UNIVERSITÁ DEGLI STUDI DI GENOVA

CORSO DI LAUREA IN INGEGNERIA MECCANICA

A.A. 2014/2015

“Numerical simulations of jet impingement cooling: OpenFOAM vs. Ansys Fluent comparison”

Relatori: Chiar.mo Prof. Ing. Alessandro Bottaro

Chiar.mo Prof. Ing. Paweł Flaszyński

Candidato: Alessandro Dapelo

Marzo 2016

In collaboration with:

Acknowledgements

The first person I would like to thank is Prof. Alessandro Bottaro, who proposed me this thesis and has given me great support throughout all the experience, showing uncommon dedication to his job. Then, I would like to thank Piotr Doerffer and Paweł Flaszyński for agreeing, without any hesitation, to supervise an external thesis and providing valuable input during all the work. I cannot forget all the assistants which whom I have shared the office and who helped me solving CFD issues. Fernando, Javier, Filip and Rohan made my stay in Gdánsk more pleasant and gave me support during the very first days in a completely new city. I want to thank also the other guys of the department, Oscar, Arthur, etc., for the support. I would like also to mention Mikko Folkersma, Damiano Natali and Joel Guerrero for the disinterested help with OpenFOAM.
I would like to remember all the other amazing people I have met in Poland, who made me feel at home in a foreign country. A special thanks goes to my flat mates, my neighbors and the dorm crew. I wish I could stay longer, sharing other unforgettable moments with you.
Finally, I want to thank Zuza, Timo, Natalia, Elizabeth, Cristina, Francisco and Matteo for the great time we had all together, and Jacob, who has become a friend rather than a landlord. You are part of my Polish family now, and I promise I will come back to visit you!

The thesis is dedicated to my family and my friends, who supported me every single day during my studies.

Abstract

This thesis aims to evaluate if OpenFOAM is already able to replace an expensive commercial code, comparing results with Ansys Fluent, one of the most known and used commercial CFD software around the world, for two cases of interest.

The first is a thermal flat plate, with simple geometry and physics, while the second is a more complex impinging jet with axisymmetric geometry.

The simple first case has been studied in order to take confidence with the new software and to confirm how both ANSYS Fluent and OpenFOAM can lead to the analytical solution, unfortunately available only for very easy cases.

With the second case, we wanted to check the capability of OpenFOAM to solve different problems: in fact this case has been solved with different solvers and boundary conditions.

Both incompressible and compressible, steady and unsteady, laminar and turbulent cases have been investigated, leading to somewhat different solutions if compared to Fluent.

OpenFOAM has showed good capabilities but a higher cost in terms of time is needed to set up properly difficult cases, especially if compared to Fluent. In particular, the learning curve is quite steep, and the lack of an integrated GUI (Graphical User Interface) and of a detailed guide doesn’t help, especially at the beginning.

Prefazione

La fluidodinamica computazionale sta diventando sempre più importante oggigiorno, estendendo le proprie capacità ad ambiti mai affrontati prima d'ora. Il costo annuale di una licenza per uno dei più conosciuti software commerciali può superare le svariate migliaia di euro. Fortunatamente, alcuni software gratuiti si stanno affacciando alla scena, offrendo funzioni e capacità simili ai concorrenti commerciali. Uno dei software open source più conosciuti è OpenFOAM; dietro al suo successo sta, oltre alle solide capacità in tutti i maggiori ambiti legati alla CFD, una vibrante comunità sempre al lavoro per fixare bugs ed aggiungere nuove funzionalità.

Questo lavoro di tesi punta a valutare il programma gratuito OpenFOAM in due casi di interesse, confrontando i risultati ottenuti con Ansys Fluent, uno dei software commerciali più usati e conosciuti. Il primo caso analizzato è costituito da una semplice lastra piana, mentre nel secondo abbiamo studiato un più complesso getto impingente, caratterizzato da geometria assialsimmetrica. Si è scelto il caso della lastra piana sia per prendere confidenza con il nuovo software, sia per vedere i risultati ottenibili in confronto alla soluzione analitica, disponibile per questa semplice geometria. Il getto è stato risolto in OpenFOAM con diversi solver e diverse condizioni al contorno, per valutare la capacità del software a trattare problemi di diversa natura; sono state condotte simulazioni comprimibili e incomprimibili, laminari e turbolente, stazionarie e instazionarie.

OpenFOAM ha dimostrato buone capacità nei casi da noi investigati, ma va tenuto in conto il maggior tempo solitamente richiesto per la messa a punto della simulazione. Inoltre la curva di apprendimento risulta particolarmente ripida, specialmente all'inizio, ed è richiesto del tempo prima di poter padroneggiare in maniera soddisfacente il software.

Il lavoro di tesi è stato svolto presso l'istituto di macchine a fluido di Danzica sotto la supervisione del Prof. Paweł Flaszyński e con la collaborazione di tutto il dipartimento di Aerodinamica.

Index

1. Introduction

Nowadays, the simulative capability in the engineering field has become more and more important. The continue rise of the computing performances and the subsequential decrease in costs give us the possibility to simulate very complex case in a reasonable time. CFD (Computational Fluid Dynamics) and Structure simulation allow us to investigate new devices without manufacturing the object itself, saving time and money. In particular, when we are studying something completely new or more complex than usual, we can simulate the behavior expected before producing the first sample. In some fields, where the technology level is very high and the cost in terms of time and resources of some manufacturing processes can be unsustainable, it provides a powerful tool to discard bad solutions and to better investigated the most interesting set-ups.

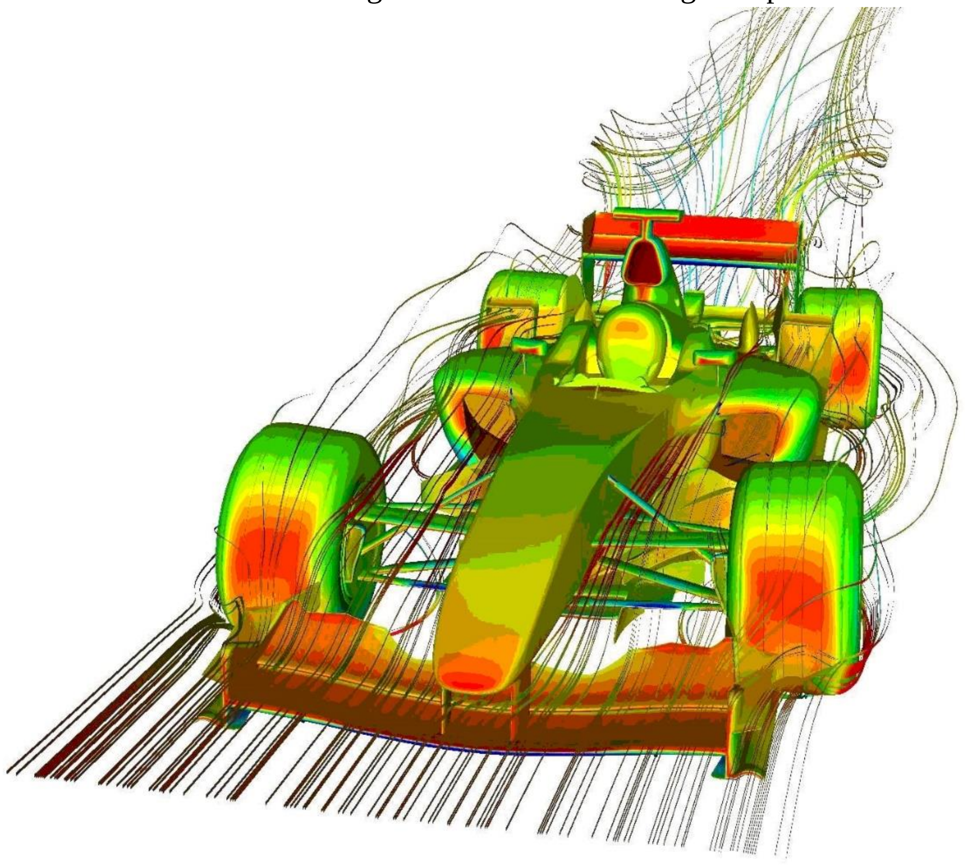


Fig. 1.1: CFD simulation of a Formula 1 car

Just to give an example, Formula 1 teams can analyze different aerodynamic solutions and manufacture only the most effective ones. This allows them to substantially improve the performances of the race cars in the few time between two races (2 weeks and even less), reducing the cost of manufacturing carbon fiber parts not as effective as wanted. Sometimes it happens that they do not obtain the desired results: to save as much time as possible, they need to simplify the equations solved by the software introducing some hypothesis. This can

lead to an unreal solution and this is one of the main reason why it is very important to compare against experiments the results obtained.
Given the geometry, numerical methods allow us to study several different flow conditions in few time, while changes in the set-up of experimental measures can require days.
One of the biggest withdraw of commercial solvers for CFD simulations is the cost: one license can cost several thousands of euros and it is limited in time extension. In this context, many universities developed their own codes, usually for their exclusive use, saving the license cost and adding some features over the main commercial codes to suit better the problems they use to investigate. In other cases, these codes has been released as open source software; the most used and known freeware code is OpenFOAM and it is also the one that gives us the possibility to deal with different problems. So we decided to compare OpenFOAM with one of the most, if not the most known commercial code: Ansys Fluent.
In this thesis we investigate if a free, open source software can compete with a commercial one.
What can we expect from a free-to-use software, developed and continuously updated by the fervent community? We try to give you an answer, which will be far to be definitive, as OpenFOAM is growing really fast adding every time new features and fixing some bugs. Moreover, the capabilities used are only a small part of the huge quantity available, and so the results we are going to show could not apply in other fields.
The comparison will not regard only the quality of the results (which is obviously the most important thing) but also the time requested from each code to set-up the case and to find the solution. These aspects are very important as well, as we want to obtain a correct solution and we want it as fast as possible: obtaining equal solutions, a considerable difference in time could push customers to pay for a commercial code.

1.1. IMP PAN Institute of Fluid-Flow Machinery - Polish Academy of Sciences

All the simulations reported in this thesis have been conducted at the Institute of Fluid Flow Machinery in Gdańsk under the supervision of Paweł Flaszyński and the Aerodinamics Department. In this department they currently use Ansys Fluent and Numeca FINE Turbo to perform their simulation.
Their activities includes:

- Investigation of flow in boundary layer, and in particular transition and unsteady phenomena, heat transfer and boundary layer transition induced by wakes.
- Vortex generators and their applications on helicopters and wind turbines for noise and drag reduction.
- Transition location effect on shock wave boundary layer interaction, as coordinator of the TFAST project, whose first phase has just ended. The project try to reduce drag and losses in order to respect emissions reduction expected by 2020 for airplanes and is supported by several important partners.
- Gas turbine cooling.

They are interested in OpenFOAM as they have a limited number of licenses for other commercial codes and the possibility of working also when all the licenses are occupied or when the licenses server is down is attracting. Another feature they are interested in is the possibility of looking inside the code; this is very interesting, especially for an institute of research, and can be quite useful when something weird is noticed in a simulation or when they do not obtain the solution expected. In fact, with the closed commercial codes it is impossible to say what is going on in such a case, as the algorithms are hidden behind the GUI and not controllable. Moreover, the open source license allows researchers to work also at home or in any other place out of the institute. This is not possible with commercial codes, like Numeca and Fluent, whose licenses are located in the institute server and not available out of the institute Web connection. As both researchers and professors are often in travel because of conferences etc., the ability to work on the move is not to be considered unimportant. The operative system used by the majority of the researchers at the institute is Linux and so they would have no problems using the terminal as input interface and workspace.

Fig. 1.2: Imp PAN building in Gdánsk

1.2. Structure of the thesis

The work of thesis has been structured in the following way:

- Chapter 1: general outlook on CFD, with a quick introduction of the topics. Some information about the institute and overview of the software utilized during the period in Poland.
- Chapter 2: in this chapter we can find just a little bit of theory. This theoretical reminder is not an in depth analysis and want just to make clear some differences between some simulations run with different settings.
- Chapter 3: some information about OpenFOAM and its capabilities, either out of the box and with custom codes.
- Chapter 4: results and comparison for the flat plate are reported; it is also present a comparison with the approximate analytical solution of Blasius' equation.
- Chapter 5: impinging jets theory and possible applications; performances with different layouts are described.
- Chapter 6: results of the simulations carried out on the impinging jet geometry.
- Chapter 7: conclusive chapter with a resume of the results obtained and some comments. The fields where an improvement is expected, in order to fill the gap with other software, are presented in a critical way.
- Appendix: it collects all the important parts of code not directly reported in the chapters. In this way it is possible to reproduce the same simulations, as in the code are presented also settings we do not deal with during the discussion, as they are less important for the comparison.

In the first part of the book it is possible to find the Abstract, both in Italian and in English, with the Acknowledgements.

1.3. Software utilized

In this paragraph we present the software adopted during this work; we want to do this to be as clear as possible in the presentation of the results and to allow readers interested to obtain the same results.

IGG mesher

The first software used has been IGG mesher by Numeca. It allows the creation of 2D and 3D meshes, supporting also Python scripts. Numeca is a leader company in providing CFD software. Born in the University of Bruxelles, it has become famous for the software FINE Turbo, especially conceived for the analysis of turbomachinery, but not only. Anyway, we did not use its solver but only the mesher. This has been done because it is a powerful and easy-to-use tool and, moreover, it has allowed us to check how it works the mesh importing procedure in both OpenFOAM and Fluent. The utilization of a third party mesh should also make more impartial the whole comparison.

Tecplot 360
Tecplot 360 is a CFD and numerical simulation software package used in post-processing simulation results. While the first contact can be quite difficult, being the software quite complex and full of functionalities, once you know where to put your hands it is a real powerful software and saves a lot of time while post-processing. Unfortunately, the utility available in OpenFOAM to export data to Tecplot did not work properly, with some data computed missing or messed up. We spent some time in order to understand which was the problem but we managed to import correctly data only saving results as *.csv* (comma separated values); so the procedure became less automatized and more mechanical, and we could not save time at all. It is possible that the utility *foamToTecplot360* is not yet supported by the OpenFOAM version utilized. Exporting data from Fluent does not present these issues and is straightforward.

OpenFOAM
We used the last version available when we started the work, i.e. OpenFOAM 2.4.x. We installed a whole package with Linux OpenSUSE 13.2. This Linux version was made by Wolf Dynamics including all the tools profitable for open source CFD. In this version are also included additional OpenFOAM applications, utilities and libraries. Paraview is present in version 4.4. At late 2015 the OpenFOAM foundation released the new version, OpenFOAM 3.0. As we did not want to change the version with a certain number of simulations already done, we did not updated it. So, all results presented in this work refer to the version 2.4. We have only tried the last version of Paraview, the number 5, to check the new post processing functionalities.

Ansys Fluent
All the simulations have been run on the version 16.2 of Fluent. We ran on both the standard and the academic version and we did not notice any difference. This result was expected, as Ansys claims the academic version to be only limited in the dimensions of the problem and not in other features. For some preliminary post processing, we have also used CFD Post, the post-processing tool by Ansys.

Excel
All the graphs in this thesis have been plotted with Excel, the spreadsheets software by Microsoft available in the Office suite. Even if it is possible to obtain the same plots also with Tecplot, I opted for Excel because of the best design of the graphs and also because I did not have the possibility to use Tecplot out of the institute, and this would not have allowed me to work at home and then, later, in Italy.

2. Theoretical Outlook

2.1. Governing equations

The governing equations for fluids consist of a continuity equation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0 \tag{2.1}$$

a momentum equation:

$$\left(\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) + \rho f_i \tag{2.2}$$

and an energy equation:

$$\left(\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u_j E}{\partial x_j}\right) == -\frac{\partial p u_j}{\partial x_i} + \frac{\partial u_i \tau_{ji}}{\partial x_j} + \frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_j}\right) + S_E \tag{2.3}$$

In the equations above, f_i is the force applied on the fluid, S_E is an energy source term and the tensor τ_{ji}:

$$\tau_{ji} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij} \tag{2.4}$$

Pressure $p=p(\rho,T)$ and internal energy $i=i(\rho,T)$ are related to the variables ρ and T thanks to the equations of state (e.g. for an ideal gas $p=\rho RT$ and $i=C_vT$), allowing us to solve the equations above.

The conservation equation for mass and momentum are more complex than they appear. They are non-linear, coupled and difficult to solve. By existing mathematical tools is difficult even to prove that a unique solution exists for particular boundary conditions. In all cases in which such a solution is possible, many terms in the equations are zero.

2.1.1. Incompressible fluids

The conservation equation presented above are the most general ones and they assume that all fluid and properties vary in space and time. With gases at low Mach numbers (<0.3) and liquids, compressible effects can be neglected and the continuity equation reduces to:

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{2.5}$$

As there is no link between the energy equation and momentum and continuity equations.

