AIAA 01–0974

A Multi-Code-Coupling Interface for Combustor/Turbomachinery Simulations

Sriram Shankaran
Juan J. Alonso
Stanford University
Stanford, CA 94305

May-Fun Liou
Nan-Suey Liu
NASA Glenn Research Center
Cleveland, OH 44135

Roger Davis
United Technologies Research Center
East Hartford, CT 06108

39th AIAA Aerospace Sciences Meeting and Exhibit
January 8–11, 2001/Reno, NV

A Multi-Code-Coupling Interface for Combustor/Turbomachinery Simulations

Sriram Shankaran[*]
Juan J. Alonso[†]
Stanford University
Stanford, CA 94305

May-Fun Liou[‡]
Nan-Suey Liu[§]
NASA Glenn Research Center
Cleveland, OH 44135

Roger Davis[¶]
United Technologies Research Center
East Hartford, CT 06108

This paper describes the design, implementation and validation of a method to couple multiprocessor solvers whose solution domains share a common surface. Using Message Passing Interface (MPI) constructs, parallel communication pathways are established between various simulation codes. These pathways allow applications to exchange data, synchronize time integrations and reinitialize communication data structures when meshes change their relative positions. At an interface with another simulation code, applications request specific flow variables, typically for a ghost/halo layer of cells or nodes. Numerical estimates of these flow variables are provided by the simulation software on the other side of the interface through three-dimensional interpolation. With an aim at achieving conservative interfacing between applications, particular instances of the requested flow variables and interpolation stencils will be used for different problems. Communication tables are built for processes involved with the exchange of information and all exchanges occur strictly between specific processes, thereby minimizing communication bottlenecks. This paradigm has been used to build a code coupling interface for a three-dimensional combustor/turbine interaction simulation in which a new massively parallel computational fluid dynamic solution procedure for turbomachinery, called TFLO, has been coupled with an unstructured-grid, parallel procedure for combustors, called NCC. Numerical and physical issues regarding the exchange of information as well as the coupling of physics-disparate analyses will be discussed. Several development test cases have been used to ensure the soundness of the communication procedures. A multi-component simulation for a dump combustor/exit duct has been performed as a demonstration of the new interface.

Introduction

COMPUTATIONAL Fluid Dynamic (CFD) simulations are an essential and integral element in the design process of modern gas turbine jet engines, providing engineering predictions of aerodynamic performance, heat transfer, and flow behavior. Steady-state flow predictions are commonplace for problems ranging in size from the design of an individual compressor or turbine blade, to large sections of a complete component such as a combustor or low-pressure turbine. Entire component unsteady or multi-component steady-flow simulations have not yet become routine in the design process because of the exceedingly large resource requirements to perform these analyses and the disparate flow physics of each component. However, multi-component interaction effects, such as combustor/turbine hot streak migration or compressor/combustor instability, are of great interest, especially at off-design conditions, due to their adverse effects on performance, durability, and operability.

Shared and distributed memory parallel computer systems and networks have helped to greatly extend the feasible size and reduce the solution time of large-scale gas-turbine design and analysis problems. Many new advances in computer hardware, network communications, and simulation software are required, however, to bring large-scale simulations into practical, everyday use. The design and analysis of gas turbine jet engines is not the only engineering or scientific arena that has these bottlenecks due to the overwhelm-

[*]Doctoral Candidate, Stanford University
[†]Assistant Professor, Member AIAA
[‡]Aersopace Engineer
[§]Aerospace Engineer, Senior Member, AIAA
[¶]United Technologies Research Center

ing problem size and the limited computer systems that can handle them. As a result, the Department of Energy (DoE) has launched the Accelerated Strategic Computing Initiative (ASCI) to promote the development of massively-large parallel computer systems and the simulation software that can take advantage of them. As part of this initiative, the current effort has focused on developing and demonstrating the capability to simulate the steady or unsteady flow through multi-components of a gas-turbine engine. In addition, plans for more ambitious simulations which are currently underway are brielfy mentioned.

In the following sections a detailed description of the integration framework and an overview of the two codes, TFLO and NCC is presented. Finally, results from coupling the two codes for a few model problems as well as a dump combustor/duct are presented.

Integration Framework

One of the crucial steps in the coupling of different simulation codes is the need to develop a framework which allows these codes to communicate with each other within a parallel computing environment. In the following sections we outline a method which aims at developing one such framework. This framework allows codes to request/provide the necessary data from/to other codes, synchronize time integration, and reset communication data structures for meshes in relative motion. The framework described here has been developed with the aim of coupling TFLO and NCC. However, it has been kept as general as possible and it should be capable of handling multiple code interfacing problems. Another aspect of coupling multiple codes is to identify the numerical nature of the coupling process. There are different ways to arrive at a numerical scheme and they can be placed in increasing hierarchy of complexity as

- loosely coupled systems in which the presence of the interface can be treated by the individual codes as a boundary condition. This involves no explicit exchange of values between the participating codes.
- moderately coupled systems in which the primary or primitive variables at the interface are exchanged between codes.
- tightly coupled systems, where in addition to the primary variables, the fluxes across the interface are also exchanged.
- simultaneously coupled systems, where algorithmic variables required by the numerical schemes of the individual codes are exchanged.

