OpenFOAM turbo tools: From general purpose CFD to turbomachinery simulations

Conference Paper · January 2011

DOI: 10.1115/AJK2011-05015

2 authors:

Hrvoje Jasak
University of Zagreb
172 PUBLICATIONS **4,338** CITATIONS
SEE PROFILE

Martin Beaudoin
Hydro-Québec
9 PUBLICATIONS **153** CITATIONS
SEE PROFILE

Proceedings of ASME-JSME-KSME Joint Fluids Engineering Conference 2011
AJK2011-FED
July 24-29, 2011, Hamamatsu, Shizuoka, JAPAN

Paper No. AJK2011-05015

OpenFOAM TURBO TOOLS: FROM GENERAL PURPOSE CFD TO TURBOMACHINERY SIMULATIONS

Hrvoje Jasak
Wikki Ltd. London, United Kingdom
FSB, University of Zagreb, Croatia

Martin Beaudoin
Institut de recherche d'Hydro-Quebec (IREQ)
Montreal, Quebec, Canada

Key Words: *OpenFOAM, Open source CFD, turbomachinery, GGI, mixing plane, C++, object orientation*

ABSTRACT

OpenFOAM is an established object-oriented library for Computational Continuum Mechanics, with emphasis on CFD. It implements physical models of fluid flow, structural analysis, heat and mass transfer using equation mimicking, with unstructured polyhedral mesh support and massive parallelism in domain decomposition mode.

In order to use OpenFOAM in turbomachinery CFD, its "general purpose" capabilities are enhanced with turbo-specific features, related to physics of rotating regions and rotor-stator interfaces. Handling for geometric simplifications of multi-blade and multi-stage rotating machines are implemented, including simple stage interfaces, non-equal pitch of blade passages, pitch-wise cyclicity and mixing plane averaging.

In this paper we describe the implementation of turbomachinery-specific features in OpenFOAM, in the spirit of object orientation and C++. Emphasis is given to the basic functionality of turbo tools, software layout in OpenFOAM, numerical formulation of stage interfaces and their place in overall code design.

The paper is concluded with examples of turbomachinery simulations, illustrating the capability of turbo tools on industrial cases of incompressible and compressible turbomachinery flows.

INTRODUCTION

Turbomachinery simulation represents a significant portion of contemporary Computational Fluid Dynamics (CFD). Simulation of multi-bladed machines brings challenges beyond those encountered in standard external aerodynamics CFD, HVAC simulations or studies of simple wall-bounded turbulent flows. Firstly, geometrical complexity of multi-bladed, multi-stage rotating equipment couples to physical complexity of unsteady rotating flow and wake-to-blade interaction. Secondly, performance prediction of a rotating machine requires accurate simulation of attached or detached turbulent boundary layers, sometimes in presence of adverse pressure gradients. This in turn implies the need for very fine mesh resolution. Thirdly, rotation makes it necessary to account for relative motion of multiple rotors and stators, either by direct topological changes in the mesh or by geometrical simplification. Fourthly, performance of turbomachines is of interest under quasi-stationary (engineering) conditions, even in the presence of clearly time-dependent effects. Finally, for practical engineering use the cost of simulation (in terms of turnover time) needs to be sufficiently low to allow detailed studies of off-design conditions, geometric shape optimisation, robust design studies *etc.*

Mesh resolution requirements for turbomachinery flows are high, even if one considers a single-passage simulation of steady turbulent flows: presence of boundary layers in adverse conditions,

shock-to-boundary-layer interaction and transient wake effects results in a challenging environment. To complicate things further, practical design of rotating machinery involves non-matching blade pitch in consecutive blade rows, removing the possibility of simplifying the geometry by imposition of symmetry constraints.

In response to this, a number of simplifications to the flow model and geometry handling is regularly applied: we shall refer to them as *turbo tools*.

Rotation can be handled in several ways:

- In *moving mesh calculations*, a rotating component mesh is transformed in solid body motion by vertex motion and presence of rotation appears in equations via the relative velocity term. No further intervention is required, either in mesh handling or operator discretisation; however, this is appropriate only for transient flow simulations;
- Geometries under solid body rotation and steady conditions may be simulated by converting the equation set into the *single rotating frame of reference* represented on a static computational mesh. Here, the velocity field appearing in equations is the relative velocity $\mathbf{u}_{rel}$, and rotation is accounted for by addition of centrifugal and Coriolis force source terms in the momentum equation;
- For multi-stage simulations, *multiple rotating frame (MRF)* of reference approach is used, again represented on a static mesh. Rotation is accounted for in a region-wise manner, with the condition that the external boundary of each rotating region forms a surface of rotation. Momentum equation may be formulated either in terms of absolute or relative velocity; the latter requires a transformation of the velocity field at the rotor-stator interface, which complicates the model.

At the interface between stationary and rotating parts, further problems arise. For transient simulations with topological changes, interface may be handled using a sliding mesh technique [1]. Interface handling techniques that allow (but are not limited to) steady-state are specific to turbomachinery CFD and are listed below. This will be the main focus of the work presented here.

- The simplest manner of handling a rotor-stator interface is the *frozen rotor* technique, where a section of a stator and a rotor are simulated in a prescribed relative position and in steady-state, using the MRF approach. Transient behaviour of the machine is then inferred (at a certain level of approximation from a steady flow result. For increased fidelity, a frozen rotor simulation may be repeated several times, varying the relative position of components. In implementation terms, it is sufficient to couple the non-matching rotor and stator patches at the interface;
- A *General Grid Interface (GGI)* accounts for the coupling of the two in an implicit coupled manner without the need to topologically change the mesh. This is achieved via a direct intervention at discretisation level;
- In simulations of steady flow in single passages at high mesh resolution, cyclic boundaries are used. Imposing cyclicity in mesh structure is both tedious and limiting in terms of quality. A *cyclic GGI* interface is used to establish accurate and implicit handling of non-matching periodic boundaries.
- Another geometric generalisation of a GGI allows one to simulate geometrically non-matching sections of a rotor and a stator, assuming periodicity of each component separately. Sections at either side of the interface are coupled by rotational repetition (of variable pitch), using a *partial overlap interface*;
- For a steady-state simplification of the above, one may apply the multiple rotating reference frame approach and perform circumferential averaging of the solution at the rotor-stator interface via a *mixing plane* interpolator.

