

Parallelization and Load Balancing of a Dynamic Mesh Method for Moving Boundary CFD

Mattias Liefvendahl and Carl Troëng
Swedish Defense Research Agency, FOI

November 19, 2008

Summary

A method for parallel CFD, for problems with large boundary motion, is presented. Two different parallelization schemes are described and compared with respect to load balancing and computing time. The methods are evaluated on a model problem consisting of the flow around a submarine with a complete propeller model. Results concerning parallel computational performance and the computed flow field are presented.

1 Introduction

Applications with moving boundaries constitute a large and diverse class of problems which, naturally, cannot be treated with standard, fixed domain, CFD-methods. In this paper, we will describe a method based on deformation and regeneration (referred to as D&R below) of the computational grid, which can be applied to many cases of great importance for industrial applications, such as rotating components (turbo-machinery, propellers), linear motion (of e.g. pistons), rudder motion and more general relative motion. For these applications, it is not sufficient to stretch or deform the grid, because the boundary motion is too large.

The D&R-method, and its implementation in OpenFOAM, was described in [9] and [10], where the discussion and results were restricted to simulations on one processor. Here we will describe how to parallelize the algorithm for simulation on distributed memory machines, typically a Linux cluster. The method includes additional steps, compared to a fixed grid simulation, such as interpolation of the flow variables between topologically different grids. Furthermore, the regeneration of the computational grid, in a limited region, introduces constraints on how the grid is distributed on the processors. Several approaches are possible and we investigate how they affect the load balancing, and thereby the parallel performance.

This paper addresses the question of parallelizability and scale-up of all parts of the algorithm. Computational results are given for a model problem of a rotating propeller attached to a model submarine hull. The demonstrator geometry consists of a simplified submarine hull and a surface ship propeller of old design. The purpose of the case is to demonstrate the simulation capability, not to investigate a real operational configuration. The results include parallel scale-up performance and flow visualizations.

We will also discuss the OpenFOAM implementation of the algorithm. This includes the use of existing utilities and solvers, as well as modification and writing of separate programs which are linked with the OpenFOAM library. The focus will be on the moving mesh, interpolation and parallelization functionalities. In our implementation, we use OpenFOAM version 1.3.

2 The dynamic mesh CFD-method

We give an overview of the D&R-method, first on an algorithmic level, and then we briefly describe its implementation for simulation on a single processor. The material in this section has been described in more detail in [9] and [10], here we collect the necessary background material for the discussion of the parallelization, in section 3, which is the main new contribution of the present paper.

Now we explain some terminology used extensively below. We talk generally about a *dynamic* meshes and differentiate between mesh *deformation* and the change of mesh *topology*. By topology, we mean here the connectivity of the mesh, i.e. the list of neighboring cells of each cell. A deforming mesh does not change topology but the mesh nodes are moving, thereby deforming the cells.

For the cells and cell faces we use OpenFOAM-terminology, see table 6.1, on page 137 of the User's Guide, [2]. Thus hexahedral cells are abbreviated to hex-cells. Tetrahedral cells to tet-cells. The six faces of a hex-cell are referred to as quad (quadrilateral) faces. A prismatic cell has two triangular faces and three quad-faces, this type of cell is referred to as a prism-cell.

2.1 High-level algorithmic description

The D&R-method was described in a quite general setting in [9]. On this level of generality, the method is to “dynamically” (during the time-stepping) decide, based on an appropriate mesh quality measure, when to regenerate the mesh, and also, in which part of the complete simulation domain this is to be done. In this paper we restrict ourselves to prescribed boundary motion described by one parameter and taking place in a part of the computational domain which can be identified before the simulation is started. The reason for this restriction is that it makes it possible to make a parallel implementation which takes advantage of the specific properties, of this restricted situation, to address issues of the parallel performance and load balancing. Our work is mainly motivated by naval hydrodynamics, and two applications in this field which falls in the category of problems treated here is rudder motion and propeller rotation. Other application areas include e.g. piston motion in internal combustion engine simulations.

In the above setting, we assume that the moving domain $V(\alpha) \subset \mathbb{R}^3$ can be decomposed into a disjoint union of three regions,

$$V(\alpha) = V_f \cup V_t(\alpha) \cup V_{rb}(\alpha).$$

Here $\alpha \in \mathbb{R}$ is the parameter describing the motion and V_f denotes a fixed region where the same mesh can be used throughout the simulation. The region V_{rb} moves as a rigid body and here the same mesh, submitted to the rigid-body rotation, is also used during the simulation. The preceding two regions are matched together by a transition region V_t . It is in the transition region that the deformation and regeneration of the mesh takes place. The computational grid is boundary fitted, meaning that it covers $V(\alpha)$ and does not extend outside of $V(\alpha)$.