The integration framework described in the following sections, aims to result in moderately coupled systems with possibility of extension to tightly coupled systems.

Problem Description

The interface between different codes requires the transfer of information from one code to the other. Typically, for problems in turbomachinery, an Unsteady Reynolds Averaged Navier-Stokes (URANS) code that handles the turbine/compressor requires information at the entrance/exit of the physical domain being simulated. This information is to be provided by the code that handles the combustion process, which can also be URANS-based or can use a Large Eddy Simulation (LES) approach. For both cell-centered and cell-vertex schemes the values of the flow variables in a row of surrounding halo/ghost cells/nodes need to be updated with information from the simulation software handling the opposite side of the interface (see Figure 1). Hence, to manage the transfer of information it is necessary to identify '*donor*' cells for each halo/ghost cell/node that requests information. If the identification of this '*donor*' is performed in an algorithmically efficient manner, codes which use different solution methodologies can be coupled efficiently. In combustor/turbomachinery simulations, the grids on either side of the interface between the codes are mostly stationary. However, with emerging technologies, there is a possibility that in the future the first blade row of the turbine might be designed to be set of rotating blades. In this situation, for every ghost/halo cell, it will become necessary to recompute its '*donor*' whenever the grids move relative to each other. It is also crucial to minimize the amount of information exchanged so that any bottlenecks that may arise due to communication may be eliminated. This is of particular importance in large-scale parallel computing applications, where the cost of the use of the communication sub-system is relatively high compared to the cost of actual computing.

In the following sections, we describe the important steps needed to set-up a framework which allows for the integration of different simulation codes. Suggestions are made for the naming of the various components of the integration framework, as well as for the algorithmic details of the implementation.

Interface For Multi-Code Integration

The two important steps involved in developing this framework are :

- Initialization - This step sets up the communication tables which allow each processor to identify the processors to/from which it needs to send/receive information. For each code, the processors involved in the interface must also identify the cells/vertices which will be used to compute the information requested by the other code. Furthermore, it is necessary to perform this initialization step whenever the grids in either code move relative to each other or when any of the grid systems are adaptively refined.

- Communication - Once the processors have gathered the requisite information, this second step is used whenever information has to be exchanged between the two codes. The communication step will be used repeatedly during the pseudo-time iterations of all participating simulation codes, and, therefore, extreme care is placed in making this step as efficient as possible.

For the sake of clarity, the following discussion assumes that the two codes to be interfaced are TFLO and NCC. However, no inherent assumptions are made in the methodology and, hence, one should be able to generalize this method to any pair of grid-based simulation codes.

Initialization Step

In order to facilitate the integration of component simulation codes into a single parallel computing environment, we will make extensive use of user-defined MPI communicators. The MPI standard allows for the creation of sets of processors which are grouped according to a commonality of the tasks to be performed. At the very least, every component code should be a member of a separate communicator. In addition, subgroups of processors within a simulation code may also belong to additional communicators if they perform a specific task which differentiates them from the other processors.

The main reason for the introduction of additional communicators is to restrict global communication operations to only the processors that need to participate in the communication. In addition, the enrollment of various processors into a specific communicator allows a simulation code to provide information to itself and other component codes without apriori knowledge of the functions that these codes may perform. Using such a communicator-based processor decomposition certain MPI global communication commands (such as MPI_BCAST and MPI_GATHER) can be restricted to members of a particular processor subset.

Creation of new Communicators

To distinguish between the processors that are running TFLO and NCC, all processors running TFLO will enroll in the TFLO_WORLD communicator, while those that run NCC will become part of NCC_WORLD. Moreover, we shall make a further distinction within each component code. Those processors from TFLO that are involved in the communication across the interface will become part of a new communicator, TFLO_AEE_WORLD, and those from NCC that perform a complementary role will enroll in NCC_AEE_WORLD. The AEE portion of the names of the communicators stands for Arbitrary Eulerian-Eulerian which represents the types of simulation codes we have thought of in the development of this environment. In general, a given simulation code will need to enroll its processors in a communicator called CODENAME_WORLD, and the subset of its processors that participate in the exchange of information across the interface will become part of the CODENAME_AEE_WORLD communicator. The processors in CODENAME_AEE_WORLD of one simulation communicate with their counter-parts in other simulations through new communicators called CODENAME1_CODENAME2. These communicators are called inter-communicators in MPI. While coupling multiple codes a suitable naming strategy can be used to identify the different inter-communicators. Note that this strategy can also be used to provide communication pathways between multiple instances of the same simulation.