The features above make a distinction between a general-purpose and turbomachinery-capable CFD code and will be described in more detail. Clearly, this is not an exhaustive list: further capability enhancements, such as turbo-specific post-processing and the harmonic balance model are likely to be a subject of future work.

The remainder of the paper shall be organised as follows. We shall start with a brief review

of object-oriented design and software layout of OpenFOAM. This will be followed by a description of stage interfaces implemented in this study. All stage interfaces are based on GGI interpolation, with the addition of appropriate coordinate transformation or averaging. Features and details of their implementation, including the software design consistent withe the OpenFOAM framework and discretisation methodology will be given. The paper shall be completed by three examples of turbomachinery CFD with OpenFOAM and a brief summary.

OBJECT ORIENTATION IN NUMERICAL SIMULATION SOFTWARE

Traditionally, high-performance numerical simulation software such as commercial CFD and structural analysis tools have been written using a *procedural programming paradigm*. Here, a set of functions operates on globally shared data in a sequential manner, making it simple to understand and easy to optimise for efficiency. Unfortunately, increasing number of features complicates the source code to the level where further development grinds to a halt: bulk of development time is spend removing software defects that result from complex model-to-model interaction.

Complexity of monolithic functional software stems from its data organisation model: globally accessible data is operated on by a set of functions. Here, each added feature interacts with all other parts of the code, potentially introducing new defects (bugs) – with growing software size, data management and code validation issues necessarily grow out of control. This implies a software release cycle of approx. 18 months, with 6 months of development time and 12 months consolidation, bug fixing and validation.

In 1980s, a programming technique called *object orientation* attempts to resolve the complexity in a "divide and conquer" approach. The idea is to recognise self-contained objects in the problem and place parts of implementation into self-contained types (classes) to be used in building the complexity. In C++, a *class* (object) consists of:

- A *public interface*, providing the capability to the user;
- *Private data*, needed to provide functionality and managed by the public interface.

As an example, consider a sparse matrix class. It stores matrix coefficients in its preferred layout (private data) and provides manipulation functions, *e.g.* matrix transpose, matrix algebra (addition, subtraction, multiplication by a scalar, vector-matrix, matrix-matrix products and ultimately linear equation solvers). Each of these operates on private data in a controlled manner but its internal implementation details are formally independent of its interface. Classes introduce new user-defined types into problem description, allowing the programmer to create a "look and feel" of the high-level code, ideally as close to the problem as possible. Crucially, data access and protection is handled at class level and protected by a public interface.

Object orientation brings new challenges in numerically intensive applications, such as scientific computing. OpenFOAM [2, 3], with its beginnings in early 1990s illustrates how a careful use of object orientation and evolutionary code development results in an efficient and versatile numerical simulation tool-set. Main ingredients of its design are:

- Expressive and versatile syntax, allowing easy implementation of complex physical model by equation mimicking;
- Extensive capabilities, including wealth of physical models, accurate and robust discretisation and complex geometry handling;
- Open architecture and open source development paradigm, where complete source code is available to all users for customisation and extension at no cost.

EQUATION MIMICKING

A natural language for physical models in continuum mechanics already exists: it is the language of partial differential equations (PDEs). Attempting to represent PDEs in their natural language in software, based on established discretisation techniques of the Finite Volume (FV) and the Finite Element (FE) method, is our stated goal.

Consider a turbulence kinetic energy equation encountered in Reynolds Averaged Navier-Stokes

(RANS) models:

$$\frac{\partial k}{\partial t} + \nabla_{\bullet}(\mathbf{u}\, k) - \nabla_{\bullet}[(\nu + \nu_t)\nabla k] = \nu_t \left[\frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right]^2 - \frac{\epsilon_o}{k_o}\, k, \quad (1)$$

one can readily recognise the main objects that form the equation: fields, k, $\mathbf{u}$, ν_t; and operators $\frac{\partial_{\cdot}}{\partial t}$, $\nabla_{\bullet}$., ∇ *etc*. Lower-level objects, such as the computational mesh, a time-advancement scheme, boundary conditions and others support the above.

Within the object-oriented framework, we shall follow the path from Eqn. (1) to its encoded version in OpenFOAM, reading:

```
solve
(
    fvm::ddt(k)
  + fvm::div(phi, k)
  - fvm::laplacian(nu() + nut, k)
== nut*magSqr(symm(fvc::grad(U)))
  - fvm::Sp(epsilon/k, k)
);
```

Correspondence between Eqn. (1) and the code is clear, even with limited programming knowledge and without reference to object-orientation or C++.

DISCRETISATION SUPPORT

Salient parts of object hierarchy in code snippet above are outlined below. Some basic types, like `scalar`, `vector`, `tensor`, `List`, `word` *etc.* underpin the system and need not be reviewed in detail.

Space and Time. In computational terms, the temporal dimension is split into a finite number of time-steps. Formally, it is sufficient to track the time step count and time increment Δt. A set of database operations associated with time-marching finds its natural home in the `Time` class, including simulation data output every n time-steps or x seconds of computational time and general time-related data handling, *e.g.* book-keeping for old-time level field data handling needed in temporal discretisation operators.

