A python approach to multi-code simulations: CHIMPS

By J. U. Schlüter, X. Wu, E. v. d. Weide, S. Hahn, M. Herrmann, J. J. Alonso AND H. Pitsch

1. Introduction

Simulations of real world engineering applications require that a large variety of physical phenomena be modeled. The areas of application for numerical simulations have been extended significantly in recent years due to the steady increase of computational resources. However, the integration of new models into existing solvers is often problematic.

Consider the example of fluid-structure interactions. In the past, flow solvers have been developed and optimized to simulate a variety of flows. Solvers for structural analysis have been developed to simulate stresses and deformations in solid bodies. So far, in order to simulate fluid-structure interactions, both approaches have to be integrated into a single solver. In the process, it usually had to be accepted that one of the models was not as accurate, optimized, or flexible as in the original solver.

Another example is that of LES-RANS hybrids. Here, a variety of flow solvers have been written for either the RANS (Reynolds-Averaged Navier-Stokes) approach or for LES (Large-Eddy Simulations). While previous LES-RANS hybrid approaches, such as Detached-Eddy Simulations (DES) (Spalart, 2000) and Limited-Numerical Scales (LNS) (Batten *et al.* 2002) combine LES and RANS in a single flow solver, the approach of coupling two existing flow solvers has the distinct advantage of building upon the experience and validation that has been put into the individual codes during their development. Also, simulations can be run in different domains at different time-steps. Both LES and RANS require a distinct set of numerical algorithms and models to work efficiently and accurately. The integration of both approaches into a single solver is often tiresome.

Here we present a different approach to address this problem: we keep the solvers separate. They run simultaneously and exchange only the necessary information at the interfaces. This allows each solver to use the best and most accurate methods for the solution of its problem, while the interaction with other solvers allows for an approach to very complex engineering applications.

In the example of fluid-structure interaction, we can execute one flow solver and one structural solver simultaneously. At the surface of the body, the structural solver receives the loads from the flow solver, and the flow solver receives the displacement of the body from the structural solver.

For the example of LES-RANS hybrids, the coupling approach allows for the use of the most appropriate approach for each zone in a domain. For example, one part can be computed using a compressible structured multi-block RANS solver and the other by a low-Mach number unstructured LES solver. This approach has been successfully applied to a variety of flow phenomena (Schlüter *et al.* 2005b), including complex real-world engineering applications (Schlüter *et al.* 2005c) (Fig. 1).

While the actual coupling between LES and RANS requires a framework that allows

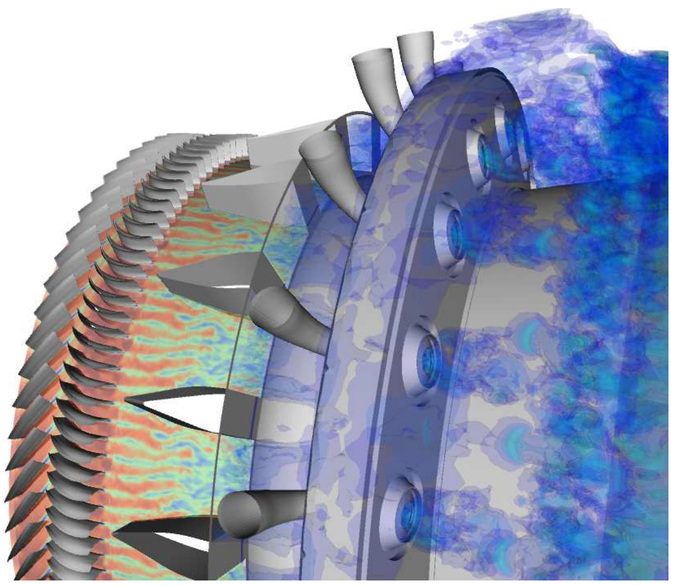

FIGURE 1. Coupled LES-RANS computation of the compressor-combustor interface of a gas turbine engine (Schlüter *et al.* 2005c). The compressor is computed with a RANS solver, the combustor with an LES solver. Isocontours of the axial velocity at the 50% plane in the compressor and diffuser. Isosurfaces of axial velocity in the combustor. A 20° segment is computed.

for conversion from the LES representation of turbulence to the RANS representation and vice versa, we will refer to previous work (Schlüter *et al.* 2004, Schlüter *et al.* 2005a, Schlüter and Pitsch, 2005d) on the details, and the reminder of the paper will concentrate on the coupling approach itself.

2. Python approach to coupling

Previous approaches to couple solvers were based on a pure MPI approach (Shankaran *et al.* 2001, Schlüter *et al.* 2003a, Schlüter *et al.* 2003b, Schlüter *et al.* 2005b) (Fig. 2). In this approach, MPI is used to establish a direct communication between the solvers. This requires that in each of the solvers, algorithms have to be implemented that perform the tasks associated with the coupling.