After simplifying the governing equations using the incompressible condition, the following system of equations is received:

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{2.5}$$

$$\left(\frac{\partial u_i}{\partial t} + \frac{\partial u_j u_i}{\partial x_j}\right) = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\nu\frac{\partial u_i}{\partial x_j}\right) + f_i \tag{2.6}$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity. The simplification introduced is not of a great value and the equation are only slightly simpler to solve. In any case it does help in numerical solution.

2.1.2. Inviscid (Euler) Flow

If the flow is far from solid surfaces, the effects of viscosity are usually very small. If we neglect viscous effects, the Navier-Stokes equations reduce to the Euler equations. While the continuity equation remains the same, the momentum equation are:

$$\left(\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i} + \rho f_i \qquad (2.7)$$

The fluid cannot stick to walls and slip is possible at solid boundaries. The Euler equations are usually used to study compressible flows at high Mach numbers: in fact at high velocities, the Reynolds number is very high and viscous and turbulence effects are important only in a small region near the walls. These equations are not easy to solve but no boundary layer near the walls have to be resolved and this allows the use of coarser grids.

2.1.3. Boussinesq Approximation

In flows in which heat transfer is present, the fluid properties are normally functions of temperature. The variations may be small and yet be the cause of the fluid motion. If the density variation is not large, one may treat the density as constant in the unsteady and convection terms. This hypothesis is called the Boussinesq approximation. It is common to assume that the density varies linearly with temperature. We can express the density term as:

$$(\rho - \rho_0) = -\rho_0 g_i \beta (T - T_0) \qquad (2.8)$$

where β is the coefficient of volumetric expansion. While this approximation introduces errors of the order of 1% for small temperature differences (e.g. 2°C for water, 15°C for air). Otherwise the error may be more substantial and it is possible to have even a qualitatively wrong solution.

2.2. Introduction to Numerical Methods

The most important key ideas for numerical solution methods for partial differential equations were established more than a century ago, but their practical use started since the 1950s, when computers appeared. Although first computers were extremely expensive and with relatively low performance, the performance-to-cost ratio has increased dramatically and even now it doesn’t show sign of slowing down. With the constant evolution of computers, interest in numerical techniques increased dramatically. The field known as computational fluid dynamics (CFD) is responsible to the solution of the equations of fluid mechanics on computers.

How it works?

To obtain an approximate solution numerically, we have to use a discretization method which approximates the differential equations by a system of algebraic equations, which can then be solved on a computer. As the approximations are applied to small domains in space and/or time, the numerical solution provides results at discrete locations in space and time. The accuracy of numerical solutions is dependent on the quality of discretization used. However we are unable to obtain accurate solutions for all flows and so we have to determine what we can produce and learn to analyze and judge the results. First, we have to consider that numerical results all always approximate. Here we have a list of reasons why we have differences between computed results and “reality”:

- The differential equations may contain approximations or idealizations;
- Approximations are made in the discretization process;
- In solving the discretized equations, iterative methods are used. Unless they are run for a very long time, the exact solution of the discretized equations is not produced.

In principle, if the governing equations are known accurately, we can achieve a solution of any desired accuracy. The fact is that for many phenomena the exact equations are either not available or numerical solution is not feasible. This oblige us to introduce models and the validation of these models relies on experimental data. Sometimes models are used even the exact treatment is possible in order to reduce costs. Using more accurate interpolation or approximations, we can reduce discretization errors but usually this increases the time and cost of obtaining the solution. We must find a compromise, an optimum between accuracy and costs. Compromises are also needed in solving the discretized equations. Direct solvers could be more accurate but are seldom used, turning out to be too costly. Iterative methods are more common but we have to take in account errors stopping the process too early. It is very important to estimate errors, as erroneous solution visualizations can be easily confused with correct ones and sometimes we can even try to explain wrong behaviors with physical phenomena. So the first rule is to examine critically results before considering them believable.

2.2.1. Finite Volume Method

We focus only on this discretization approach as it is the one used in both OpenFOAM and Ansys Fuent. So we do not treat Finite Difference Method (FDM) and Finite Element Method (FEM).

The FVM method uses the integral form of the conservation equations. The solution domain is subdivided into a finite number of contiguous control volumes (CVs), and the equations are applied to each CV. At the centroid of each CV lie a computational node where the variable values are calculated. To express variable values at the CV surface in terms of nodal (CV-center) values interpolation is used. As a result, we obtain an algebraic equation for each CV, in which a number of neighbor nodal values appear. This method can be used with any type of grid and so it is suitable also for complex geometries. As the grid defines only the control volume boundaries, it does not need to be related to a coordinate system. The method is conservative (respect the conservation laws) by construction: surface integrals, representative of convective and diffusive fluxes, are the same for CVs sharing the boundary.

All terms that need to be approximated have physical meaning and this is the reason why the FVM is easy to program and very popular among engineers.

The main disadvantage of FVM compared to FDM schemes is the greater difficulty to develop methods of order higher than second in 3D. This is strictly related to the fact that FV approach requires 3 levels of approximation: interpolation, differentiation and integration.

Far from wanting to describe the method in details, we get into details just for what concern interpolation and differentiation practices. In fact in modern codes it is possible to select different options and it is up to the user to choose which one suits better for the case investigated.

2.2.1.1. Interpolation and Differentiation Practices

The approximations to the integrals require the values of variables at locations other than computational nodes (CV centers). To calculate the convective and diffusive fluxes, the value of Φ and its gradient normal to the cell face at one or more locations on the CV surface are needed. Volume integrals may also require these values. So they have to be expressed in terms of the nodal values by interpolation. A big number of possibilities are available, so we are going to mention just the most commonly used.

Upwind Interpolation Scheme

Upwind schemes denote a class of numerical discretization methods for solving hyperbolic PDE. They simulate in a solution sensitive way the direction of propagation of information in a flow field, using differencing biased in the direction determined by the sign of characteristic speeds. Approximating Φ_e by its value at the node upstream of "e" is equivalent to using backward- or forward-difference approximation for the first derivative (depending on the flow direction):

$$\Phi_e = \begin{cases} \Phi_P \ \ if\ (\boldsymbol{v}\cdot\boldsymbol{n})_e > 0 \\ \Phi_E \ \ if\ (\boldsymbol{v}\cdot\boldsymbol{n})_e < 0 \end{cases} \tag{2.9}$$

This is the only approximation that unconditionally satisfies the boundedness criterion and so it will never yields oscillatory solutions. To achieve this, it is numerically diffusive. The numerical diffusion can lead to particularly serious type of error, especially in peculiar flow conditions (e.g. flow oblique to the grid). Diffusion smears out peaks and rapid variations in the variables and, since the rate of error reduction is only first order, very fine grids are required to get accurate solutions.

Gauss Linear Interpolation

It approximate the value at CV-face center using linear interpolation between the two nearest nodes. At location "e" on a Cartesian grid we have:

$$\Phi_e = \Phi_E\lambda_e + \Phi_P(1-\lambda_e) \tag{2.10}$$

where λ_e is the linear interpolation factor and is defined as:

$$\lambda_e = \frac{x_e - x_P}{x_E - x_P} \tag{2.11}$$

Equation (*) is second-order accurate as can be easily shown by using Taylor series expansion. As with all approximations of order higher than one, this scheme may produce oscillatory solutions. Linear interpolation is the simplest second-order scheme and is the most widely used. When "e" is located midway between P and E (e.g. a uniform grid), the approximation is second-order accurate. When the grid is not uniform, the approximation is formally first-order accurate (Δx multiplied by the grid expansion factor), but the error reduction is similar to that of a second-order approximation even on non-uniform grids.

Gauss linearUpwind Interpolation

This scheme is the one used in the first of the two impinging jet cases simulated. Gauss linearUpwind interpolation fuses the upwind and linear schemes presented above, then it is second order accurate. Comparing the linear scheme with linearUpwind scheme, the simulation results more stable. It is possible to calculate nonphysical results (it happened to us) but this scheme does not diverge the solution in a way the standard linear scheme will do it.

MUSCL

The MUSCL scheme can provide highly accurate numerical solutions for a given system, even in cases where the solutions exhibit shocks, discontinuities or large gradients. The acronym stands for *Monotonic Upstream-Centered Scheme for Conservation Laws* and it was introduced in a paper by *van Leer.* The scheme presents second order spatial accuracy. The idea behind it is to replace the piecewise constant approximation of Godunov's scheme by reconstructed states, derived from cell-averaged states obtained from the previous time-step. For each cell, slope limited, reconstructed left and right states are obtained and used to calculate fluxes at the cell boundaries (edges). This scheme is known to be very stable and the algorithm gives only physical results.

Actually, with the term MUSCL we represent not only a scheme but a one-parameter family of schemes; it is the choice of this parameter that gives different schemes. The formula has been derived for a structured grid; it assumes equal spacing between grid points and requires local co-linearity of the points. This is achieved by an orthogonal structured grid. In other papers, they added a curvature factor, to take account of non co-linearity, and stretching factor, to consider non local grid spacing, in order to make it usable for a higher number of meshes.

Quadratic Upwind Interpolation (QUICK)

We can improve the approximation of the variable profile between P and E using a parabola rather than a straight line. To construct a parabola one more point is needed. Neglecting the full discussion, we can say that this quadratic interpolation scheme has a third-order truncation error on both uniform and non-uniform grids. Using this interpolation scheme with the mid-point approximation of the surface integral, we obtain an overall approximation accuracy of the second-order. Although the QUICK approximation is slightly more accurate than the linear one, both schemes converge in a second-order manner and it is hard to find big differences.

2.2.2. Turbulent Flows

The majority of the flows encountered in engineering practice is turbulent and requires a different treatment. The first consideration to do is that turbulent flows are highly unsteady and three-dimensional: instantaneous velocity fields fluctuates rapidly in all three spatial dimensions. As vortex stretching is one of the principal mechanisms by which the intensity of turbulence is increased, vorticity is way higher than in laminar case.

Turbulence improves stirring process and helps classical diffusion mechanism: in this case it is named turbulent diffusion. All these properties are important and the effects produced may or may not be desirable, depending on the application. Intense mixing is useful when we need heat transfer and it can be increased by orders of magnitude by turbulence. On the other hand, increased mixing of momentum results in increased frictional forces (about one order of magnitude depending on turbulence intensity).

Design-engineers need to be able to predict these effects in order to achieve the best performances possible.

One of the most used approaches to predicting turbulent flows is based on equations obtained by averaging the equations of motion over time, over a coordinate in which the mean flow does not vary. This approach leads to a set of equations called the Reynolds-averaged Navier-Stokes (RANS) equations.

Other two approaches used especially in CFD are LES and DNS. The first acronym stands for Large Eddy Simulation and solves the largest scale motions of the flow while approximating or modeling only the small scale motions; it can be considered as a compromise between RANS and DNS. In DNS (Direct Numerical Simulation) the Navier-Stokes equations are solved for all of the motions in a turbulent flow. Moving from RANS to DNS, the cost of the computation increases and it is also requested a finer grid. In the past DNS approach was not an option if not for easy cases because it was too much demanding from the computational point of view. With the advent of new HPC with huge quantities of RAM, it has become possible to run DNS with more ease but the time requested is not suitable for industrial applications yet, and DNS is relegated in the research field and it is used to validate less demanding models.

In the next paragraph, the RANS approach is presented, while we do not present LES and DNS as we did not deal these models.

2.2.2.1. RANS

In order to model the Reynolds Averaged Navier Stokes equation, all variable have to be splitted into a time-averaged part and a fluctuating part:

$$\varphi = \bar{\varphi} + \varphi' \tag{2.12}$$

The time-averaged part is calculated as $\bar{\varphi} = \int_T \varphi(x,t)\,dt$. For compressible flows often another form of decomposition is done using Favre-averaging.
Here variables are decomposed as:

$$\theta = \tilde{\theta} + \theta'' \tag{2.13}$$

where $\tilde{\theta} = \overline{\rho\theta}/\bar{\rho}$. Thus θ'' includes not only the turbulent fluctuations but also the density fluctuations. After Favre averaging the velocity and energy and performing a standard time-averaging for ρ and p, we derive the following equations:

$$\frac{\partial\bar{\rho}}{\partial t} + \frac{\partial\bar{\rho}\tilde{u}_i}{\partial x_i} = 0 \tag{2.14}$$

$$\left(\frac{\partial\bar{\rho}\tilde{u}_i}{\partial t} + \frac{\partial\bar{\rho}\tilde{u}_j\tilde{u}_i}{\partial x_j}\right) = -\frac{\partial\bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\overline{\tau_{\imath\jmath}} + \overline{\rho u_i'' u_j''}\right) \tag{2.15}$$

Here $\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij}$ and $\overline{\tau_{\imath\jmath}} = \widetilde{\tau_{\imath\jmath}} + \overline{\tau_{\imath\jmath}''}$.

For the energy equation a similar transformation is done with the resulting Favre averaged equation,

$$\left(\frac{\partial\bar{\rho}\tilde{E}}{\partial t} + \frac{\partial\bar{\rho}\tilde{u}_j\tilde{E}}{\partial x_j}\right) = -\frac{\partial\bar{p}\tilde{u}_j}{\partial x_j} + \frac{\partial\overline{u_\imath\tau_{\jmath\imath}}}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\overline{q_\jmath}\right) - \frac{\partial}{\partial x_j}\overline{u_j'' p} - \frac{\partial}{\partial x_j}\overline{\rho u_j'' E''} \tag{2.16}$$

This is the decomposition made in compressible codes. For incompressible codes the Favre decomposition is not used and the RANS equations are derived from the standard time-averaging. For solving the RANS equations assumptions have to be made regarding the turbulent terms. In fact they do not form a closed set so this method requires the introduction of approximations, in other words a turbulence model.

2.2.2.2. Turbulence models

When modeling the governing equations with RANS, a closure equation (or more depending on the chosen model) is needed to solve the problem. There are several models, everyone with a different modeling approach.

Considering the RANS equations, the objective of the turbulence model is to compute Reynolds stresses. In this essay only the Spalart-Allmaras and the *k-ω SST* turbulence models are presented, as they are the ones we used during our campaign of simulations. In particular *k-ω SST* model is one of the most suitable for the flow conditions investigated, blending the advantages of other models and avoiding their shortcomings.

Spalart-Allmaras

The Spalart-Allmaras is a one equation turbulence model, where the kinematic eddy viscosity is calculated through a transport equation and the lenght scale is found from an algebraic expression. This model provide a computationally cheap way of calculating the boundary layers in aerodynamics. In the Spalart-Allmaras model an eddy viscosity parameter is calculated, which is related to the eddy viscosity through:

$$\nu_t = \tilde{\nu} f_{v_1} \tag{2.17}$$

The Reynolds stresses are calculated using the following assumption:

$$-\overline{\rho u_i' u_j'} = \rho \tilde{\nu} f_{v_1} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \tag{2.18}$$

A transport equation is set up for $\tilde{\nu}$ to find the eddy viscosity

$$\frac{\partial \rho \tilde{\nu}}{\partial t} + \frac{\partial \rho \tilde{\nu} \bar{u}_k}{\partial x_k} =$$

$$\frac{1}{\sigma_\nu} \left[\frac{\partial}{\partial x_k} \left((\mu + \rho \tilde{\nu}) \frac{\partial \tilde{\nu}}{\partial x_k} \right) + C_{b_2} \rho \frac{\partial \tilde{\nu}}{\partial x_k} \frac{\partial \tilde{\nu}}{\partial x_k} \right] + C_{b_1} \rho \tilde{\nu} \tilde{\Omega} - C_{\omega_1} \rho \left(\frac{\tilde{\nu}}{\kappa y} \right)^2 f_\omega \tag{2.19}$$

where $\tilde{\Omega} = \Omega + \frac{\tilde{\nu}}{(\kappa y)^2} f_{v_2}$, and Ω is the mean vorticity. The wall functions are dependent on the following variables $f_{v_2} = f_{v_2}\left(\frac{\tilde{\nu}}{\nu}\right)$ and $f_\omega = f_\omega \left(\frac{\tilde{\nu}}{\tilde{\Omega} \kappa^2 y^2} \right)$. The turbulent lenght scale can be found from κy, where y is the distance from the wall.

The model constants are set as $\sigma_\nu = 2/3$, $\kappa = 0.4187$, $C_{b_1} = 0.1355$, $C_{b_2} = 0.622$ and $C_{\omega_1} = 0.56203$. There are other constants in the wall functions formulation, but they are not reported here. These values are taken from the original formulation: folllowing version of the same turbulence model may differ. In particular Spalart himself proposed some modifications.

k-ω SST

Now we are going to see in detail this model, which is the one mainly adopted in this thesis.

It has been developed by Menter in 1993 to deal with the strong freestream sensitivity of the standard *κ-ω* turbulence model and improve the predictions of adverse pressure gradients and separation.

The *κ-ω* model is more accurate than the *κ-ε* in the near wall layers, and has therefore been successful for flows with moderate adverse pressure gradients, but fails for flows with pressure induced separation; moreover, it shows a strong sensitivity to the values of *ω* in the freestream outside the boundary layer. This drawback has largely prevented the *ω*-equation from replacing the *ε*-equation as the standard scale-equation in turbulence modelling, despite its superior performance in the near wall region.