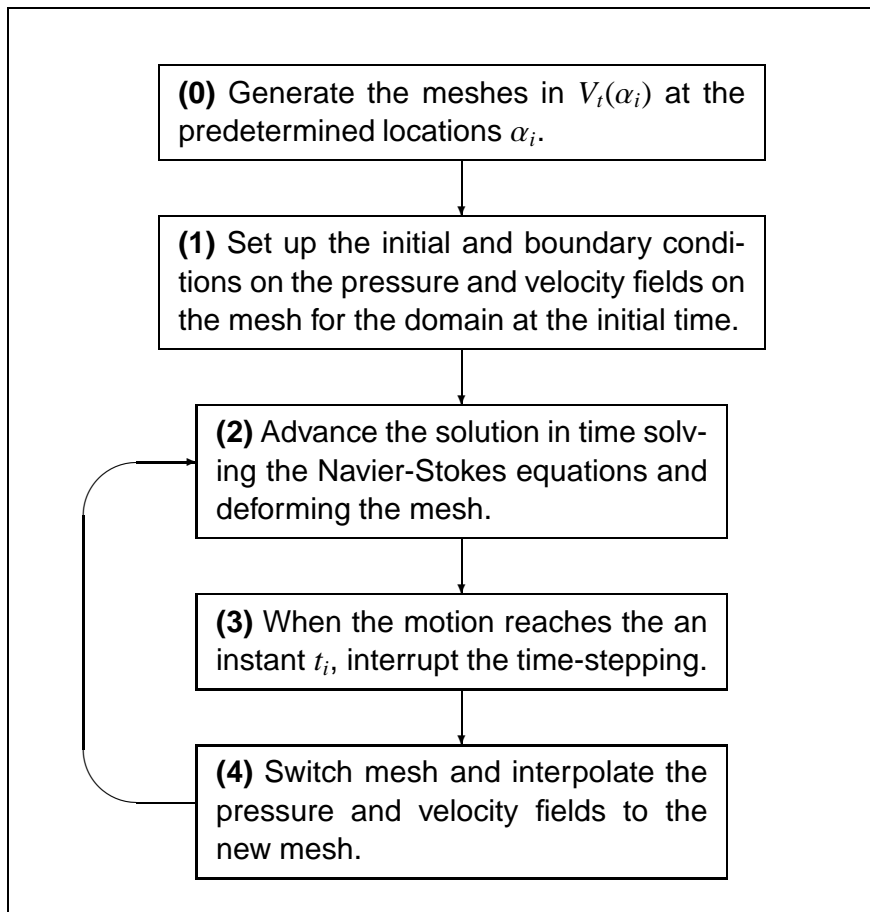


Figure 1: Flow chart for the D&R-method restricted to prescribed one-parameter motion of the boundary.

The above construction is illustrated with the model problem for which we present computational results in section 4. In general we have in mind a “craft” (mainly contained in V_f) with a moving component, e.g. rudder or propeller (contained in V_{rb}).

Since the boundary motion is described by one parameter it is possible to generate the necessary meshes in V_t before the flow simulation is started. Suppose that the motion will take place in the interval $\alpha \in [A_L, A_R]$, then this interval can be divided into subintervals where (topologically) different meshes are used. For this we introduce α_i according to,

$$A_L = \alpha_0 < \alpha_1 < \dots < \alpha_i < \dots < \alpha_n = A_R.$$

The i :th mesh is then used in the interval $\alpha \in [\alpha_{i-1}, \alpha_i]$ and for motion in this interval, the mesh is deformed by one of the algorithms described in section 2.2. When the motion reaches one of the end points of this interval, we switch mesh to the one “covering” the corresponding interval. The numbers α_i and the corresponding $V_t(\alpha)$ -meshes should be constructed by taking the following trade-off into account. Large α -intervals lead to poor quality mesh near the ends of the interval but at the same time leads to few mesh switches. Typically the meshes in V_t are constructed using automatic tet-meshing, see below, at the center of the corresponding α -interval.

Since the motion is prescribed, the parameter is a (given) function of time $\alpha = \alpha(t)$. Thus the mesh switching locations α_i corresponds to mesh switching instants t_i such that $\alpha_i = \alpha(t_i)$.

In figure 1, we show a flow chart illustrating the complete D&R-method. Step (0), concerning mesh generation, is described further below in section 2.2. Step (1) and (2) consists of the solution of the Navier-Stokes equations in a domain with *small* boundary motion so that the mesh can be deformed, and regeneration is not possible. This is not the focus of this paper, more information about the methods employed can be found in [9] and [7]. Step (3) concerns the predetermined instants/locations (t_i/α_i) when the mesh is to be switched. Step (4) is the interpolation of the flow field to the new mesh, this is described in some detail in section 2.3

2.2 Mesh generation, cell quality and mesh deformation

At each time instant the computational domain is covered by a finite volume mesh consisting of hex-, tet-, prism- or pyramid cells. In the D&R-method the, meshes for the regions V_f and V_{rb} can be generated quite independently of each other. In practice, the mesh generation here can be complicated due to the geometry of the moving and fixed parts. In V_t the mesh degrades during the motion and must be regenerated often. We restrict ourselves to considering automatic tetrahedral meshing in this region. We have used two different softwares for the tet-meshing of V_t . The first is TRITET, [11], which is an in-house FOI-code using the advancing front method. The second software is NETGEN, [1]. which is Open Source under LGPL (GNU Lesser General Public License) and uses an algorithm based on Delaunay triangulation.

We use the so called aspect ratio as a quality measure, to monitor the degradation of tet cells during the mesh deformation. The aspect ratio Q of a tet is defined as,

$$Q = \beta \frac{r}{h}, \quad \text{where } \beta = 2\sqrt{6} \quad \text{and} \quad r = \frac{3V}{A}. \quad (1)$$

Here β is a normalization factor which ensures that $Q = 1$ for an equilateral tet, r is the radius of the largest possible sphere inscribed in the tet, h is the length of the longest edge of the tet and V is the volume and A is the total area of the tet cell.