The coupling is performed in several steps. In a handshake routine, the solvers exchange the location of the nodes that are on the interface of their own domain. Each solver determines where the interface nodes are within its own domain and how to interpolate the data from its own mesh to the location of the interface nodes requested by other solvers.

During the actual computation, the solvers exchange interface data directly. After each time-step, the solvers interpolate their own solution on the interface nodes of the other solver and send the information to their peers.

The disadvantage of this approach is that the implementation of the communication algorithms into a new solver takes some effort. Since each MPI command in one solver requires a corresponding MPI command in the other solver, the implementation may be tedious and prone to errors. Furthermore, the search and interpolation routines must be implemented in each solver separately.

In order to improve the interoperability of the solvers, we decided to approach the

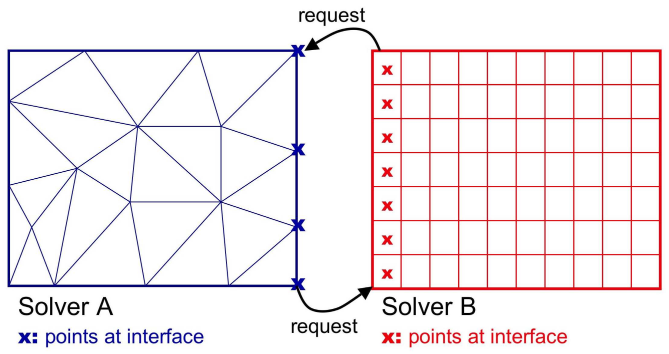

FIGURE 2. Direct coupling approach: solvers communicate the location of their interface points and the other solver determines how to provide data at these locations.

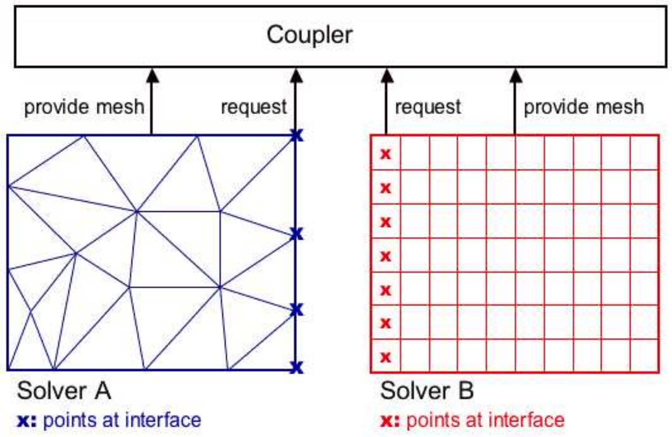

FIGURE 3. CHIMPS approach: solvers communicate location of their interface points and their mesh and solution to the coupler. The coupler determines how to provide information to the solver at the interface nodes.

coupling in a different manner. Instead of implementing all coupling routines (communication, search, and interpolation) in all solvers separately, we have developed a separate software module that performs these tasks and facilitates the coupling process. The idea is to remove the workload, especially the search, interpolation, and communication routines, from the solvers. The solvers communicate only with the coupler software (Figure 3). The coupler performs all searches and interpolations. In order to perform these tasks, the coupler requires knowledge of the meshes and of the solutions of the solvers.

The coupling software module that we have developed is called **C**oupler for **H**igh-Performance **I**ntegrated **M**ulti-**P**hysics **S**imulations (CHIMPS). It is based on the script language Python. This script language, together with its parallel version pyMPI, allows for the simplification of the communication between the solvers and CHIMPS. Instead of defining MPI calls, python functions are defined allowing for more freedom in the implementation. The communication is then handled like a function call with arguments, with the data being passed in the argument list.

3. Terminology

First, we want to define some terms that we will use to specify CHIMPS:

• An **API** (**A**pplication **P**rogramming **I**nterface) is a detailed specification of the interface routines, including arguments and their types, which can be used to interact with a software Module. APIs are intended to be composed of a number (possibly large) of simple, elemental functions that can be combined in multiple ways to perform complicated operations with the module.

• A **Solver** is a set of functions and/or subroutines that carry out the solution of a problem described by a set of governing equations and boundary conditions. In our work this normally means a flow solver, but the definition can be extended to any other type of solver (NASTRAN, for example).

• A **Module** is a set of functions and/or subroutines that have been compiled into a shared object (dynamically loaded library, *.so or *.dll) and that have been appropriately wrapped so that they are accessible through the Python interpreter.