This brought to the development of the zonal (BSL and SST) models.

The zonal formulations are based on blending functions, which ensure a proper selection of the *κ-ω* and *κ-ε* zones without user interaction. The main additional complexity in the formulation compared to standard models lies in the necessity to compute the distance from the wall, required in the blending functions. This is computed solving a Poisson equation and is therefore compatible with modern CFD codes.

Originally used for aeronautics applications, SST model has made its way into most industrial, commercial and many research codes. It has greatly benefited from the strength of the underlying turbulence models. In particular, the accurate and robust near wall formulation of the Wilcox model has substantially contributed to its industrial usefulness, going far beyond pure aerodynamics.

After the original formulation, there have been several areas of improvement; robustness optimization have brought the model to the same level of convergence as the standard *κ-ε* model with wall functions and the improved near-wall formulation has reduced the near wall grid resolution requirements.

A large number of model validation studies and applications can be found on the internet.

Now we can take a look of the equations.

The eddy viscosity is:

$$\mu_t = \rho k/\omega \tag{2.20}$$

and the Reynolds stresses are derived from the Boussinesq assumption.

$$\tau_{ij} = -\rho\overline{u_i' u_j'} = \mu_t\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) - \frac{2}{3}\rho k \delta_{ij} \tag{2.21}$$

The transport equation for *k* is

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial}{\partial x_i}(\rho k \bar{u}_i) = \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + 2\mu_t S_{ij} S_{ij} - \frac{2}{3}\rho k \frac{\partial \bar{u}_i}{\partial x_j}\delta_{ij} - \beta^* \rho k \omega \tag{2.22}$$

where

$$S_{ij} = \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \tag{2.23}$$

The *ω* is

$$\frac{\partial(\rho\omega)}{\partial t} + \frac{\partial}{\partial x_i}(\rho\omega\bar{u}_i) = \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_{\omega,1}}\right)\frac{\partial \omega}{\partial x_i}\right] +$$
$$+\gamma_2\left(2\rho S_{ij} S_{ij} - \frac{2}{3}\rho\omega\frac{\partial \bar{u}_i}{\partial x_j}\delta_{ij}\right) - \beta_2\rho\omega^2 + 2\frac{\rho}{\sigma_{\omega,2}\omega}\frac{\partial k}{\partial x_k}\frac{\partial \omega}{\partial x_k} \tag{2.24}$$

The model constants are $\sigma_k = 1.0$, $\sigma_{\omega,1} = 2.0$, $\sigma_{\omega,2} = 1.17$, γ_2=0.44, $\beta_2 = 0.083$ and $\beta^* = 0.09$. Blending functions are implemented to assure smooth transition between the *κ-ω* and the *κ-ε* model is done. The model is still evolving: developers are trying to extend its capabilities also in those applications where it does not suit properly. Check out Menter's papers for further information about this model and its blending functions.

2.3. OpenFOAM modeled equations

In OpenFOAM the governing equations are not solved as a coupled system of equations but the continuity, momentum and energy equations are all solved separately. In this section, a short description of equations and algorithms of the solvers used in this paper is provided.

2.3.1. simpleFoam

As seen, it is a steady-state incompressible solver. Due to the incompressibility the energy equation does not have to be solved, as there is no link between the momentum and continuity equations. Instead the momentum equation is solved and a pressure correction is performed, thus a physical pressure is not solved but rather pressure differences are found.

If the divergence is taken of the momentum equation and the continuity equation is used to expand the term $\frac{\partial}{\partial x_k}\left(\frac{\partial}{\partial t}\rho u_k\right)$, a Poisson equation for the pressure is found.

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left[\frac{\partial}{\partial x_j}\left(\rho u_i u_j - \tau_{ij}\right)\right] + \frac{\partial(\rho g_i)}{\partial x_i} + \frac{\partial^2 \rho}{\partial t^2} \qquad (2.25)$$

In the case of constant density and viscosity, it reduces to:

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left[\frac{\partial}{\partial x_j}\left(\rho u_i u_j\right)\right] \qquad (2.25)$$

The SIMPLE algorithm, which the simpleFoam solver is based upon, is solving the momentum equation and the Poisson pressure equation. As OpenFOAM utilizes collocated grid, Rhie-Chow interpolation is used for the pressure-velocity coupling.

2.3.2. pimpleFoam

This is a transient solver for incompressible fluids and is a merged solver consisting of the PISO and SIMPLE algorithms. The solver is using SIMPLE algorithm in an inner loop with an outer PISO loop. The SIMPLE algorithm neglects the velocity corrections as they are unknown, which results in slow convergence. In the PISO loop these are taken into consideration, and thus PIMPLE scheme should achieve a faster convergence.

3. OpenFOAM

3.1. Historical outlook

OpenFOAM (Field Operation And Manipulation) is an open-source collection of libraries developed for simulating continuum mechanics. OpenFOAM (originally, FOAM) was created by Henry Weller from the late 1980s at Imperial College, London, to develop a more powerful and flexible general simulation platform than FORTRAN, the standard one at the time. This led to the choice of C++ as programming language, due to its modularity and object oriented features. In 2004, Henry Weller, Chris Greenshields and Mattijs Janssens founded OpenCFD Ltd to develop and release OpenFOAM. Then OpenCFD was acquired by Silicon Graphics International (SGI) and at the same time, the copyright was transferred to the OpenFOAM Foundation. This not-for-profit organization manages and distributes OpenFOAM to the general public. In 2012 ESI Group acquired OpenCFD Ltd from SGI. In 2014, Weller and Greenshields left ESI Group and continue the development and management of OpenFOAM, on behalf of the OpenFOAM Foundation, at CFD Direct.

Weller maintains the official version of OpenFOAM which has closed-source development and periodically releases open-source version updates. Hrvoje Jasak, another developer, mantains a fork of OpenFOAM called Extend-Project (also known as OpenFOAM-extend) which includes additional modeling capability not found in the official version and is centered on community-driven development.

3.2. Fundamentals

Based on finite volume approach to discretize and solve the governing equation, it is primarily focused on CFD but it can deal as well with computational solid mechanics, computational aero-acoustics, computational electromagnetics, etc.

It can be used in massively parallel computers without the need to pay for separate licenses.

While it comes with its own mesh generation tools, mesh generated using many of the major mesh generators can be converted to OpenFOAM format and this makes it very versatile.

Written in C++, an object oriented programming code, it is designed to be a flexible and programmable environment for numerical simulations by using a high level programming language that is direct representation of the equations being solved; thanks to this, it is very easy to read, understand, modify or add further capabilities.

An object-oriented program (OOP) typically makes use of the following concepts: Classes and Objects, Member Functions, Private vs. Public Members. These concepts allow efficient organization and design of programs and are essential when dealing with very complex systems.

C++ language was developed in the early 1980's by Stroustrup with the aim of combining the flexibility and efficiency of C systems programming with OOP paradigms. It simply makes convenient to use any standard programming which fits the task at hand, without forcing any one particular style on all users.

This is what makes C++ such a powerful scientific programming language.

With high level programming new solvers can be implemented with relative ease. Users do not need a deep knowledge of object-orientation and C++ programming to a solver but should know the principles behind object-orientation and classes, and to have a basic knowledge of some C++ code syntax; an understanding of the underlying equations, models and solution method and algorithms is far more important. Furthermore, it is possible to modify existing solvers or use them as the starting point for new ones. OpenFOAM design itself encourages code and libraries re-use. Low level functions, e.g. mesh handling, parallelization, etc., are not needed for special coding at the top level, as they are natively implemented. The ease which whom we can setup new solvers allows researchers to tackle nonstandard continuum mechanics problems, looking beyond the capabilities of commercial CFD software.

Another unique feature compared to other commercial code is the access to complete source: we can always check what we are going to do just having a look at the source code. So no secret modeling tricks nor cutting corners are hidden in the code. This is not possible with all the other commercial software: all the functionalities are available through a Graphical User Interface (GUI), which does not allow to see the code.

This makes OpenFOAM ideal for research and development.

OpenFOAM doesn't come with a GUI, even if some third-party solutions are available (e.g. HELYX-OS). Everything is controlled through the terminal and this can create some troubles, especially if the user is not familiar with the LINUX environment. For post-processing and flow visualization, plugins to ParaView are readily available; ParaView is an open source, multi-platform data analysis and visualization application with user-friendly graphical interface and can be opened typing the command '*paraFoam*' in the terminal.

So the main problem while starting to use OpenFOAM is to gather the necessary knowledge to operate with the pre-existing functionalities, and this can require a lot of time. Fortunately both community-based and professional support are available.

3.3. OpenFOAM Structure

3.3.1. Case Structure

In the Fig. 3.1 is shown the basic directory structure for an OpenFOAM case, which contains the minimum set of files required to run an application:

'constant' directory → it contains a full description of the case mesh in a subdirectory 'polyMesh' and files specifying physical properties for the application concerned (e.g. 'transportProperties' or 'thermophysycalProperties').

'0' folder → it represents the time directory of the initial time (time "zero") and contains initial values and boundary conditions that the user must specify in order to define the problem, while other 'time' folders contain results written to file by OpenFOAM. Note that the OpenFOAM fields must always be initialized, even when the solution does not strictly require it, as in steady-state problems.

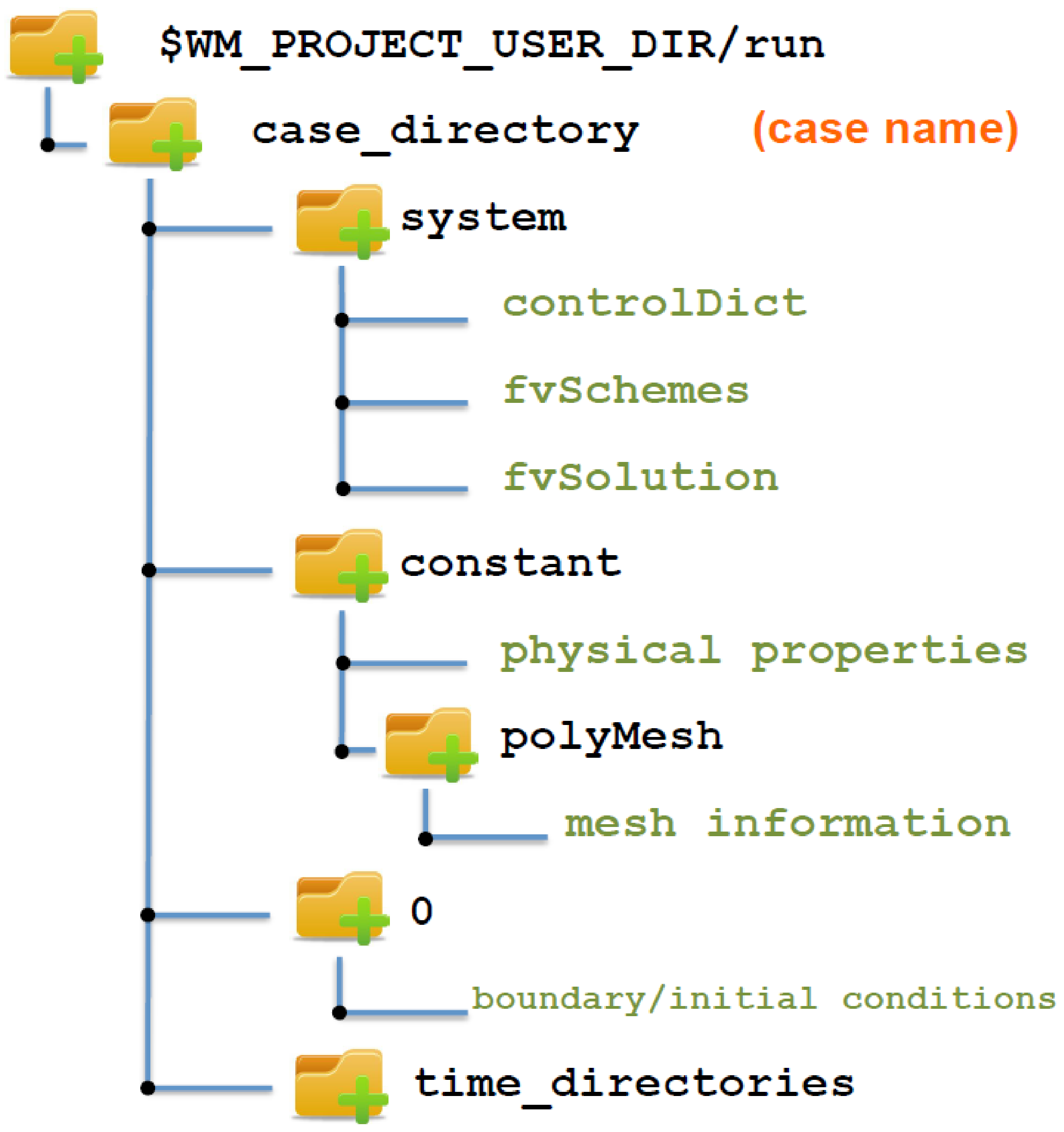


Fig. 3.1: Directory structure of an OpenFOAM case

‘system’ directory → for setting parameters associated with the solution procedure itself; it contains run-time control and solver numeric. In this directory we find at least the followings three files:

controlDict where run control parameters are set including start/end time, time step and parameters for data output

fvSchemes where we set discretization schemes used in the solution.

fvSolution where we set equation solvers, tolerances and other algorithm controls for the run.

It is common to find also other files, in which are written the settings of optional dictionaries utilized in the simulation.

‘time’ directories → containing files of data for particular fields. The name of each directory is based on the simulated time at which the data is written. The number of time directories written can be chosed changing the *writeInterval* value in the *controlDict* file located in the ‘system’ folder. The smaller is the writeInterval value, the higher will be the number of the time step saved. Therefore we can run out of memory choosing a too small value; usually it is good practice to run the first iterations with small write intervals, being quite easy to have some issues at the very beginning of the simulation, and then to increase the interval as soon as we see that the solver is running properly. If for instance the solver blows up after few iterations (quite easy if something is not set properly), smaller write intervals allow the user to see in detail what happened just before the error, simplifying the debugging.

In addition to these essential folders, we can also find other files, such as log files, post-processing files, etc. depending on the utilities we use while running the simulation. As the nature of these files can be very different, we will not see them in depth.

3.3.2. Standard Solvers

Here a list of standard solvers is described; not all the solvers implemented in OpenFOAM are listed, but only the ones used during our simulations and some of the most known.

‘Basic’ CFD codes

- potentialFoam solves for the velocity potential from which the flux –field is obtained and velocity field by reconstructing the flux. It is usually used to initialize the flow (some iterations) before the specific solver for the case studied.

Incompressible

- icoFoam: transient solver for incompressible, laminar flow of Newtonian fluids.
- pisoFoam: transient solver for incompressible flow.
- simpleFoam: steady-state solver for incompressible, turbulent flow.
- pimpleFoam large time-step transient solver for incompressible flow using the PIMPLE (merged PISO-SIMPLE) algorithm.

Compressible

- rhoPimpleFoam: transient solver for laminar or turbulent flow of compressible fluids for HVAC and similar applications.
- rhoSimplecFoam: steady-state SIMPLEC solver for laminar or turbulent RANS flow of compressible fluids.
- rhoSimpleFoam: steady-state SIMPLE solver for laminar or turbulent RANS flow of compressible fluids.
- sonicFoam: transient solver for trans-sonic/supersonic, laminar or turbulent flow of a compressible gas.

Heat transfer and buoyancy-driven flows

- buoyantPimpleFoam: transient solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer. There is also the version with the Boussinesq approximation (buoyantBoussinesqPimpleFoam).
- buoyantSimpleFoam: steady-state solver for buoyant, turbulent flow of compressible fluids, including radiation, for ventilation and heat-transfer. Also for this solver there is a version with the Boussinesq approximation (buoyantBoussinesqSimpleFoam).
- chtMultiRegionFoam: it is a combination of *heatConductionFoam* and *buoyantFoam* for conjugate heat transfer between a solid region and a fluid region.
  There is a steady-state version called chtMultiRegionSimpleFoam.

Other solvers.

The official release includes solvers for the following topics:

- Multiphase flow
- Combustion
- Direct Numerical Simulation (DNS)
- Particle-tracking flows
- Molecular dynamics methods
- Direct simulation Monte Carlo methods
- Electromagnetics
- Stress analysis of solids
- Finance

OpenFOAM includes many multiphase flow solvers covering a wide range of applications; due to the fact that we did not deal with multiphase flows, we are not going to describe them in details. The same applies for Combustion, DNS and all the other families of solvers.

It is not always easy to gather information on the solvers implemented, no matter which is the field of application. We can always check the code but sometimes it requires time we do not have. It is faster to find information in specialized forum, where it is likely that a related discussion is already open, especially if the solver is commonly used.