Mesh deformation is a critical component of the overall method. A good deformation method makes it possible to use the (topologically) same mesh for a longer deformation interval $[\alpha_{i-1}, \alpha_i]$ and/or increases the numerical accuracy by good “preservation” of mesh cell quality during the boundary motion. This is however not the focus of the present paper, see [9] and [10] for more information, and section 4 below for our example computation. Interesting methods for mesh deformation include: (1) Computation of the mesh node motion by a weighting of the distance to the moving and stationary boundaries, used in e.g. [8]. (2) Node motion based on the solution of an elliptic equation with variable coefficients, this is implemented in e.g. OpenFOAM version 1.3. (3) A mechanical spring analogy for the grid, [3]. (4) Node motion based on the solution of the elastic (biharmonic) equation, [6].

2.3 OpenFOAM implementation for single processor simulation

In this section we briefly describe our implementation of the D&R method on a single processor using OpenFOAM. More information about this can be found in [10], here we only present the material necessary for the following section.

The “main” program is a shell script which implements the flow chart of figure 1 by executing a series of OpenFOAM utilities and solvers, as well as moving and renaming files and editing files (controlDict). There are three main components involved:

- (A) Pre-processing. Construction of the different meshes needed for the complete α -interval of the simulation.
- (B) Mesh deformation.
- (C) Solution of the Navier-Stokes equations on a deforming mesh.
- (D) Interpolation of the flow field (at every mesh switch).

During the pre-processing stage (A), we use different software to generate the meshes in the regions V_f , V_i and V_{rb} as described in section 2.2. To construct the complete meshes for the domain V we use the OpenFOAM-utilities `mergeMeshes` and `stitchMesh`.

Concerning (B), we restrict ourselves in this paper to the simple situation of mesh deformation in concentric cylinders, which is appropriate for rotating components as our example below in section 4. In this situation it is possible to use explicit algebraic expressions for the node motion between the concentric cylinders as a function of the rotation angle, see [10] for a complete description of this mesh motion implementation.

For (C), the solution of the Navier-Stokes equations on a deforming mesh (without mesh switches) we use a modified version of the application `icoFoamAutoMotion`. The interface for reprogramming the mesh motion algorithm is very simple to use. Essentially, it only consists of the following line of source code which is executed once every time step.

```
mesh.movePoints(newPoints);
```

Here `mesh` is of type `fvMesh` and `newPoints` are the new node coordinates and is of type `pointField`.

(D): When switching to a new mesh, the solution is interpolated to the new mesh with the application `mapFields`.

3 Parallelization

In the section we describe our extension of the D&R-algorithm as described in section 2, to the case of parallel simulation. This is the main new development presented in this paper. The simulations are to be carried out on a Linux cluster using LAM/MPI as provided by OpenFOAM, see section 3.4.2 of the User's Guide, [2]. We remark that modern multi-kernel processor technology can cause confusion in the terminology. Here we use "OpenFOAM-terminology". Thus, for example, a computing "node" with two processors with two kernels each, we will refer to simply as four processors, since LAM/MPI will treat it as four processors and the decomposed (`decomposePar`) OpenFOAM-case will contain four processor directories.

Any computational method with a dynamic or adaptive mesh (i.e. a mesh changing topology during the simulation) encounters difficulties with load balancing, for parallel simulation, which do not occur for fixed grid simulations. As the mesh topology changes the mesh cells must be redistributed among the processors. The main trade-off is between the cost (and algorithmic complexity) of redistributing mesh cells and the inefficiency caused by deteriorating load balancing. It is difficult to devise a general method for all (general) situations, and in this paper we present two solutions for the D&R-method with the restrictions on boundary motion described in section 2.

The OpenFOAM library provides the parallelization of the solver for the Navier-Stokes equations on a deforming (without switch) mesh. The main difficulty is the interpolation step and the associated handling of the two topologically different meshes between which we switch/interpolate. We now turn to this and describe our simplest approach which we denote **P1**. We denote the mesh from which we interpolate by M_i and the mesh to which we interpolate M_{i+1} . At the switching instant t_i both M_i and M_{i+1} cover the domain $V(t_i)$ and on mesh M_i we have computed a velocity and a pressure field, denoted by \mathbf{v} and p respectively.

P1 - *We treat the mesh decomposition of M_i and M_{i+1} completely independently. Thus we apply `decomposePar` to both meshes with the `metis` decomposition method. For the interpolation it is now necessary to reconstruct (using `reconstructPar`) the complete mesh before interpolation and then after interpolation we decompose it again before resuming the time advancement on mesh M_{i+1} .*

We see that **P1** is straightforward to implement. The main and very significant drawback of this method is that the mesh switching is treated essentially on one processor. A very limiting factor for the size (number of mesh cells) of the problems which can be treated by **P1** is the internal memory on the processor used for mesh switching (i.e. running `reconstructPar`, `mapFields` and `decomposePar`). Considering a "large" problem, with $O(10^7)$ cells, then these operations must be run on a dedicated node and not on one of the nodes on the cluster assigned by the typical queue system. If a dedicated node is *not* used, then the maximum size of problems to be treated is limited by the internal memory of the nodes of the cluster. For example, with nodes with 1 Gbyte of internal memory, the maximum number of cells allowed is approximately $1.5 \cdot 10^6$ cells before memory swapping (and very slow execution). The application `mapFields` requires most memory.

To be able to simulate larger problems than those allowed by **P1**, we have devised a second parallelization method, denoted by **P2**. This method relies on the fact that it is only for the mesh in the region V_t that interpolation is necessary. For V_f and V_{rb} , the same mesh is used before and after the mesh switch.