4. Requirements on the coupler software

The following requirements must be satisfied by CHIMPS:

• Stand-alone: The CHIMPS software must be an independent module that does not require particular knowledge of the nature of the solvers that it is coupling to perform its functions.

• Parallel: CHIMPS must execute in parallel with close attention paid to the parallel performance, efficiency, and scalability (CPU and memory) of all its functions. CHIMPS must also work properly and transparently in a single-processor environment.

• Multi solver: CHIMPS must allow for the coupling of an arbitrary number of solvers.

• High-performance: CHIMPS should accommodate a variety of different search routines as they become available and must establish communication patterns whose performance is as close to direct, point-to-point MPI communication.

• Communication: All communication/exchange of information of the solvers with CHIMPS occurs via the Python interpreter. MPI is never to be used to communicate from the solvers to CHIMPS directly.

• Minimum memory impact: CHIMPS should be able to clean up the memory allocated for all of its operations if it is requested to do so, with zero net impact. The function chimps.deallocate() should free up all of the overhead memory that was required to perform its pre-processing functions. While CHIMPS is operating, it is expected to minimize the memory impact on running the solver applications, although concessions may be made to simplify the interface with the codes.

• Two main responsibilities: CHIMPS functions should be divided into those that belong in a pre-processing step, which may be repeated multiple times as needed if the relative positions of the mesh change in time, and those that participate in the interpolation/communication step.

• CHIMPS is responsible for all searches (both volume and surface/distance) and all interpolations. The solvers that are coupled simply provide information (in the form of standardized representations of both the mesh and the solution) to CHIMPS.

• Diagnostics routines: Information must be given back to the user regarding the success of the interface operations. This includes both information on whether a suitable donor cell/face was found or not, as well as whether the other solvers can provide data about the fields that are requested.

- All CHIMPS operations must be accessible from Python, and Python is the preferred method of communication.
- Component API functions may be combined either directly or via variable argument lists to simplify the interface with the solvers and to simplify the interface on the whole. However, a large number of API functions may be developed to address future uses of the interface in situations not foreseen by the development team.
- Clear standardization of mesh and solution formats for various different types of mesh.
- Flexibility to support different types of coupling to ensure accuracy and stability.

5. Mesh format

The communication of the mesh and the solution from the solver to CHIMPS requires a mesh standard. For the time being we assume a node based mesh and solution. We currently use several cell centered solvers with CHIMPS, and the solvers interpolate their solution to the nodes.

The mesh is provided to CHIMPS in an unstructured format. The nodes are provided in xyz-coordinates to CHIMPS, and the connectivity of the nodes follows a set of specified order. The solution is provided to CHIMPS based on the node list handed to CHIMPS in the mesh format.

6. Code execution

We now want to demonstrate how CHIMPS and the solvers are used within a python script. This is the main script, which launches the solvers and the coupler software. The script conrols the execution of the solvers. A launch of the pyMPI and the coupler script `chimps.py` could look like this:

```
mpirun -np 8 pyMPI coupler.py
```

The following is a simplified script. We will discuss the meaning of the function calls in detail following the example script.

```
#! /usr/bin/env python
# File: chimps.py

# *******************************************************
# This is a sample script coupling Solver A and Solver B.
# *******************************************************
# Standard Python modules
import sys # ... etc.

# Extension modules
import Numeric
import mpi

# Import CHIMPS
import chimps
```

```
# Import solvers
if (solver A):
    import solver_a as a
else
    import solver_b as b

# coupler: initialize
cpl_inputfile='coupler.inp'
chimps.initialize(cpl_inputfile)

# solver: initialize
if (solver A):
    a.init
else:
    b.init()

(...)    # Exchange of interface information

# get/set interface xyzs
if (solver A):
    [xyzs,ids] = a.get_interface_xyzs(nxyzs)
else:
    [xyzs,ids] = b.get_interface_xyzs(nxyzs)

chimps.set_xyzs(xyzs,ids, solver)

# get local mesh
if (solver A):
    [tetra_conn,pyra_conn,prisms_conn,hexa_conn,node_coor] =
            a.get_local_domain(n_tetra,n_pyra,n_prism,n_hexa,n_nodes)
else:
    [tetra_conn, pyra_conn, prisms_conn, hexa_conn, node_coor] =
            b.get_local_domain(n_tetra, n_pyra, n_prism, n_hexa, n_nodes)

chimps.set_local_domain(node_coor, tetra_conn, pyra_conn, prisms_conn, hexa_conn, solver)

# coupler: find donor cells
chimps.find_donors()

# determine which solution fields are needed

(...)    # Exchange of information what variables are requested
         # and what variables are needed from the local mesh
         # so that all interfaces receive the necessary data

# get the solution
if (solver A):
    solution = a.get_local_solution(nFields, n_nodes)
```

```
else:
    solution = b.get_local_solution(nFields, n_nodes)

chimps.set_local_solution(solution, mysolver)

# interpolate data to requested xyzs
chimps.interpolate()

# Provide interpolated xyzs to python
xyz_solution = chimps.get_interpolated_xyzs(nFieldsMax, nxyzs, mysolver)

if (solver A):
    a.set_interface_data(nFields, nxyzs, xyz_solution)
else:
    b.set_interface_data(nFields, nxyzs, xyz_solution)

# Iteration loop
for i in range(n_TimeSteps):
    if (solver A):
        a.RunIterations()
        solution = a.get_local_solution(nFields, n_nodes)
    else:
        b.run(1)
        solution = b.get_local_solution(nFields, n_nodes)
    #
    # set solution
    chimps.set_local_solution(aSolution, mysolver)
    #
    # interpolate
    chimps.interpolate()
    #
    # Provide interpolated data to python
    xyz_solution = chimps.get_interpolated_xyzs(nFieldsMax, nxyzs, mysolver)
    #
    # Provide interpolated data to solvers
    if (solver A):
        a.set_interface_data(nFields, nxyzs, xyz_solution)
    else:
        b.set_interface_data(nFields, nxyzs, xyz_solution)

# shutdown solvers and coupler
if (solver A):
    a.finalize()
else:
    b.finalize()

chimps.deallocate_all()
```

```
print " ALL DONE!"
```