Identification of Processors at the Interface

Once the new interface communicators have been created, every processor in each simulation code must determine for itself whether it belongs to either one of these groups. This determination may be made based on a variety of criteria. For example, special boundary condition flags may be specified in the input file of each simulation program. Alternatively, processors may conduct a series of geometric tests to determine whether any part of their mesh lies on an apriori defined interface surface. However, this choice is entirely left to the developers of each simulation program.

Although typically all processors participating in the communication across the interface will lie directly on the interface, it is possible that other processors that do not lie directly on the interface will also have to provide information. This situation may arise when the size of the cells on both sides of the interface vary greatly. In such a situation, the halo of a large cell may lie inside a processor that is not in direct contact with the interface. In the event that one or more of the required cells are not identified, the value of the requested data can be computed from the updated values of the neighboring cells/nodes in the local processor.

Compilation of Information to Be Requested

Once the processors that are involved in the exchange of information are identified and properly enrolled into the new communicators, these processors need to compile a list of locations and variables for which they require information from the other code. For each location at which information is needed the following pieces of information will be compiled:

- Processor ID : ID of the processor in which the requested location resides.

- Block/Cell/Node ID : Block/Cell/Node number in the local processor where the information provided by the other code will be stored. The format of this data depends greatly on the data structure used by the simulation code. In TFLO, for example, each processor may handle an arbitrary number of blocks from the multiblock mesh. However, a one-dimensional array is used as the underlying data structure, and then, the Cell/Node ID will correspond to the position in this one-dimensional array.

- x,y,z location : Cartesian coordinates at which interpolated data is required.

- Information Request Flag : This flag is an integer value which encodes the specific variables, their number, and the order in which these variables will be provided. For example, TFLO may request ρ, k, and ω from NCC, while NCC may only need p, ρu, and T.

Each processor in TFLO_AEE_WORLD and NCC_AEE_WORLD compiles this list which is then broadcast to all its counterpart processors and only to them.

The initial exchange of information is to be done in three '*steps*'.

- During the first '*step*', each and every processor in the TFLO_AEE_WORLD and NCC_AEE_WORLD groups will *broadcast* to the other group the complete list of locations and additional information described above for that specific processor.

- The receiving processors must then, in a second '*step*' sort through all the entries in all the lists provided during the broadcast operation and identify those that they can provide information for. Internally, each processor will generate information regarding the interpolation weights necessary to provide the information requested, and will store this information using a data structure appropriate to the code in question.

- In the final third '*step*', all processors must communicate with those processors they can provide data to and inform them of the specific requests that can be fulfilled. In theory, all data requests by a given processor will be fulfilled by a combination of processors in the group on the other side of the interface. However, there may be situations in which this will not be true; orphaned cells may occur for which the error checking and extrapolation procedures described below will be necessary.

Identifying 'donor' cells and Send buffers

Each processor that receives the complete list that is broadcast in the first '*step*', builds a different list for each of the processors to which it will provide interpolated data during the communication step. Hence, each processor, scans the entries from all the lists that are broadcast to it and determines those entries for which it can provide information. The process of identifying the entries for which each processor can provide information is left to the developers of the respective simulation codes. This task constitutes the second '*step*' described in the previous section. During the process of identifying the entries each processor is responsible for, each processor builds a separate list for each processor on the other side of the interface to which it will provide information. The entries in this list will contain the following entries:

- Processor ID : ID of the processor to which the data is to be sent to.

- Block/Cell/Node ID : Block/Cell/Node number on the remote processor to which the data is sent to.

- Data : Interpolated data that will be used to update the values in the Cell/Node ID of the processor that requested those data.

Each processor in TFLO_AEE_WORLD and NCC_AEE_WORLD sends this information to the appropriate processors on the other side of the interface. The receiving processors map this information, to an array that is stored locally in each processor, to determine the cells/nodes that will be updated in the communication step.

Step three of the above process is illustrated through the following example. If the list received by processor 15 (say) which is in TFLO_AEE_WORLD from processor 48 (say) which is in NCC_AEE_WORLD is

15, 1001, p
15, 2035, ρ, u, v, p
15, 5019, ρ

The local array that is built by processor 15 will be

48, 1001, Information Request Flag for p
48, 2035, Information Request Flag for ρ,u,v,p
48, 5019, Information Request Flag for ρ

Each receiving processor builds this local array which contains entries in the order in which they will be received from the processor that sends this information. This local array is built, to prevent the need to transmit the cell/node ID during the communication step. Hence, during the communication step, each processor builds only a list of interpolated data that it will communicate to the processor that needs the information in the exact same order specified above. The local processors, then determine the cells/nodes to be updated using the pointer list that was compiled during the initialization step.