P2 - The meshes on the transition region M_i^t and M_{i+1}^t are distributed to the same number processors. The same applies to the meshes in the fixed region M_i^f/M_{i+1}^f and the rigid-body motion region M_i^{rb}/M_{i+1}^{rb} . In all regions `decomposePar` is used with `metis`. The decomposition can be done before the simulation is started. For the interpolation we reconstruct only the mesh M_i^t , interpolate this and decompose it before resuming the time advancement on mesh M_{i+1} .

The method **P2** increases the size considerably of the problems which can be computed. The restriction that the meshes of the different regions must be distributed on different groups of processors can impede the load balancing, which of course is a drawback. It is also necessary to implement a special reconstruct/decompose step for the transition zone V_t . In fact this has not been done yet and our current implementation requires that the transition zone is computed on one single processor, we refer to this as method **P2**¹.

4 Example: Flow around a submarine with a complete propeller model

In this section we illustrate the D&R-algorithm, as described in sections 2 and 3 above, by presenting computational results for a model problem which contains most of the computational complications of a realistic application. Using the parallelized D&R-method, we compute the flow around a submarine with a complete propeller model. The geometries do not reflect realistic submarine designs. The hull is generic, and was designed at the David Taylor Research Center (DTRC), U.S.A. for the purpose of CFD validation with experimental data. The hull is called SUBOFF and its shape is described in [5]. The propeller is chosen to be E779 which was designed and experimentally investigated at the Italian Ship Model Basin (INSEAN), also for the explicit purpose of CFD validation. Velocity measurements for the propeller in open water conditions (homogeneous inflow) are presented in e.g. [4].

Next in this section, we describe the general set-up and parameters of the problem, then, in section 4.1, we present results of the computational performance and compare the two parallelization methods **P1** and **P2** described in section 3. Finally, in section 4.2 we present the computed flow field to illustrate the time-scales and flow features which are characteristic for this type of simulation. There is no quantitative comparison since no experiments have been performed for the configuration.

The parameters we have chosen for the simulation are given in table 1. A full scale submarine simulation is associated with fluid flow at a very high Reynolds number, $Re_L = \mathcal{O}(10^8)$. The parameters we have chosen give a significantly lower Reynolds number which can be thought of as model scale, as would be used in experiments in a towing tank. The submarine hull is enclosed in a large cylindrical domain with a prescribed constant velocity on the inflow boundary, zero-gradient for the pressure field on the outflow boundary and the slip-condition on the remaining part of the cylindrical surface. As initial condition, we use a constant values $\mathbf{v} = (V_\infty, 0, 0)$ and $p = 0$ throughout the domain.

The first step of the D&R-method is to identify the mesh regions V_f , V_t and V_{rb} . As V_{rb} we chose a small cylinder enclosing the propeller and rotating with it as a rigid body. The

Quantity	Notation	Expression	Unit	Value
Submarine length	L	-	m	1.092
Propeller diameter	D_P	-	m	0.065
Submarine hull diam.	D_S	-	m	0.127
Kinematic viscosity	ν	-	m ² /s	$6 \cdot 10^{-5}$
Inflow velocity	V_∞	-	m/s	1
Propeller rotation freq.	n	-	1/s	27.78
Advance number	J	$V_\infty/(nD_P)$	-	0.554
Reynolds number	Re_L	$V_\infty L/\nu$	-	18 000

Table 1: Notation and parameters for the SUBOFF + E779 test case.

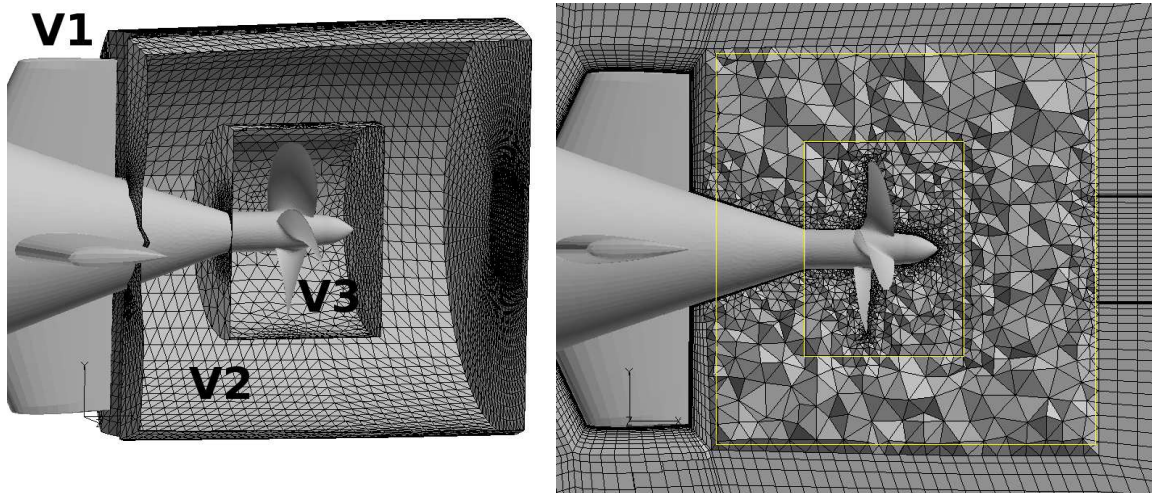
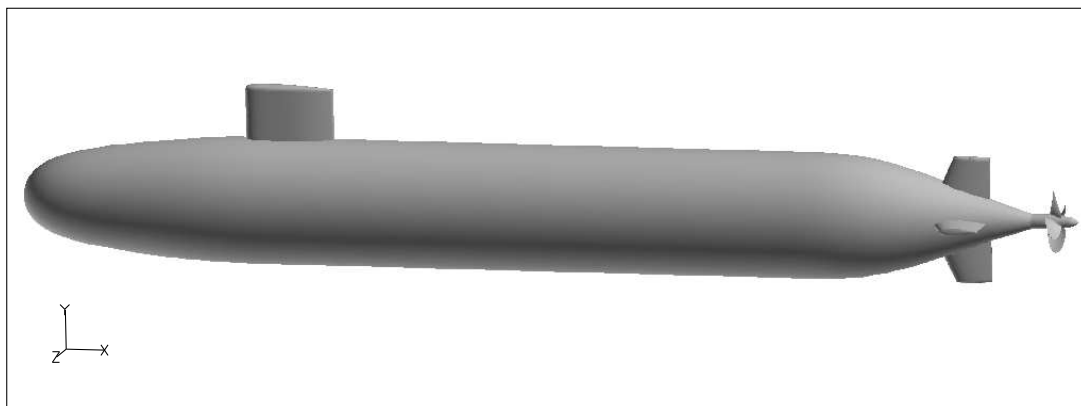