First, some standard python modules are imported into python. These are necessary for some of the basic python operations. Then the coupler and the solvers are imported, which means that they are ready to be used.

The first operations are performed with the initialization of the code modules. The coupler initialization reads a parameter file that contains information about the codes and the interfaces. The solver initialization routine groups together all operations of the solver together that are performed prior to the actual computation.

The next step is the communication of the location of the interface nodes from the solvers to CHIMPS. The communication pattern here is an example of how the communication between solvers and CHIMPS takes place. Generally, functions with the prefix `‘‘get’’` provide data from the modules to the python script, whereas functions with the prefix `‘‘set’’` provide data from the python script to the modules. Here, data is transferred from the solvers to CHIMPS. First, the solver routines `get_interface_xyz` are called. The coordinates of the interface nodes are returned in the argument list. The data is then provided to the coupler with the function `set_xyzs` using the argument list. Note that in parallel execution, only interface nodes that are on the local processor are provided.

In the next step, the mesh is transferred from the solvers to CHIMPS. Essentially the same procedure is used using the functions `get_local_domain` and `set_local_domain`. Note again that in parallel execution, only the local meshes on the local processor are transferred.

In `find_donors`, parallel Alternating Digital Tree (ADT) search routines determine where the local interface nodes can be found in the other solvers' domain. The parallel searches determine on which processor and in which element of the mesh each interface node can be found. It also provides information about interpolation weights.

The following step is not explicitly shown in the example script because it consists of a number of calls following the same pattern as above. Here, the solvers provide the number and names of the requested variables of each of their local interfaces. The names follow the CFD General Notation System (CGNS) (Legensky *et al.* 2002) for data identifiers. CHIMPS then determines the local subset of unique variables that are needed on the local mesh. For example, if on a local mesh nodes from two interfaces are found and the interfaces request [ρ, u, v, w] and [u, v, w, T] respectively, the local unique subset would be [ρ, u, v, w, T]. If no interface nodes are on the local mesh, no variables will be requested from the solvers.

All previous steps are preprocessing steps that define the handshake routine for static meshes. If moving meshes are used, these steps can be repeated as needed after the location of the interface nodes or the local meshes have changed.

The next task is that of the communication of solution variables. In the sample script, one such communication step is used prior to the actual iteration loop. The data exchange is done in three steps. First, the requested local solution variables from the solvers are communicated from the solvers to CHIMPS (`get_local_solution` and `set_local_solution`). Then the coupler performs all interpolation steps and sends the data to the requesting processors (`interpolate`). As a last step, the data is transferred from CHIMPS to the solvers (`get_interpolated_xyzs` and `set_interface_data`).

