CRANFIELD UNIVERSITY

Nicholai Chapman

Grid-Based CFD Optimization

SCHOOL OF ENGINEERING

MSc THESIS

CRANFIELD UNIVERSITY
School of Engineering

MSc THESIS
ACADEMIC YEAR 2007-2008

Nicholai Chapman

Grid-Based CFD Optimization

Supervisor: Karl Jenkins

September 2009

This thesis is submitted in partial fulfilment of the requirements
for the degree of Master of Science

Acknowledgements

I am grateful to my supervisor Dr. Karl Jenkins for giving me the possibility to do this project and for his guidance and assistance.

I am very thankful to Mike Riley for his valuable assistance since the beginning of the project and all his suggestions.

I am also very thankful to Naveed Akram for sharing his knowledge on OpenFOAM with me and helping me through the meshing process.

I am thankful to the AMAC department for allowing me to use a computer in the PhD lab during this project.

I am also thankful to Germán Moltó for accepting to be my supervisor in my home university and for his time and suggestions.

Also thanks to Patrick Verdin for his assistance when needed.

I would also like to thank the Nimrod Toolkit development team at Monash University for providing the use of Nimrod.

I want to express my thanks to my friends Hafida Boutahar, Helena Delgado and Iván Rodríguez for their company and support during my thesis which made things much easier.

I am also very thankful to Estela Ajenjo for her faithful support even in the distance.

Finally I am extremely grateful to my mother and grandmother to whom I owe everything and have encouraged me all my life to become what I am now.

Abstract

In this thesis a design optimization tool applied to computational fluid dynamics is presented. The aim is to develop a new optimization method using powerful free software such as Nimrod and OpenFOAM as well as automating the optimization process. The tool is tested on a two dimensional aerofoil were its three parameters are optimized to obtain its maximum lift-to-drag ratio. The optimization is carried out by Nimrod/O which executes a script in charge of integrating the process. Salome will first create the CAD model and mesh it using a Python script and the CFD toolbox OpenFOAM is then used to solve the mesh. Two different optimization algorithms are used and compared and the speed up is evaluated by running the process on several processors.

Contents

Chapter 1: Introduction

A common procedure in engineering when designing a product is to build real prototypes and perform experiments on them. Computational science and engineering provides techniques which bring a different perspective to this way of engineering.

A user can build a computational model which simulates the physical properties of the product to be engineered and perform experiments on it rather than building a prototype. This way several different scenarios can be explored and the user can study many design alternatives reducing the design cycle and even saving money and improving the quality of the product.

However, these computational models are computationally expensive and require the use of plenty of computational power to achieve accurate results. These models must be executed many times during the optimization process and for that reason it's interesting to be able to run the executions concurrently. Large clusters of computers can be used for this purpose or, even better, the Grid.

Grid computing has become a field of increasing research and has expanded the possibilities for computing, allowing users to share resources and therefore having access to more computational power and storage. Therefore grid computing has become a field of interest when dealing with computational intense problems such as design optimization and is already being applied for this purpose.

For all this, it is interesting to create a tool which allows the user not only to automate the optimization process, but to give him at the same time the resources to do it efficiently.

Chapter 2: Literature review

2.1 Aerodynamic concepts

2.1.1 Aerofoil parameters

A cross-section of an aircraft wing is called an aerofoil, as shown in figure 1. Many efforts are put on finding the optimal shape of an aerofoil according to specific design requirements. Three parameters are of special importance in this project: camber, thickness and angle of attack.

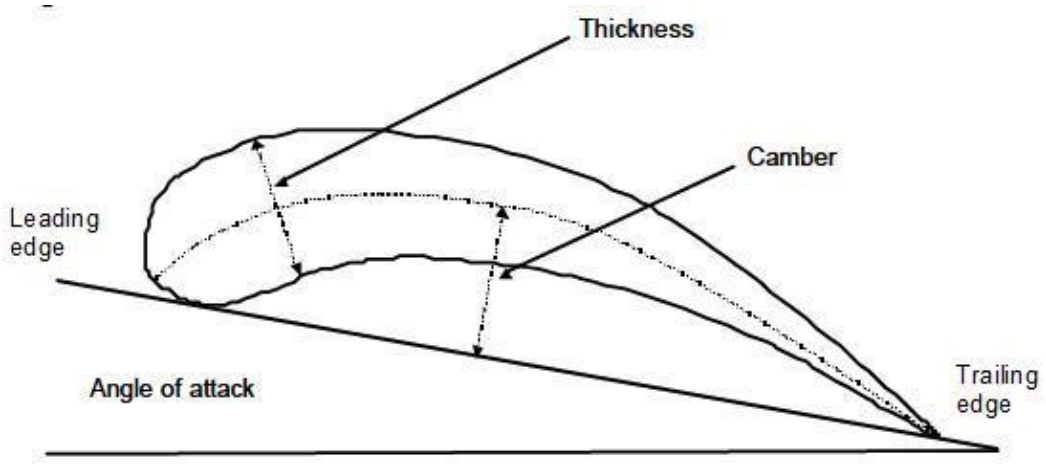

Figure 1: Aerofoil structure [3]

As said in [1] and [2], a straight line drawn from the aerofoil's leading edge to its trailing edge is called the chord line and the length of this line is known as the chord. If a curved line is drawn, equidistant between the upper and lower surfaces of the aerofoil and from one edge to the other, this line is called the mean camber line. The camber of an aerofoil is the maximum distance between its chord line and its mean camber line, measured perpendicular to the chord line. The thickness is the distance between the upper and lower surfaces of the aerofoil, and is also measured perpendicular to the

chord line. The angle of attack is the angle between the chord line and the direction of arrival of the relative wind, defined in [2] as "the flow velocity far ahead of the body".

2.1.2 Forces

According to [2], aerodynamic forces on a body are due to two sources: the pressure distribution over the body surface, which is perpendicular to it, and the shear stress distribution over the body surface, which is parallel to it. The sum of the net effect of these distributions result in an aerodynamic force which can be split into two components. Lift is defined as the component of this force which is perpendicular to the flow direction, while drag is defined as the component of this force which is parallel to the flow direction.

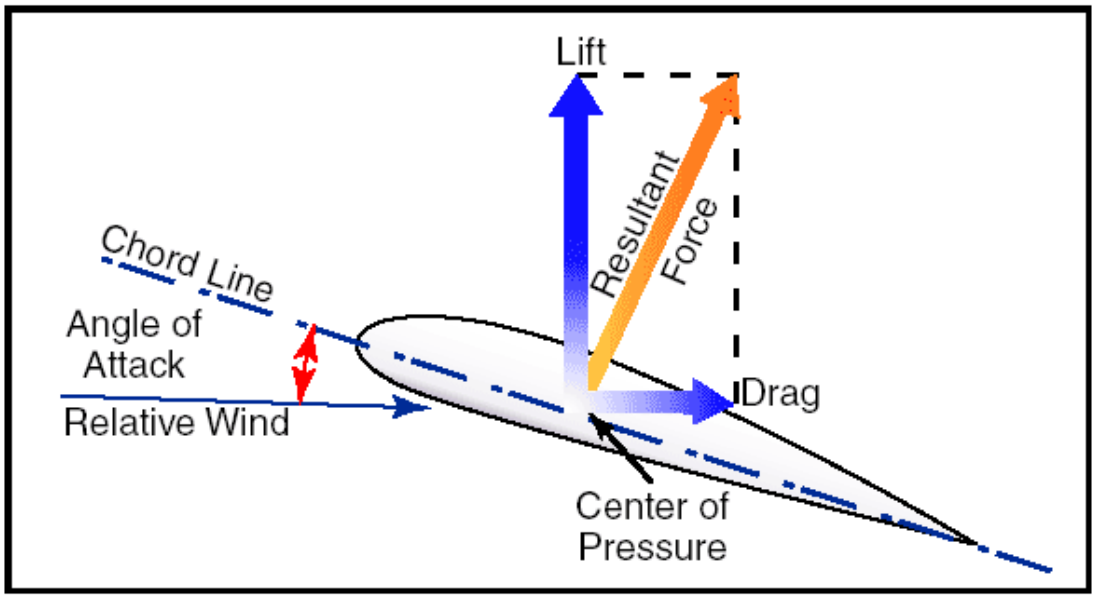

Figure 2: Lift and drag forces [4]

A very important matter in aeroplane design is the lift-to-drag ratio, as it is a measure of the aerodynamic efficiency of the aeroplane. As said in [5], "it only makes sense that maximum aerodynamic efficiency should lead to minimum thrust required". The lift-to-drag

ratio is also said to represent the aeroplane's benefit-to-cost-ratio, as lift represents the economic value of the aeroplane and drag the cost of providing the economic value. [1]

As it can be seen in figures 3 and 4, lift and drag are related to angle of attack. For small values of angle of attack, lift increases linearly while drag does it very gradually. At higher angles of attack lift starts to increase slower and at some point it reaches its maximum value. Drag starts to increase faster with higher angles of attack. When the point of maximum lift is reached, further increases in the angle of attack will result in less lift, a phenomenon which is called stall.

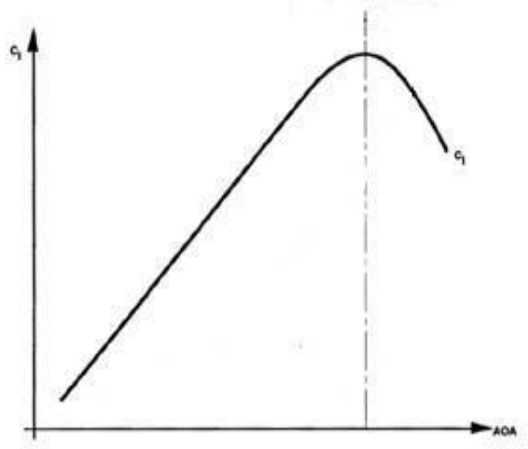

Figure 3: Lift coefficient curve [6]

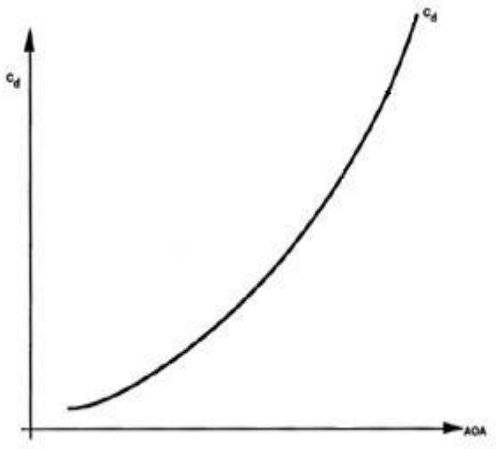

Figure 4: Drag coefficient curve [6]

2.2 Algorithms applied in design optimization

Many are the techniques and algorithms which have been applied in engineering for optimization purposes. In the field of aerodynamics, many efforts have been put in shape optimization of aerofoils.

2.2.1 Gradient-based methods

Gradient-based methods use gradient information and are very efficient when the derivative information of the problem can be easily

obtained. Their main advantage is that they are fast search methods as low number of function evaluations are needed. However, they can't be applied directly to many problems. [7]

Gradient-based methods are very popular and have been widely applied. In [8] a gradient-based method using the steepest descent was implemented. The objective was to reduce the drag coefficient on an aerofoil keeping the lift coefficient within some established ranges. In [9] an adjoint-based method was applied to aerodynamic optimization. According to the author, this method has several advantages relative to other gradient-based methods as "the computational effort required to calculate the gradients is independent of the number of the design variables."

Although these methods can be very efficient, they may not be efficient for design automation. The reason for this is that they don't ensure a global optimum, as they search for a local optimum. Therefore, the optimization process must be started repeatedly from different initial points in order to find a global optimum. [9]

2.2.2 Evolutionary algorithms

Evolutionary algorithms appeared from the interest of imitating nature to develop new algorithms and they are based on the idea of using the theory of evolution as an algorithm [10] [11].