Figure 2: Above, we have the submarine hull and propeller geometry (SUBOFF + E779), the enclosing large cylinder is not shown. The two lower pictures illustrate the division into sub-domains and the meshing of the stern region. To the left is indicated the three mesh regions, the fixed region (V1), the transition region (V2) and the region rotating as a rigid body (V3). To the right we see the mesh on the center plane. Here it can be seen where hex and tet cells are used respectively. The concentric cylinders used for the mesh motion computation are illustrated with the yellow lines.

	Prism	Hex	Tet	All
Mesh in V_f	0	755	0	755
Mesh in V_{rb}	71	0	197	268
Mesh in V_t	7	0	71	78
Complete mesh	78	755	268	1 101

Table 2: Number of cells (in *kCells*, thousands of cells) of the different types and in the different regions. The last column contains the sum of the values in the three previous columns. The complete mesh thus consists of approximately $1.1 \cdot 10^6$ cells and the transition region contains 7% of the total number of cells.

transition region V_t is chosen between this cylinder and a larger one enclosing it. The remaining part of the domain, containing the submarine hull, is then the fixed region V_f . This decomposition is illustrated in figure 2.

The second step consists of generating the mesh for the different sub-regions. We will describe this very briefly next. It is common practice to prefer hex cells in naval hydrodynamics because of the streamlined shape of ship hulls and the expected good accuracy of aligned block-structured hex cell grids. Therefore, the region V_f is meshed using hex cells. For the propellers however, it is often exceedingly time-consuming to construct a hex mesh. For this reason, the region V_{rb} is meshed with tet cells. Both for the hull and the propeller it is necessary to have flat cells next to the boundary to resolve the thin boundary layer. In V_f the boundary layer mesh consists of flat hex cells while flat prism cells are used on the propeller blades and hub.

The final phase of the mesh generation consists of generating a set of meshes for V_t to be used throughout the complete propeller rotation and which connects the regions V_f and V_{rb} . Here several design choices can be made, but since this is not the focus of the present paper, we just describe our solution for this example simulation. The main part of V_t is meshed using tet cells. To connect with the hex mesh in V_f we split every quad face on the boundary between V_f and V_t into two triangles, which are used as input to the tet mesher. A specific complication of the present case is the propeller axis which also needs a boundary layer mesh. The propeller axis is meshed using flat prism cells which are matched to the boundary layer mesh in V_f and V_{rb} . This is thus a slight modification of our general description in section 2 where we stated that V_t is meshed using tet cells. The resulting number of cells of the different types and in the different regions is summarized in table 2.

For the mesh deformation we have a special case with node motion taking place in between concentric cylinders. This makes it possible to derive algebraic expressions for the node motion. These expressions are based on a linear weighting of the rigid body rotation of the inner cylinder and the stationary outer cylinder. Only the geometrical shape of the cylinders, and the location of the node to be moved, are involved in the expression. It is thus not necessary to, for example, search for the closest neighboring node on the boundary. We do not give the node motion expressions here since it would then be necessary to introduce relatively much notation. We refer to [10] for a fuller description of this mesh deformation method. In the lower left picture in figure 2 the cylinders are indicated for the present case. For a complete rotation we use 36 different meshes for V_t . Every one of these is used for 10° rotation and they are generated at the middle of their respective intervals. Thus the

first mesh is generated at a rotation angle of 5° and it is used in the interval $[0^\circ, 10^\circ]$, the second is generated at 15° and is used in the interval $[10^\circ, 20^\circ]$, etcetera. The cell quality during the a complete propeller revolution is illustrated in figure 3.