Anyway, refer to the User's Guide for further information.

Other than the standard solvers, there are a big number of custom ones; sometimes they are very similar to existing solvers but add some feature needed by the user. In other cases, they are completely different and solve some problems that are impossible to be investigated with the existing solvers; as fluid dynamics studies are becoming more and more specialized and focused on particular cases never tackled before, it is easy to understand why new solvers are appearing with a certain frequency. Most useful custom solvers are usually natively implemented in the following official releases but, before, a validation process has to be done.

3.3.3. Standard Utilities

Here we are going to describe the most useful utilities included in OpenFOAM and the ones we used during the simulations as well.

Pre-processing

- changeDictionary: utility to change dictionary entries, e.g. can be used to change the patch type in the field and polyMesh/boundary files.
- mapFields: maps volume fields from one mesh to another, reading and interpolating all fields present in the time directory for both cases. Parallel and non-parallel cases are handled without the need to reconstruct them first.
- setFields: sets values on a selected set of cells/patchfaces through a dictionary.

Mesh generation

- blockMesh: a multi-block mesh generator.
- extrudeMesh: extrude mesh from existing patch (by default outwards facing normals; optional flips faces) or from patch read from file.
- snappyHexMesh: automatic split hex mesher. Refines and snaps to surface.

Mesh conversion

- fluentMeshToFoam: coverts a Fluent mesh to OpenFOAM format including multiple region and region boundary handling.
- foamMeshToFluent: writes out the OpenFOAM mesh in Fluent mesh format.

Mesh manipulation

- checkMesh: checks validity of a mesh.
- createBaffles: makes internal faces into boundary faces. Does not duplicate points, unlike (in order to do that there is a different command '*mergeOrSplitBaffles*'.
- createPatch: utility to create patches out of selected boundary faces. Faces come either from existing patches or from a *faceSet.*
- orientFaceZone: corrects orientation of *faceZone.*
- transformPoints: tranforms the mesh points in the polyMesh directory according to the translate, rotate and scale options.

Other mesh tools

- modifyMesh: manipulates mesh elements.
- refinementLevel: tries to figure out what the refinement level is on refined cartesian meshes. Run before snapping.
- collapseEdges: collapses short edges and combines edges that are in line.

Post-processing data converters

- foamDataToFluent: translates OpenFOAM data to Fluent format.
- foamToEnsight: translates OpenFOAM data to Ensight format.
- foamToTecplot360: Tecplot binary file format writer.
- foamToVTK: Legacy VTK file format writer.
- smapToFoam: translates a STAR-CD *SMAP* data file into OpenFOAM field format.

Post-processing velocity fields

- Co: calculates and writes the Courant number obtained from field *phi* as a *volScalarField.*
- Mach: calculates and optionally writes the local Mach number from the velocity field *U* at each time.
- Pe: calculates the Peclet number *Pe* from the flux *phi* and writes the maximum value, the *surfaceScalarField* Pef and *volScalarField* Pe.
- vorticity: calculates and writes the vorticity of velocity field *U.*

Post-processing at walls

- wallGradU: calculates and writes the gradient of *U* at the wall.
- wallHeatFlux: calculates and writes the heat flux for all patches as the boundary field of a *volScalarField* and also prints the integrated flux for all wall patches.
- wallShearStress: calculates and reports the turbulent wall shear stress for all patches, for the specified times.
- yPlus: calculates and reports y^+ for the near-wall cells of all wall patches, for the specified times for laminar, LES and RAS.

Post-processing turbulence

- createTurbulenceFields creates a full set of turbulence fields.
- R: calculates and writes the Reynolds stress *R* for the current time step.

Post-processing patch data

- patchAverage: calculates the average of the specified field over the specified patch.
- patchIntegrate: calculates the integral of the specified field over the specified patch.

Sampling post-processing

- probeLocations: samples one (or more) specified fields in one (or more) specified locations.
- sample: sample field data with a choice of interpolation schemes, sampling options and write formats.

Parallel processing

- decomposePar: automatically decomposes a mesh and fields of a case for parallel execution of OpenFOAM.
- reconstructPar: reconstructs fields of a case that is decomposed for parallel execution of OpenFOAM.
- reconstructParMesh: reconstructs a mesh using geometric information only.
- redistributePar: redistributes existing decomposed mesh and fields according to the current settings in the decomposeParDict file.

Some of these utilities need a file in the 'system' directory (e.g. extrudeMeshDict), where we are supposed to setup their options; others do not need this file but we have to specify in some fields after the command itself in the terminal.

3.3.4. Turbulence models

Any solver that includes turbulence modelling reads the *turbulenceProperties* dictionary, included in the '*constant*' folder. Within that file is the *simulationType* keyword that controls the type of turbulence modelling to be used, either:

- *laminar:* uses no turbulence models.
- *RAS:* uses Reynolds-averaged stress (RAS) modelling.
- *LES:* uses large-eddy simulation (LES) modelling.

We used RAS modelling for our simulations, so we will see only turbulence models suitable for RAS. The most known and used are:

- laminar: no turbulence model implemented.
- kEpsilon: standard κ-ε model.
- kOmega: standard κ-ω model. Suitable only for incompressible fluids.
- kOmegaSST: κ-ω SST model.
- realizableKE: realizable κ-ε model.
- Spalart-Allmaras 1-eqn mixing-length model.

The turbulence models used have been described in the previous chapter.

4. Flat Plate

4.1. Introduction

In order to get used to the new software and to compare the solutions for both OpenFOAM and Fluent in a case in which the analytical solution is available, we started to investigate a flat plate. In fact for a steady 2D laminar simulation of a flat plate is available the Blasius solution of the boundary layer.

4.2. Mesh

We used three different meshes to evaluate the effect of the dimensions of the domain. In particular, two of them have been created with Ansys Mesher, while the last one is an IGG mesh. The last one differs from the first two meshes because it presents a portion of domain at the inlet without the wall, where the flow can develop before meeting the wall.
Our mesh is a structured mesh with:

- 50 divisions along x, constant spacing.
- 60 divisions along y, refined near the wall (Bias factor equal to 70).

According to the settigs above, it is composed by 3000 cells.
In Fig. 4.1, we can take a look at it.

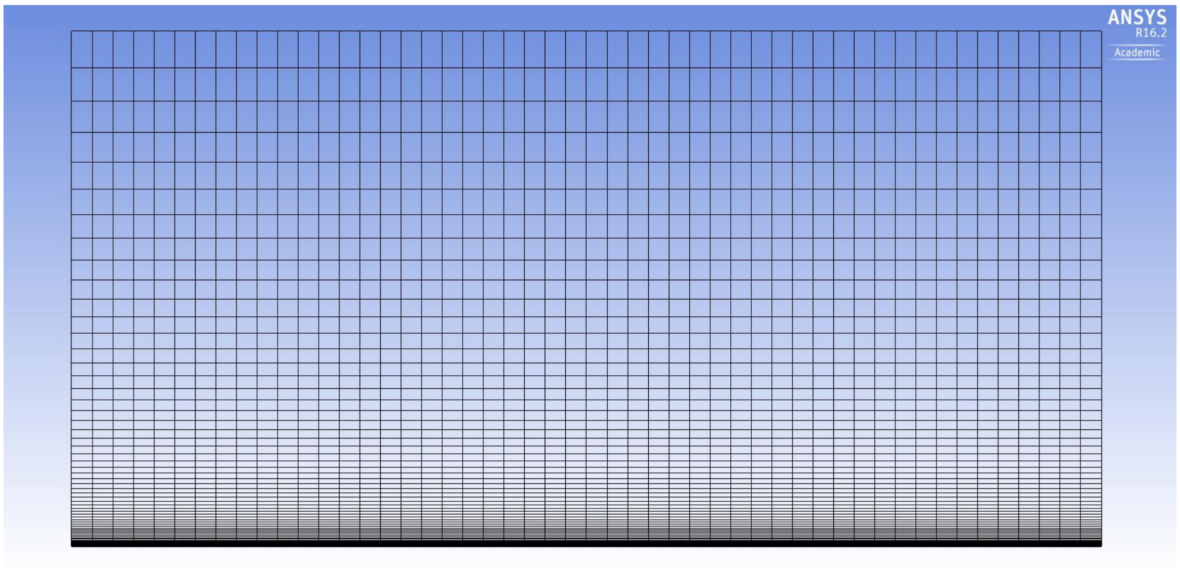


Fig. 4.1: Domain Mesh.

The edges have been named as in Fig 4.2 in order to assign boundary conditions.

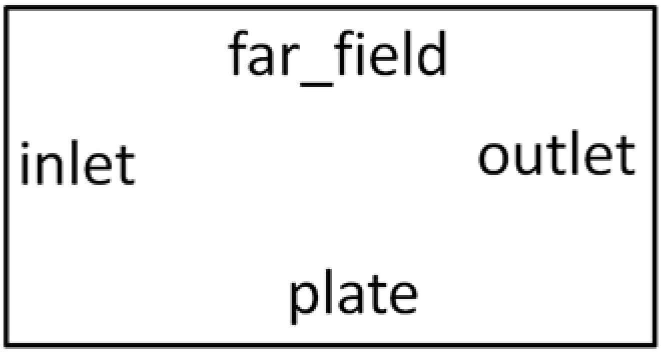


Fig. 4.2: Patch Names.

The mesh passed the quality check of both Fluent and OpenFOAM (the criteria are slightly different). In order to check the validity of my solution, we compute again on a finer mesh (100x120, 12000 elements). The numerical solution of the two velocity fields did not vary at all so we can say that problem is mesh converged; therefore, our first mesh is suitable for solving this problem.

4.3. Boundary Conditions

These are the settings chosed for our simulation:

- Re=10000
- U=1 m/s
- ρ=1 $\frac{kg}{m\cdot s}$
- μ=$10^{-4}\frac{kg}{m\cdot s}$
- L (length)=1 m
- pOutlet=0 Pa

These values are not necessarily physical. In fact, they have been chosen to obtain the desired Reynolds number, with $Re = \frac{\rho\cdot U\cdot L}{\mu}$.

As the height of our rectangular domain was not given, first we had to choose an adequate value. We decided to set the height of the boundary to ten times the boundary layer thickness expected. The boundary layer thickness has been found with the following approximation:

$$\frac{\delta 99\%}{x} = \frac{5}{\sqrt{Re_x}} \tag{4.1}$$

For x=L, it gives $\delta 99\%$=0.05m. Thus, we chose a height equal to 0.5m.

Fluent boundary settings:

- Inlet: *velocity-inlet* (velocity field given).
- Outlet: *pressure-outlet* (gauge pressure given).
- Plate: *wall* (standard wall boundary condition).
- Far_field: *symmetry* (being an “open” edge, the flow conditions must be the same on both sides).

In OpenFOAM, equivalent boundary conditions have been used. We tried also to use the b.c. *zeroGradient* for the Far_field, but we got exactly the same results.

4.4. Solvers setup

For incompressible flow, the energy equation is decoupled from continuity and momentum equations. We need to solve the energy equation only if we are interested in determining the temperature distribution. We did not deal with temperature, at least at first. So accordingly the *Energy equation* was set to off. We also left the *Viscous Model* set to *Laminar.*

The pressure momentum coupling in Ansys has been done with a SIMPLE scheme. I used the following spatial discretization:

- Gradient: Least Squares Cell Based
- Pressure: Standard
- Momentum: Second Order Upwind

In OpenFOAM, we utilized the solver *icoFoam*, which solves incompressible laminar Navier-Stokes equations.

4.5. Results

In Fig. 4.3, velocity profiles at the outlet are represented, together with the analytical solution by Blasius.

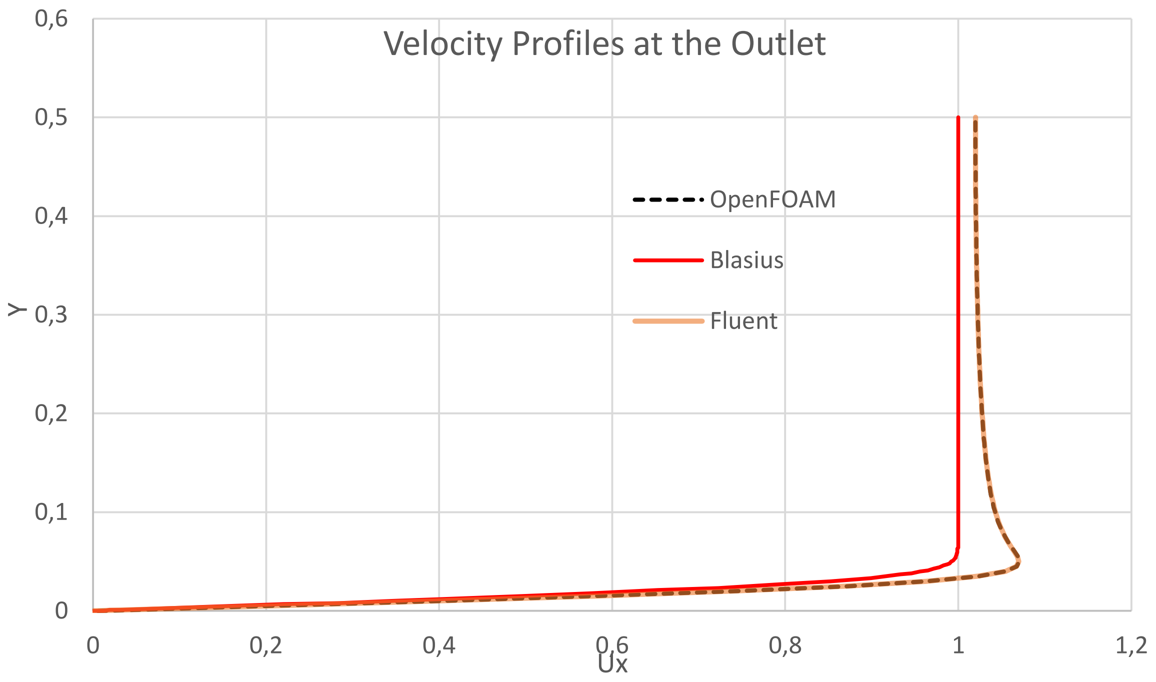


Fig. 4.3: Outlet Velocity Profiles.

Both the profiles have an overshoot in the velocity profile that we cannot find in classical boundary layer theory. This is a real effect that is missed by the boundary layer theory due to one of the assumptions it makes, namely that the outer flow is the inviscid flow past a flat plate. This is valid if the boundary layer is infinitely thin, which is true only when Reynolds number tends to infinity. At a finite Reynolds number as in our case, the boundary layer has a thickness, which displaces the outer flow and causes the overshoot seen in our solutions.
We ran also a similar simulation with only a different length of the plate to confirm what said before; the overshoot decreases along the plate, as the boundary layer develops. The length at which it is no more present is strictly related to the Reynolds number of the simulation.
In the boundary layer study and not only, it is common to represent velocity profiles normalized with the velocity at the inlet, in order to compare results for different setups. In this case, velocity profiles represented are equal to the normalized one, as the inlet velocity is equal to 1 m/s. Therefore, further plots are not needed.

Regarding pressure contours (Fig. 4.4, Fig 4.5), the behaviors are analogue. Also isopressure lines show a similar shape, and the differences can be justified with the color scale, which is slightly different, also because of the higher maximum pressure in Fluent at the plate. In any case, pressure distribution is well reproduced for both solvers.

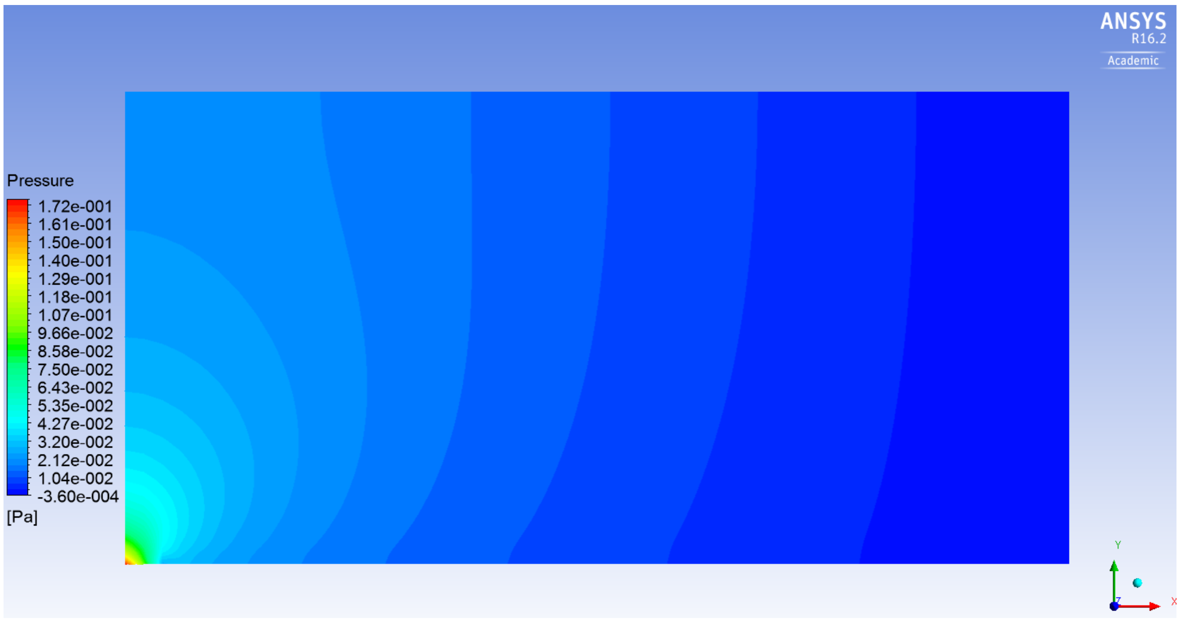


Fig. 4.4: Fluent pressure contour.