Genetic algorithms are probably the best well known evolutionary algorithms. A population of individuals is maintained where only the fittest individuals survive. The individuals evolve due to mutation and crossover operations, finally converging to the best individual [10].
Contrary to gradient-based methods, genetic algorithms search the

design space from multiple points and work on function evaluations alone, which makes them robust and suitable to parallel computing [9].

Genetic algorithms have been widely applied to aerodynamic optimization [9] [12] [13]. However, one of their weaknesses is that they require many function evaluations which lead to poor computational efficiency. Due to this, they have been coupled with other optimization techniques in order to preserve their benefits but improving the computational efficiency, as can be seen in [14] where a genetic algorithm was used with a gradient-based method.

2.3 Grid computing in design optimization

Many engineering optimization design problems, as shape optimization of aerofoils, require great amounts of computational power. CFD simulations, for instance, must be called repetitively while searching through the design space. These simulations, which take most of the optimization process´ time, are computationally expensive and can require hours of computation.

Grid computing is a type of distributed computing which focuses on large-scale resource sharing. Users can share resources such as computational power which allow to overcome problems that were infeasible with previous technologies. [15]

Therefore grid computing is being taken into account as an interesting solution when dealing with computational intense problems [16] [17] [18] [19] [20]. In [18], for example, grid computing was used to overcome the computational cost of using genetic algorithms by means of parallelization and distributed

executions while in [16] and [19] grid computing was applied to computational fluid dynamics optimizations which require high computational power.

2.4 Nimrod as an automatic design optimization tool

Nimrod supports distributed executions and optimizations in one tool. Nimrod has been successfully applied in several studies. In [21] Nimrod was applied to a number of parameterized computational experiments where Nimrod efficiently distributed the work and therefore reduced the execution time of the experiments.

More specifically, Nimrod/O was applied in [22] to a mechanical design problem where it was able to find multiple local optima by performing several searches in parallel, starting from different initial points. In [23] Nimrod/O was tested in a number of different case studies using the P-BFGS algorithm, showing that speedup can be achieved for some problems.

Nimrod has also been used in the design of aerofoils. In [3] Nimrod/O was used to optimize a two dimensional aerofoil which was governed by three parameters where the goal was to maximize the lift to drag ratio. The aerofoil was meshed with Gambit and solved with Fluent, and two different strategies where applied to the optimization: simplex and P-BFGS. According to the authors, the experiments were a success as the results were better than those previously obtained without the use of Nimrod/O. Moreover, Nimrod/O gave the possibility to test the experiment with two different algorithms by just changing the plan file.

Chapter 3: Software Description

3.1 Nimrod

Nimrod is a specialized parametric modeling system developed by Monash University. It provides a simple declarative language which allows the user to create parametric experiments which can be executed across distributed computers. Nimrod manages the whole experiment and provides tools which automate the process, distributing the necessary files to run the experiment, running it, monitoring it and finally gathering the results.

3.1.1 Nimrod/O

Nimrod/O "allows a user to run an arbitrary computational model as the core of a non-linear optimization process" [24]. It accepts a declarative plan file in which the user can specify the domain and type of the parameters, any constraints imposed on the solution, the tasks to perform the experiment, the output variable to be optimized and the optimization method to use out of the four built-in optimization algorithms which it currently employs: Simplex, BFGS, Divide and Conquer and Simulated Annealing.

Nimrod/O uses Nimrod in order to perform distributed executions. The jobs are passed to Nimrod/G or EnFuzion which allow them to be run in the Grid or in a cluster of computers. A cache is placed between Nimrod/O and Nimrod which reduces the number of calculations if it receives a set of parameters which have already been calculated. A persistent database is attached to this cache in case the Nimrod/O is terminated prematurely.

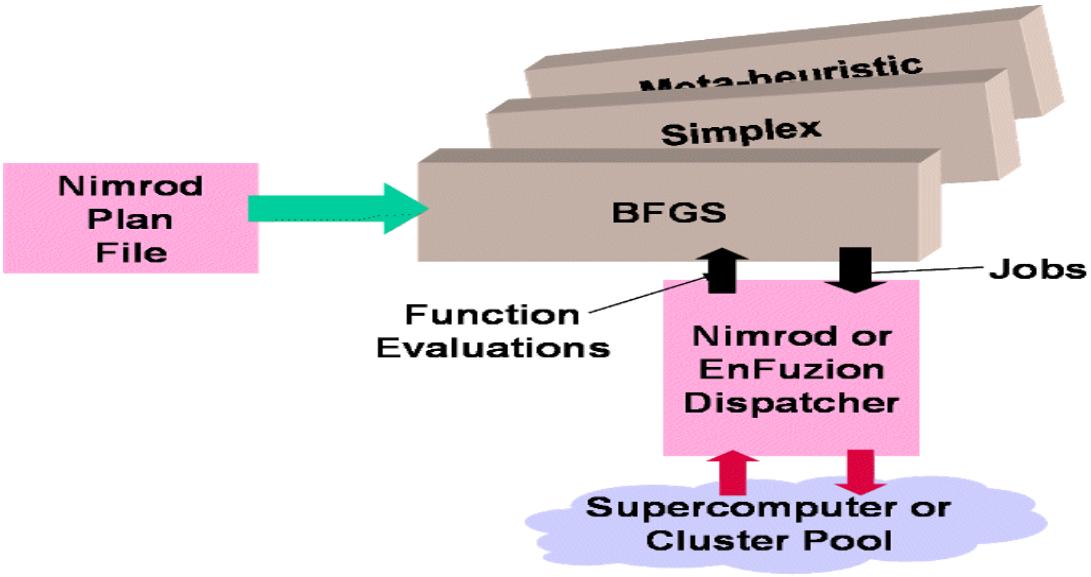

Figure 5: Nimrod/O architecture [25]

3.1.2 Algorithms

Two of the algorithms supplied in Nimrod have been used in this project.

3.1.2.1 Simplex

The algorithm is based on the method of Nelder and Mead. A simplex is "a geometrical figure consisting, in N dimensions, of N+1 points, or vertices, and all their interconnecting line segments." [24] For example, in two dimensions a simplex is a triangle and in three dimensions it's a tetrahedron.

The algorithm is started with N+1 points and their associated cost function evaluations. It then iterates replacing at each step the worst vertex by a better one until the cost function values at all points fall within a desired tolerance of each other. For example, if the simplex is a triangle as shown in figure 6, and the worst point is w, n is the next worst and b is the best, a straight line would be drawn between w and c, being c the middle point between n and b. On this line we

consider four new points which are candidates to replace w in the simplex:

- p: the reflection of w in c
- q: the midpoint of p and c
- r: the midpoint of w and c
- s: a point past p and equally far from p as p is from c

[26]

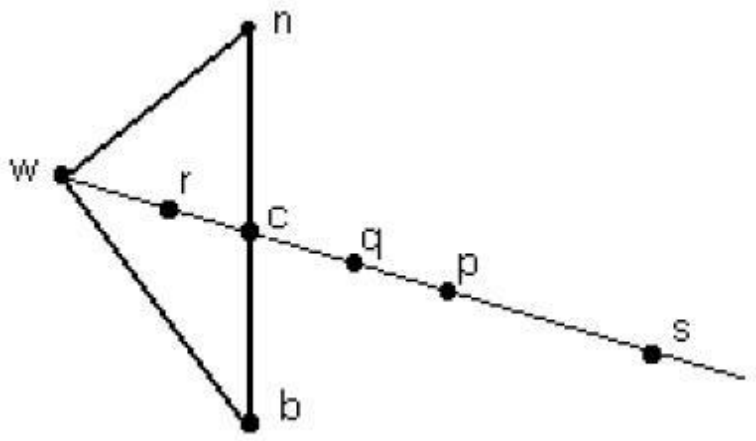

Figure 6: Simplex example [26]

3.1.2.2 BFGS

This algorithm is based on a quasi-Newton method. The BFGS method maintains an approximation to the inverse of the Hessian, H, of f, where:

$H_{ij} = \delta^2 f / \delta x_i \delta x_j$

New approximations to the solution vector x are derived by:

- Computing a search direction, $d = -H^{-1} \nabla f(x)$

- Finding a new solution vector x using a line search, i.e. $x_+ = x + \lambda d$
- Updating the inverse of H using the current approximation to H, x and x_+

[24]

3.2 OpenFOAM

OpenFOAM is a free and open source CFD toolbox developed by OpenCFD Ltd. It is basically a set of C++ libraries which are used to create applications. These applications are solvers, which are designed to solve specific problems in engineering mechanics, and utilities, which are designed to perform pre- and post-processing tasks involving data manipulation, such as mesh generation or data visualisation. [27] [28]

OpenFOAM contains numerous solvers, utilities and libraries which cover a wide range of problems. However, one of the main advantages of using OpenFOAM is that the user can create new solvers and utilities as it is open. [27]

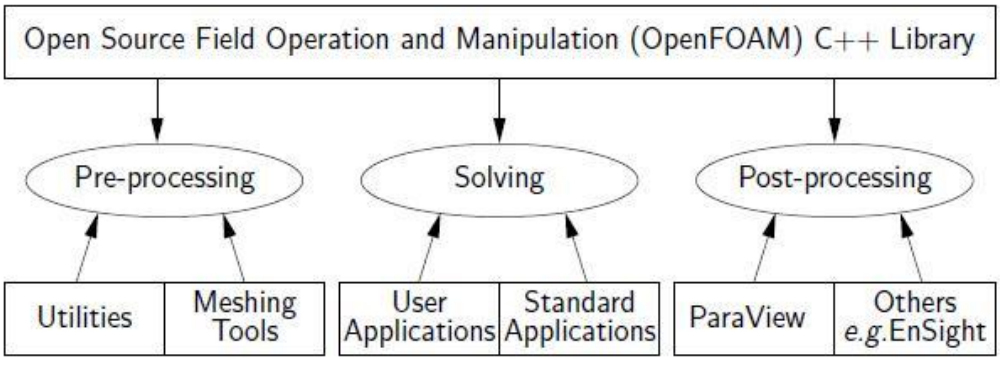

Figure 7: Overview of OpenFOAM structure [27]

3.3 Salome

Salome is a free and open source platform for numerical simulation. It provides a number of modules which allow the user to work on CAD models and mesh them. It also provides a user interface and a Python console through which all the functionalities of Salome are accessible. [29]

Among its several modules two of them are used in this project: the GEOM and the SMESH modules. The GEOM module allows the user to create or modify a CAD model as well as importing or exporting the models in several formats. With the SMESH module the user can import or export a mesh from and into several formats or create its own mesh with a number of different algorithms and hypothesis which it supplies according to the dimensions of the model.

3.3.1 Dump Study

Salome can be launched in batch mode and operated with the use of Python scripts. It comes with an option called Dump study which automatically generates a set of Python scripts from data created with Salome GUI. It is highly useful as there is no need for the user to script himself. The scripts can be stored and loaded later to re-create the content of the original study. [29]

3.4 CommandLineToIGES

In this project a command-line aerofoil generator which was previously created for another project has been used. It depends on Open Cascade libraries and this version is Linux based.

This code uses the NACA 4-digit standard to generate the vertices before interpolating these to create the aerofoil's outline. The program receives as an input the Angle of Attack (degrees), which rotates around the leading edge, Thickness [0-100]% and Camber [0-100]%. It is also possible to specify the chord, which is set to 0.5 meters as default. Other options are implemented such as specifying the output path or the output filename of the aerofoil. The use is as follows:

CommandLineToIges --AoA 1 --camber 8 --thickness 8 --chord 1.25 --outputPath /root/my_foils --outputFilename sample
[30]

Chapter 4: Nimrod Installation and Test

4.1 Installation

All the steps are in the Nimrod/O Users' Guide.

Once Nimrod/O has been downloaded, the archive must be expanded using the following command:

tar xvf nimrodo.<version>.tar

This creates the following directory structure:

```
nimrodo.<ver> ----- source ---------- nimrodo
                                          |
                                          |--- nimcache
                                          |
                                          |--- nimdisp
                                          |
                                          |--- beader
```

Then, to configure and install Nimrod/O for a Unix system:

1. Go to the *source* subdirectory. Enter

   ./configure --prefix=<installation_dir>

   Here *<installation_dir>* is the directory where Nimrod/O will be installed. This directory must already exist.