OpenFOAM implements *polyhedral mesh handling*, where a `cell` is described as a list of faces closing its volume, a `face` is an ordered list of point labels and points are gathered into an ordered list of (x, y, z) `vectors`. Low-level implementation is discretisation-independent, where the `polyMesh` class provides connectivity information and mesh metrics as well as operations like mesh deformation, topological changes or adaptive refinement. For convenient use with discretisation, basic mesh information is wrapped. The `fvMesh` class, for example, supports the Finite Volume Method (FVM), while the `tetFemMesh` supports the Finite Element (FE) solver. In both cases, basic mesh structure and functionality is shared: a single mesh can simultaneously support the FVM and FEM solver without duplication of data and functionality.

Field Variable. Continuum mechanics operates on fields, discretised as a list of typed values at predefined locations in the mesh. This is implemented in three layers. A `vectorField` class consists a list of vectors and vector field operations: addition, subtraction, scalar multiplication, magnitude, dot- and cross-products *etc.* Arbitrary rank tensors and fields are defined by `VectorSpace` expansion in the same manner.

Boundary conditions, encoded as patch fields, carry *behaviour* in addition to face values. For example, a `fixedValue` field carries values which do not change on assignment. Other types, like a `fixedGradient` field class "evaluate" boundary values given internal field and a surface-normal gradient. Patch fields constitute a family of related classes: each calculates its boundary value, but does so in its own specific way.

Grouping the field data with its spatial dependence (reference to a mesh), boundary conditions and a dimension set creates a self contained *Geometric Field* object. Example are the `volScalarField k` or `volVectorField U` in the code snippet above.

Matrix, Linear System and Linear Solver. A sparse matrix $[A]$ and linear system $[A][x] = [b]$ hold the result of discretisation and provide the machinery for its solution. It suffices to say that code organisation as presented above allows the FE and FV discretisation to share sparse matrix implemen-

tation and linear solver technology, resulting in substantial code re-use.

Discretisation Method. Discretisation operators assemble an implicit or explicit representation of operators, based on the following three operations.

Interpolation, which evaluates field variable between computational points, based on prescribed spatial and temporal variation (shape function), *e.g.* cell-to-face interpolation.

Differentiation, where calculus operations are performed on fields to create new fields. For example, the following code:

```
volVectorField gP = fvc::grad(p);
```

creates a new FV vector field of pressure gradient given a pressure field `p`. Calculus operators carry the `fvc::` prefix.

Discretisation operates on differential operators (rate of change, convection, diffusion), and creates a discrete counterpart of the operator in sparse matrix form. Discretisation operators carry the `fvm::` prefix.

PHYSICAL MODELLING

Physical Modelling Library. Taking object orientation further, one can recognise object families at the physics modelling level. For example, all RANS turbulence models answer to the same interface: evaluate the Reynolds stress term $\overline{\mathbf{u}'\mathbf{u}'}$ in the momentum equation based on the mean flow velocity $\mathbf{u}$. Grouping them together guarantees interchangeability and decouples their implementation from the rest of the flow solver. A turbulence model contributes a discretisation matrix to the momentum equation, usually consisting of a diffusion term and explicit correction and no special knowledge of a particular turbulence model is needed at the top level.

Physics Solver. The components described so far act as a numerical tool-kit used to assemble various physics solvers. Each flow solver is a standalone tool, and handles only a narrow set of physics, eg. turbulent flow with LES, or partially premixed combustion. Capability of such solvers is underpinned by a combination of complex geometry support and massive parallelisation.

OpenFOAM TURBO TOOLS

The "general purpose" CFD capability of OpenFOAM brings us close to practical turbomachinery simulations, but not sufficiently so. First successful attempts relied on *dynamic mesh handling* [1, 4] and automatic mesh motion [5, 6]. Here, components in relative motion are connected using a sliding interface which, relying on polyhedral mesh support, re-connect two parts into a single domain. A sliding interface operates by projecting the pair of sliding surface onto each other and replacing original faces with facets, thus defining a singly connected polyhedral mesh, Fig. 1. In cases of dynamic mesh motion, face cutting is repeated for each topologically new relative position of two sides.

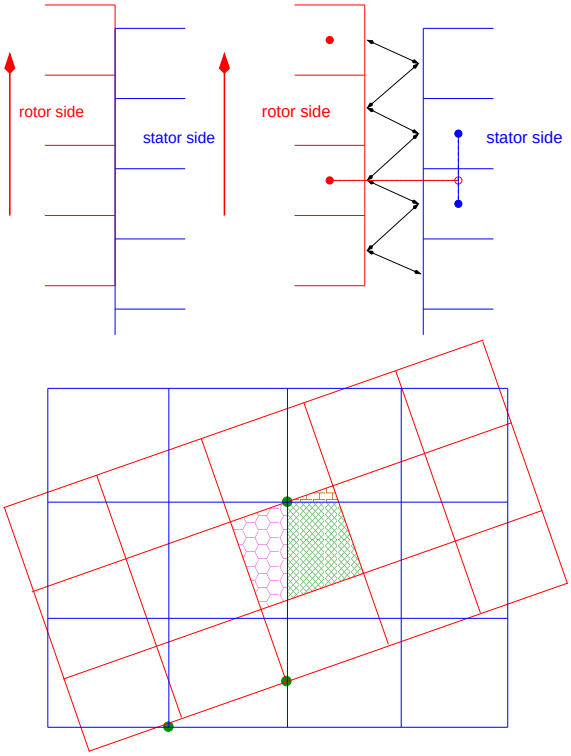

Figure 1: SLIDING INTERFACE AND GGI IN ACTION: FACETS REPLACING A FACE.

The beauty of a sliding interface and other topological changes is that no further intervention in the code is necessary: equation discretisation handles

the topology change as a moving mesh, without the need for solution mapping.