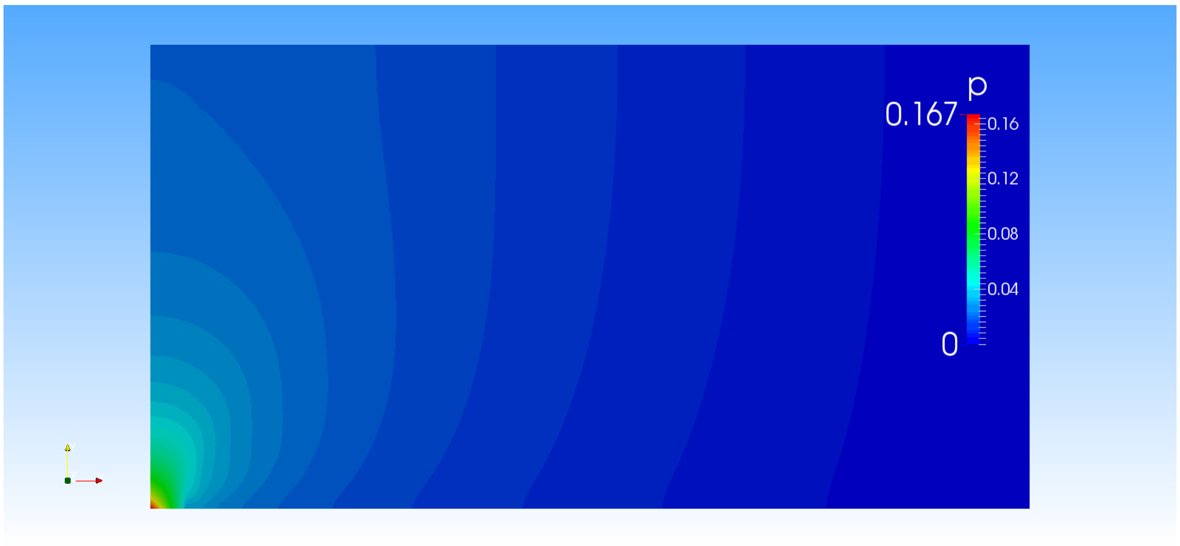


Fig. 4.5: OpenFOAM pressure contour.

Considering skin friction plots, they are almost equal for Fluent and OpenFOAM (Fig. 4.6). Compared to Blasius behavior, we have some differences in the first part of the plate; moving towards the Outlet, the difference becomes smaller and smaller.

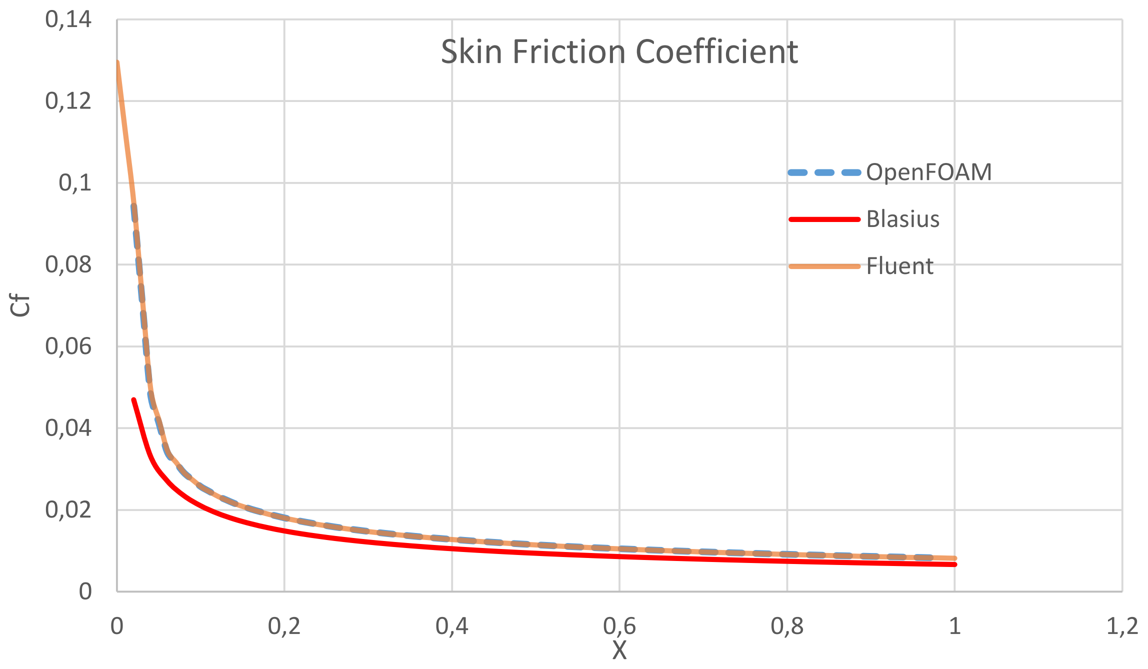


Fig. 4.6: Skin Friction Coefficient.

4.6. Thermal Flat Plate

After the first easy case, we decided to add the thermal solution. We considered a slightly different domain with *Tflow=300K* and *Tplate=350K*. In this case, the Energy equation has to be enabled in order to solve the temperature field.
The changes in the mesh consist in the modification of its size and in the addition of a portion of domain at the inlet before the plate. This shrewdness allows the flow to develop before the plate, reproducing a more natural flow condition; in fact with the plate starting from the inlet, at its leading edge we have an unphysical state: the no-slip condition forces the velocity at the wall to zero, while the inlet boundary condition consists in a fixed velocity other than zero. Though this discrepancy leads to 2 different boundary condition at the same location, we did not notice any strange behavior in the quantities measured in the previous case.
The new domain is 8m long and 3.75m high. The portion of developing flow before the plate is equal to 0.5m. The number of horizontal divisions is 400, while along the vertical we have 225 divisions. If in the horizontal direction we have no clustering, along the vertical we opted for an hyperbolic tangent distribution, in order to have a finer mesh at the wall and to solve the boundary layer in a proper way. Due to the extreme simplicity of the geometry, the quality of the mesh is very good. The minimum orthogonality is 90°, while the maximum aspect ratio and expansion ratio are respectively 21 and 1.01. This mesh has been entirely done with IGG mesher by NUMECA.

In Fig 4.7 we can have a look at the new domain.

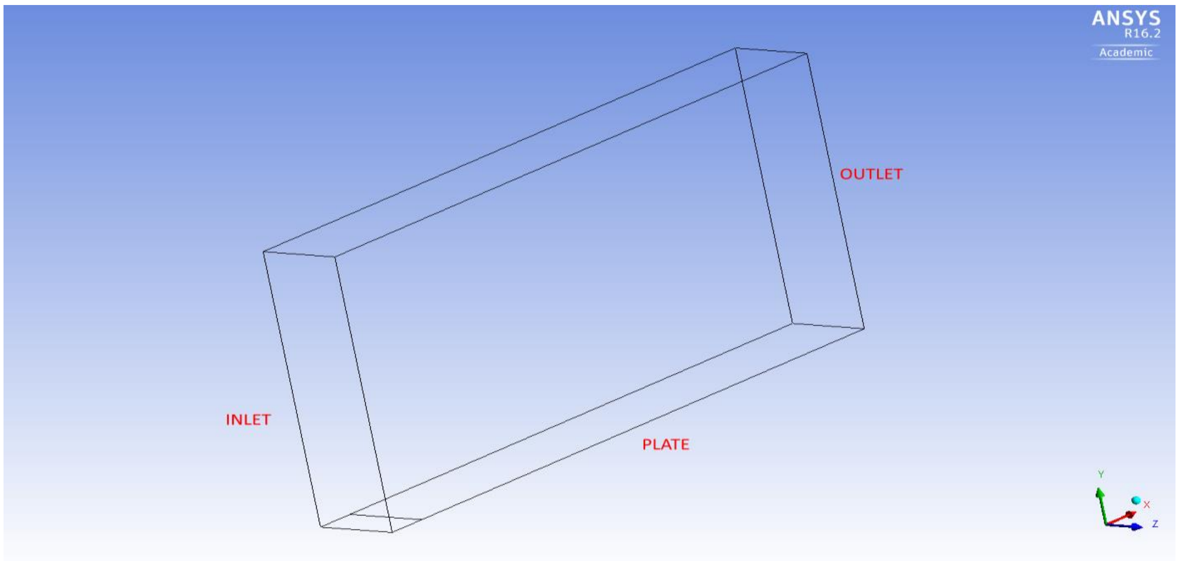


Fig. 4.7: Domain of the new mesh.

The boundary conditions are the same used for the previous case.
In order to compare results from Fluent and OpenFOAM in case of turbulence, we added the Spalart-Allmaras turbulence model with strain/vorticity production (it is the only one implemented in OpenFOAM). We maintained the default settings of the turbulence model for both the solver.

4.6.1 Results

Pressure contours are another time equal. While absolute values change, because of the different layout, what said about the behavior in the previous case is valid also for this solution. So, we decided to report only Fluent pressure contour for this case (Fig. 4.8).

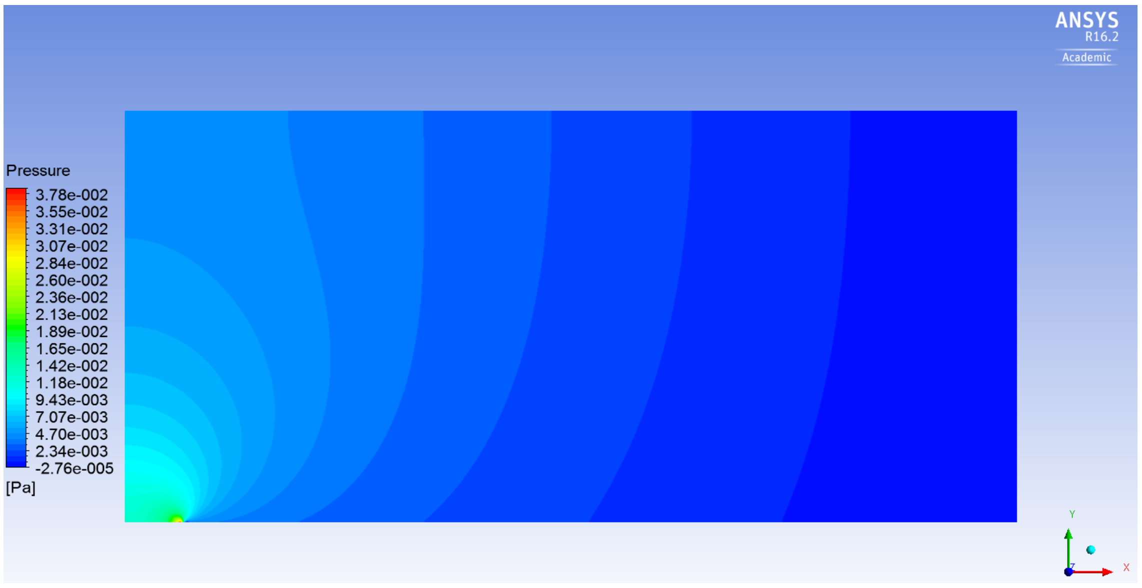


Fig. 4.8: Fluent pressure contour.

After pressure, we consider velocity field. First, let us compare velocity field after the developing zone. As we can see in Fig 4.9, the developing zone allows the flow to approach the plate in a more physical way without leading to differences. In Fig. 4.10, we can see velocity profiles at the outlet of our domain. Also at the outlet the behaviors are identical. So we can say that the evolution of the boundary layer is reproduced in the same way by the two solvers. We can also affirm that there are no big differences in the implementation of Spalart-Allmaras turbulence model, as they would have led to different shear layer behaviors.

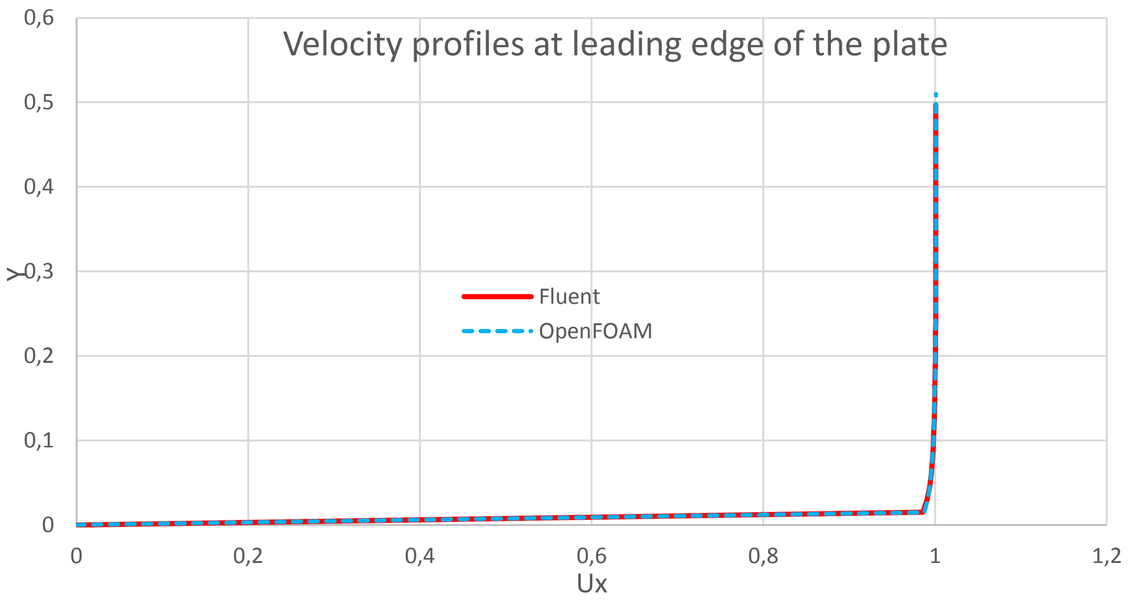


Fig. 4.9: Velocity profile at the leading edge of the plate.

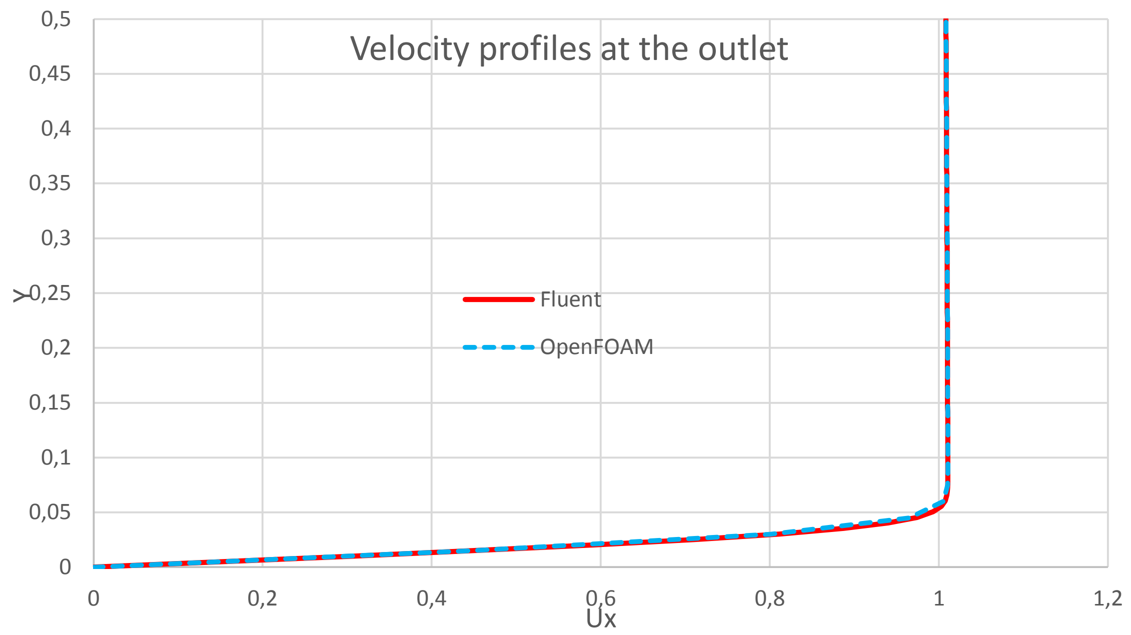


Fig. 4.10: Velocity profiles at the outlet.