2. Enter

   make install

This process should create the following directory structure

<installation_dir> ---------- bin
|
|-- lib --- nimrodo --- test

To test that Nimrod/O is working (independently of enFuzion and Nimrod/G):

1. Ensure that <installation_dir>/bin is in your path.

2. Move to the directory <installation_dir>/lib/nimrodo/test (or copy the contents of that directory into your current directory).

3. Enter the command

   nimrodo -t 2 -d s

[26]

As the experiments were going to be run locally, there was no need to install Nimrod/G or enFuzion.

4.2 Test

In order to get familiarized with Nimrod/O, it was first tested with a simple mathematical function. A simple C++ code was first developed which, given parameters x and y, gives as an output the result of the objective function for those two parameters. As Nimrod/O looks for the minimum as default and the aim was to maximize the objective function, the result had to be multiplied by -1. By doing this Nimrod/O would search for the biggest negative answer, which will be

the maximum.

Once the C++ code was working, the following schedule file was made:

example1.shd

```
parameter x integer range from 0 to 25
parameter y integer range from 0 to 25

constraint 2*x+y <= 18
constraint 2*x+3*y <= 42
constraint 3*x+y <= 24
constraint x >= 0
constraint y >= 0

task main
        copy nimrodexample1 node:nimrodexample1
        node:execute ./nimrodexample1 $x $y > result
        copy node:result output.$jobname
endtask

method simplex
        starts 1
                tolerance 0.000
        endstarts
end method
```

The first part of this schedule file lists the parameters which constitute the design space. For each parameter its data type and domain are specified.

The second part defines the constraints. The constraints can be hard, in which case they can't be violated, or soft, which can be violated but will then give a penalty value to the objective function which is proportional to the dimension of the violation. In this case the constraints are hard, which is the default option. The syntax for a constraint is:

constraint [hard|soft] <constraint_condition>

The third part specifies the tasks which must be carried out by the dispatcher to compute the objective function. Here the C++ program will be copied from the root to a node which will then execute it and copy back the result to the root. However, when working locally there is no need to move files between nodes. When the file is copied back to the root it is given a unique name as determined by the dispatcher jobname. Nimrod/O requires that the objective value is the first string in the file output.$jobname.

The last part specifies the optimization method to be used. In this case a simplex search will be started from one random point with the specified tolerance. The initial points can also be selected manually and more than one optimization method can be specified. [26]

This schedule file was first run locally, with the following results:

```
-------------- Final Results:  Block '1 (unnamed)' ------------
Starting point Optimum At Point Iters. Evals.  Batches
(2,8)             -32  (4,10)    7      8 (21)  5 (8)
```

As the optimum is 33 with (x,y) = (3,12) so the result wasn't the expected one, Nimrod/O was ran again changing the starts of the simplex method to 5 and 10, finally obtaining the optimum values after 10 starts.

```
-------------- Final Results:  Block '1 (unnamed)' -----------
Starting point Optimum At Point Iters. Evals.   Batches
(2,8)             -32  (4,10)    7      8 (21)   5 (8)
(5,4)             -31  (5,8)     9      13 (30)  6 (10)
(5,1)             -30  (6,6)     9      15 (32)  7 (11)
(0,2)             -32  (4,10)    14     19 (35)  8 (15)
(4,3)             -28  (6,5)     7      8 (21)   5 (8)
```

```
-------------- Final Results:  Block '1 (unnamed)' -----------
Starting point Optimum At Point Iters. Evals.    Batches
(2,8)             -32  (4,10)    7      8 (21)    5 (8)
(5,4)             -31  (5,8)     9      13 (30)   6 (10)
(5,1)             -30  (6,6)     9      15 (32)   7 (11)
(0,2)             -32  (4,10)    14     19 (35)   8 (15)
(4,3)             -28  (6,5)     7      8 (21)    5 (8)
(3,8)             -33  (3,12)    9      11 (28)   5 (10)
(0,9)             -33  (3,12)    57     11 (370)  6 (107)
(3,12)            -33  (3,12)    5      2 (9)     2 (5)
(0,13)            -26  (0,13)    6      6 (23)    4 (7)
(1,1)             -27  (7,3)     12     14 (40)   7 (14)
```

Chapter 5: Process Description

5.1 Overview

In figure 8 we can see a structure of the workflow.

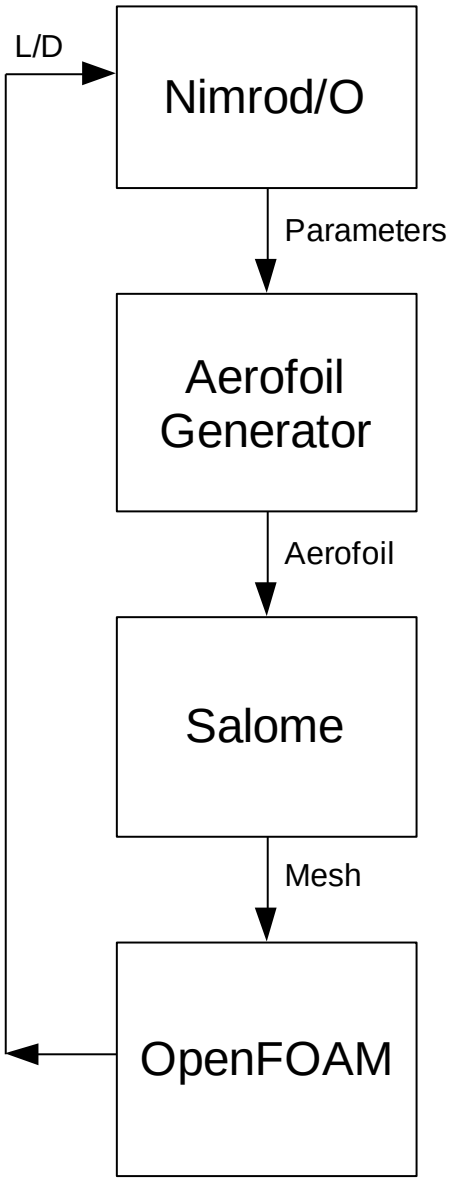

Figure 8: Workflow

Nimrod/O governs the optimization generating a new set of parameters for each evaluation. CommandLineToIGES takes these new set of parameters as an input and generates the aerofoil file.

This file is imported into Salome, which creates the final CAD model and meshes it, finally exporting the mesh to the OpenFOAM directory which will solve it and output the lift and drag coefficients which are used to calculate the lift-to-drag ratio and give the value to Nimrod/O. All this process is run on a AMD Phenom(tm) 9850 Quad-Core Processor with a 2’5MHz frequency.

5.2 Constructing the CAD model

The aerofoil file created with CommandLineToIGES must be first imported into Salome’s geometry module:

File -> Import (IGES files)

Then a face is created using the imported aerofoil which will be used to work with:

New Entity -> Build -> Face

A quadrangle must be built around the aerofoil. An appropriate size for this quadrangle would be of five chord length distance for inlet, bottom and top walls and 10 chord length distance for the outlet wall. However, this would lead to a mesh with a huge amount of elements, which would take a very long time to solve in OpenFOAM. Due to time restrictions, the accuracy of the results is not an aim of this project so the quadrangle was reduced to one chord length for inlet, top and bottom and two for outlet, which will lead to a much smaller mesh.

To create the quadrangle the following steps must be followed:

1. Four points must be created around the aerofoil:

New Entity -> Basic -> Point

In this case the points were created in (-500, 500, 0), (-500, -500, 0), (1500, -500, 0) and (1500, 500, 0)[1]

2. A quadrangle face is created using the previous points:

New Entity -> Blocks -> Quadrangle Face

At this point some problems which must be explained arose. OpenFOAM works with 3D meshes but in this project a 2D aerofoil was going to be solved. The procedure to solve a 2D case in OpenFOAM is to create a 3D mesh which is just one cell thick in the third dimension.

The first approach to do this was to mesh the 2D geometry and once done that extrude it:

Modification -> Extrusion

selecting the whole mesh, using a vector on the z direction and selecting number of steps = 1. By doing so, the expected mesh was achieved but the problem was that groups of faces for the boundary conditions could not be created. This was due to the fact that the original geometry was 2D and therefore wasn't composed by faces.

A different approach was then tried out which consisted in creating an extrusion mesh. The first step to do this is to convert the geometry into 3D. Both of the previously created faces must be extruded. To do so a vector is created:

[1] A chord length is 0,5 m which is represented as 500 in Salome.

New Entity -> Basic -> Vector

in the extrusion direction (z in this case). Then both faces can be extruded:

New Entity -> Generation -> Extrusion

selecting the face to extrude as the base shape and the previously created vector.

At this point the aerofoil is in 3D as well as the quadrangle around it but one final operation must be made to obtain the final geometry as the aim is to mesh the surface around the aerofoil:

Operations -> Boolean -> Cut

selecting the extruded quadrangle as the main object and the extruded aerofoil as the tool object.

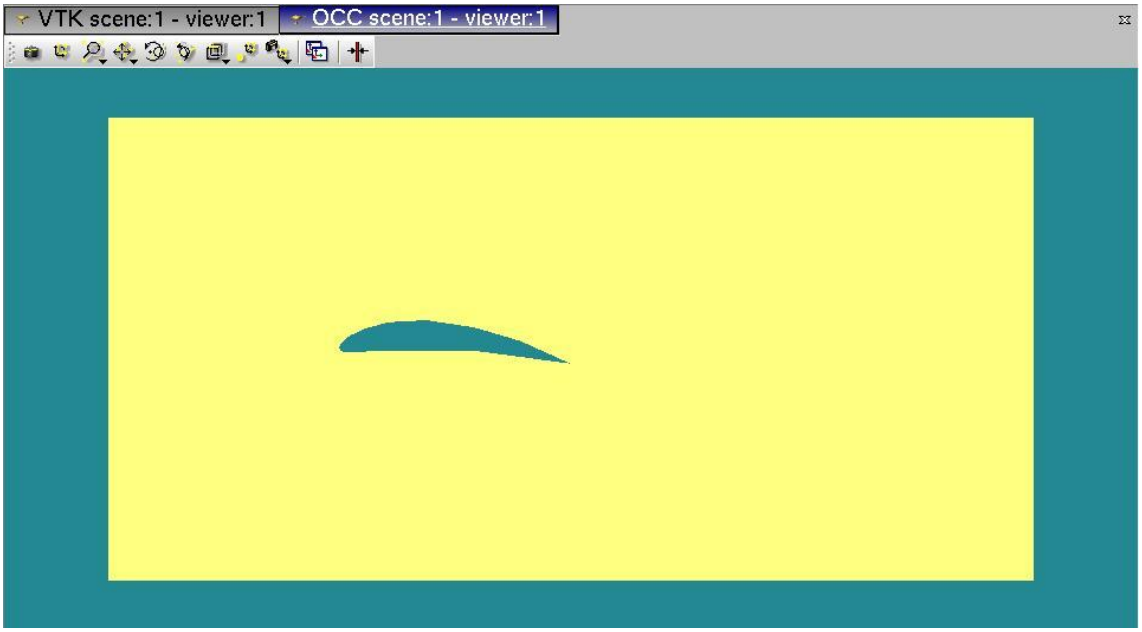

Figure 9: Cut1

After this operation a volume is obtained with an empty aerofoil shape in the middle, as can be seen in figure 9, and the geometry is now ready to be meshed. This final geometry will be referred from now on as *Cut1.*

Although the geometry is ready to be meshed, some further operations must be done which will be used during the mesh generation. First a group must be created which is composed by the edges of Cut1 in the extrusion direction. This will be used to create a mesh which is one cell thick. To do so, right click on Cut1 and select:

Create Group

selecting edges as *elements type* and adding the correct edges to the group.

In order to be able to select these edges, Cut1 must be exploded first:

New Entity -> Explode

selecting Cut1 as the main object and "edges" in Sub Shapes Type. Also, the same operation must be done for "faces", as it will be used later.