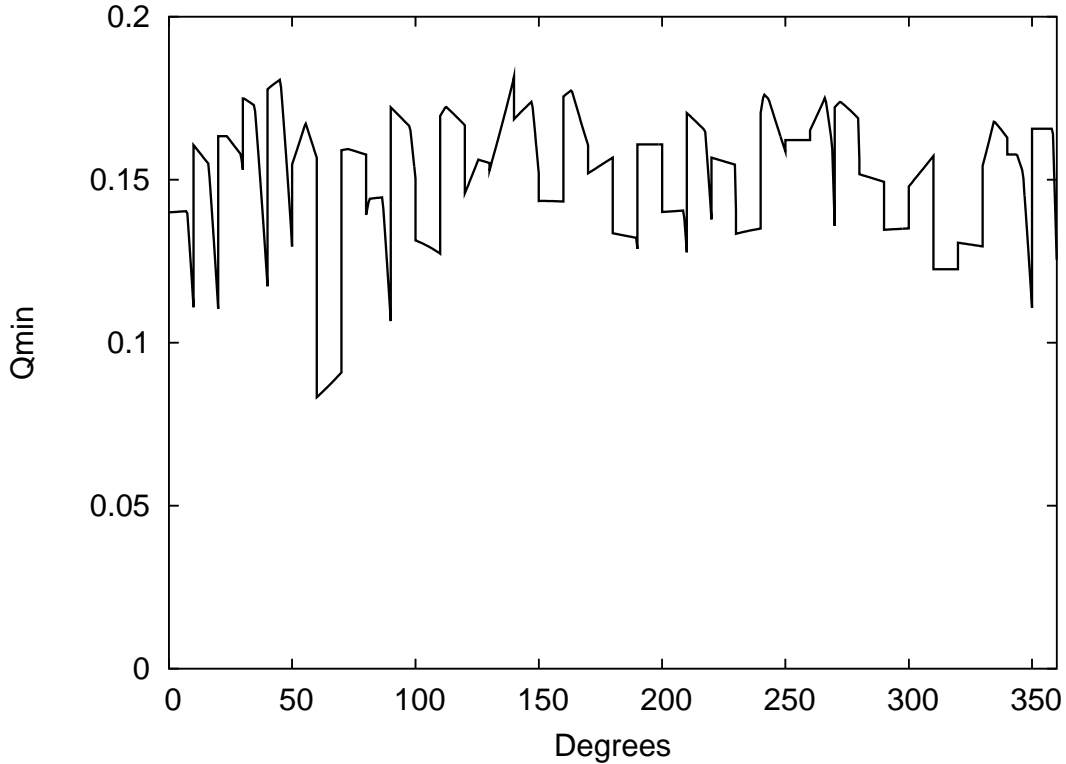


Figure 3: Quality, as defined by the aspect ratio given in equation (1), of the worst cell Q_{min} , as a function of rotation angle. The discontinuities in the curve indicate mesh switches which occur every 10° .

4.1 Performance results

Here we present the computational time for the two parallelization schemes **P1** and **P2**¹ described in section 3, applied to the current problem for simulation on 1-10 processors. The main part of the simulation time is spent during the solution of the Navier-Stokes equations on the deforming mesh, unless the internal memory is insufficient for the necessary operations in **P1** as discussed in section 3. We introduce the following notation. The number N denotes the total number of nodes used in the simulation, n_{cell} is the maximum number of cells on any processor and n_{face} is the maximum number of boundary faces on any processor. In the case of **P2**, each mesh region is assigned to separate groups of processors. We denote the number of processors used for the fixed region by N_f , the number of processors for the transition region by N_t and the number of the rigid-body rotation region by N_{rb} . We thus have $N = N_f + N_t + N_{rb}$.

In tables 3 and 4 the computational times for the flow solution can be compared. The computed time interval is the same for all simulations and it is shorter than the time between two mesh switches, so it is only the item (2) in the flow chart of figure 1 that is accounted for. The simulations have also been performed on a single processor and values in the speed-up column are the ratio of the computation time to that on a single processor. We note that **P2** gives less flexibility in the choice of number of processors but that both 5 and

N	n_{cell}	n_{face}	Time(s)	Speed-up
2	553 324	4 258	723	1.28
4	277 968	8 518	457	2.03
8	140 277	9 719	265	3.51
10	113 426	10 545	220	4.23

Table 3: The computational time for the solution of the Navier-Stokes equations on a deforming grid with a decomposition produced by method **P1**.

N_{rb}	N_t	N_f	N	n_{cell}	n_{face}	Time(s)	Speed-up
1	1	3	5	268 248	16 298	421	2.16
2	1	5	8	152 440	16 993	225	4.14

Table 4: The computational time for the solution of the Navier-Stokes equations on a deforming grid with a decomposition produced by method **P2**¹.

8 processors give good load balancing and gives virtually the same execution speed as the comparable simulations with **P1**. We note that the scaling is far from linear but we believe that this is connected with the processor architecture of the FOI Linux cluster used for this study. The purpose of the comparison to compare **P1** and **P2**¹ and there is no reason to believe that the scaling is worse than for any other typical OpenFOAM solver.

Next we compare the time advancement stage with the mesh switching stage. On a single processor the typical computational time to advance 10^o is approximately 6 hours, depending on the current flow status which influences the necessary number of iterations in the linear equation solvers. As seen from the speed-up given in tables 3 and 4 this is reduced to less than 1h 30min for some simulations. The combined time for **P1** for the reconstruction, interpolation and decomposition is less than 10 minutes. Thus clearly, the time advancement is the most computationally expensive part and it is crucial to have a good load balancing so that this stage is not slowed down. From tables 3 and 4 we see that it was possible to achieve this for the present problem in the range 1-10 processors and the parallelization method **P2**¹ which allows for the simulation of much larger problems than **P1**. Finally we remark that, for the present problem with 1.1 Mcells, the computational time would not decrease significantly if more processors are used, since then the interprocessor communication becomes dominant since each processor is allocated very few cells.