Following the handshake and the initial data exchange is the iteration loop. In this loop, the solvers advance their solution (e.g `RunIterations()`) for either a predetermined number of iterations (for steady state simulations) or for a predetermined time-step (for

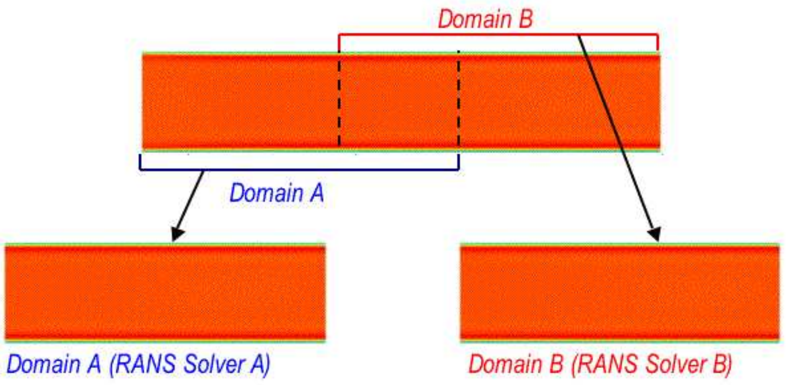

FIGURE 4. Coupled RANS-RANS computation of annular channel flow using CHIMPS: Domain decomposition.

unsteady simulations). This is followed by an exchange of interface data as described above.

Once the iteration loop is terminated, the solvers and CHIMPS call finalizing routines (`finalize` and `deallocate_all`) that ensure that the solvers and CHIMPS shut down properly.

7. Solver adaptation

Here, necessary steps for the adaptation of a solver to use with CHIMPS are described. The first step is the most complex part. The solver has to be 'pythonized'. In order to do this, the code has to be subdivided into three parts. The first part calls all initialization routines until the iteration routine. The second part is that of the actual iteration routine, and the third part is that of finalizing the solver, which includes all functions that are called after the iteration loop until the termination of the program. Assuming that the solvers are written in a well structured manner, the main program of the solver should already have this structure. Hence, all that remains is the wrapping of the lower level functions into python, which can be done with utilities such as `f2py` for Fortran and `SWIG` for C/C++. Only functions called from python need to be pythonized.

In the next step, all communication routines must be written. These are relatively simple routines that exchange information with the python script through the argument list. These functions can be seen as subroutines that exchange information with a main program. All that needs to be done is to either receive data in the argument list following the CHIMPS API and store it in appropriate form for the solver or to prepare data from the solver and bring it into the form of the CHIMPS API. The pythonization of these functions is once again performed with the appropriate tools.

Once the interface data is received from CHIMPS, it is the solver's responsibility to use the data in an appropriate way as a boundary condition in its own domain. For example, in the context of LES-RANS, coupling boundary conditions have been developed that allow for the use of RANS data in LES and vice versa (Schlüter *et al.* 2004, Kim *et al.* 2004, Schlüter *et al.* 2005a, Schlüter and Pitsch, 2005d).

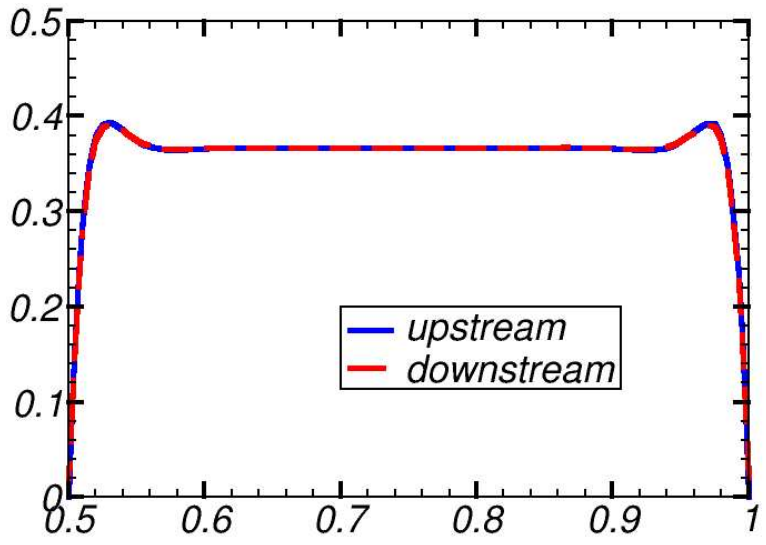

FIGURE 5. Coupled RANS-RANS computation of annular channel flow using CHIMPS. Momentum profiles at the interface.

8. CHIMPS verification

In the following we want to demonstrate the verification the data exchange on simple test cases.

8.1. *Predefined solution*

In the first verification case, we exchange a predefined solution and determine the interpolation error. The test case is that of two boxes with an overlapping interface. The coupled codes are the unstructured LES solver CDP (Moin & Apte, 2004) and the structured multi-block solver SUmb (Yao *et al.* 2000).

The solution field is defined as $u = x$, $v = y$, and $w = z$, where u, v, w are the flow velocities in the three coordinate directions and x, y, z are the coordinates. The solvers are requesting u, v, w from one of their interfaces. Once the exchange has been performed, the data received from the other solver can be compared with the value at the originally requested node location and the interpolation error can be estimated.