While a sliding interface is appropriate for cases where the complete sliding rotor-stator interface is modelled, Fig. 2, single-passage simulations, Fig. 3, cannot be easily performed. Furthermore, cost of transient simulations with topological change occuring at every time-step is about 20% higher than in static mesh simulations.

To mitigate this, we shall develop non-intrusive interface handling techniques operating at discretisation level (as opposed to topological mesh changes) and generalise them for various interface types.

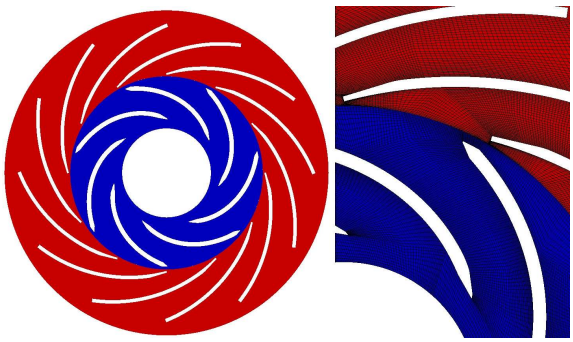

Figure 2: ERCOFTAC CENTRIFUGAL PUMP.

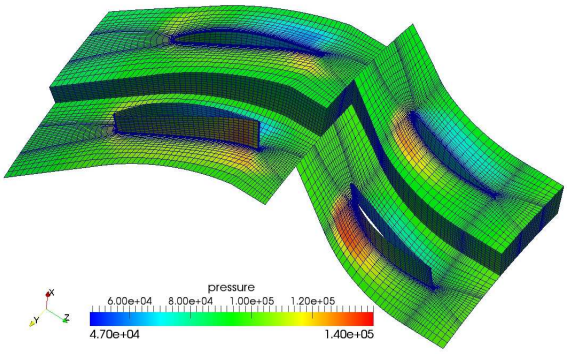

Figure 3: SINGLE PASSAGE SIMULATION, PARTIAL OVERLAP AND CYCLIC GGI.

General Grid Interface (GGI). Referring to Fig. 1 one can establish that a cell left of the interface communicates with a number of cells on the right, depending on relative face overlap. It is possible to interpret the effect of facets by a facet-area-based interpolation scheme across the interface, which acts on discretisation identically as a sliding interface. Interpolation stencil is established by geometrical consideration and weighting factors are calculated as area ratio of the facet and the complete face, [7]. With careful discretisation, implicitness of the coupling, second order of accuracy and strict mass conservation across the interface is preserved.

In practice, mesh sliding with a GGI interface operates in the manner identical to a topologically sliding mesh, but changes are contained in weighting factor calculation.

Cyclic GGI. A non-intrusive technique such as GGI allows us to easily handle cases where the mesh does not match across the pitch, Fig. 4. Flexibility of the code in handling non-matching cyclicity greatly facilitates mesh generation for more complex blade shape.

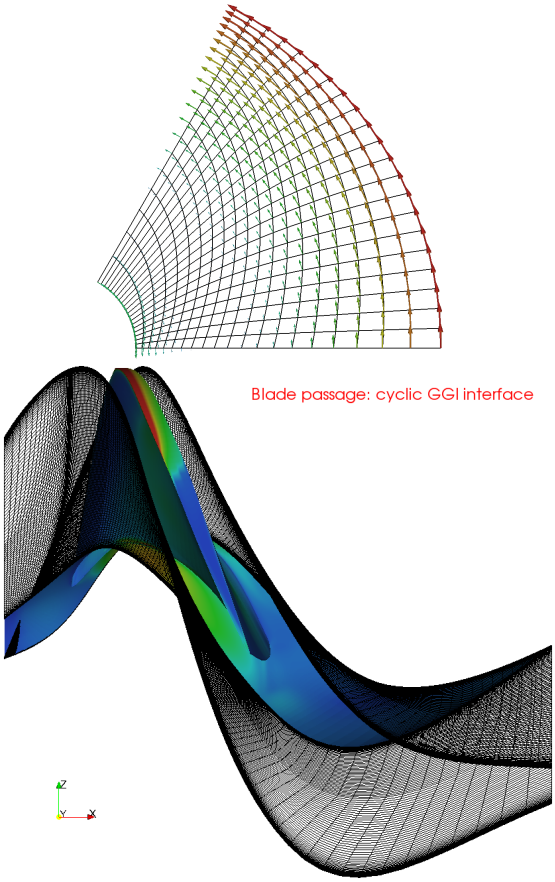

Figure 4: NON-MATCHING CYCLIC GGI.

In terms of implementation, the GGI interpolation is combined with a coordinate transformation across the patch. The slave side of a cyclic GGI is transformed to fall on top of the master and weights evaluation is used without change. For coupled boundary update (implicit or explicit), field

data next to the cyclic boundary will undergo a coordinate transformation to account for *e.g.* angle span. Basic numerics and implicit coupling is identical to the GGI interface above.

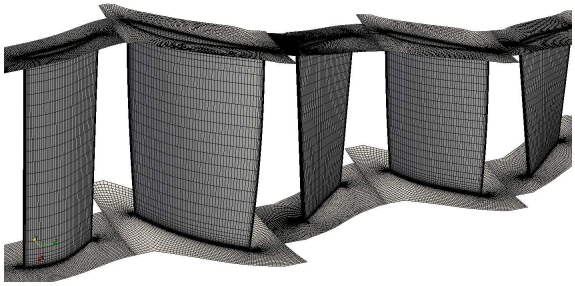

Figure 5: PARTIAL OVERLAP GGI INTERFACES IN A MULTI-STAGE COMPRESSOR.