5.3 Generating the mesh

The mesh is created changing to the mesh module, selecting:

Mesh -> Create Mesh

on Cut1 and applying the following algorithms:

- 3D Algorithm: 3D extrusion (doesn't require hypothesis)
- 2D Algorithm: Quadrangle (doesn't require hypothesis)
- 1D Algorithm: Wire discretisation
- 1D Hypothesis: Average Length = 15

If the mesh is computed at this point it would give an error. Some more steps are required. As a mesh which is only one cell thick is needed, a submesh must be created. The group formed by the edges of Cut1 in the extrusion direction, which has been created previously in the geometry module, is used to create a submesh:

Mesh -> Create Sub-mesh

The following algorithm and hypothesis must be applied:

- 1D algorithm: Wire discretisation
- 1D hypothesis: Nb. Segments = 1

One more step must be done before computing the mesh. Two more submeshes must be created selecting as the geometry first the face of Cut1 which represents the *initial face* (the face representing the quadrangle face and the aerofoil before the extrusion) and second the face of Cut1 which represents the equivalent to the initial face after the extrusion or *destination face*. These faces are obtained with the explode operation which has been done previously in the geometry module. The first submesh is created selecting the initial face and applying the following algorithm and hypothesis:

- 2D Algorithm: Triangle (Mefisto)
- 2D Hypothesis: Length From Edges

The second submesh is created selecting the destination face and applying:

- 2D Algorithm: Projection 2D
- 1D Hypothesis: Source Face

selecting the initial face as "Face" for the hypothesis.

Now the mesh is ready to be computed by doing right click on the mesh and selecting:

Compute

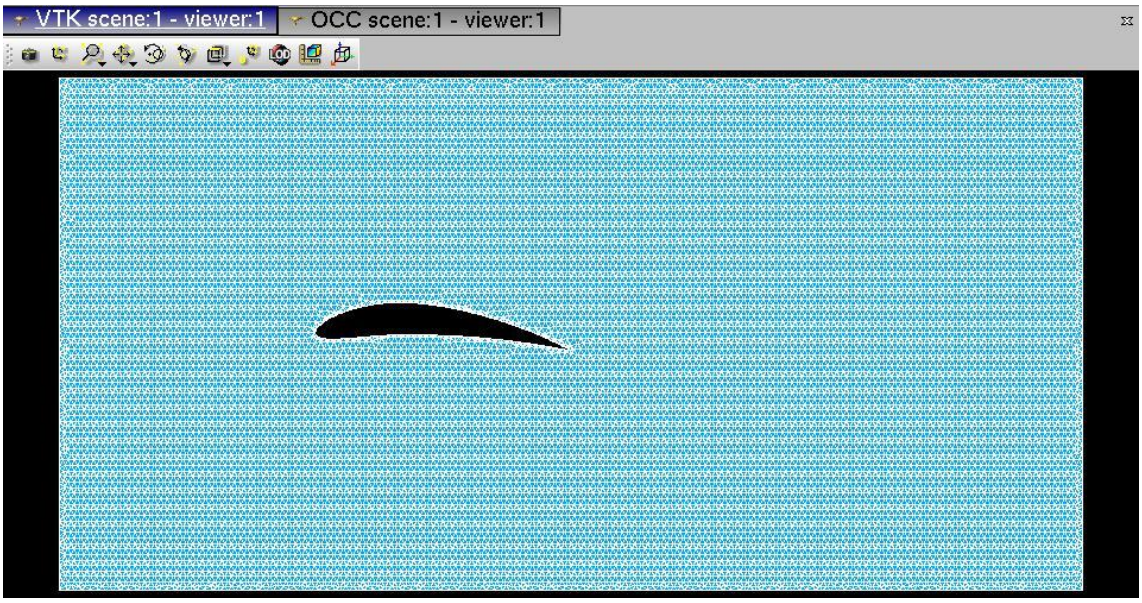

Figure 10: Mesh

Although this mesh is ready to be solved in OpenFOAM, a further refinement can be made to it. The highest point of interest are the elements around the aerofoil so we can refine the mesh to obtain more elements around it. To do so, some steps should be made in the geometry module before generating the mesh. The initial face and the destination face of Cut1 must be exploded into edges. Then a

submesh can be created to refine the elements around the aerofoil by selecting first as the geometry the edge which represents the aerofoil in the initial face and applying:

- 1D Algorithm: Wire discretisation
- 1D Hypothesis: Average length = 3

and a second submesh selecting as the geometry the edge which represents the aerofoil in the destination face and applying:

- 1D Algorithm: Projection 1D
- 1D Hypothesis: Source Edge

The mesh can be then computed and smaller elements around the aerofoil will be obtained, as seen in figure 11. This mesh, which will be used for the optimizations, will have approximately 40000 elements.

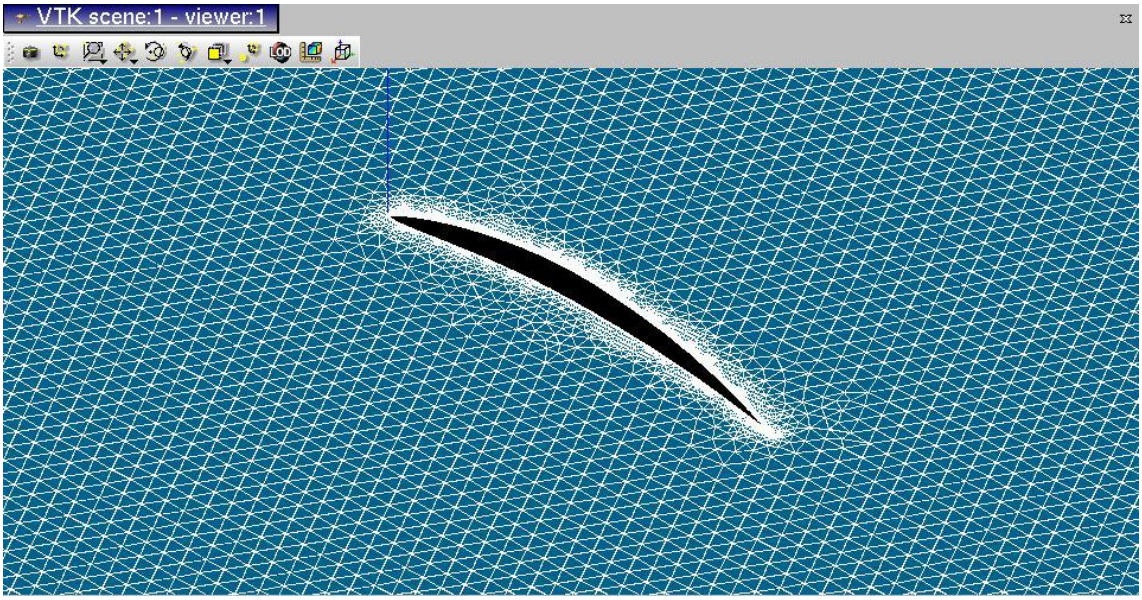

Figure 11

The value for the average length depends on how small the elements are wanted. At the beginning, an average length = 1 was used. The

aim was to refine these elements as much as possible as it was the point of maximum interest. The first tests were running as expected but at some point some problems started to arise as sometimes Salome was being unable to compute the mesh, which wasn't allowing the optimization process to continue. The problem was coming from this step as the elements around the aerofoil were being refined too much. Changing this value to a higher one was solving the problem and Salome was again being able to compute the mesh. The value was changed to three and Salome started working without problems.

Finally, some groups of faces for the boundary conditions in OpenFOAM must be created by using the previously exploded faces of Cut1. Doing right click on the mesh, selecting *Create Group* and following these steps:

- Element Type = Face
- Group Type = Group on geometry
- Geometrical object = face which represents the aerofoil

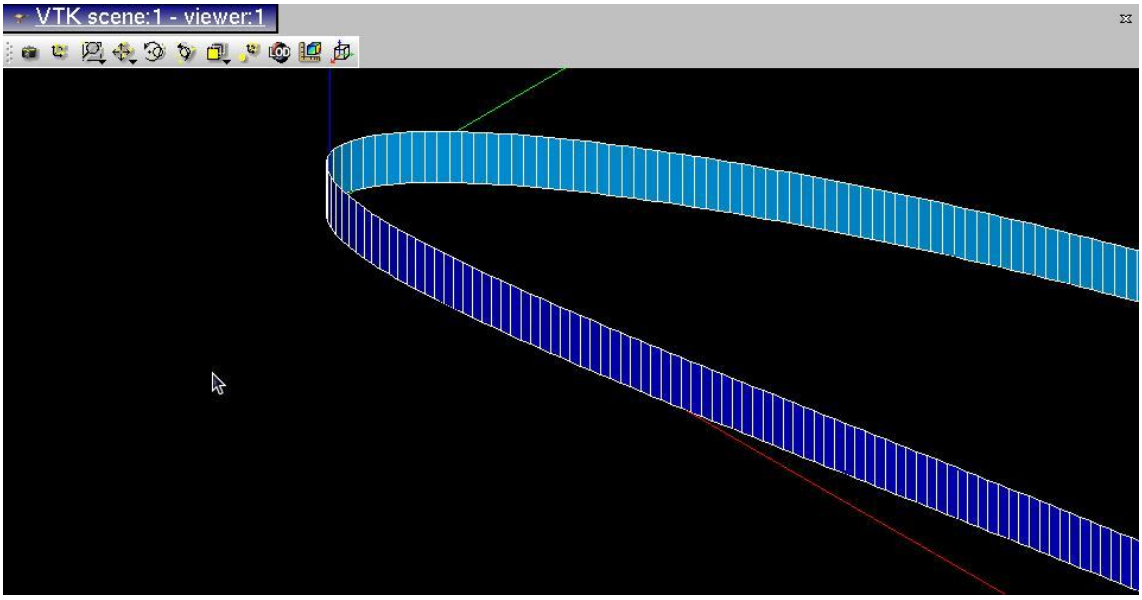

Figure 12: Aerofoil faces

In this case the faces of the aerofoil are selected, as shown in figure 12. These steps must be repeated for inlet, outlet, top, bottom, right (initial face) and left (destination face).

Now the mesh is completed and ready to be exported to the OpenFOAM directory:

File -> Export -> UNV file

5.4 Dump Study and Batch Mode

Once the mesh is generated, the process has to be automated using a script. Salome's option *Dump study* (File -> Dump Study) is used to generate automatically a python script of the session, which will be run in batch mode. Salome is run in batch mode using the option –t and to load a script the option –u is used:

./runSalome -t -u myscript.py

The option -k is also used to kill all the Salome running sessions and & to run it in background:

./runSalome -t -u myscript.py -k &

The reason to run Salome in background is that when it is run in batch mode the prompt of the terminal changes, which doesn't allow the following instructions in the script to be executed. To overcome this problem it is run in background. However, the rest of the instructions must not be executed until Salome has finished generating the mesh. Therefore, a simple C++ program was coded which waits for Salome to create the file with the mesh before the

rest of instructions can be executed (see *wait_for_unv.cc*)

One final change must be done before the Python script is ready. Salome has an option in its Geometry Module named *"Basic properties"* which gives the length, surface and volume of an object. The value of the surface of the walls around the aerofoil is needed to change a file later in OpenFOAM. Salome provides a command to obtain these values:

geompy.BasicProperties(Shape)

This command is used to add to the script the necessary lines to obtain the surface and write the value into a file.

```
area = geompy.BasicProperties(Face_3)
f = open('/home/nicky/Simulation/workfile','w')
s = str(area[1])
f.write(s)
f.close()
```

5.5 Solving in OpenFOAM

The Python script outputs the mesh into .unv format. The first step is to convert this mesh to OpenFOAM:

ideasUnvToFoam mymesh.unv

One important observation must be now made. Salome and OpenFOAM don't work with the same units. While OpenFOAM is working in meters, Salome does it in millimeters. Therefore the points must be converted before starting to work with the mesh:

transformPoints -scale "(0.001 0.001 0.001)"

Some changes must be made to some files before solving the case. When the mesh is converted to OpenFOAM, by default the boundaries in */constant/polyMesh/boundary* receive type *patch*. A C++ program was coded (see *edit_boundary.cc*) to change some of these boundaries. The aerofoil, top and bottom boundaries must be of type *wall*, while right and left boundaries must be of type *empty*.