With different resolutions and different levels of parallelization the relative error was consistently below 10^{-6} using single precision. Several unstructured meshes have been tested with the RANS interface intersecting a number of different element types with the same result.

8.2. *RANS-RANS coupling*

As the first flow computation with CHIMPS, we chose an annular channel flow. The pipe is 2.5 diameter D long with an outer radius of $R_O = 0.5D$ and an inner radius of $R_I = 0.25D$. The Reynolds number is $Re = 1000$. The inflow is defined as a plug flow.

The domain is split into two subdomains (Fig. 4), each computed by a different instance of the RANS flow solver SUmb. The axial extend of the first subdomain is $0 < x < 1.5D$, and for the second domain $1D < x < 1.5D$. Hence there is an overlap between both domains. Both meshes are identical, which means that the meshes at the interface match in radial and azimuthal direction but not in axial direction.

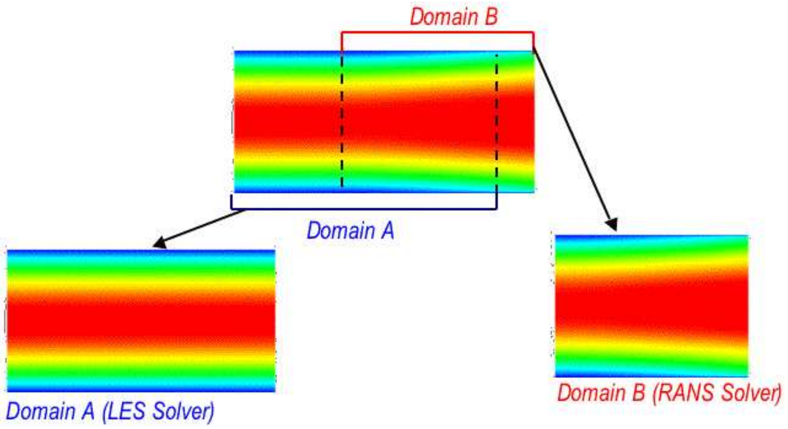

FIGURE 6. Coupled LES-RANS computation of channel flow using CHIMPS: Domain decomposition.

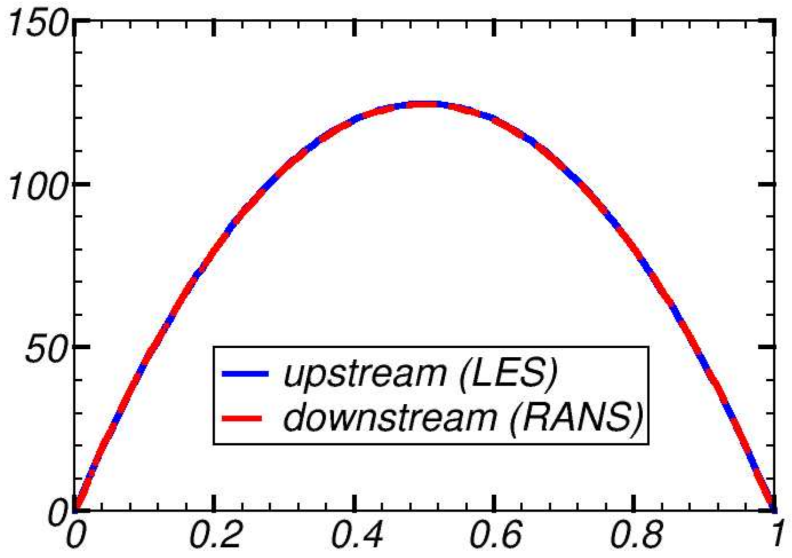

FIGURE 7. Coupled LES-RANS computation of channel flow using CHIMPS. Momentum profiles at the interface.

Figure 5 shows the momentum profiles at the interface ($x = 1$) in the upstream and in the downstream domain. It can be seen that they match perfectly. The error is within 1%.

8.3. *LES-RANS coupling*

In the next step, an LES solver is coupled with a RANS solver. Here, only the laminar case is computed in order to exclude effects of the different turbulence models in both domains.