In the thermal flat plate is no more observable the overshoot of the velocity field at the boundary of the shear layer. In fact, in this case the plate is 8 times longer and the boundary layer has all the space needed to develop. At the outlet, the outer flow had already been displaced by the growing boundary layer and so it could level the velocity profile, avoiding the overshoot noticed in the previous simulation.

Now let us turn to another topic: wall quantities. These quantities are very important in the analysis of a flow along a plate as we are more interested in what happens at the wall than in the other parts of the domain; in practical problems they are used to predict cooling performance and drag.

Starting with the heat flux, we have:

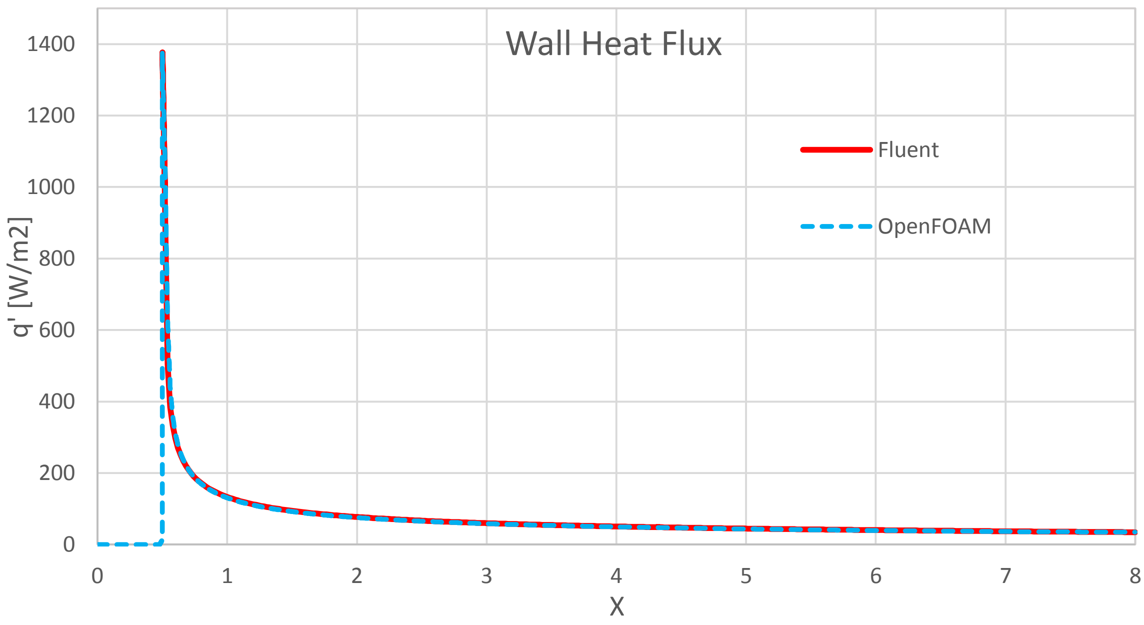


Fig. 4.11: Heat Flux at the plate.

As we can clearly see, there are no differences between the two behaviors. Both the profiles have been calculated automatically by the two software; in particular in OpenFOAM we used the utility *wallHeatFlux*.

Regarding the shear stresses at the plate, they are plotted in Fig. 4.12.
Also for this quantity, there are no major differences: the behavior is exactly the same and only Fluent presents an higher maximum. Still, the difference is a few percentage points. While in Fluent the values were calculated automatically, the OpenFOAM utility *wallShearStress* did not work properly and so we proceeded with the manual calculation. We use the following formula:

$$\tau_w = \mu \left(\frac{\partial U}{\partial y}\right)_{y=0} \tag{4.2}$$

where μ is the dynamic viscosity of the fluid.
To obtain the gradient of U along y at the wall, we opted for the utility *wallGradU*.

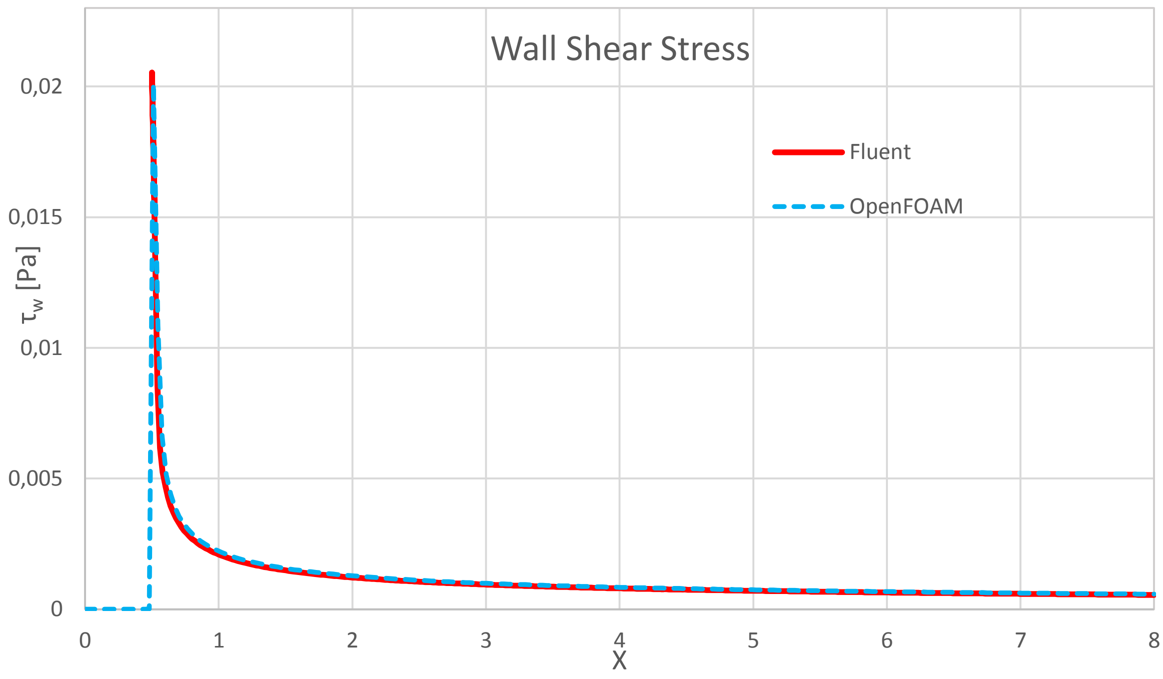


Fig. 4.12: Shear Stresses at the wall.

Then, we calculated the skin friction; for this quantity there are no utilities in OpenFOAM, so we calculated it manually as already done for the wall shear stress. The formula used is the following:

$$C_f = \frac{\tau_w}{\frac{1}{2}\rho U_\infty^2} \tag{4.3}$$

where τ_w is known, the density ρ is part of the solution of the simulation and U_∞ is the free-stream velocity (taken outside the boundary layer or at the inlet). In our case, $U_\infty = 1$ m/s.

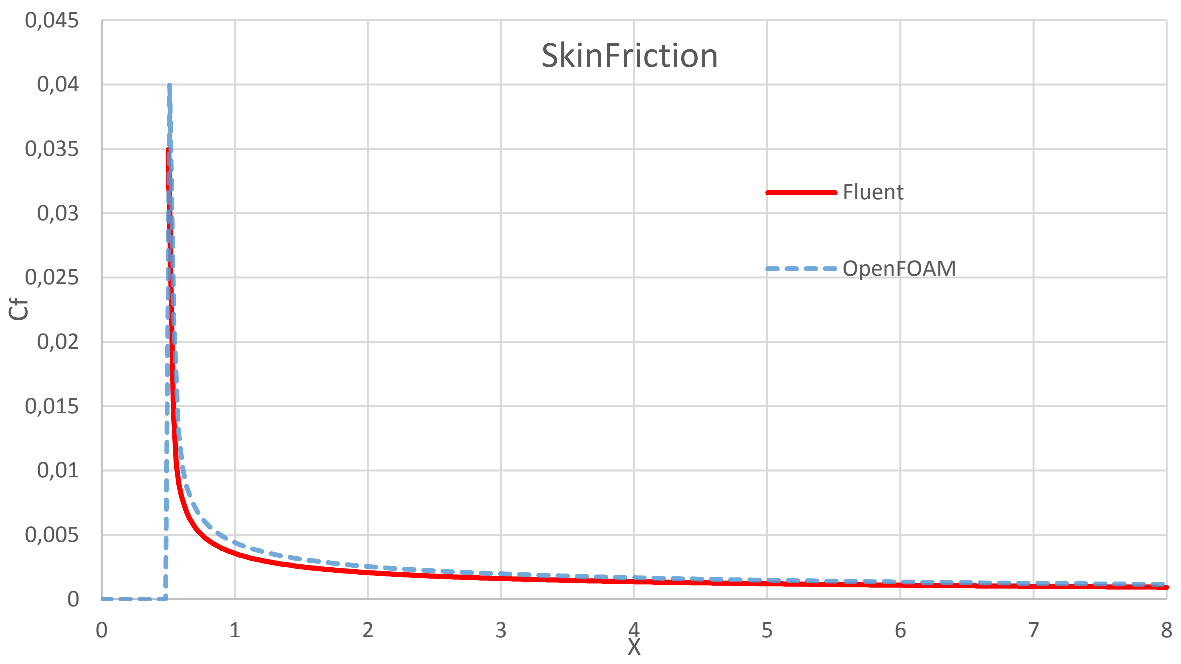


Fig. 4.13: Skin Friction profiles at the wall.

As attested by Fig. 4.13, for this quantity the difference is more marked.
Having obtained a slightly higher value of τ_w in Fluent, we thought that also C_f would have been higher than in OpenFOAM, because of the direct bond between the shear stress and the skin friction. This is not showed in the behaviors we have found.
Indeed, the OpenFOAM value is higher. It was hard to explain such a trend. Moreover, while in the previous plots the two curves were exactly superimposed, here there are some differences not only in the maximum value, but also in the shape of the profiles just after the maximum.
We tried to find out what could have been the reason of such a difference; first, we investigated the density field at the wall, but both showed very similar results. Also the free-stream velocity can only be the same, as we took the velocity at the inlet in our manual calculation of the skin friction, and all the boundary conditions are equivalent.
The difference could only lie in a different definition of skin friction. Gathering some infos about on the user's manual, it came out that Fluent does not use the actual density along the plate while calculating the skin friction but the reference one, that we can find in the *"Reference Values"* dialog in the software. Computing the skin friction in a similar way in OpenFOAM led to a much more similar behavior (Fig. 4.14), in respect also of the wall shear stress plot.
As it appears clear in the plots reported, the agreement between the two software is very good, also adding the solution of the Energy equation and a turbulence model. Minor differences can be explained with slightly different implementation of the numerical schemes in the two codes. In any case, it is not possible to look inside Fluent code and this did not allow us to compare the two codes and to find out any differences. It is not excluded that in Fluent are present some proprietary modifications of the standard implementation for the solvers used in our simulations.

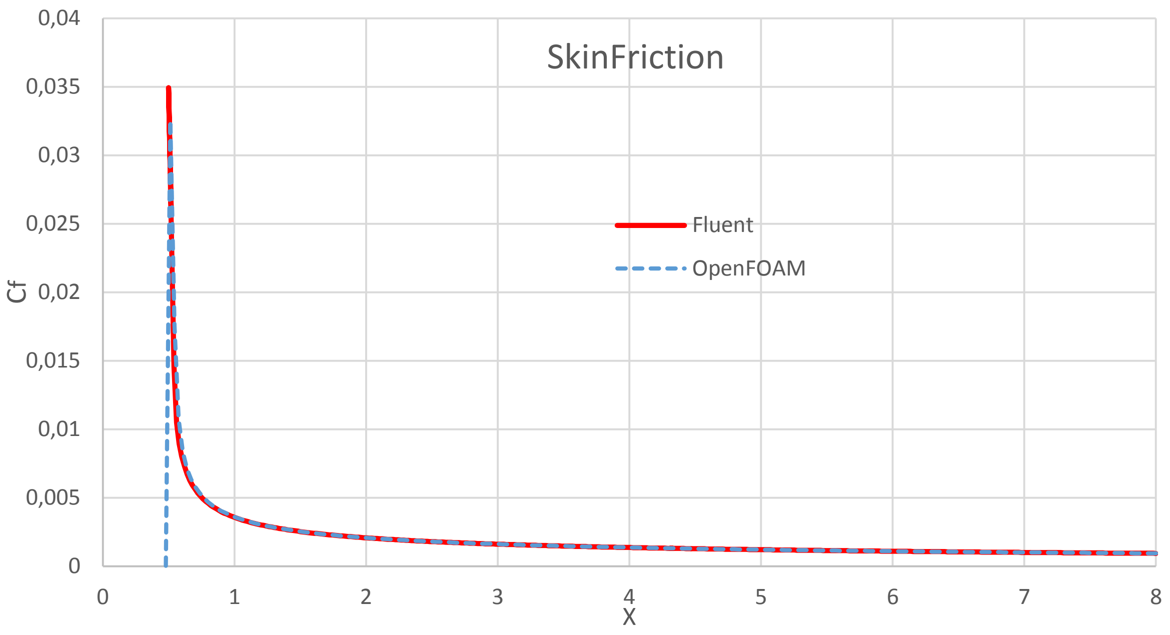


Fig. 4.14: New Skin Friction profile.

Looking at wall heat flux and skin friction plots, we can notice how the behaviors are similar. In fact our configuration satisfies all the hypothesis behind the Reynolds analogy: the evolving fluid is air (so *Prandtl* and *Schmidt* number are almost equal) and there is no form drag or pressure gradient; though our simulation is turbulent, very near the wall the fluid motion is smooth and laminar (laminar sublayer), and molecular conduction and shear are important. So, it is possible to find a relation between the heat transfer and the skin friction coefficient. This analogy comes in help when we want to obtain a first approximation for heat transfer knowing the shear stress.

5. Impinging Jet

Cooling processes are often used in industry and in most applications high heat transfer rates are requested. High heat transfer rates are especially needed where the energy consumption of the process is relatively high.

Jet impingement achieves locally high heat/mass transfer on an interested surface. For these reasons, the impinging jet cooling technique has been widely used in many industrial systems such as gas turbine cooling, rocket launcher cooling and high-density electrical equipment cooling in order to remove a large amount of heat. Its applications cover also materials processing and manifacturing. A new field of application is related to vertical/short take-off and landing jet devices: abrasion and heat transfer by impingement are studied as side effects (Fig 5.1).

Fig. 5.1: Harrier Jet Vertical take off.

5.1. Why to use impinging jets?

Considering turbine cooling applications, impinging jet flows may be used to cool several different sections of the engine such as the combustor case (combustor can walls), turbine case/liner, and the critical high-temperature turbine blades. The gas turbine compressor offers a steady flow of pressurized air at temperatures lower than those of the turbine and of the hot gases flowing around it.

The blades are cooled using pressurized bleed flow, typically available at 600°C. The bleed air must cool a turbine immersed in gas of 1400°C total temperature, which requires transfer coefficients in the range of 1000-3000 W/m^2. The ability to cool these components in high-temperature regions allows higher cycle temperature ratios and higher efficiency, improving fuel economy, and raising turbine power output per unit weight. Modern turbines have gas temperatures in the main turbine flow in excess of the temperature limits of the materials for the blades, meaning that the structural strength and component life are dependent upon effective cooling flow. Compressor bleed flow is commonly used to cool the turbine blades, at an acceptably low temperature. This air can be routed to a perforated internal wall to form impinging jets directed at the blade exterior wall. Upon exiting the blade, the air may combine with the impinging jet device with internal fins, smooth or roughened cooling passages, and effusion holes for film cooling. Spacing and location of jet and effusion holes are studied in order to concentrate the flow in the regions requiring the highest cooling effect. Though the use of bleed air carries a performance penalty, the small amount of flow extracted has a small influence on bleed air supply pressure and temperature. Moreover, turbofan engines provide cooler fan air at lower pressure ratios, which can be routed directly to passages within the turbine liner.

Compared to other heat or mass transfer arrangements that do not employ phase change, the jet impingement device offers efficient use of the fluid, and high transfer rates. If compared with conventional convection cooling by confined flow parallel to the cooled surface, jet impingement provides heat transfer coefficients that are up to three times higher at a given maximum flow speed, because the impingement boundary layers are much thinner, and often the spent flow after the impingement serves to turbulate the surrounding fluid. Given a required heat transfer coefficient, the flow required from an impinging jet device may be two orders of magnitude smaller than that required for a cooling approach using a free wall-parallel flow. Furthermore, it offers a compact hardware arrangement. If more uniform coverage over larger surfaces is needed, multiple jets may be used.