As was mentioned in the previous section, there might be situations in which some cells/nodes on a processor requesting information do not find a donor from the processors on the other side of the interface. The local arrays can easily determine the identity of these orphaned cells/nodes by direct comparison between the request list and the lists sent by the processors on the other side of the interface. The simplest way to handle this scenario would be to use an extrapolation routine to update the values in this cell/node from its neighboring values.

The processors in TFLO_AEE_WORLD and NCC_AEE_WORLD also compile a local array which allows them to build the send buffers during the communication step. This list would contain the following information:

- Processor ID : ID of the processor to which the requested entry is to be sent to.
- Interpolation weights : These entries are specific to the nature of the interpolation and they are to be decided by the developers of the simulation code. This entry should be programmed to allow for ease of use of different interpolation stencils.

Communication Step

In the communication step, each processor compiles a list of interpolated data to be sent. Hence, the send buffers during the communication step would only contain the following information for each processor:

- Data : Interpolated data that will be used to update the values in the Block/Cell/Node ID of the processor that requested the data.

Only the actual data is needed here, since the initialization '*step*' described in previous sections has carefully created lists of pointers that translate the information in an ordered communication buffer into specific memory locations of the ghost/halo cells of the receiving processor.

To reiterate, each processor which receives this information, then uses the local arrays that it built during the initialization step, to determine which cell/node needs to be updated.

Overview of TFLO

The unsteady Reynolds Averaged Navier-Stokes equations are solved using a cell-centered discretization on arbitrary multiblock meshes. The solver is parallelized using domain decomposition, an SPMD (Single Program Multiple Data) strategy, and the Message Passing Interface (MPI) Standard.[1]

The solution procedure is based on efficient explicit modified Runge-Kutta methods with several convergence acceleration techniques such as multigrid, residual averaging, and local time-stepping. These techniques, multigrid in particular, provide excellent numerical convergence and fast solution turnaround. Turbulent viscosity is computed from a $k-\omega$ two-equation turbulence model. The dual-time stepping technique[2–4] is used for time-accurate simulations that account for the relative motion of rotors and stators as well as other sources of flow unsteadiness.

The multiblock strategy facilitates the treatment of arbitrarily complex geometries using a series of structured blocks with point-to-point matching at their interfaces. This point-to-point matching ensures global conservation of the flow variables. The structure of the mesh is specified via a connectivity file which allows for arbitrary orientations of the blocks. Two layers of halo cells are used for inter-block information transfer and an efficient communication scheme is implemented for the halo cell data structures. The load of each processor is balanced on the basis of a combination of the amount of computation and communication that each processor performs. Communication of halo cell values is conducted at every stage of the Runge-Kutta integration and in every level of the multigrid cycle in order to guarantee fast convergence rates. A general and parallel efficient procedure has been developed to handle the inter-blade row interface models, for multistage turbomachinery simulations.

Overview of NCC

The National Combustion Code (NCC)[5] is an unstructured-mesh solver for the solution of the time dependent compressible Navier-Stokes equations with turbulent combustion. A finite-volume, cell-centered scheme is employed with the explicit four-stage Runge-Kutta algorithm to advance the solution. Local pseudo-time stepping and residual smoothing are used to accelerate the convergence. Turbulence closure is obtained via either a high or a low Reynolds number $k-\epsilon$ model. A Lagrangian scheme based on particle tracking and the dilute spray approximation is used to solve the liquid-phase equations. NCC has been run on various massively parallel platforms.

Choice of Variables

The location and choice of variables that are exchanged between the participating codes are dependent on how we wish to model the coupled systems at hand. As mentioned earlier, in this approach we aim to build a set of moderately coupled systems with the possibility of extension to tightly coupled systems.

Our experiments with coupling TFLO and NCC revealed that the implementation specifics of each code also plays an important role in determining the set of variables. Differing assumptions for the material properties of air (R, C_p, γ) in the two codes, produce differing states of the gas at the interface. This makes it impossible to derive a set of variables which maintains continuity in all variables of interest, namely density, pressure, temperature, velocity and energy. Furthermore, NCC and TFLO model the dimensional and non-dimensional forms of the governing equations. This demanded that the two codes have knowledge of the non-dimensionalization procedure used in the other code. When the conditions of the flow warrant a turbulent simulation, variables that enable the estimate of the turbulent viscosity need to be exchanged. Differing turbulence models in the two codes required the identification of a new set of turbulence variables that need to be exchanged at the interface. Another factor that had to be accounted for was the disparate numerics in the two codes. For example, the implementation of pre-conditioning in NCC demands knowledge of the minimum velocity in the whole computational domain and hence this had to be included in the set of variables that were exchanged between the two codes.