Also, the value *Aref* in the */system/controlDict* file must be updated using the surface value which the Python script writes into a file. This is also done with a C++ program (see *edit_controldict.cc*).

These changes must be done in each evaluation during the optimization process. Some other changes are done just once before starting the optimization. The number of iterations is set to 3000 in the */system/controlDict* file (endTime field) as it was proved during the tests to be enough to converge. Also, the tolerance for the solution is set to 1e-04 in the */system/fvSolution* file. Finally, in the */system/controlDict* file must be specified that the forces must be calculated (see *controlDict*) as the aim is to obtain the lift and drag coefficients.

The case must now be prepared to run in parallel. The method of parallel computing used by OpenFOAM is known as domain decomposition, in which the geometry and associated fields are broken into pieces and allocated to separate processors for solution. To run a parallel case the first step required is therefore to decompose the domain using the *decomposePar* utility. There is a dictionary associated with decomposePar named *decomposeParDict*. This dictionary must be copied into the system folder of the case and

edited depending on the number of subdomains into which the case will be decomposed, four in this case, which is done by editing the number in the entry *"numberOfSubdomains"* [27]. The method used to distribute the work among the processors can also be selected. In this case *metis* has been used, distributing the work equally among all the processors.

The decomposePar utility is executed by typing:

decomposePar

and now the case is ready to run in parallel, which is done with the following command:

mpirun -np 4 simpleFoam -parallel > log

were the number of processors (4) are specified, the solver used (simpleFoam) and the output is sent to a log file.

Once the case has completed running, the decomposed fields and mesh must be reassembled for post-processing using the *reconstructPar* utility. [4] This is executed by typing:

reconstructPar

Finally, the lift and drag coefficients are written into the forceCoeffs/1/forceCoeffs.dat file. A C++ program (see *lift_drag.cc*) will extract these coefficients and return the lift-to-drag ratio which Nimrod/O will copy as a result.

5.6 Running Nimrod/O

Nimrod/O is run using the following command:

nimrodo –f schedule.shd -l

were –l is used to run the optimization locally and schedule.shd is the schedule file used.

In the schedule file the angle of attack, camber and thickness are specified. As the camber range is set from 0-12, a constraint is set so that the camber must be higher than zero.

The first task of the schedule file is to execute the script which automates all the process of creating the CAD, the mesh and solving it. This script accepts the angle of attack, camber and thickness as an input. Once the script has finished its job, the lift_drag program is executed to obtain the lift-to-drag ratio. This value is copied into a file named *result*, which is copied into a final filename which identifies it.

Finally the optimization method is specified, with the number of starts and the tolerance which has been set to 0.01.

A file called *nimdisp.options* must also be created in the working directory. This file will contain the line:

concurrency 1

which will set the number of concurrent jobs to one.

Chapter 6: Experimentation

6.1 Camber and thickness optimization

The aim of the first experiment was to test the process and make sure that the optimization process was working correctly. Therefore only two parameters, camber and thickness, were optimized, maintaining a constant angle of attack of 0 degrees. The inlet velocity is set to 73.45 m/s. The results are shown in table 1 and the progress of the optimization is shown in figure 13.

Starting point	L/D	Camber	Thickness	Eval	Time
(10.9626, 10.3086)	38.6014	4.54541	1.5	59	5:40:06

Table 1

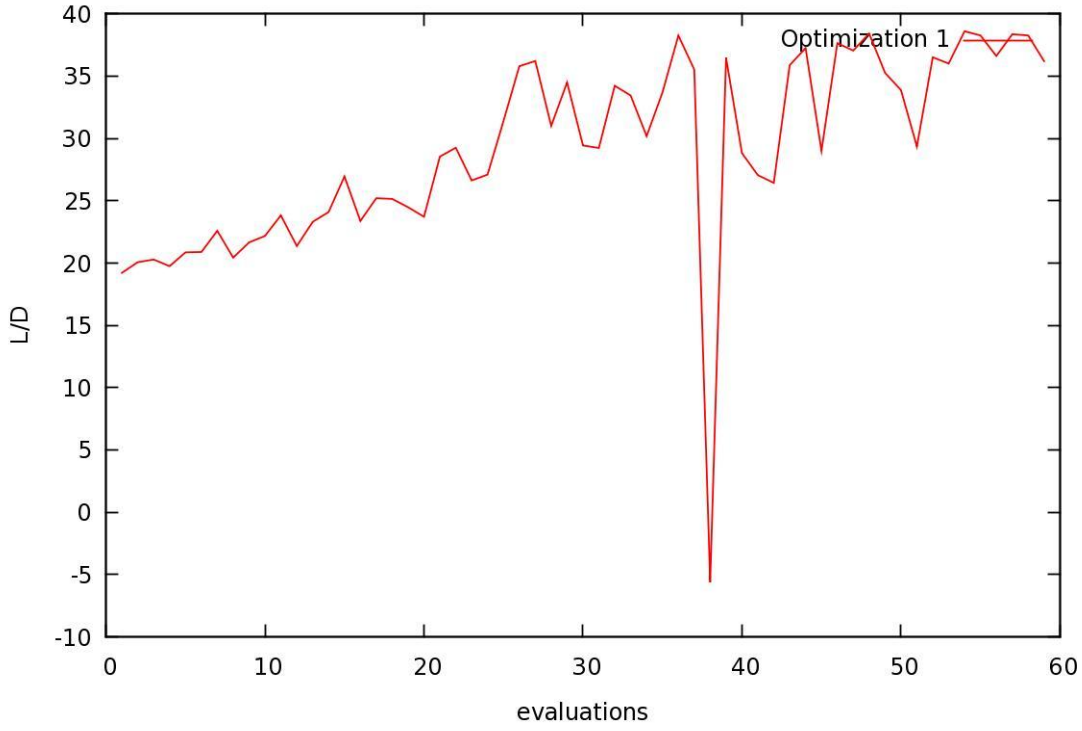

Figure 13

The starting point was decided randomly by Nimrod/O. The L/D value was acceptable and the process ended without problems.

In figure 13 a negative value for L/D can be seen. This is due to a failed evaluation which crashed, returning the L/D value according to the coefficient at the moment of the crash. This can happen sometimes when working in parallel.

6.2 Angle of attack optimization

In the next experiment the angle of attack had to be included to the optimization. However, before doing this, another experiment was made to check that the angle of attack optimization was working correctly. To do this the optimum values of camber and thickness in the previous experiment were used, keeping these values constant and optimizing only the angle of attack. As said in the literature review, as the angle of attack increases the L/D increases too until the stall angle is reached. Therefore, a better L/D value than in the previous experiment should be obtained after this optimization as the angle of attack was zero before. The results are shown in table 2.

Starting point	L/D	AoA	Camber & Thickness	Eval	Time
3.65419	43.7026	0.85419	(4.54541, 1.5)	17	1:47:49

Table 2

As expected, a higher value for L/D was obtained, being the optimal angle of attack of 0.85419. In figure 14 can be seen how the L/D increases with the angle of attack until certain value where it begins to decrease again. Figure 15 shows the progression of the

optimization, where it can be seen how the values increase progressively as the algorithm finds better values of the parameters.

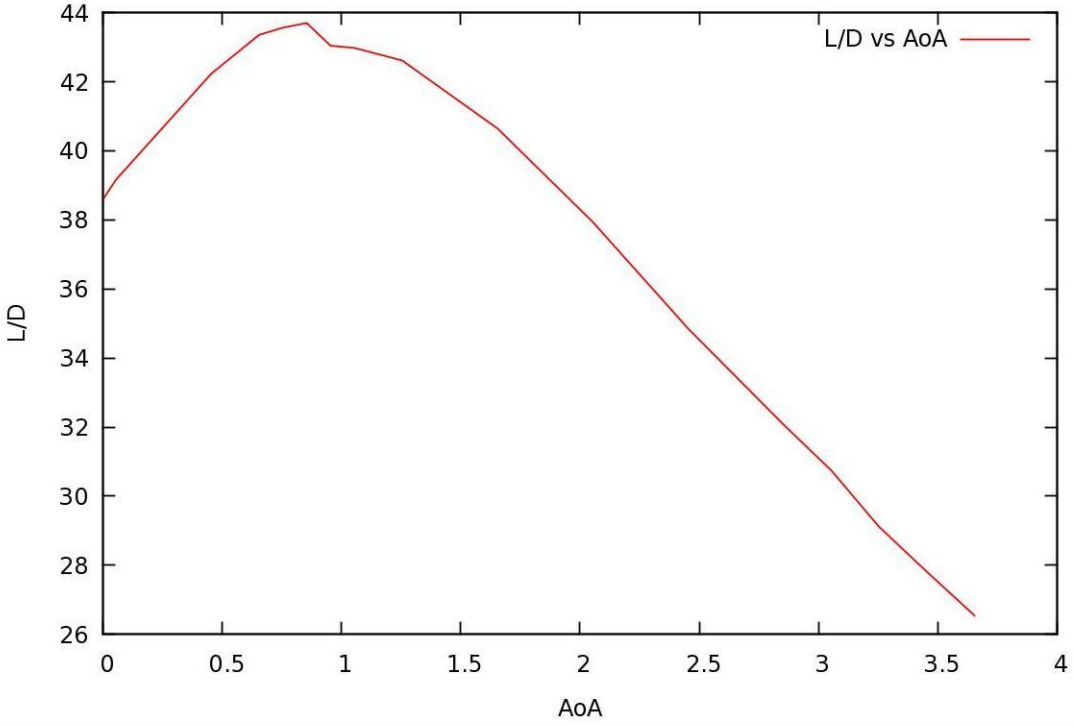

Figure 14

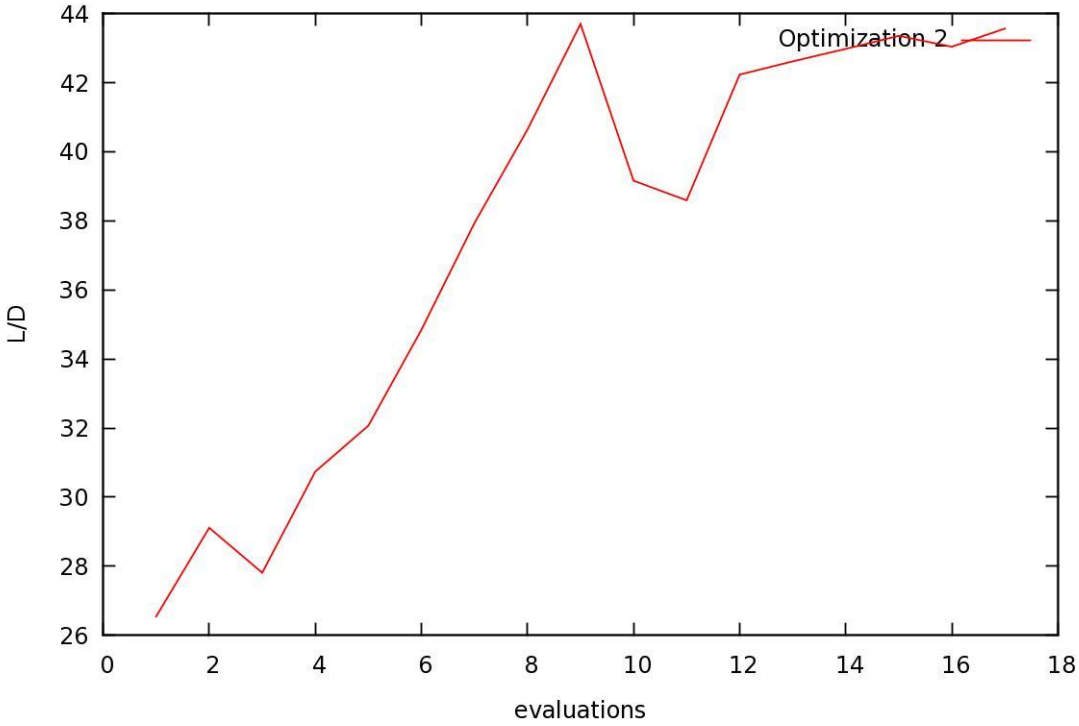

Figure 15

6.3 AoA, camber and thickness optimization using simplex

As everything was working correctly, an optimization was done for all three parameters using the simplex method. The results are shown in table 3.