In any case, we have also to consider some disadvantages:

1. In case of moving targets with very uneven surfaces, the jet nozzles may have to be located too far from the surface. For jets starting at a large height above the target (over 20 jet nozzle diameters) the decay in kinetic energy of the jet as it travels to the surface may reduce average Nu by 20% or more.
2. The hardware changes necessary to implement an impinging jet device may degrade structural strength (one reason why impinging jet cooling is more easily applied to turbine stator blades than to rotor ones).
3. In static applications where very uniform surface heat or mass transfer is required, the resulting high density of the jet array and corresponding small jet height may be impratical to construct and implement, and at small spacing jet-to-jet interaction may degrade efficiency.

5.2. Impinging jet region

Jets can be classified as submerged if they discharge into an ambient fluid of similar physical properties (e.g. air in air) and unsubmerged if the properties of the two fluids are quite different (e.g. water in air).

The flow of a submerged impinging jet can be divided in several distinct regions, as shown in Fig 5.2.

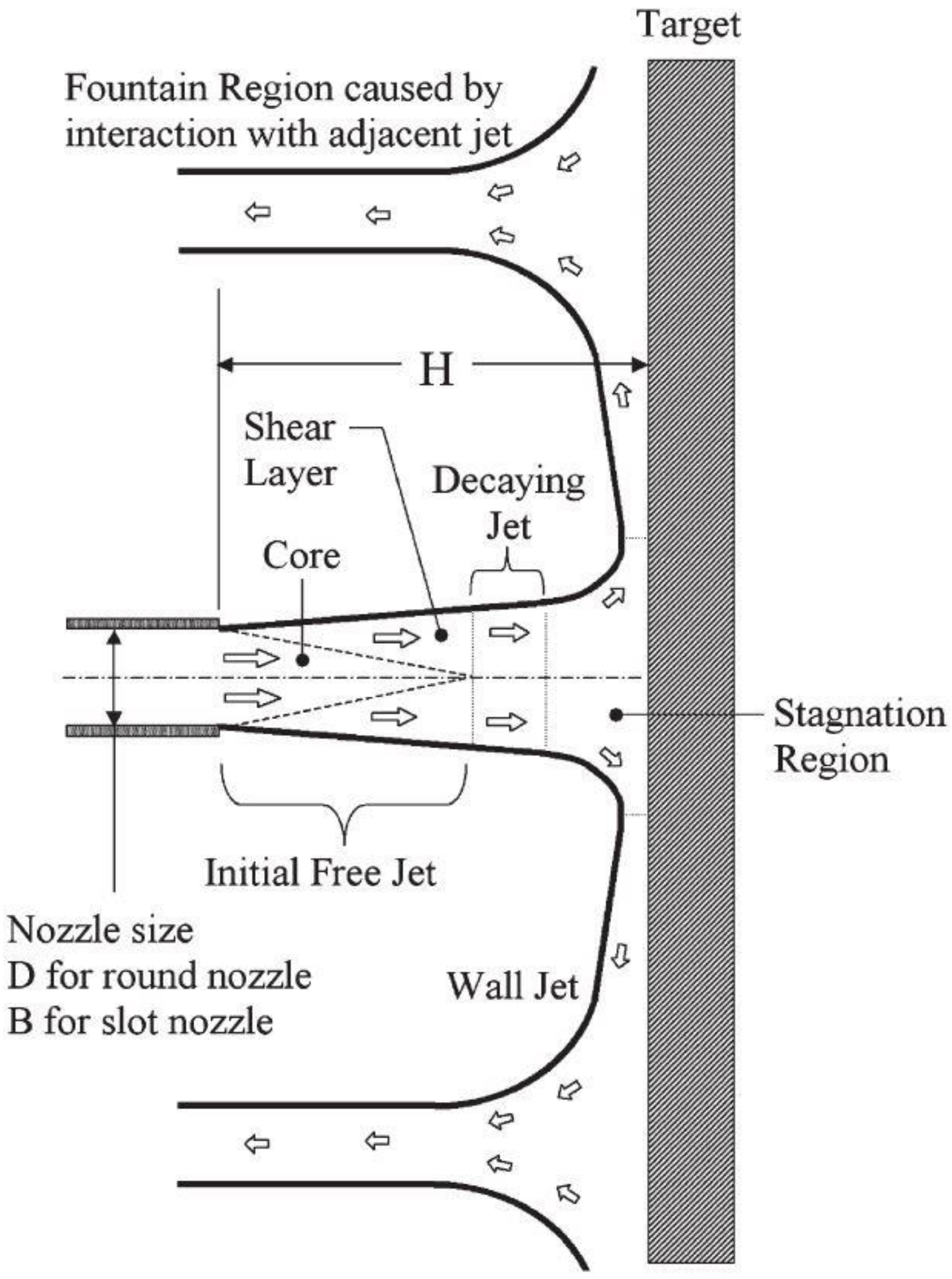


Fig. 5.2: The flow regions of an impinging jet.

The jet comes out from a nozzle or opening with a velocity and temperature profile and turbulence characteristics dependent upon the upstream flow. For a pipe-shaped nozzle, also called a tube nozzle or cylindrical nozzle, the flow develops into the parabolic velocity profile common to pipe flow plus a moderate amount of turbulence developed upstream. In contrast, a flow delivered by application of differential pressure across a thin, flat orifice will create an initial flow with a fairly flat velocity profile, less turbulence, and a downstream flow contraction (vena contracta). Typical jet nozzles designs use either a round jet with an axisymmetric flow profile or a slot jet, a long, thin jet with two-dimensional flow profile.

After it exits the nozzle, the emerging jet may pass through a region where it is sufficiently far from the impingement surface to behave as a free submerged jet. Here, the velocity gradients in the jet create a shearing at the edges of the jet which transfes momentum laterally outward, pulling additional fluid along with the jet and raising the jet mass flow (Fig. 5.3).

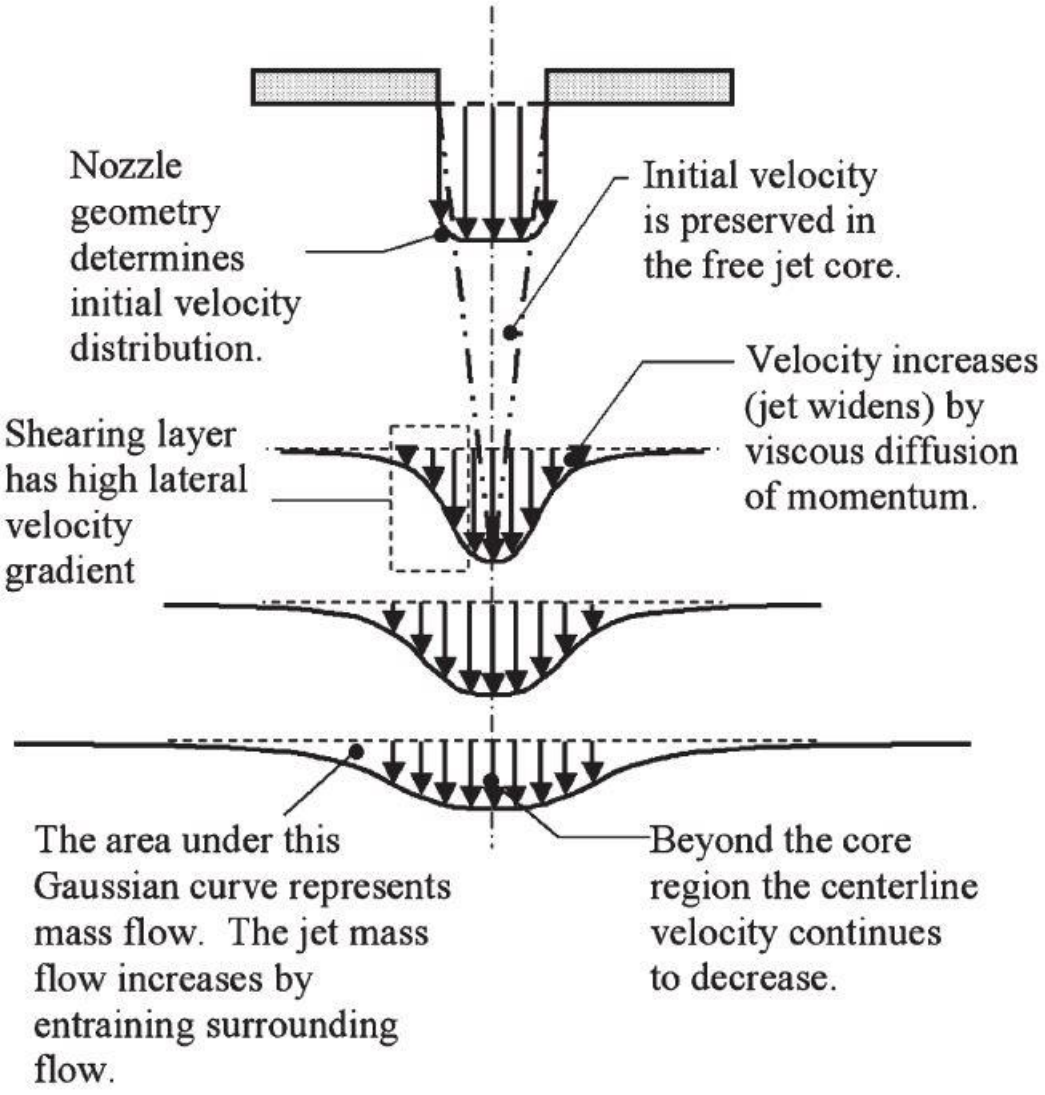

Nozzle geometry determines initial velocity distribution.
Initial velocity is preserved in the free jet core.
Velocity increases (jet widens) by viscous diffusion of momentum.
Shearing layer has high lateral velocity gradient
The area under this Gaussian curve represents mass flow. The jet mass flow increases by entraining surrounding flow.
Beyond the core region the centerline velocity continues to decrease.

Fig. 5.3: The flow field of a free submerged jet.

During this process, the jet loses energy and the velocity profile is widened in spatial extent and decreased in magnitude along the sides of the jet. Flow interior to the widening shear layer remains unaffected by this momentum transfer and forms a core region with a higher total pressure, though it may experience a drop in velocity and pressure decay resulting from velocity gradients present at the nozzle exit. A free jet zone may not exist if the nozzle lies within a distance of two diameters (2D) from the target. In such cases, the nozzle is close enough to the elevated static pressure in the stagnation region for this pressure to influence the flow immediately at the nozzle exit. In the decaying jet, the axial velocity component in the central part decreases, with the radial velocity profile resembling a Gaussian curve that becomes wider and shorter with distance from the nozzle outlet. In this region, the axial velocity and jet width vary linearly with axial position. Martin provided a collection of equations for predicting the velocity in the free jet and decaying jet regions based on low Reynolds number flow.

As the flow approaches the wall, it loses axial velocity and turns. This region is named stagnation region or deceleration region. The flow builds up a higher static pressure on and above the wall, transmitting the effect of the wall upstream. The nonuniform turning flow experiences high normal and shear stresses in the deceleration region, which greatly influence local transport properties. The resulting flow pattern stretches vortices in the flow and increase the turbulence. The stagnation region typically extends 1.2 nozzle diameters above the wall for round jets. After turning, the flow enters a wall jet region where the flow moves laterally outward parallel to the wall. The wall jet has a minimum thickness within 0.75-3 diameters from the jet axis, and then continually thickens moving farther away from the nozzle. This thickness may be evaluated by measuring the height at which wall-parallel flow speed drops to some fraction (e.g. 5%) of the maximum speed in the wall jet at that radial position. The boundary layer within the wall jet begins in the stagnation region, where it has a typical thickness of no more than 1% of the jet diameter. The wall jet has a shearing layer influenced by both the velocity gradient with the respect to the stationary fluid at the wall (no-slip condition) and the velocity gradient with respect to the fluid outside the wall jet. As the wall jet progresses, it entrains flow and grows in thickness, and its average flow speed decreases as the location of highest flow speed shifts progressively farther from the wall. Due to conservation momentum, the core of the wall jet may accelerate after the flow turns and as the wall boundary layer develops. For a round jet, mass conservation results in additional deceleration as the jet spreads radially outward.

5.3. Nondimensional Heat and Mass Transfer Coefficients

One of the most important parameters for evaluating heat transfer coefficients is the Nusselt number:

$$Nu = hD_h/k_c \tag{5.1}$$

where h is the convective heat transfer coefficient, D_h is the nozzle diameter and k_c is the thermal conductivity of the fluid.

Usually, h is defined as:

$$h = \frac{-k_c \frac{\partial T}{\partial \vec{n}}}{T_{0jet} - T_{wall}} \tag{5.2}$$

where $\frac{\partial T}{\partial \vec{n}}$ is the temperature gradient component normal to the wall.

The selection of Nusselt number to measure the heat transfer describes the physics in terms of fluid properties, making it independent of the target characteristics. The jet temperature used is the adiabatic wall temperature of the decelerated jet flow, a factor of greater importance at increasing Mach numbers. The non-dimensional recovery factor describes how much kinetic energy is transferred into and retained in thermal form as the jet slows down:

$$recovery\ factor = \frac{T_{wall} - T_{0jet}}{U_{jet}^2/2c_p} \tag{5.3}$$

This definition may introduce some complications in laboratory work, as the test surface is rarely held at a constant T, and more frequently held a constant heat flux. Experimental work (Goldenstein *et al.*) showed that the temperature recovery factor varies from 70% to 110% of the full theoretical recovery, with lowered values in the stagnation region of a low-*H/D* jet (*H/D*=2), and 100% elevated stagnation region recoveries for jet with *H/D*=6 and higher. The recovery comes closest to uniformity for intermediate spacings around *H/D*=5.

In a similar way, the nondimensional Sherwood number defines the rate of mass transfer:

$$S_h = k_i D / D_i \tag{5.4}$$

con:

$$\frac{D_i \frac{\partial C}{\partial \vec{n}}}{(C_{0jet} - C_{wall})} \tag{5.5}$$

where $\frac{\partial C}{\partial \vec{n}}$ is the mass concentration component normal to the wall.

With sufficiently low mass concentration of the species of interest, the spatial distribution of concentration will be similar to that one of the temperature. Studies of impinging air jets frequently use the nondimensional relation:

$$Nu/Sh = ({}^{Pr}/_{Sc})^{0.4} \tag{5.6}$$

Where Pr is Prandtl number (the ratio of fluid thermal diffusivity to viscosity) and Sc is Schmidt, defined as the ratio of momentum diffusivity and mass diffusivity:

$$Pr = \frac{\nu}{\alpha} = \frac{\mu c_p}{k} = \frac{viscous\ diffusion\ rate}{thermal\ diffusion\ rate} \tag{5.7}$$

$$Sc = \frac{\mu}{\rho D} = \frac{viscous\ diffusion\ rate}{molecular\ (mass)\ diffusion\ rate} \tag{5.8}$$

Other nondimensional parameters selected to describe the impinging jet heat transfer problem are:

- *H/D*: nozzle height to nozzle diameter ratio;
- *r/D*: nondimensional radial position from the center of the jet;
- *z/D*: nondimensional vertical position measured from the wall;
- *Tu*: nondimensional turbulence intensity, usually evaluated at the nozzle;
- Re_0: Reynolds number U_0D/v
- *M*: Mach number (the flow speed divided by speed of sound in the fluid), based on nozzle exit average velocity (of smaller importance at low speeds, i.e. M<0.3);
- P_{jet}/D: jet center-to-center spacing (pitch) to diameter ratio, for multiple jets;
- A_f: free area (=1-[total nozzle exit area/total target area]);
- f : relative nozzle area (= total nozzle exit area/total target area)

The fluid properties are conventionally evaluated using the flow at the nozzle exit as a reference location. A complete description of the problem also requires knowledge of the velocity profile at the nozzle exit, or equivalent information about the flow upstream the nozzle, as well boundary conditions at the exit of the impingement region.

The geometry and flow conditions for the impinging jet depend upon the nature of the target and the fluid source (compressor or blower). In cases where the pressure drop associated with delivering and exhausting the flow is negligible, the design goal is to extract as much cooling as possible from a given air mass flow. Turbine blade cooling is an example of such an application; engine compressor air is available at a pressure sufficient to choke the flow at the nozzle. As the bleed flow is a small fraction of the overall compressor flow, the impinging jet nozzle pressure ratio varies very little with changes in the amount of airflow extracted. At high pressure ratios the jet emerges at a high Mach number. In the limit case, the flow exits as an underexpanded supersonic jet. This jet forms complex interacting shock waves and a stagnation "bubble" directly below the jet, which can degrade heat transfer (shown in Fig 5.3). If for incompressible duct flow it is easy to predict the power draw from the compressor or the blower just by multiplying the pressure rise Δp by the volumetric flow Q and then considering also some efficiency factors, when dealing with turbine-cooling solutions where the compressibility is significant the problem becomes more complex. In this case the blower pressure rise Δp depends on the total of the pressure losses, considering the ones in the blower intake, in the flow path leading to the nozzle, in the jet region (due to jet interaction and jet confinement) and in the region at the exit of the target zone. When space is not critical, the intake and nozzle supply pathways are relatively open (in fact there is no reason to accelerate the flow far upstream of the nozzle). When possible, the goal is to mantain the flow at low speed until it nears the nozzle exit, and then to accelerate it to the required velocity using a smoothly contracting nozzle. In such a case the majority of the loss occurs at the nozzle: for a cylindrical nozzle, this loss will be at least equal to the nozzle dump loss, giving a minimum power requirement of $(0.5\rho U_{jet}^2 Q)$. In general, we can say that a reduction of losses is possible at the expense of greater hardware volume and complexity, and this cannot always be done.

