioning has been performed together with a feasibility study and analysis of the effective performance of the hardware acceleration. Several partitioning algorithms have been included in the TAU code, and high parallel simulations using up to 1024 processors have been performed to test the efficiency and scalability of the selected new algorithms for industrial configurations. The conclusion here is that the graph partitioner algorithm maintains the speedup much closer to the linear one for a high number of processors in all the tests performed.

Moreover, the current status of the development of a mixed hw-sw computation platform for simulation has been presented focusing in precision and speedup estimations which shows a promising technology when the precision format required is limited. Word lengths must be determined by error analysis and precision requirements and they will affect the final speedup achieved. Future efforts in hardware acceleration will be performed to achieve a complete implementation with optimal balance between area and performance and a mixed hardware-software cluster architecture to test the effective acceleration into a simulation platform.

Acknowledgments

The research described in this paper made by INTA and DLR researchers has been supported under the INTA activity Termofluidodinamica (INTA's code: IGB4400903), by the Spanish Ministry of Education and Science (MEC) under the DOMINO Project(CIT-370200-2007-22) and under the high performance activity for the TAU code at $C^2A^2S^2E$ at the Institute of Aerodynamics and Flow Technology in Braunschweig. In particular, this work has been possible due to the close cooperation between the two organizations.

References

[1]Jameson, A., "Computational aerodynamics for aircraft design." *Science Magazine*, Vol. 245, 1989, pp. 361 – 371.

[2]Kroll, N., Gerhold, T., Melber, S., Heinrich, R., Schwarz, T., and Schöning, B., "Parallel Large Scale Computations for Aerodynamic Aircraft Design with the German CFD System MEGAFLOW," *Parallel Computational Fluid Dynamics - Practice and Theory*, 2002.

[3]Mavriplis, D., "Unstructured Mesh Discretizations and Solvers for Computational Aerodynamics." 18th *Computational Fluid Dynamics Conference*, AIAA, Miami, FL, 25-28 June 2007.

[4]Mavriplis, D., "Parallel Performance Investigation of an Unstructured Mesh Navier-Stokes Solver." *International Journal of High Performance Computing*, Vol. 2(16), 2002, pp. 395–406.

[5]Schwamborn, D., Gerhold, T., and Heinrich, R., "The DLR Tau-code: recent applications in research and industry." *Proceeding of ECCOMAS CFD 2006*, Egmond aan Zee, Netherlands, 5.-8. September 2006.

[6]Alrutz, T., "Investigation of the parallel performance of the unstructured DLR-TAU-Code on distributed computing systems." *Proceedings of Parallel CFD 2005*, edited by Elsevier, Elsevier, Washington, DC (USA), May 2005, pp. 509 – 516.

[7]Wylie, B. J. N., Geimer, M., Nicolai, M., and Probst, M., "Performance analysis and tuning of the XNS CFD solver on BlueGene/L." *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Vol. LNCS 4757 of *Proceedings of the 14th European PVM/MPI User's Group Meeting, Paris, France*, Springer, 2007, pp. 107–116.

[8]Gropp, W. D., Kaushik, D. K., David, Ý., Keyes, E., and Smith, B. F., "Performance modeling and tuning of an unstructured mesh CFD application." *In Proceedings of SC2000. IEEE Computer Society*, 2000, pp. 0 – 7803.

[9]"ZOLTAN Parallel Partitioning, Load Balancing and Data-Management Services," http://www.cs.sandia.gov/Zoltan.

[10]Sillen, M., "Evaluation of Parallel Performance of an Unstructured CFD-Code on PC-Clusters." *17th AIAA Computational Fluid Dynamics Conference*, AIAA, Toronto, Ontario, Canada, 2005.

[11]Puttegowda, K. and Worek, W., "A Run-time Reconfigurable System for Gene-sequence Searching." *Proceedings of the International VLSI Design Conference*, 2003.

[12]Mahmoud, B., Bedoui, M. H., Essababah, R., and Essabbah, H., "Nuclear medical image treatment system based on FPGA in real time." *International Journal of Signal Processing*, 2004.

[13]Andres, E., Carreras, C., Caffarena, G., Nieto, O., and Palacios, F., "A Methodology for CFD Acceleration Through Reconfigurable Hardware." *46th AIAA Aerospace Sciences Meeting and Exhibit*, edited by AIAA, AIAA, Reno, Nevada, Jan. 7-10 2008.

[14]Strenski, D., "Computational Bottlenecks and Hardware Decisions for FPGAs." *FPGA and Structured ASIC Journal*, 2006.

[15]various, "Technical Documentation of the DLR TAU-Code." Tech. rep., Institut of Aerodynamics and Flow Technology, 1994-.