Starting point	L/D	AoA, Camber, Thickness	Eval	Time
(3.65419, 8.45624, 12.6781)	26.913	(1.01795, 11.9062, 1.5)	60	5:43:07

Table 3

The result obtained in this optimization was poor as the value obtained for L/D was much lower than the one in the first experiment. This evidences that the optimization is sensitive to the starting point. Several optimizations must be started from different initial points to search the design space and find the global optimum.

The same optimization was repeated starting from a different point. The results are shown in table 4.

Starting point	L/D	AoA, Camber, Thickness	Eval	Time
(0, 4, 2)	43.569	(0.912346, 4.62593, 1.5)	40	3:50:20

Table 4

The L/D value was much better than in the previous experiment and the optimum values of the parameters were very close to the ones in experiment 1 and 2. However, as the initial point was close to the optimum values, this optimization required much less evaluations, saving almost two hours of computation.

Once again a new optimization was carried out with a closer initial point to the optimum values in the last experiment than the initial point in that same experiment. The results are shown in table 5.

Starting point	L/D	AoA, Camber, Thickness	Eval	Time
(1, 5, 2)	42.3841	(1.12099, 4.74815, 1.61111)	47	4:34:41

Table 5

The optimum values were quite similar, obtaining a slightly lower L/D. Although the initial point was closer to the optimum than in the previous experiment, the optimization took some more evaluations to finish.

This time the experiment was done again from an initial point close to the previous optimums. The results are shown in table 6.

Starting point	L/D	AoA, Camber, Thickness	Eval	Time
(1, 4, 1.5)	43.1336	(1, 5.2, 1.5)	23	2:09:10

Table 6

This optimization took much less evaluations and the L/D value was slightly higher than before. It can be seen that the initial point of the optimization is very important.

A last experiment was done with a starting point which was further to the range of solutions which were being obtained than the initial points of the previous experiments. The results are shown in table 7.

It can be seen that the L/D value and the optimum angle of attack, camber and thickness are similar to the ones in the previous experiments. However the number of evaluations is higher, as the starting point is far from the optimum solution.

Starting point	L/D	AoA, Camber, Thickness	Eval	Time
(3, 6, 8)	43.0906	(1.04444, 4.53333, 1.5)	51	4:37:51

Table 7

After all this experiments it can be concluded that the choice of the initial point is very important to obtain best results, as the solution is sensitive to this choice.

The best solution after these experiments was of 43.569 for L/D, while angle of attack, camber and thickness were of 0.912346, 4.62593 and 1.5 respectively. The angle of attack and thickness have similar values to the ones in [3], although the value of the camber is quite different. The value of the L/D is higher in [3], but these results can't be compared to those as the experiment may be set with different parameters such as the inlet velocity.

6.4 AoA, camber and thickness optimization using BFGS

An optimization was also done using the BFGS algorithm. The results are shown in table 8.

Starting point	L/D	AoA, Camber, Thickness	Eval	Time
(1, 10, 2)	32.2171	(1.28356, 9.74414, 1.5)	108	11:35:29

Table 8

This optimization shows that the BFGS algorithm requires much more evaluations than the simplex method to find the solution. The idea was to do more optimizations using the BFGS method from different starting points, and also to do those same optimizations with the simplex method to compare their results from the same starting points. However, the optimization when using the BFGS algorithm only finished for the starting point used in this optimization.

This not only happened for the BFGS algorithm, although it happened much more often than when using the simplex method. The problem occurs because Salome is unable to mesh the aerofoil and the whole optimization is waiting for it to do it. The error happened often when performing the cut operation. After loading the aerofoil into Salome to see its shape, sometimes an abnormal aerofoil had been generated, as shown in figure 16, which was the source of the error. In other cases the aerofoil seemed to be perfectly alright, but Salome was still unable to mesh it.

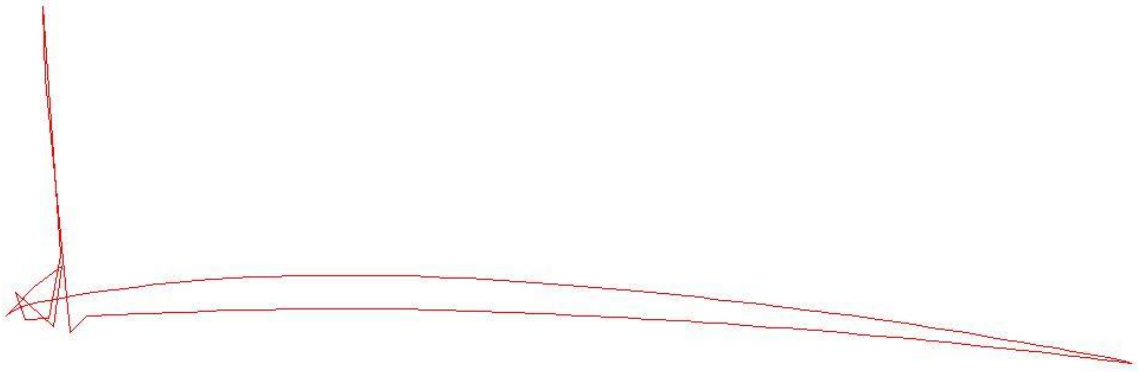

Figure 16: abnormal aerofoil

As only one optimization using the BFGS algorithm finished, the results are not conclusive and they can't be compared to the simplex method. However, during some of the failed optimizations it could be

observed, before the optimization failed, how the BFGS algorithm needed more evaluations than the simplex method.

6.5 Speedup

Finally, one of the previous optimizations was repeated using different number of processors to test the speedup. The optimizations were done using the simplex method and with (0,4,2) as a starting point. After 40 evaluations, the time needed for the optimizations is shown in table 9.

Num proc	Time
1	12:33:06
2	7:01:49
4	3:50:20

Table 9

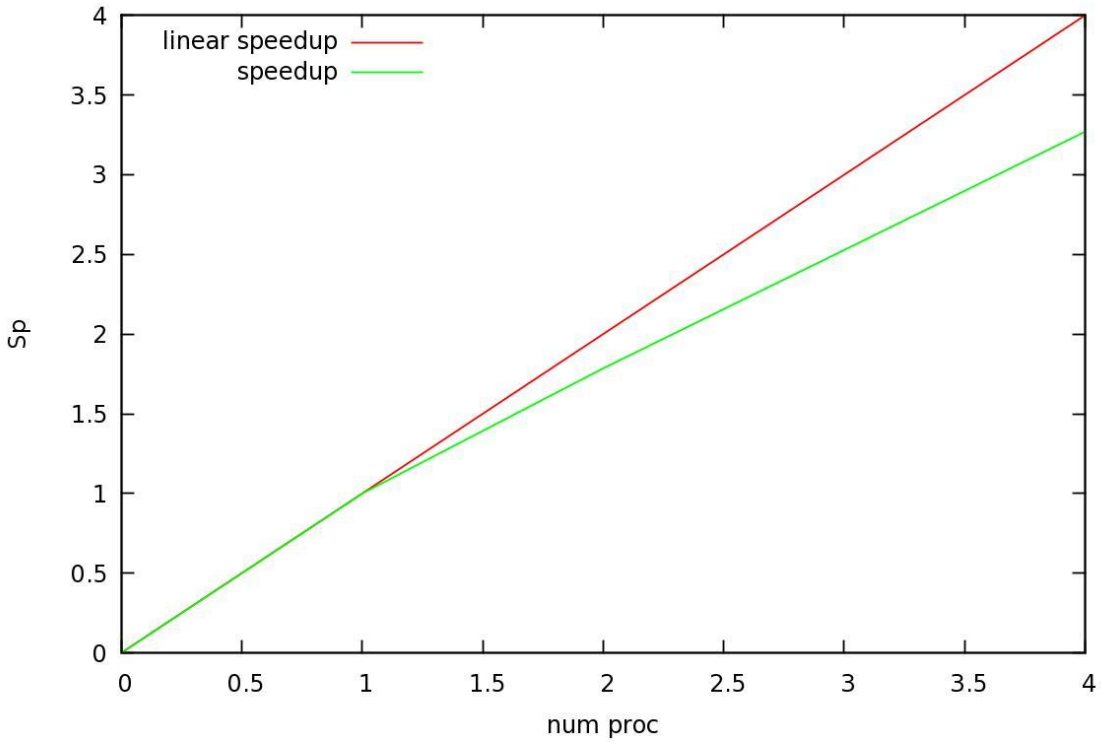

Figure 17: speedup

The speedup using two processors is of 1.78 and of 3.27 for four processors. These results suggest the potential of running the experiments under a grid environment. Moreover, when running more complex computational models the speed up will be more decisive as the bulk of the time is spent on computing the CFD code.

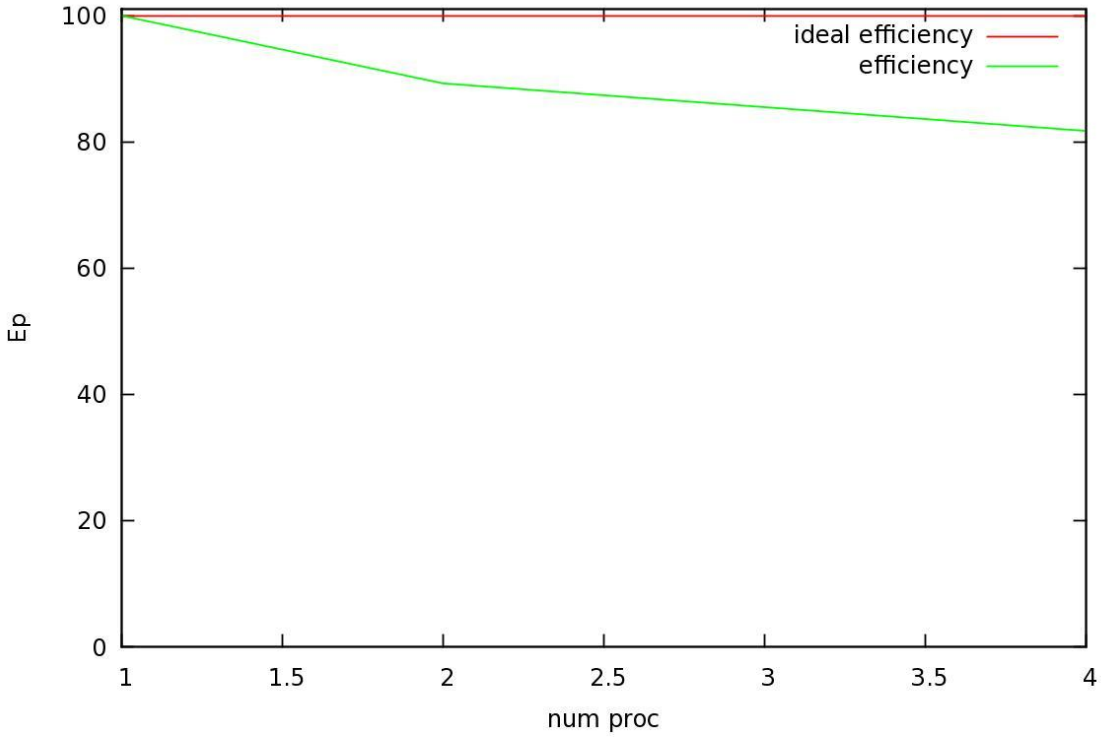

Figure 18: efficiency

Chapter 7: Conclusions and further work

A new automatic design optimization tool has been developed and has been applied successfully to a two dimensional aerofoil. During the experiments, the importance of selecting the initial point when doing the optimization has been evidenced. Also, the time of doing the optimization using different number of processors has been compared, evidencing a significant speedup when increasing its number.