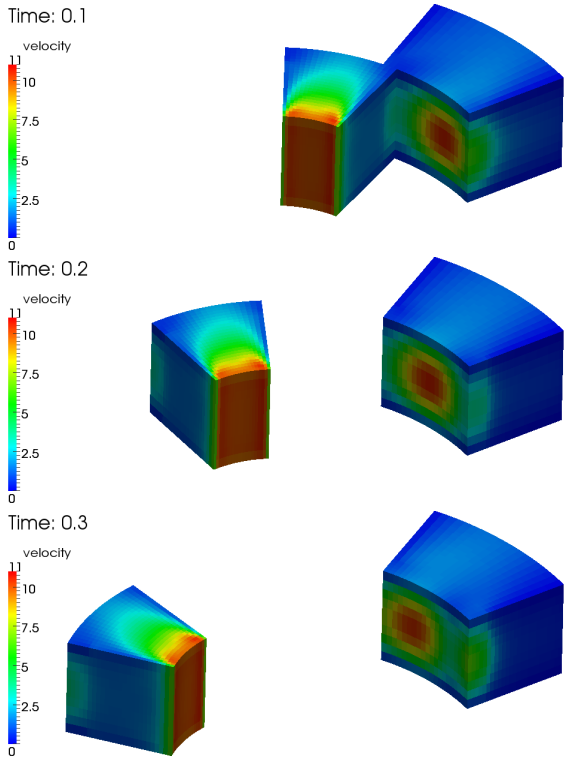

Figure 6: PARTIAL OVERLAP INTERFACE IN ROTATION.

Partial Overlap Interface. In a partial overlap interface, Fig. 5, one chooses to simulate non-matching pairs of rotor-stator interfaces, assuming rotational symmetry. It is assumed that the uncovered part of the interface corresponds to a copy of existing geometry, transformed by the cyclic angle. Depending on user specification, the "upstream" and "downstream" mesh may include either a single or multiple blade passages, possibly allowing for a certain degree of passage-to-passage flow variation.

Simulations of this kind may be performed as transient moving mesh cases, Fig. 6, or in a steady-state mode, using MRF.

In terms of implementation, partial overlap interface is a variant of GGI where patches are expanded to create a complete mapping surface. Once a pair of mapping surfaces is established, field data is transformed accordingly and a GGI interpolator is used. For field updates, only a part of mapping surface is sampled, depending on the relative position of two sides.

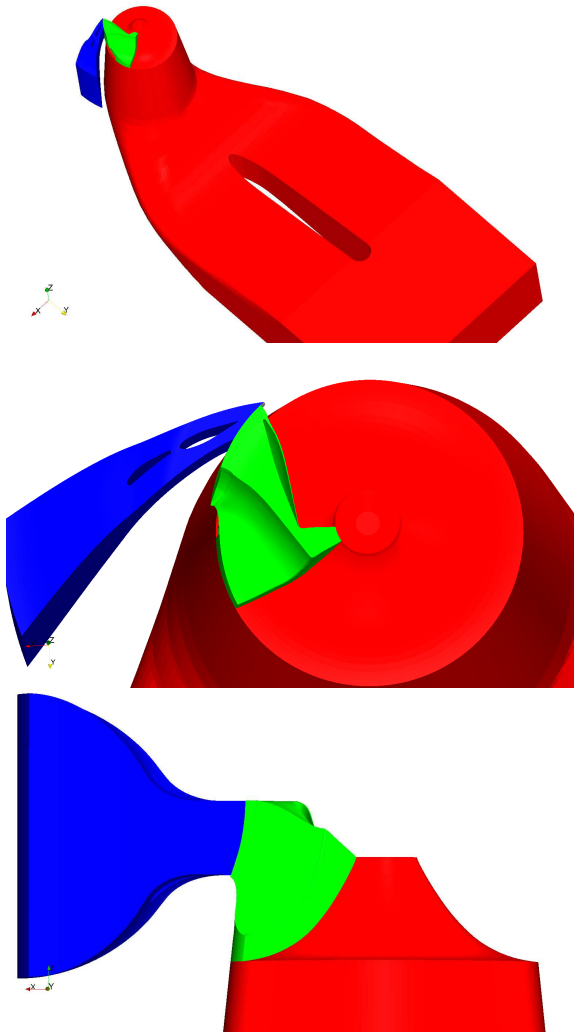

Figure 7: A STEADY-STATE SIMULATION OF A HYDRO-TURBINE ASSEMBLY USING A MIXING PLANE AND MRF.

Mixing Plane Interface. The final variant of a stage interface incorporates surface averaging to remove unsteadiness caused by passage-to-passage flow variation and blade wakes from the solution at the shadow side, both upstream and downstream.

While a mixing plane filter causes a one-time loss associated with mixing, it is extremely useful in practical simulations because it allows a fundamentally transient problem to be studied in steady-state, with the help of MRF. Example assembly of a steady-state turbine problem, with a section of inlet vane (blue), a rotor section (green) and a full draft tube geometry (red) is shown in Fig. 7.

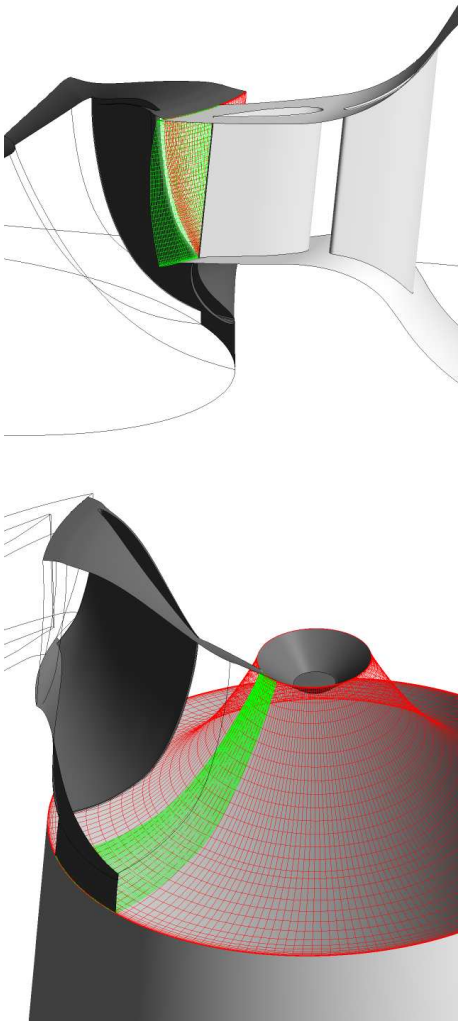

Figure 8: MESH AT A MIXING PLANE.