4.2 Flow field results

In this section we show some illustrations of the computed flow field to give the reader some idea of the type of flows involved. As discussed above, this is a model problem with a much lower Reynolds number than a full scale submarine. It however a very suitable problem for a test of principle illustrating the parallelized D&R-method.

In figure 4, the axial velocity in the stern region is shown at two different time instants. We note that at the later time we cannot see a larger number of propeller blade wakes even though the propeller wake clearly extends further downstream. This is due to insufficient mesh resolution in the corresponding region, which in turn lead to high numerical diffusion which smears out the sharp blade wake structures. By introducing a higher mesh resolution in this region more of the propeller wake structure can be captured.

In figure 5, we visualize the magnitude of the velocity on cross-planes and stream lines.

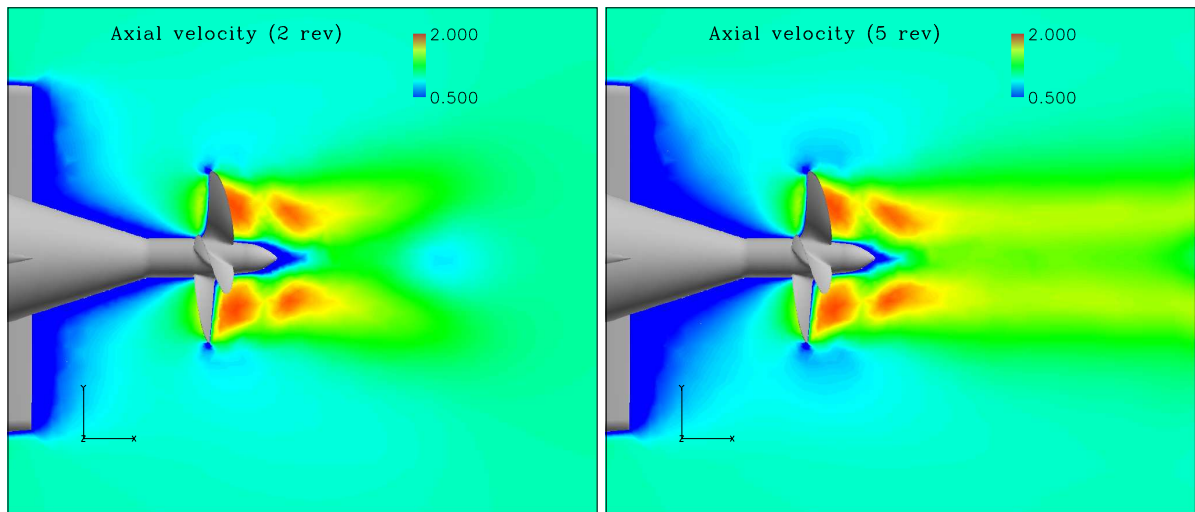


Figure 4: The axial velocity component on the center plane. In the left picture, the propeller has completed two revolutions, $t=0.072$ s, and in the right picture it has completed five propeller revolutions, $t=0.180$ s. As indicated by the color bar, the velocity interval is $0.5\text{m/s} < v_x < 2.0\text{m/s}$. The propeller provides a strong thrust and immediately behind it, the flow speed is twice the free-stream velocity. We note how the propeller wake has evolved from 2 to 5 revolutions. The “pulsating” behavior, which can be seen in both pictures, with two high velocity regions behind the blades, is caused by the blade passages. The thin region with low velocity in between is the blade wake. In a real application (high Reynolds number) these pulsations “survive” much further downstream. Here we have a relatively low Reynolds number and coarse mesh resolution, so they are quickly dissipated into a smoother, high velocity, propeller wake, and we only see two high velocity pulsations both after 2 and 5 propeller revolutions.