Although the different programs involved have been successfully integrated, the tool is not yet as robust as it should be. A problem has been noticed during the meshing process, where several aerofoils could not be meshed. Further work must be done at this stage to clearly identify the source of the error.

Highly accurate results were not an aim of this project and therefore the mesh can be improved to achieve more accurate results. The boundary walls of the CAD model can be expanded and a mesh with more and better distributed elements can be generated.

The process has been tested on a single computer with four processors. Installing Nimrod/G will allow the user to execute the process on a computational grid and achieve significant speedup. This will not only speed up the actual optimization, but will also allow working with more complex meshes which give better results.

New parameters can also be introduced to the aerofoil. By doing so a different program to generate the aerofoils must be used or implemented, or the one used in this thesis can be improved. Again this would increase the time required for the optimization and the

installation of Nimrod/G becomes important to use as much computational power as possible.

References

[1] Brandt, Steven A. (2004), *Introduction to aeronautics: a design perspective* (2nd ed), American Institute of Aeronautics and Astronautics, Reston.

[2] Anderson, John David (2007), *Fundamentals of aerodynamics* (4th ed), McGraw-Hill Higher Education, Boston.

[3] Abramson, D., Lewis, A., Peachey, T., Fletcher, C. (2001), *An automatic design optimization tool and its application to computational fluid dynamics*, SuperComputing 2001.

[4] http://www.free-online-private-pilot-ground-school.com/images/forces_airfoil.gif, accessed 03/08/09.

[5] Anderson, John D. (2005), *Introduction to flight* (5th ed), McGraw-Hill, Boston.

[6] http://www.apstraining.com/images_remote/video-lift-drag-curve.jpg, accessed 05/08/09.

[7] Kalyanmoy, Deb. (2005), *Optimization for engineering design: algorithms and examples*, Prentice Hall, New Delhi.

[8] Garcia, M.J., Boulanger, P., Giraldo, S. (2008), *CFD based wing shape optimization through gradient-based method,* in EngOpt 2008 - International Conference on Engineering Optimization, 01-05 June 2008, Rio de Janeiro, Brazil.

[9] Oyama, A., Obayashi, S., Nakahashi, K., Nakamura, T. (2000), *Aerodynamic optimization of transonic wing design based on evolutionary algorithm, in* third International Conference on nonlinear problems in Aviation and Aerospace, 10-12 May 2000, Daytona Beach, FL, USA.

[10] Gen, M., Cheng, R. (2000), *Genetic algorithms & engineering optimization,* John Wiley, Chichester.

[11] Ashlock, D. (2006), *Evolutionary computation for modeling and optimization,* Springer, New York.

[12] Epstein, B., Peigin, S. (2006), *Optimization of 3D wings based on Navier-Stokes solutions and genetic algorithms,* International Journal of Computational Fluid Dynamics, 20:2, 75 — 92.

[13] Obayashi, S., Tsukahara, T., Nakamura, T. (2000), *Multiobjective genetic algorithm applied to aerodynamic design of cascade airfoils,* IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 47, NO. 1.

[14] Vicini, A., Quagliarella, D. (1998), *Airfoil and wing design through hybrid optimization strategies*, in AIAA Applied Aerodynamics Conference N°16, 15 June 1998, Albuquerque, NM, USA.

[15] Foster, I., Kesselman, C., Tuecke, S. (2001), *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.

[16] Song, W., Keane, A., Cox, S. (2003), *CFD-based shape optimisation with grid-enabled design search toolkits*, in Proceedings of UK e-Science All Hands Meeting 2003, 2-4 September 2003, Nottingham, UK, EPSRC, 619-627.

[17] Lad, B.S., Pillai, M.U., Gupta, Y. (2006), *A grid computing tool for engineering simulations*.

[18] Song, W., Ong, Y.S., Ng, H.K., Keane, A., Cox, S., Lee, B.S. (2004), *A service-oriented approach for aerodynamic shape optimization across institutional boundaries*, In, *Proceedings of the 8th ICARCV Control, Automation, Robotics and Vision Conference,* Institute of Electrical and Electronics Engineers, New York, USA.

[19] Katsaros, G., Campos, F., Kyriazis, D., Varvarigou, T., *CFD automatic optimisation using OpenFOAM in grid environments*, in OpenFOAM International Conference 2007, 26-27 November 2007, Beaumont House, Old Windsor, UK.

[20] Song, W., Keane, A.J., Eres, M.H., Pound, G.E., Cox., S.J. (2003), *Two Dimensional Airfoil Optimisation Using CFD in a Grid Computing Environment*, Euro-Par 2003 Parallel Processing, Lecture Notes in Computer Science, 525-532.

[21] Lewis, A., Abramson, D., Sosic, R., Giddy, J. (1995), *Tool-based parameterisation: an application perspective*.

[22] Peachey, T., Abramson, D., Lewis, A., Kurniawan, D., Jones, R. (2003), *Optimization using Nimrod/O and its application to robust mechanical design*.

[23] Abramson, D., Lewis, A., Peachey, T., (2001), *Case studies in*

automatic design optimisation using the P-BFGS algorithm, P-BFGS Algorithm", 2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, 22-26 April 2001, pp 104 – 109.

[24] Abramson, D., Lewis, A., Peachey, T., *Nimrod/O: a tool for automatic design optimization using parallel and distributed systems*.

[25] Monash eScience and grid engineering laboratory (MeSsAGE), 20 July 2009, MeSsAGE Lab - Nimrod Toolkit, accessed 05/08/09 http://messagelab.monash.edu.au/NimrodO.

[26] T. C. Peachey and Monash University, *Nimrod/O Users' Guide for Version 2.6.x.*

[27] OpenCFD Limited, *OpenFOAM UserGuide*.

[28] OpenCFD Limited, OpenFOAM® - The Open Source Computational Fluid Dynamics (CFD) Toolbox, accessed 06/08/09, http://www.opencfd.co.uk/openfoam/.

[29] OPEN CASCADE SAS (OCC), Salome-Meca, http://www.salome-platform.org/ex/doc3.2.6/GUI/doc/salome/gui/GUI/index.htm, accessed 03/08/09.

[30] Mike Riley, *Linux-based aerofoil IGES generator*.

Appendix A: C++ programs

wait__for_unv.cc

```
#include <iostream>
#include <fstream>

using namespace std;

main (int argc, char ** argv){

  ifstream fitx;

  do{
    fitx.open("/home/nicky/OpenFOAM/Simulations/Aerofoil/mesh.unv");
  }while(!fitx);

  fitx.close();
}
```

lift_drag.cc

```
#include <iostream>
#include <fstream>
#include <string.h>

using namespace std;

main (int argc, char ** argv){

  ifstream fitx;
  double drag, lift, result;


fitx.open("/home/nicky/OpenFOAM/Simulations/Aerofoil/forceCoeffs/1/fil
e.txt");
  if(!fitx)
    cerr << "error!";

  fitx >> drag >> lift;

  result = (lift/drag)*(-1);

  cout << result << endl;

  fitx.close();
}
```

edit_controldict.cc

```
#include <iostream>
#include <fstream>
#include <string.h>

using namespace std;

main (int argc, char ** argv){

  ifstream fitx1, fitx2;
  FILE *fp;
  char buffer[128];
  char buffer2[16];
  int i=0;;

  fitx1.open("workfile");
  if(!fitx1)
    cerr << "error!";


fitx2.open("/home/nicky/OpenFOAM/Simulations/Aerofoil/system/controlDi
ct");
  if(!fitx2)
    cerr << "error!";


fp=fopen("/home/nicky/OpenFOAM/Simulations/Aerofoil/system/controlDict
", "r+");
  while(fitx2.getline(buffer, 256)) {
    if(i==70){
      strcpy(buffer, "Aref ");
      fwrite(buffer, 1, strlen(buffer), fp);
      fitx1.getline(buffer,16);
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, ";");
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
    }
    else{
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
    }
    i++;
  }

  fitx1.close();
  fitx2.close();
  fclose(fp);

}
```

edit_boundary.cc

```
#include <iostream>
#include <fstream>
#include <string.h>

using namespace std;

main (int argc, char ** argv){

  ifstream fitx;
  FILE *fp;
  char buffer[128];
  int i=0;


fitx.open("/home/nicky/OpenFOAM/Simulations/Aerofoil/constant/polyMesh
/boundary");
  if(!fitx)
    cerr << "error!";


fp=fopen("/home/nicky/OpenFOAM/Simulations/Aerofoil/constant/polyMesh/
boundary", "r+");
  while(fitx.getline(buffer, 256)) {
    switch(i) {
    case 21:
      strcpy(buffer, "        type            wall;");
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
      break;

    case 39:
      strcpy(buffer, "        type            wall;");
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
      break;

    case 45:
      strcpy(buffer, "        type            wall;");
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
      break;

    case 51:
      strcpy(buffer, "        type            empty;");
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
      break;

    case 57:
      strcpy(buffer, "        type            empty;");
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);
```

```
      break;

    default:
      fwrite(buffer, 1, strlen(buffer), fp);
      strcpy(buffer, "\n");
      fwrite(buffer, 1, strlen(buffer), fp);

    }
    i++;
  }

  fitx.close();
  fclose(fp);

}
```

Appendix B: Scripts

script

```
cd /home/nicky
./CommandLineToIges --AoA $1 --camber $2 --thickness $3 --outputPath
/home/nicky/Simulation/ --outputFilename aerofoil
rm /home/nicky/OpenFOAM/Simulations/Aerofoil/mesh.unv
rm -r /home/nicky/OpenFOAM/Simulations/Aerofoil/processor*
cd /opt/SALOME-MECA-2008.1-GPL/SALOME/SALOME3/V3_2_9NoDebug
./runSalome-debianForSalome -t -u mesh.py -k &
cd /home/nicky/Simulation
./wait_for_unv
sleep 3
cd /home/nicky/OpenFOAM/Simulations/Aerofoil
ideasUnvToFoam mesh.unv
transformPoints -scale "(0.001 0.001 0.001)"
cd /home/nicky/Simulation
./edit_boundary
./edit_controldict
sleep 2
cd /home/nicky/OpenFOAM/Simulations/Aerofoil
decomposePar
sleep 2
mpirun -np 4 simpleFoam -parallel > log
sleep 2
reconstructPar
cd /home/nicky/OpenFOAM/Simulations/Aerofoil/forceCoeffs/1
tail -1 forceCoeffs.dat | awk '{print $2}' > file.txt
tail -1 forceCoeffs.dat | awk '{print $3}' >> file.txt
```

mesh.py

```
### This file is generated by SALOME automatically by dump python
functionality

import sys
import salome

salome.salome_init()

sys.path.insert( 0, '/opt/SALOME-MECA-2008.1-
GPL/SALOME/SALOME3/V3_2_9NoDebug')

import mesh_GEOM
mesh_GEOM.RebuildData(salome.myStudy)
import mesh_SMESH
mesh_SMESH.RebuildData(salome.myStudy)

if salome.sg.hasDesktop():
      salome.sg.updateObjBrowser(1)
```