At the mixing plane, meshes do not match and do not offer full coverage Fig. 8.

A mixing plane interface defines a set of oriented ribbons at the coupled boundary over which the solution is averaged, schematically shown in Fig. 9.

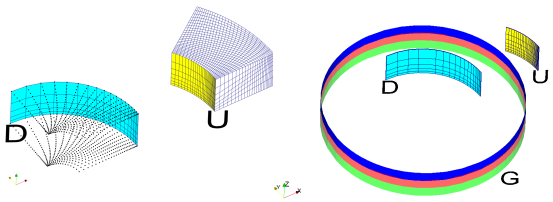

Figure 9: MIXING PLANE INTERPOLATOR.

Size distribution of mixing plane ribbons, denoted as G in Fig. 9, [8]. is established automatically from the upstream and downstream patch, marked as U and D, respectively. Each side will see the solution at its shadow as a set of averaged values for each ribbon. Clearly, ribbons of the mixing plane may be defined in an arbitrary coordinate system and orientation, which in turn defines the manner of filtering.

Based on the above, implementation of a mixing plane interface is straightforward. When the 1-D ribbon profile is established, ribbons are expanded spanwise. Two GGI interpolators are set up, with the ribbon patch acting as master in both. On data transfer, fields from patches U and D are transformed into the reference position at the ribbon and interpolated onto the master. Geometrical structure of mixing plane ribbons with respect to original patches will implicitly perform averaging. The shadow side uses the second GGI interpolator to expand the ribbon data onto the complete patch. Mixing plane averaging takes place naturally within two GGI interpolations, as a consequence of shape of the mixing patch and facet area-based assembly of GGI weights [7].

IMPLEMENTATION LAYOUT

In what follows, some details of implementation of turbo tools in OpenFOAM shall be given.

Rotating Frames of Reference. Single rotating reference frame simulations are achieved through reformulation of the momentum equation in terms of relative velocity, with addition of centrifugal and Coriolis forces to the source terms. Looking at examples of code implementation, this is a straightforward task. In order to facilitate user interaction, boundary conditions for $\mathbf{u}_{rel}$ are extended to allow

the user to specify a boundary velocity either in a rotating or in a stationary frame of reference.

For multiple rotating frames, rotation-related sources are only present in parts of the domain. In the *absolute velocity formulation*, $\mathbf{u}$ is used as the working variable and the MRF model assembles the Coriolis source term from regions rotating at different speeds. Volumetric face flux is calculated directly for the fictitious moving mesh velocity, based on $\boldsymbol{\omega} \times \mathbf{r}$ for the rotating coordinate system. In the *relative velocity formulation*, $\mathbf{u}_{rel}$ is chosen as the working variable and forces resulting from rotation are added to the momentum equation in the rotating reference frame. For consistency, jump conditions in $\mathbf{u}_{rel}$ are required at stage interfaces.

The absolute velocity formulation, is preferred: it does not require a transformation of the velocity vector at the interface between rotating zones, making the code easier to handle.

Coupled Boundaries. All stage interfaces described in this paper full under the interface of a coupled boundary condition in OpenFOAM and share a number of desirable properties:

- Interpolation operator is symmetric in all cases and no further information is required, *e.g.* on flow direction at the interface or integral variable adjustments;
- Data transfer is localised to the cell layer next to the interface and may be accompanied by coordinate transformation depending on interface type;
- GGI weights calculation algorithm is shared between different types of interfaces. GGI interpolation operator based on geometric weights assembly [7] has proven to be strictly conservative in the context of FV discretisation.

OpenFOAM handles coupled implicit interfaces through a set of virtual function updates, occuring at patch field evaluation and matrix-vector operations which represent a basis of linear equation solvers. Other places in the code are served by geometric interpolation coefficients and face area vectors. When a coordinate transformation is required in data exchange across the interface, it is served through a virtual function interface, fetching and transforming information from the shadow side.

Examples of implicit coupled interface in OpenFOAM include a cyclic or processor boundary. All new boundary conditions adhere to the same interface: as a result, no further changes are required anywhere in the software and all implemented physical models can readily use the turbo tools functionality.

EXAMPLES OF TURBOMACHINERY SIMULATIONS

In what follows, we shall illustrate the performance of OpenFOAM flow solvers utilising turbo tools.

The ERCOFTAC Centrifugal Pump. A detailed study of the ERCOFTAC centrifugal pump with a vaned diffuser has been presented by Petit *et al.* [9]. Fig. 2 shows the mesh with the rotor coloured in blue; a mesh detail shows that the rotor-stator interface does not match at and stage of the simulation.

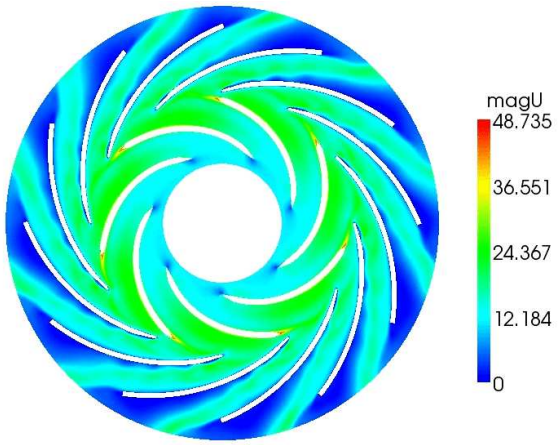

Figure 10: ERCOFTAC CENTRIFUGAL PUMP, VELOCITY FIELD.