[16]Gerhold, T., Galle, M., Friedrichs, O., and Evans, J., "Calculation of Complex Three-Dimensional Configurations employing the DLR TAU-Code." Aiaa-97-0167, AIAA, 1997.

[17]"CentaurSoft," http://www.centaursoft.com.

[18]"Python Open Source," http//:www.python.org.

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

[20]Edwards, J. R., "Comparison of eddy viscosity-transport turbulence models for three dimensional, shock-separated flowfields." *AIAA Journal*, Vol. 34, No. 4, 1996, pp. 756 – 763.

[21]Schmitt, V. and Charpin, F., "Pressure Distributions on the ONERA M6 Wing at Transsonic Mach Numbers." Experimental Database for Computer Program Assessment. AGARD-AR-138, pp. B1-1-B1-44, AGARD, May 1979.

[22]Brodersen, O., Monsen, E., Ronzheimer, A., Rudnik, R., and Rossow, C. C., "Computation of aerodynamic coefficients for the DLR-F6 configuration using MEGAFLOW," *New Results in Numerical and Experimental Fluid Mechanics II*, Vol. 72, 06 1999, pp. 85 – 92.

[23]Brodersen, O. and Stürmer, A., "Drag Prediction of Engine - Airframe Interference Effects using Unstructured Navier-Stokes Calculations." Aiaa-2001-2414, AIAA, 2001.

[24]Laflin, K. R., Klausmeyer, S. M., Zickuhr, T., Vassberg, J. C., Wahls, R. A., Morrison, J. H., Brodersen, O., Rakowitz, M. E., Tinoco, E. N., and Godard, J.-L., "Data Summary from Second AIAA Computational Fluid Dynamics Drag Prediction Workshop." *Journal of Aircraft*, Vol. 42, No. 5, 2005, pp. 1165–1178.

[25]Vassberg, J. C., Tinoco, E. N., Mani, M., Brodersen, O. P., Eisfeld, B., Wahls, R. A., Morrison, J. H., Zickuhr, T., Laflin, K. R., and Mavriplis, D. J., "Summary of the Third AIAA CFD Drag Prediction Workshop." Aiaa paper 2007-0260, AIAA, Reno, Nevada, January 2007.

[26]Boman, E., Devine, K., Heaphy, R., Hendrickson, B., Leung, V., Ann, L., Vaughan, C., Catalyurek, U., Bozda, D., and Teresco, J., "Zoltan Library User Guide," Agard, Sandia National Laboratories, Inc., 2007.

[27]Berger, M. J. and Bokhari, S. H., "A partitioning strategy for nonuniform problems on multiprocessors." *IEEE Transactions on Computers*, Vol. C-36(5), May 1987, pp. 570–580.

[28]Shephard, M. S., Flahertya, J. E., de Cougny, H. L., Ozturan, C., Bottasso, C. L., and Beall, M. W., "Parallel automated adaptive procedures for unstructured meshes." AGARD R-807, pages 6.1-6.49, In Parallel Comput. in CFD, Neuilly-Sur-Seine, 1994.

[29]Taylor, V. E. and Nour-Omid, B., "A Study of the Factorization Fill-in for a Parallel Implementation of the Finite Element Method." *International Journal for Numerical Methods in Engineering*, Vol. 37, 1995, pp. 3809–3823.

[30]Krumbein, A., "personal communication," .

[31]Rakowitz, M., Heinrich, S., Krumbein, A., Eisfeld, B., and Sutcliffe, M., "Computation of Aerodynamic Coefficients for Transport Aircraft with MEGAFLOW," *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, edited by Springer, Vol. 89, Springer Verlag, 2005, pp. 135–150.

[32]Schenk, M., "Modification of the ALVAST geometry for CFD calculations," DLR IB 129-95/25, 1995.

[33]Kiock, R., "The ALVAST Model of DLR," DLR Institusbericht 129-96/22, DLR, 1996.

[34]Brandvik, T. and Pullan, G., "Acceleration of a 3D Euler Solver Using Commodity Graphics Hardware." *46th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, Jan. 7-10 2008.

[35]Martin, M., Andres, E., and Carreras, C., "Solving the Euler Equations for Unstructured Grids on Graphical Processor Units." *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing*, Trondheim, Norway, May 13-16 2008.

[36]Bolz, I., Farmer, E. G., and Schroeder, P., "Sparse matrix solvers on the GPU: conjugate gradients and multigrid." *IEEE Transactions on Graphics (TOG)*, 2003.

[37]Carreras, C., Lopez, J., Sierra, R., Rubio, A., Caffarena, G., Pejovic, V., Nieto, O., Andres, E., Baeza, A., and Palacios, F., "A Feasibility Study on the Application of FPGAs for the Hardware Acceleration of CFD Simulations," Tech. rep., DOVRES Technical Report, 2008.