mesh_GEOM.py

```
### This file is generated by SALOME automatically by dump python
functionality
### of GEOM component

import geompy
import math

def RebuildData(theStudy):
      geompy.init_geom(theStudy)
      global Face_5, Edge_20, Edge_2, Group_1, Edge_19, Face_7,
Edge_18, Face_1, Edge_9, Edge_1, Edge_15, Vertex_3, Edge_8, Edge_22,
Edge_14, Edge_21, Vertex_4, Edge_7, Face_2, Edge_13, Face_4, Vertex_1,
Extrusion_2, Edge_6, Face_6, Edge_25, Edge_12, Cut_1, Edge_24, Face_8,
Vertex_2, Edge_5, Edge_23, Edge_11, Quadrangle_Face_1, Edge_17,
Edge_4, Edge_16, Extrusion_1, aerofoil_igs_1, Edge_3, Face_3,
Vector_1, Edge_10
      aerofoil_igs_1 =
geompy.Import("/home/nicky/Simulation/aerofoil.igs", "IGES")
      Vertex_1 = geompy.MakeVertex(-500, 500, 0)
      Vertex_2 = geompy.MakeVertex(-500, -500, 0)
      Vertex_3 = geompy.MakeVertex(1500, -500, 0)
      Vertex_4 = geompy.MakeVertex(1500, 500, 0)
      Vector_1 = geompy.MakeVectorDXDYDZ(0, 0, 1)
      Face_1 = geompy.MakeFaceWires([aerofoil_igs_1], 1)
      Quadrangle_Face_1 = geompy.MakeQuad4Vertices(Vertex_1, Vertex_2,
Vertex_3, Vertex_4)
      Extrusion_1 = geompy.MakePrismVecH(Quadrangle_Face_1, Vector_1,
5)
      Extrusion_2 = geompy.MakePrismVecH(Face_1, Vector_1, 5)
      Cut_1 = geompy.MakeCut(Extrusion_1, Extrusion_2)
      [Face_2,Face_3,Face_4,Face_5,Face_6,Face_7,Face_8] =
geompy.SubShapeAllSorted(Cut_1, geompy.ShapeType["FACE"])
      [Edge_1,Edge_2,Edge_3,Edge_4,Edge_5,Edge_6,Edge_7,Edge_8,Edge_9,
Edge_10,Edge_11,Edge_12,Edge_13,Edge_14,Edge_15] =
geompy.SubShapeAllSorted(Cut_1, geompy.ShapeType["EDGE"])
      listSubShapeIDs = geompy.SubShapeAllIDs(Cut_1,
geompy.ShapeType["EDGE"])
      None
      [Edge_16,Edge_17,Edge_18,Edge_19,Edge_20] =
geompy.SubShapeAllSorted(Face_5, geompy.ShapeType["EDGE"])
      Cut_1 = geompy.GetMainShape(Face_5)
      Cut_1 = geompy.GetMainShape(Face_5)
      None
      [Edge_21,Edge_22,Edge_23,Edge_24,Edge_25] =
geompy.SubShapeAllSorted(Face_6, geompy.ShapeType["EDGE"])
      Cut_1 = geompy.GetMainShape(Face_6)
      Cut_1 = geompy.GetMainShape(Face_6)
      None
      Face_5 = geompy.GetMainShape(Edge_17)
      None
      Face_6 = geompy.GetMainShape(Edge_22)
      area = geompy.BasicProperties(Face_3)
      area2 = area[1]/1000000
      f = open('/home/nicky/Simulation/workfile','w')
      s = str(area2)
      f.write(s)
      f.close()
      Group_1 = geompy.CreateGroup(Cut_1, geompy.ShapeType["EDGE"])
      geompy.UnionIDs(Group_1, [6, 9, 16, 23, 44])
```

```
Cut_1 = geompy.GetMainShape(Group_1)
Cut_1 = geompy.GetMainShape(Group_1)
geomObj_1 = geompy.GetSubShape(Cut_1, [6])
geomObj_2 = geompy.GetSubShape(Cut_1, [9])
geomObj_3 = geompy.GetSubShape(Cut_1, [16])
geomObj_4 = geompy.GetSubShape(Cut_1, [23])
geomObj_5 = geompy.GetSubShape(Cut_1, [44])
geompy.addToStudy( aerofoil_igs_1, "aerofoil.igs_1" )
geompy.addToStudy( Vertex_1, "Vertex_1" )
geompy.addToStudy( Vertex_2, "Vertex_2" )
geompy.addToStudy( Vertex_3, "Vertex_3" )
geompy.addToStudy( Vertex_4, "Vertex_4" )
geompy.addToStudy( Vector_1, "Vector_1" )
geompy.addToStudy( Face_1, "Face_1" )
geompy.addToStudy( Quadrangle_Face_1, "Quadrangle Face_1" )
geompy.addToStudy( Extrusion_1, "Extrusion_1" )
geompy.addToStudy( Extrusion_2, "Extrusion_2" )
geompy.addToStudy( Cut_1, "Cut_1" )
geompy.addToStudyInFather( Cut_1, Face_2, "Face_2" )
geompy.addToStudyInFather( Cut_1, Face_3, "Face_3" )
geompy.addToStudyInFather( Cut_1, Face_4, "Face_4" )
geompy.addToStudyInFather( Cut_1, Face_5, "Face_5" )
geompy.addToStudyInFather( Cut_1, Face_6, "Face_6" )
geompy.addToStudyInFather( Cut_1, Face_7, "Face_7" )
geompy.addToStudyInFather( Cut_1, Face_8, "Face_8" )
geompy.addToStudyInFather( Cut_1, Edge_1, "Edge_1" )
geompy.addToStudyInFather( Cut_1, Edge_2, "Edge_2" )
geompy.addToStudyInFather( Cut_1, Edge_3, "Edge_3" )
geompy.addToStudyInFather( Cut_1, Edge_4, "Edge_4" )
geompy.addToStudyInFather( Cut_1, Edge_5, "Edge_5" )
geompy.addToStudyInFather( Cut_1, Edge_6, "Edge_6" )
geompy.addToStudyInFather( Cut_1, Edge_7, "Edge_7" )
geompy.addToStudyInFather( Cut_1, Edge_8, "Edge_8" )
geompy.addToStudyInFather( Cut_1, Edge_9, "Edge_9" )
geompy.addToStudyInFather( Cut_1, Edge_10, "Edge_10" )
geompy.addToStudyInFather( Cut_1, Edge_11, "Edge_11" )
geompy.addToStudyInFather( Cut_1, Edge_12, "Edge_12" )
geompy.addToStudyInFather( Cut_1, Edge_13, "Edge_13" )
geompy.addToStudyInFather( Cut_1, Edge_14, "Edge_14" )
geompy.addToStudyInFather( Cut_1, Edge_15, "Edge_15" )
geompy.addToStudyInFather( Face_5, Edge_16, "Edge_16" )
geompy.addToStudyInFather( Face_5, Edge_17, "Edge_17" )
geompy.addToStudyInFather( Face_5, Edge_18, "Edge_18" )
geompy.addToStudyInFather( Face_5, Edge_19, "Edge_19" )
geompy.addToStudyInFather( Face_5, Edge_20, "Edge_20" )
geompy.addToStudyInFather( Face_6, Edge_21, "Edge_21" )
geompy.addToStudyInFather( Face_6, Edge_22, "Edge_22" )
geompy.addToStudyInFather( Face_6, Edge_23, "Edge_23" )
geompy.addToStudyInFather( Face_6, Edge_24, "Edge_24" )
geompy.addToStudyInFather( Face_6, Edge_25, "Edge_25" )
geompy.addToStudyInFather( Cut_1, Group_1, "Group_1" )
pass
```

mesh_SMESH.py

```
### This file is generated by SALOME automatically by dump python
functionality of SMESH component

import salome, SMESH
import smesh

## import GEOM dump file ##
import string, os, sys, re
sys.path.insert( 0, os.path.dirname(__file__) )
exec("from "+re.sub("SMESH$","GEOM",__name__)+" import *")


def RebuildData(theStudy):
      aFilterManager = smesh.smesh.CreateFilterManager()
      smesh.smesh.SetCurrentStudy(theStudy)
      import StdMeshers
      pattern = smesh.GetPattern()
      Mesh_1 = smesh.Mesh(Cut_1)
      Regular_1D = Mesh_1.Segment()
      Average_length_1 = Regular_1D.LocalLength(15)
      Quadrangle_2D = Mesh_1.Quadrangle()
      Prism_3D = Mesh_1.Prism()
      Nb_Segments_1 = smesh.smesh.CreateHypothesis('NumberOfSegments',
'libStdMeshersEngine.so')
      Nb_Segments_1.SetNumberOfSegments( 1 )
      Nb_Segments_1.SetDistrType( 0 )
      SubMesh_1 = Mesh_1.GetSubMesh( Group_1, 'SubMesh_1' )
      status = Mesh_1.AddHypothesis(Regular_1D,Group_1)
      status = Mesh_1.AddHypothesis(Nb_Segments_1,Group_1)
      MEFISTO_2D = Mesh_1.Triangle(geom=Face_5)
      SubMesh_2 = MEFISTO_2D.GetSubMesh()
      Length_From_Edges_2D_Hyp_for_Triangulator__1 =
MEFISTO_2D.LengthFromEdges()
      Projection_2D = Mesh_1.Projection2D(geom=Face_6)
      SubMesh_3 = Projection_2D.GetSubMesh()
      Source_Face_1 =
Projection_2D.SourceFace(Face_5,None,None,None,None,None)
      Average_length_2 = smesh.smesh.CreateHypothesis('LocalLength',
'libStdMeshersEngine.so')
      Average_length_2.SetLength( 3 )
      SubMesh_4 = Mesh_1.GetSubMesh( Edge_17, 'SubMesh_4' )
      status = Mesh_1.AddHypothesis(Regular_1D,Edge_17)
      status = Mesh_1.AddHypothesis(Average_length_2,Edge_17)
      Projection_1D = Mesh_1.Projection1D(geom=Edge_22)
      SubMesh_5 = Projection_1D.GetSubMesh()
      Source_Edge_1 = Projection_1D.SourceEdge(Edge_17,None,None,None)
      isDone = Mesh_1.Compute()
      Face_3_1 = Mesh_1.Group(Face_3)
      Face_2_1 = Mesh_1.Group(Face_2)
      Face_8_1 = Mesh_1.Group(Face_8)
      Face_7_1 = Mesh_1.Group(Face_7)
      Face_4_1 = Mesh_1.Group(Face_4)
      Face_6_1 = Mesh_1.Group(Face_6)
      Face_5_1 = Mesh_1.Group(Face_5)
      Mesh_1.ExportUNV( '/opt/OpenFOAM/Simulations/Aerofoil/mesh.unv'
)
```

```
      ## set object names
      isGUIMode = 1
      if isGUIMode and salome.sg.hasDesktop():
            smesh.SetName(Mesh_1.GetMesh(), 'Mesh_1')
            smesh.SetName(Regular_1D.GetAlgorithm(), 'Regular_1D')
            smesh.SetName(Average_length_1, 'Average length_1')
            smesh.SetName(Quadrangle_2D.GetAlgorithm(),
'Quadrangle_2D')
            smesh.SetName(Prism_3D.GetAlgorithm(), 'Prism_3D')
            smesh.SetName(Nb_Segments_1, 'Nb. Segments_1')
            smesh.SetName(SubMesh_1, 'SubMesh_1')
            smesh.SetName(MEFISTO_2D.GetAlgorithm(), 'MEFISTO_2D')
            smesh.SetName(SubMesh_2, 'SubMesh_2')
            smesh.SetName(Length_From_Edges_2D_Hyp_for_Triangulator__1,
'Length From Edges (2D Hyp. for Triangulator)_1')
            smesh.SetName(Projection_2D.GetAlgorithm(),
'Projection_2D')
            smesh.SetName(SubMesh_3, 'SubMesh_3')
            smesh.SetName(Source_Face_1, 'Source Face_1')
            smesh.SetName(Average_length_2, 'Average length_2')
            smesh.SetName(SubMesh_4, 'SubMesh_4')
            smesh.SetName(Projection_1D.GetAlgorithm(),
'Projection_1D')
            smesh.SetName(SubMesh_5, 'SubMesh_5')
            smesh.SetName(Source_Edge_1, 'Source Edge_1')
            smesh.SetName(Face_3_1, 'Face_3')
            smesh.SetName(Face_2_1, 'Face_2')
            smesh.SetName(Face_8_1, 'Face_8')
            smesh.SetName(Face_7_1, 'Face_7')
            smesh.SetName(Face_4_1, 'Face_4')
            smesh.SetName(Face_6_1, 'Face_6')
            smesh.SetName(Face_5_1, 'Face_5')

            salome.sg.updateObjBrowser(0)

      pass
```

Appendix C: Nimrod schedule file

schedule.shd

```
parameter aoa float range from 0 to 4
parameter camber float range from 0 to 12
parameter thickness float range from 1.5 to 14

constraint camber>0

task main
execute ./script $aoa $camber $thickness
execute ./lift_drag > result
copy result output.$jobname
endtask

method simplex
starts 1
tolerance 0.01
endstarts
endmethod
```