Due to full coverage between the rotor and stator, both dynamic mesh and GGI may be used. Simulation has been performed in transient mode using a GGI interface. Fig. 10 shows a snapshot of the velocity field magnitude. The rotor-stator interface is invisible in the solution, indicating no problems with the performance of GGI. Substantial passage-to-passage flow variation has been observed. For

further details of case setup, flow conditions and experimental comparison of simulation results, the reader is referred to [9].

Simulation of Marine Propulsors. Kim *et al.* [10, 11] report a series of simulations on prediction of cavitation and performance of marine propellers and axial flow water-jer propulsors. For the water-jet, the case setup is schematically identical to Fig. 2, with a single fully covered GGI interface. Simulations of propellers in open water utilises the geometry of a single blade passage and a steady-state simulation in a single rotating frame of reference. For such meshes, a cyclic GGI is used to handle pitch-wise non-matching mesh. Fig. 11 shows the pressure distribution on the blade and a mesh structure of cyclic GGI patch.

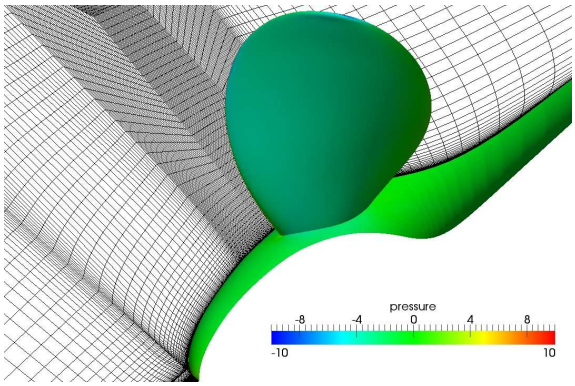

Figure 11: SINGLE PROPELLER BLADE WITH A CYCLIC GGI INTERFACE.

Mixing plane simulation of a Francis Turbine. Beaudoin *et al.* [8] report a steady-state simulation of a Francis turbine, using the MFR model and two mixing plane interfaces. A 195-MW Francis turbine is shown schematically in Fig. 7. Details of the mesh at the interface between the single passage mesh for the Francis runner and the complete geometry of the draft tube are shown in Fig. 8. The mesh for the complete geometry consists of 2.2 million cells, Fig. 12. Simulations have been performed at part load, maximum efficiency and maximum power.

Fig. 13 shows the pressure distribution at distributor mid-height for the stator and runner at three operating conditions, illustrating the performance of the mixing plane.

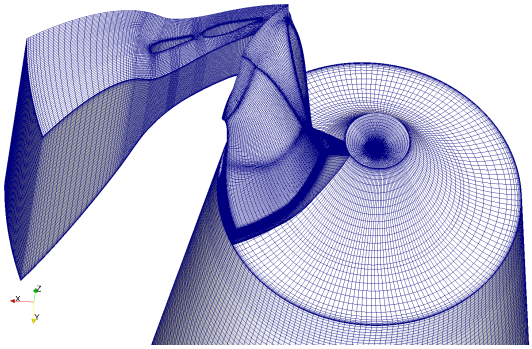

Figure 12: FRANCIS TURBINE MESH ASSEMBLY WITH TWO MIXING PLANES.

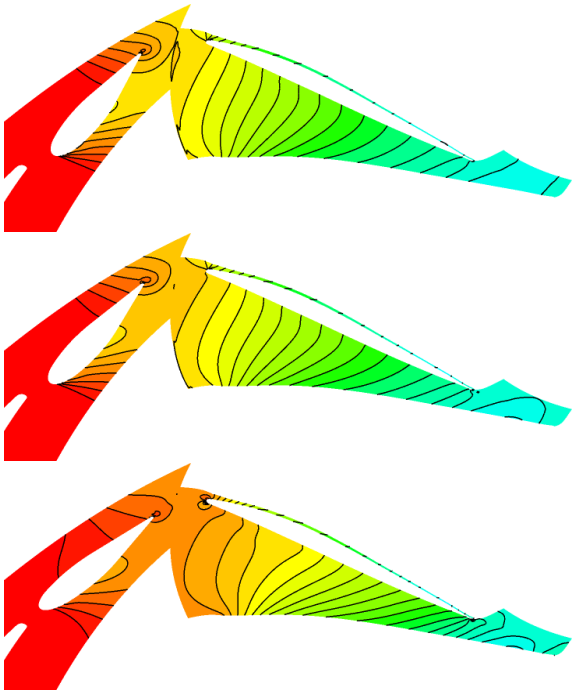

Figure 13: PRESSURE DISTRIBUTION AT DISTRIBUTOR MID-HEIGHT.

The most notable difference in flow structure is visible in the draft tube. Under part-load conditions, the vortical flow is not well organised and shows substantial transient effects, while at optimum efficiency and single large-scale vortex is formed instead. FOr further details, the reader is referred to [8].