To resolve these issues, a choice of variables that allowed for the primitive variables to be continuous across the interface while allowing for a jump in the energy was adopted and it has produced the best set of results for the problems that were tested. Specifically, in the following analysis, the quantities that were exchanged between the two codes were: $\rho, \rho u, \rho v, \rho w, T, p$. Note that this set will provide continuity in the primitive variables but there will be a jump in the energy due to the differing methods to estimate the energy. This set of variables loosely couples the systems in question. Extensions can be made to this set to include the fluxes through the appropriate cell faces on either side of the interface. However, this demands the knowledge of the intersection of the planar grid from either side of the interface, a challenging task while coupling unstructured and structured grids. The above set of variables were requested for the cell centers of the two layers of face halo cells in TFLO and the cell and face center values of the one layer of halo cells in NCC. For turbulent flow simulations, in addition to the set of variables mentioned above, the turbulent kinetic energy and the turbulent viscosity were exchanged between the two codes. However, this did not provide satisfactory answers. Further research is required to determine an optimal set.

Results

The results from using this integration module to couple TFLO to NCC for a few model problems are presented in this section.

Coupling TFLO and TFLO

Inviscid Transonic flow over a bump

The inviscid transonic flow over a two dimensional bump was modeled by dividing the domain into two equal regions. The two sub-domains form an interface at the maximum height of the bump (Figure 2). The integration module was used to exchange information across the interface. The inlet Mach number was held at 0.675 and the ratio of the back pressure to the inlet total pressure was held at 0.737. The individual blocks are structured grids and have dimensions $32 \times 17 \times 9$. The pressure contours of the steady state flow-field are shown in Figure 3 (a). The contours are fairly continuous through the interface suggesting that no spurious disturbances were introduced by the coupling process. The convergence histories of the two codes were also not significantly affected by the coupling process.

Coupling NCC and NCC

Inviscid Transonic flow over a bump

The same procedure that was done to couple TFLO to TFLO was repeated with NCC. The grids, inlet and exit conditions were the same as was used in coupling TFLO to TFLO. The pressure contours obtained from this coupling process are shown in Figure 3 (b). Note that the contours are continuous through the interface.

Coupling NCC and TFLO

Inviscid Transonic flow over a bump

The flow over the bump was now modeled by coupling TFLO and NCC. NCC was used on the left half of the bump and TFLO was used on the down-stream side. The inlet Mach number and exit conditions were the same as before. The pressure contours are shown in Figure 4. Again, the contours are fairly continuous through the interface. A few of the contours are not strictly continuous which could be resolved by a more optimal set of variables that are exchanged between the TFLO and NCC. Also, note that the ranges of color scales are slightly different between the two figures.

Dump combustor/Duct

The inviscid flow through a backward facing circular channel is modeled by coupling NCC and TFLO. NCC simulates the region within the dump combustor and TFLO handles the flow through the duct (Figure 5). NCC uses an unstructured tetrahedral grid (13922 elements) and TFLO uses a multi-block structured grid (5 blocks, each of dimension $17 \times 17 \times 17$). The grid for the whole geometry is shown in Figure 5. The inlet velocity to the dump combustor was fixed at 10 m/s and the back pressure was held at at 1 atm. The velocity vectors are shown in Figure 6. The uniform flow down-stream of the dump combustor passes through the interface and exits through the duct. There are no noticeable glitches near the interface suggesting that the numerical errors introduced by the interface are negligible.

Conclusions

The development of a framework to integrate multiple simulations was outlined in this paper. The framework was tested for a few model problems by coupling the participating codes (TFLO and NCC) to one another and to themselves. The computational results from these experiments validate the implementation of the interface code. Further research needs to be done to identify a general 'rule' to determine an optimal set of flow variables that the participating codes will exchange. Ongoing work is focussed on coupling turbulent simulations and future work will aim at coupling a turbulent combustion simulation to a turbulent flow simulation in the turbine.

Acknowledgements

The authors would like to thank the U.S. Department of Energy (DoE) for its generous support under the ASCI Program. We would also like to thank Dr. Jixian Yao, Stanford University, Dr. T. H. Shih, NASA Glenn Research Center, for generating the TFLO and NCC grids respectively for the dump combustor and Dr. Jeff Modor for the valuable discussions on the NCC integration framework. The authors would also wish to thanks United Technologies Research Center, and Pratt and Whitney for their support.

References

[1]Yao, J., Jameson, A., Alonso, J. J., and Liu, F., "Development and Validation of a Massively Parallel Flow Solver for Turbomachinery Flows," AIAA Paper 00-0882, Reno, NV, January 2000.

[2]Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," AIAA Paper 91-1596, AIAA 10th Computational Fluid Dynamics Conference, Honolulu, HI, June 1991.

[3]Alonso, J. J., Martinelli, L., and Jameson, A., "Multigrid Unsteady Navier-Stokes Calculations with Aeroelastic Applications," AIAA Paper 95-0048, AIAA 33rd Aerospace Sciences Meeting and Exhibit, Reno, NV, 1995.