The test case is that of a plane channel with a channel height $H = 1$ (Fig. 6). The upstream domain is the LES domain ranging from $0 < x < 1.5H$. The second domain is the RANS domain ranging from $1H < x < 2.5H$. A parabolic flow is used as LES

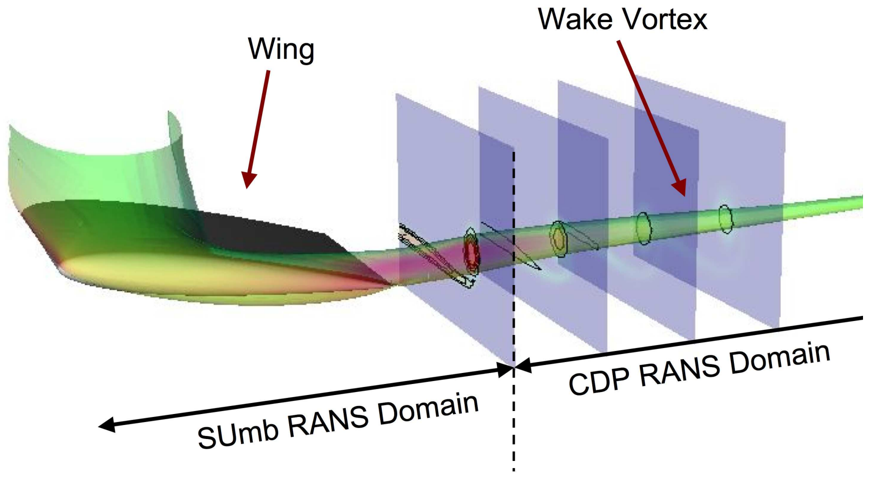

FIGURE 8. Setup for coupled RANS-RANS computation of a wing tip vortex (Geometry corresponding to (Dacles-Mariani *et al.* 1995).

inflow conditions in the LES domain upstream. Both meshes are non-matching with interpolations taking place in all three directions.

Figure 7 shows the velocity profiles at the interface ($x = 1$) in the upstream and in the downstream domain. Again, it can be seen that they match perfectly. The error is within 0.5%.

9. Applications

The CHIMPS approach for multi-code simulations is currently in use for a variety of applications.

In the DARPA helicopter quieting program at Stanford, a compressible RANS solver and an incompressible RANS solver are coupled for the prediction of the wing tip vortex of a helicopter. The prediction of the propagation of such a vortex is difficult, since the simulation of the flow around the wing requires a compressible approach. However, the numerical dissipation of a compressible RANS solver dissipates the vortex quickly. Therefore, the far field is computed with an incompressible RANS solver using a low-dissipation algorithm. Figure 8 shows the application of this concept in the prediction of the wing vortex of a wind tunnel experiment. The near field of the wing is computed with the compressible solver SUmb and the far field with the incompressible solver CDP. The communication between the solvers is handled by CHIMPS.

Another area of application is that of gas turbine engines. Here, the turbomachinery components – the compressor and the turbine – are computed with a compressible RANS solver and the combustor with a low-Mach-number LES solver. The computation of a Pratt & Whitney high-spool (high-pressure compressor, combustor, and high-pressure turbine) has been initiated using the CHIMPS approach for communication (Fig. 9).

A different kind of application has evolved in the form of multi-phase flows. Here we use an LES solver to compute the flow field of a domain and a level set solver for the determination of the location of the fluid surface. The interesting approach here is that both solvers compute the same physical domain, but they both compute different physical phenomena. The level set solver computes only the evolution of the level set equation

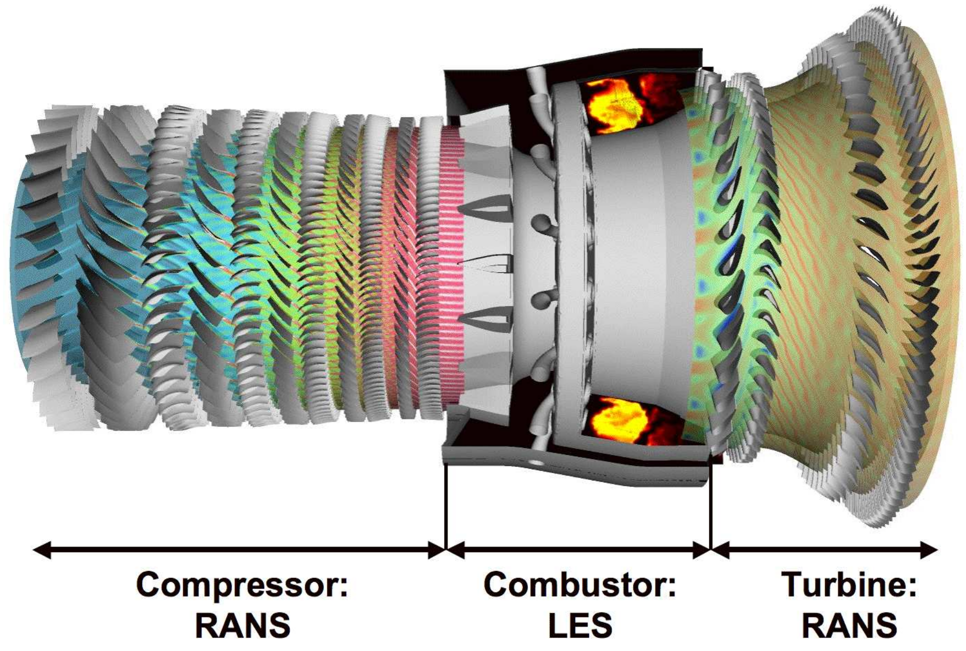

FIGURE 9. Full engine simulation: turbomachinery components are computed with RANS, combustor with LES. The communication between the solvers is handled by CHIMPS.