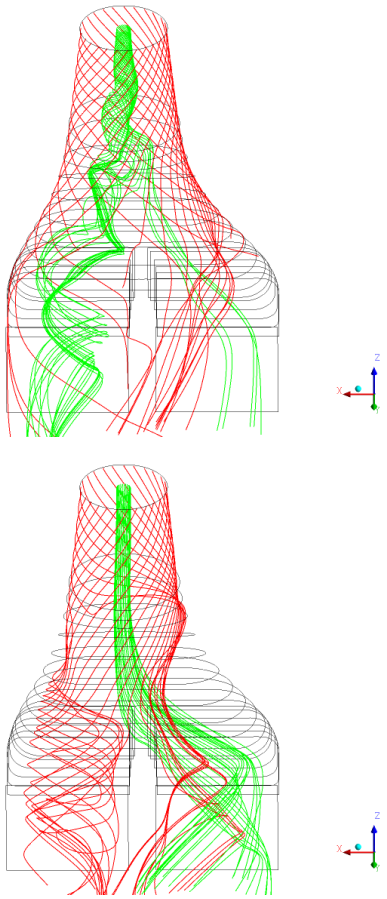

Figure 14: FLOW FIELD IN DRAFT TUBE.

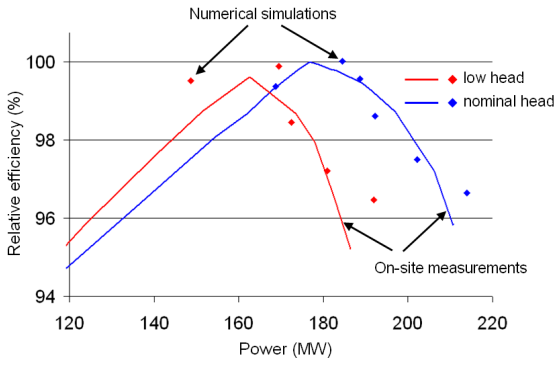

Figure 15: SIMULATED AND MEASURED EFFICIENCY OF A FRANCIS TURBINE.

For the chosen geometry, on-site efficiency measurements are available for low head and nominal head. Losses and efficiency predicted by CFD simulations can be compared to on-site experimental measurements in shown in Fig. 15. Agreement at this stage is not perfect, with some of the blame attributed to lack of mesh resolution in critical parts of the domain, simplification of flow conditions at the mixing plane and choice of turbulence model.

SUMMARY

This paper describes the formulation and implementation of turbo tools in OpenFOAM, designed to enable the code to deal with typical techniques for CFD simulations of rotating machinery. Object-oriented design allows us to deploy the handling for rotor-stator interfaces and simulation in rotating frames of reference with minimal intervention in existing source code.

Implemented features include handling for single- amd multiple rotating frames of reference and a suite of rotor-stator interface handling techniques. A General Grid Interface (GGI) mimics the operation of a topological sliding interface by direct intervention at discretisation level. Interpolation stencil and weighting factors in GGI are calculated from geometric considerations and are proven to be strictly conservative. Other interfaces re-use the basic interpolation functionality, accompanied by appropriate coordinate transformation or averaging.

With turbo tools, capabilities of OpenFOAM in turbomachinery simulations are significantly improved, aiming towards a truly versatile and efficient open source CFD solution.

ACKNOWLEDGEMENT

Authors would like to acknowledge the contribution of partners and industrial sponsors supporting this work in OpenFOAM, in particular our colleagues in the Turbomachinery Special Interest Group.

BIBLIOGRAPHY

[1] Jasak, H: “Dynamic Mesh Handling in OpenFOAM”: 48th AIAA Aerospace Sciences Meeting, AIAA Paper, January 2009: Orlando, FL.

[2] Weller, H.G., Tabor, G., Jasak, H., and Fureby, C.: “A tensorial approach to computational continuum mechanics using object orientated techniques”, *Computers in Physics*, 12(6):620 – 631, 1998.

[3] Jasak, H.: “Multi-Physics Simulations in Continuum Mechanics”, In *Proceedings of 5th International Conference of Croatian Society of Mechanics, Trogir*, page ?? Croatian Society of Mechanics, September 2006.

[4] Jasak, H. and Tuković, Ž: “Dynamic Mesh Handling in OpenFOAM Applied to Fluid-Structure Interaction Simulations”, In Pereira, J. C. F. and Sequeira, A., editors, *ECCOMAS CFD 2010*, June 2010.

[5] Jasak, H. and Tuković, Ž.: “Automatic Mesh Motion for the Unstructured Finite Volume Method”, *Transactions of FAMENA*, 30(2):1–18, 2007.

[6] Lucchini, T., D’Errico, G., Jasak, H., and Tuković, Ž.: “Automatic mesh motion with topological changes for engine simulation”: SAE Technical Paper 2007-01-0170, 2007.

[7] Beaudouin, M. and Jasak, H.: “Development of a General Gid Interface for turbomachinery simulations with OpenFOAM”: Open Source CFD International Conference, November 2008.

[8] Page, M., Beaudoin, M., and Giroux, A.-M.: “Steady-state Capabilities for Hydroturbines with OpenFOAM”, *Int. J. of Fluid Machinery and Systems*, 4(1):160–170, Jan-Mar 2011.

[9] Petit, O., Page, M., Beaudoin, M., and Nilsson, H.: “The ERCOFTAC Centrifugal Pump OpenFOAM Case-Study”, In *3rd IAHR International Meeting of the Workgroup of Cavitation and Dynamic Problems in Hydraulic Machinery and Systems*, pages 523–532, Brno, Czech Republic, October 2009.

[10] Kim, S.-E., Scroeder, S., and Jasak, H.: “A Multi-Phase CFD Framework for Predicting Performance of Marine Propulsors”, In *The Thirteenth International Symposium on Transport Phenomena and Dynamics of Rotating Machinery (ISROMAC-13)*, Honolulu, Hawaii, April 2010.

[11] Schroeder, S., Kim, S.E., and Jasak, H.: “Toward Predicting Performance of an Axial Flow Waterjet Including the Effects of Cavitation and Thrust Breakdown”, In *First International Symposium on Marine Propulsors , Trondheim, Norway*, June 2009: http://www.marinepropulsors.com/.