[4]Belov, A., Martinelli, L., and Jameson, A., "Three-Dimensional Computations of Time-Dependent Incompressible Flows with an Implicit Multigrid-Driven Algorithm on Parallel Computers," Proceedings of the 15th International Conference on Numerical Methods in Fluid Dynamics, Monterey, CA, June 1996.

[5]Liu, N.-S. and Quealy, A., "A Multidisciplinary Design/Analysis Tool for Combustion," NASA CP 1999-208757, NASA HPCCP/CAS Workshop Proceeding, Cleveland,OH, January 1999.

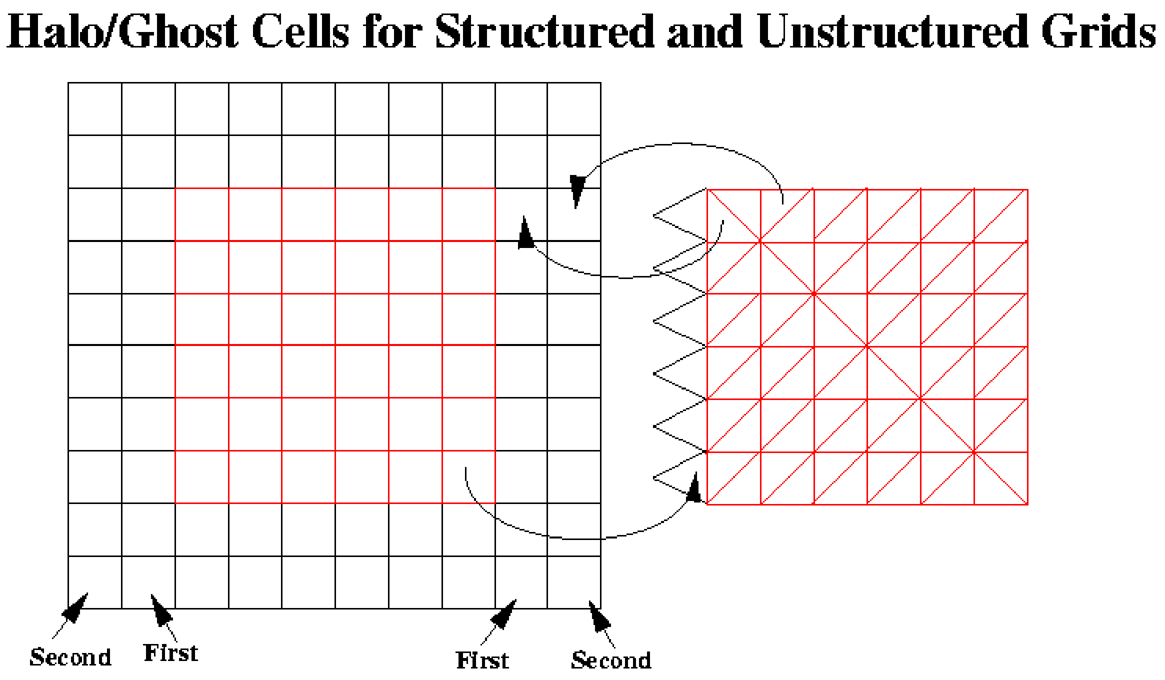


Fig. 1 Grid structures for TFLO and NCC

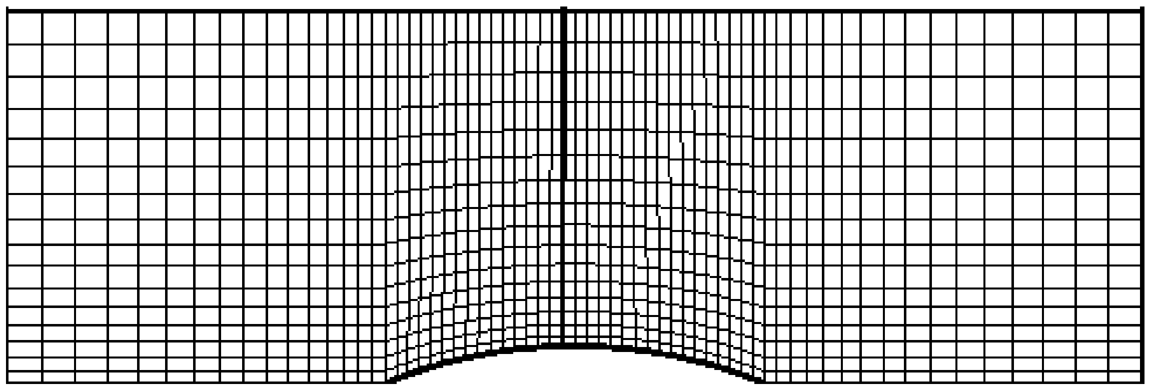
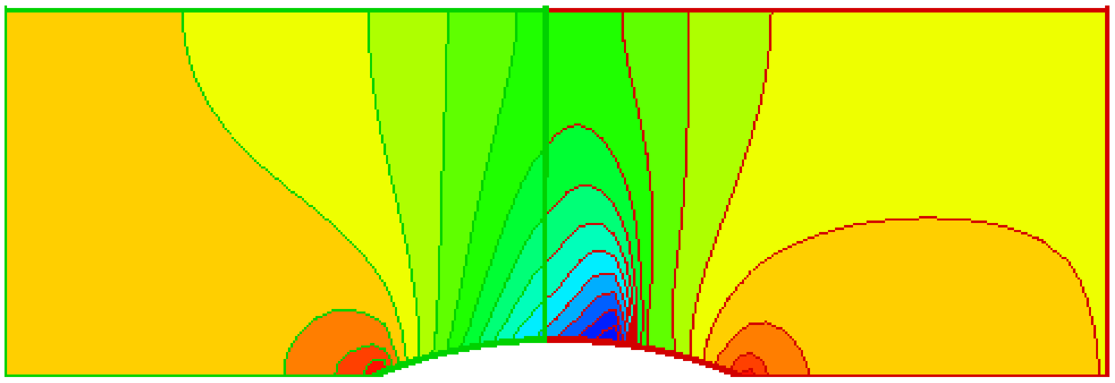
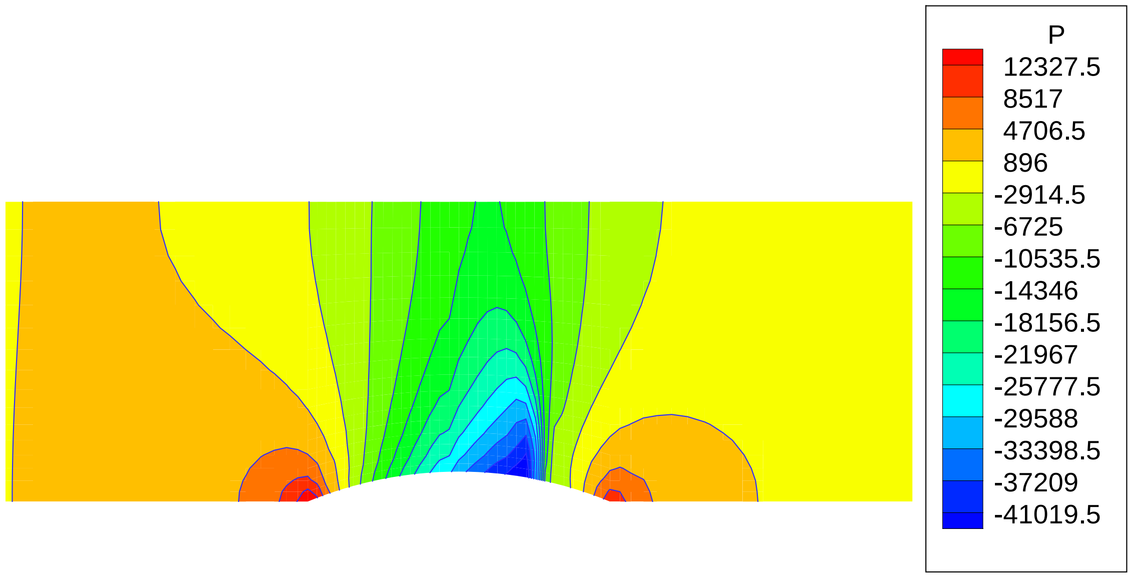


Fig. 2 Section of the grid

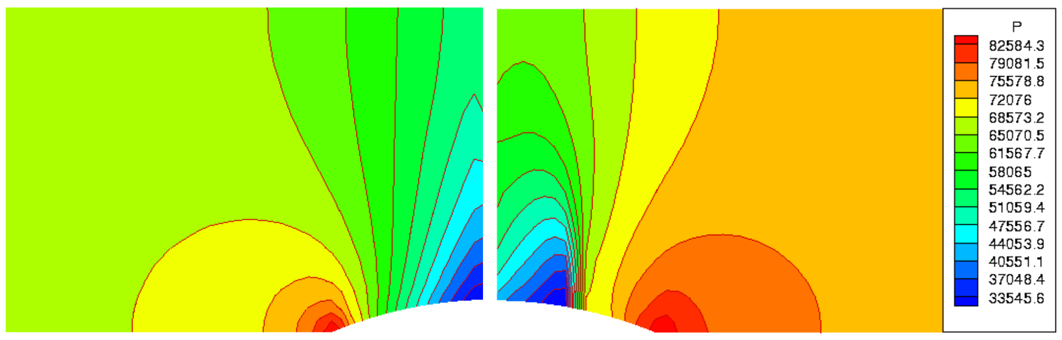


a)



b)

Fig. 3 Transonic Bump : a) TFLO coupled to TFLO, b) NCC coupled to NCC



a) b)