based on the flow variables from the LES solver. Since the level set equation needs to be solved only in the vicinity of an interface and the solver can use structured grids, the level set mesh can have a much higher resolution. It then returns density weights to the LES solver. This application shows that the coupling approach can be used not only to decompose domains but also to decompose physical phenomena. Here, the coupling approach can be used for the implementation of models into an existing solver.

10. Conclusions

We have developed a software module that allows for the coupling of multiple solvers. The advantage of the module is that it is written in a general fashion and solvers can be adapted easily to communicate with other solvers. The software module performs many of the required coupling tasks such as searches, interpolations, and process-to-process communication. The adaptation of solvers to the coupling module is facilitated compared with previous approaches.

The verification and the accuracy of the approach have been demonstrated. In addition, some of the current applications of this software have been presented.

Acknowledgments

We are thankful for support from the US Department of Energy under the ASC program and from DARPA under the Helicopter Quieting program. We also thank Pratt & Whitney for helpful discussions.

REFERENCES

Batten, P., Goldberg, U. & Chakravarthy, S. 2002 LNS-An approach towards embedded LES. *AIAA-2002-0427*.

Legensky, S., Edwards, D., Bush, R., Poirier, D., Rumsey, C., Cosner, R. & Towne, C. 2002 CFD general notation system CGNS: status and future directions. *AIAA-2002-0752*.

Kim, S., Schlüter, J., Wu, X., Pitsch, H. & Alonso,J. J. 2004 Integrated simulations for multi-component analysis of gas turbines: RANS boundary conditions. *AIAA-2004-3415*.

Dacles-Mariani, J., Zilliac, G. G., Chow, J. S. & Bradshaw, P. 1995 Numerical/Experimental study of wingtip vortex in the near field. *AIAA J.*, **33**, 1561–1568.

Moin, P. & Apte, S. 2004 Large-eddy simulation of realistic gas turbine combustors. *AIAA-2004-0330*.

Schlüter, J., Shankaran, S., Kim, S., Pitsch, H. & Alonso, J. J. 2003a Integration of RANS and LES flow solvers for simultaneous flow computations. *AIAA-2003-0085*.

Schlüter, J., Shankaran, S., Kim, S., Pitsch, H. & Alonso, J. J. 2003b Towards multi-component analysis of gas turbines by CFD: Integration of RANS and LES flow solvers. *ASME-GT2003-38350*.

Schlüter, J., Pitsch, H. & Moin, P. 2004 Large-eddy simulation inflow conditions for coupling with Reynolds-averaged flow solvers. *AIAA J.*, **42**, 478–484.

Schlüter, J., Pitsch, H. & Moin, P. 2005a Outflow conditions for integrated large-eddy simulation/Reynolds-averaged Navier-Stokes simulations, *AIAA J.*, **43**, 156–164.

Schlüter, J., Wu, X., Shankaran, S., Kim, S., Pitsch, H. & Alonso, J. J. 2005b A framework for coupling Reynolds-averaged with large-eddy simulations for gas turbine applications. *ASME J. Fluids Eng.*, **127**, 806–815.

Schlüter, J., Wu, X., Kim, S., Pitsch, H. & Alonso, J. J. 2005c Integrated simulations of a compressor/combustor assembly of a gas turbine engine. *ASME-GT2005-68204*.

Schlüter, J. & Pitsch, H. 2005d Anti-aliasing filters for coupled Reynolds-averaged/large-eddy simulations, *AIAA J.*, **43**, 608–616.

Shankaran, S., Liou, M.-F., Liu, N.-S., Davis, R. & Alonso, J. J. 2001 A multi-code-coupling interface for combustor/turbomachinery simulations. *AIAA-2001-0974*.

Spalart, P. R. 2000 Trends in Turbulence Treatments. *AIAA-2000-2306*.

Yao, J., Jameson, A., Alonso, J. J. & Liu, F. 2000 Development and validation of a massively parallel flow solver for turbomachinery flows. *AIAA-2000-0882*.