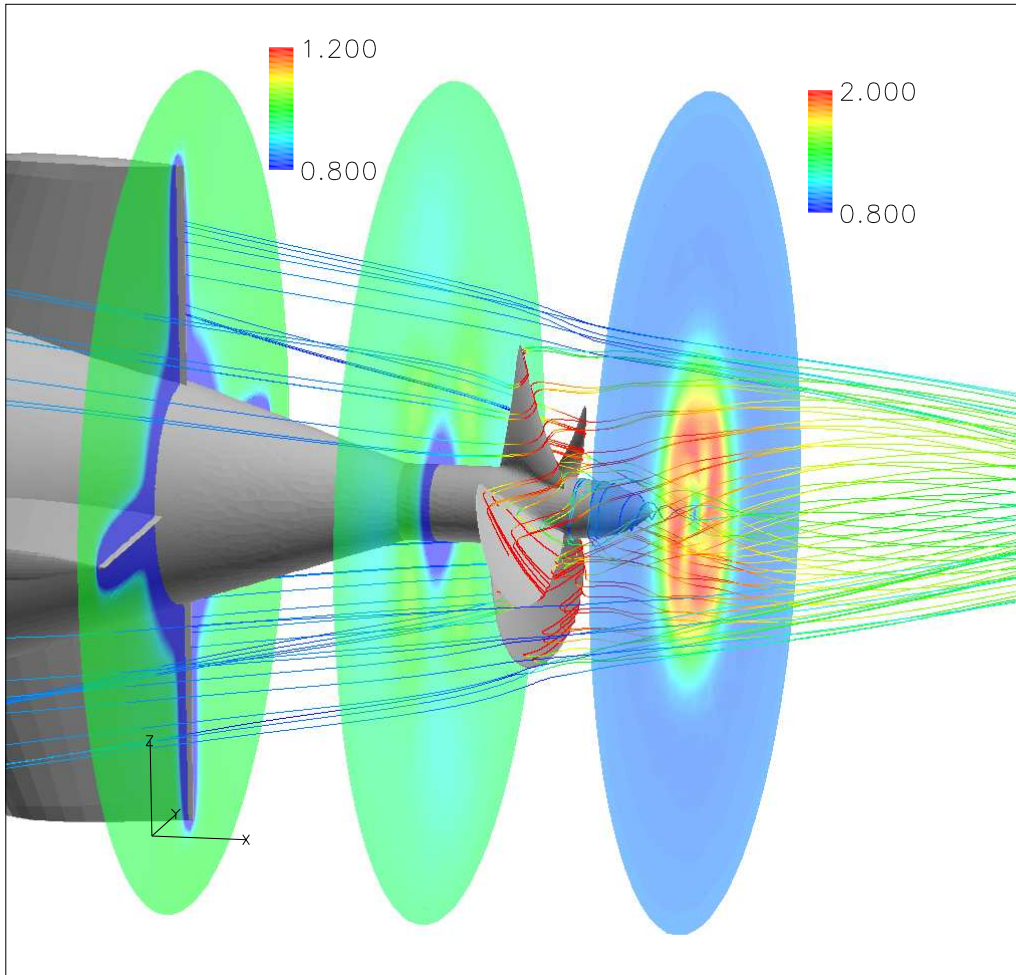


Figure 5: Magnitude of the velocity $|\mathbf{v}|$ after five propeller revolutions. The streamlines are colored by $|\mathbf{v}|$ and on the cross-planes we have color plots of the same magnitude. Different intervals are however used in order to visualize the structures downstream of the propeller where $|\mathbf{v}|$ is significantly larger. For the two upstream cross-planes we use the interval $0.8m/s < |\mathbf{v}| < 1.2m/s$ and for the downstream cross-plane and the coloring of the streamlines we use the interval $0.8m/s < |\mathbf{v}| < 2.0m/s$. Structures generated by the rudders, their interaction with the flow into the propeller region and their downstream evolution can be analyzed in this and similar visualizations.

This type of post-processing can be used to analyze the rudder-propeller interaction and the effect of upstream structures on the loading of the propeller.

5 Concluding remarks

We have described the D&R-algorithm, and a parallel implementation of it using OpenFOAM, for CFD with moving boundaries, executed on Linux clusters. Two methods for parallelization were compared with a focus on the load balancing and on factors limiting the maximum size (in terms of mesh cells) of the problem that can be simulated.

The methods were tested on a model computation of the flow around a submarine with a complete (rotating) propeller model. For this problem it was demonstrated that the proposed implementation scales well and leads to execution times comparable to a corresponding standard fixed grid CFD-simulation with a equally sized grid.

The method has been developed to be used for high Reynolds number applications. For this, it will be necessary to include turbulence modeling in the form of URANS (Unsteady Reynolds Averaged Navier-Stokes equations) or LES (Large Eddy Simulation). In the present paper we use the Navier-Stokes equations without turbulence modeling, since this situation includes all the complications associated with the dynamic mesh which is the focus of the paper, and it is relatively straightforward to include turbulence modeling.

Future developments include the practical demonstration of the good scaling properties for large problems with $O(10^7)$ cells. Furthermore, we currently we have only implemented the parallelization method **P2**¹. This will be extended to the general **P2**-method.

Acknowledgments

The authors wish to acknowledge the financial support from the Defense Science and Technology Agency of Singapore and from the Swedish Armed Forces.

References

- [1] Homepage of Netgen. www.hp fem.jku.at/netgen.
- [2] OpenFOAM 1.3 User Guide, 2006.
- [3] F. J. Blom. Considerations on the spring analogy. *Int. J. Numer. Meth. Fl.*, 32:647–668, 2000.
- [4] F. Di Felice, D. Di Florio, M. Felli, and G. P. Romano. Experimental Investigation of the Propeller Wake at Different Loading Conditions by Particle Image Velocimetry. *J. Ship Research*, 48(2), 2004.
- [5] N. C. Groves, T. T. Huang, and M. S. Chang. Geometric Characteristics of DARPA SUBOFF Models. Technical Report DTRC/SHD-1298-01, David Taylor Research Center, March 1989.
- [6] B. Helenbrook. Mesh deformation using the biharmonic operator. *Int. J. Numer. Meth. Eng.*, 56:1007–1021, 2003.
- [7] Demirdzic I. and Peric M. Finite Volume Method for Prediction of Fluid Flow in Arbitrarily Shaped Domains with Moving Boundaries. *Int. J. Num. Fluids*, 10:771–790, 1990.

- [8] M. Liefvendahl and E. Lillberg. Computational methods for unsteady fluid force predictions using moving mesh large eddy simulation. In *43rd AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA-2005-1090, January 2005.
- [9] M. Liefvendahl and C. Troëng. Deformation and Regeneration of the Computational Grid for CFD with Moving Boundaries. In *45th AIAA Aerospace Sciences Meeting and Exhibit*, number AIAA-2007-1458, January 2007.
- [10] M. Liefvendahl and C. Troëng. Simulation of incompressible flow with large boundary motion by deforming and regenerating the computational mesh. In *OpenFOAM International Conference*, November 2007. Windsor, UK.
- [11] L. Tysell. An Advancing Front Grid Generation System for 3D Unstructured Grids. In *ICAS-94-2.5.1*, 1994.