Fig. 4 NCC coupled to TFLO, a) NCC, b) TFLO

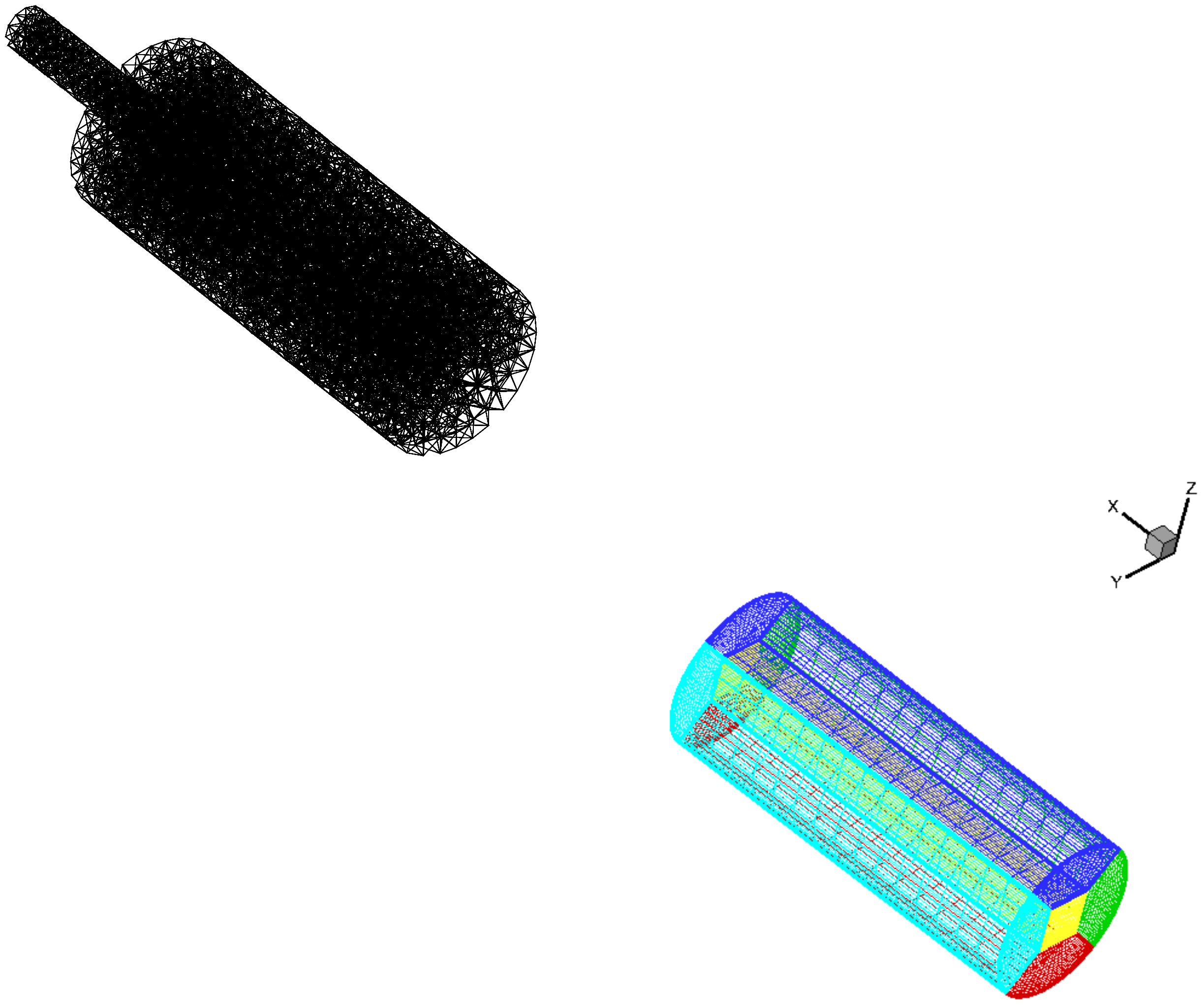
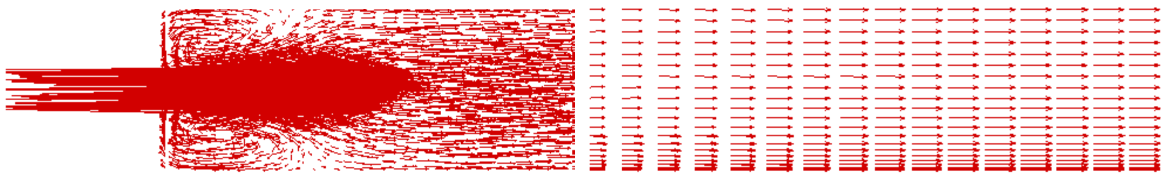


Fig. 5 NCC coupled to TFLO, Grids for Dump Combustor/duct



a) b)

Fig. 6 NCC coupled to TFLO, a) NCC, b) TFLO