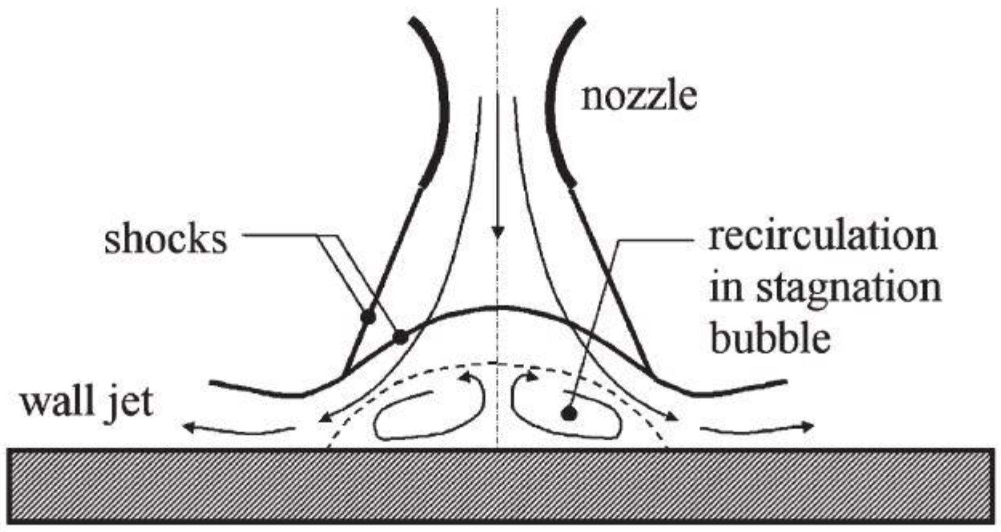


Fig. 5.4: Underexpanded jet flow pattern.

5.4. Turbulence Generation and Effects

Jet behavior is typically categorized and correlated by its Reynolds number $Re=U_0D/\nu$, defined using initial average flow speed (U_0), the fluid viscosity (ν) and the characteristic lenght that is the nozzle exit diameter D or twice the slot width $2B$ (the slot jet hydraulic diameter). At $Re<1000$ the flow exhibits laminar flow properties. At $Re>3000$ the flow has fully turbulent features. A transition region occurs with $1000<Re<3000$. Turbulence has a large effect on the heat and mass transfer rates; analytical solution is available for fully laminar jets, but they provide less heat transfer at a given flow rate, so much literature exists for turbulent impinging jets. Typical gas jet installations for heat transfer span a Reynolds number range from 4000 to 80000. H/D usually ranges from 2 to 12. Ideally, Nu increases as H decreases, so a designer would prefer to select the smallest tolerable H value, noting the effects of exiting flow, manifacturing capabilities, and physical constraints, and then select nozzle size D accordingly. For small-scale turbomachinery applications jet arrays commonly have D values of 0.2-2mm, while for larger scale industrial applications, jet diameters are commonly in the range of 5-30mm. In any case the diameter is heavily influenced by manifacturing and assembly capabilities.

The specific turbulent kinetic energy k gives a measure of the intensity of the turbulent flow field. This can be nondimensionalized by dividing it by the time-averaged kinetic energy of the flow to give the turbulence intensity, based on a velocity ratio:

$$Tu = \sqrt{\frac{\overline{u_j' u_j'}}{\bar{u}_i \bar{u}_i}} \tag{5.9}$$

Turbulence can be generated not only in the impinging jet flow field itself, but also upstream of the nozzle. This can happen because of the coolant flow distribution, but sometimes may be forced in order to increase the heat transfer coefficients, by inserting various obstructions in the jet supply pipe upstream of or at the nozzle. It was experimentally shown that this increase the lenght of the jet core region, thus reducing the *H/D* at which the maximal Nu_{avg} is reached. The downstream flow and heat transfer characteristics are sensitive to both the steady time-averaged nozzle velocity profile and fluctuations in the velocity over time. Knowledge of these turbulent fluctuations and the ability to model them are vital for understanding and comparing the behavior and performance of impinging jets.

The primary source of turbulence in the initial region is the shear flow on the edge of the jet. It starts very thin on a sharp nozzle but then it grows in area along the axis of the jet. At higher Reynolds numbers, the shear layer generates flow instability, similar to the Kelvin-Helmholtz instability. Fig 5.3 represents qualitatively the pattern of motion at the edges of the unstable free jet.

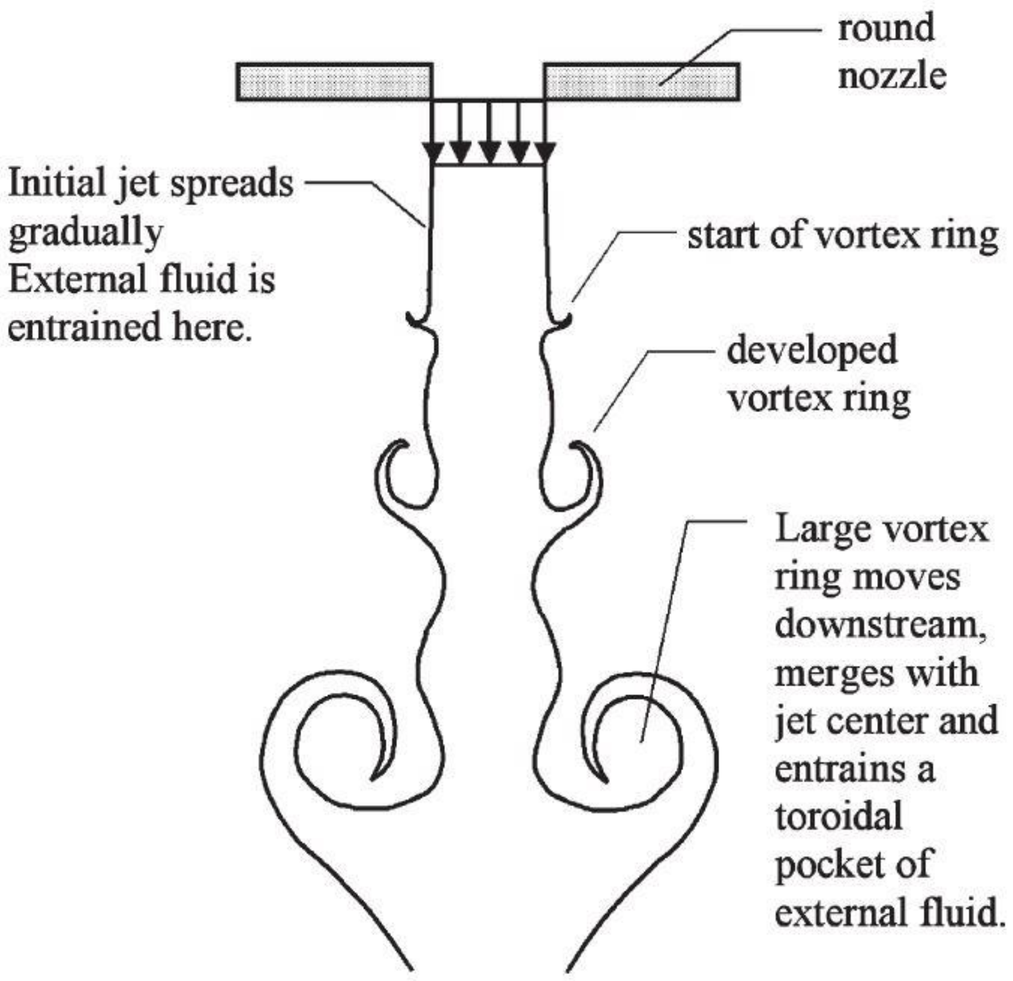

round nozzle
Initial jet spreads gradually External fluid is entrained here.
start of vortex ring
developed vortex ring
Large vortex ring moves downstream, merges with jet center and entrains a toroidal pocket of external fluid.

Fig. 5.5: Instability in the turbulent free jet.

At high flow speeds (*Re>1000*) the destabilizing effects of shear forces may overcome the stabilizing effect of fluid viscosity/momentum diffusion. The position of the shear layer and its velocity may develop oscillations in space. Further downstream, the magnitude and spatial extent of the oscillations grow to form large-scale eddies along the sides of the jet. The largest eddies have a lenght scale of the same order of magnitude as the jet diameter and persist until they either independently break up into smaller eddies or meet and interact with other downstream flow features. The pressure field of the stagnation region further stretches and distorts the eddies, displacing them laterally until they arrive at the wall.

Some experiments (by Hoogendorn) found the development of turbulence in the local *Nu* on the target stagnation region as well as the magnitude. For pipe nozzles and for contoured nozzles at high spacing ($z/D>5$) the *Nu* profiles had a peak directly under the jet axis. For contoures nozzles at $z/D=2$ and 4 with low initial turbulence ($Tu \approx 1\%$), the maximum *Nu* occured in the range $0.4<r/D<0.6$ with a local minimum at $r=0$, typically 95% of the peak value.

In the decaying jet region, the shear layer extends throughout the center of the jet. This shearing promotes flow turbulence, but on smaller scales. Here the flow may form small eddies and turbulent pockets within the center of the jet, eventually developing into a unstructured turbulent flow field with little or no coherent structures in the entire jet core.

In the deceleration region, additional mechanisms take part in influencing flow field turbulence. The pressure gradients cause the flow to turn, influencing the shear layer and turning and stretching large-scale eddies, in contrast to the developed wall jet where shear strains dominate.

Traveling along the wall, the flow may make a transition to turbulence in the way of a regular parallel wall jet, beginning with a laminar flow boundary region and then reaching turbulence. For transitional and turbulent jets, the flow approaching the wall already has substantial turbulence. This turbulent flow field may contain large fluctuations in the velocity component normal to the wall.

Large-scale turbulent flow structures in the free jet have a great effect upon transfer coefficients in the stagnation region and wall jet. The vortices formed in the free jet-shearing layer, categorized as primary vortices, may penetrate into the boundary layer and exchange fluids of differing kinetic energy, temperature or concentration. The primary vortex scrub away the boundary layer as it travels against and along the wall and this increases the local heat and mass transfer. The turbulent flow field along the wall may also cause formation of additional vortices categorized as secondary vortices. Turbulent fluctuations and associated pressure gradient fluctuations can produce local flow reversals along the wall, initiating separation and the formation of secondary vortices (Fig 5.6). These vortices cause local rise in heat/transfer rates and, like the primary vortices, they result in overall losses of flow kinematic energy and may cause local regions of lower transfer rates. Recent studies (Chung and Luo) at various Reynolds numbers showed that large-scale vortex activity along the wall may generate a secondary peak in transfer coefficients and causes most of the variation in *Nu* over time. Some investigations suggest that the turbulence in this region is generated by increased shear forces in the thin accelerating region immediately outside the stagnation region. Time-averaged numerical modeling by Zuckerman & Lior for $H/D=2$ showed that the shear layer in the upper portion of the wall jet generates the majority of turbulence in the flow field.

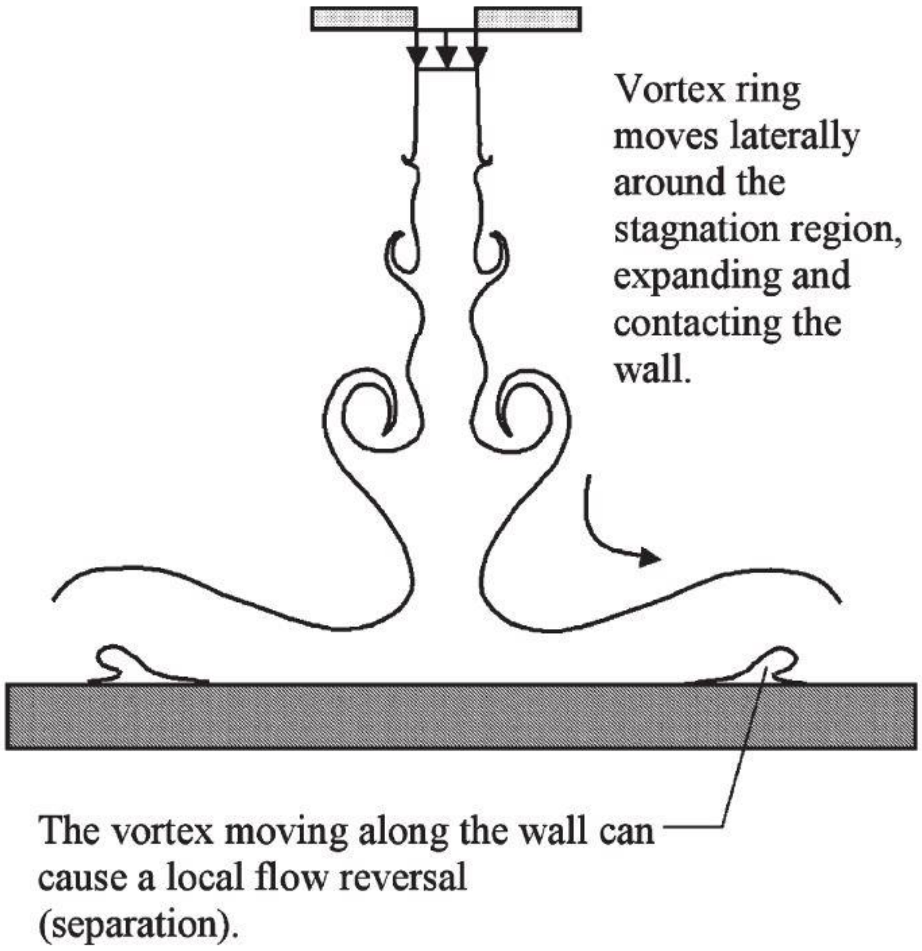

Vortex ring moves laterally around the stagnation region, expanding and contacting the wall.
The vortex moving along the wall can cause a local flow reversal (separation).

Fig. 5.6: Vortex motion in the impinging jet.

This high-turbulence region grows streamwise and also spreads in the wall-normal direction. The location of the secondary peak coincides with the location of the highest turbulent kinetic energy adjacent to the wall. These numerical results correlate well with the findings of Narayanan *et al.,* who found that maximum *Nu* occured in regions with high outer wall-jet region turbulence, rather than in regions exhibiting high turbulence only in the near-wall portion of the wall jet. Their specific conclusion was that the outer region turbulence caused an unsteadiness in the thermal boundary layer outside of the stagnation region.

5.5. Jet Geometry

Flow and turbulence effects and the heat or mass transfer rates are strongly influenced by the geometry of the impinging jet device. These include tubes or channels and orifices. An enlarged plenum upstream of the orifice may dampen supply pressure oscillations and smooth velocity and temperature profiles but its implementation is not always possible because of the lack of space in some applications. The nozzle type affects the heat transfer rate.

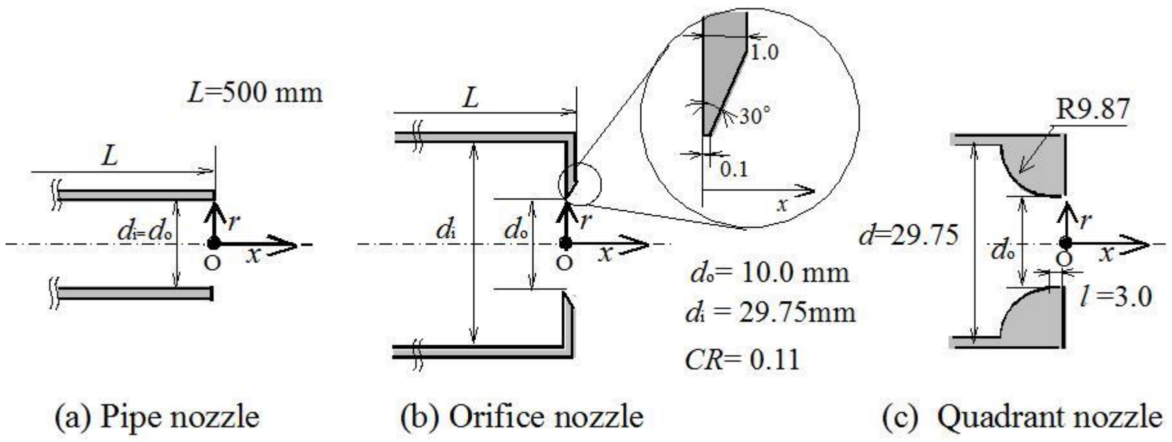
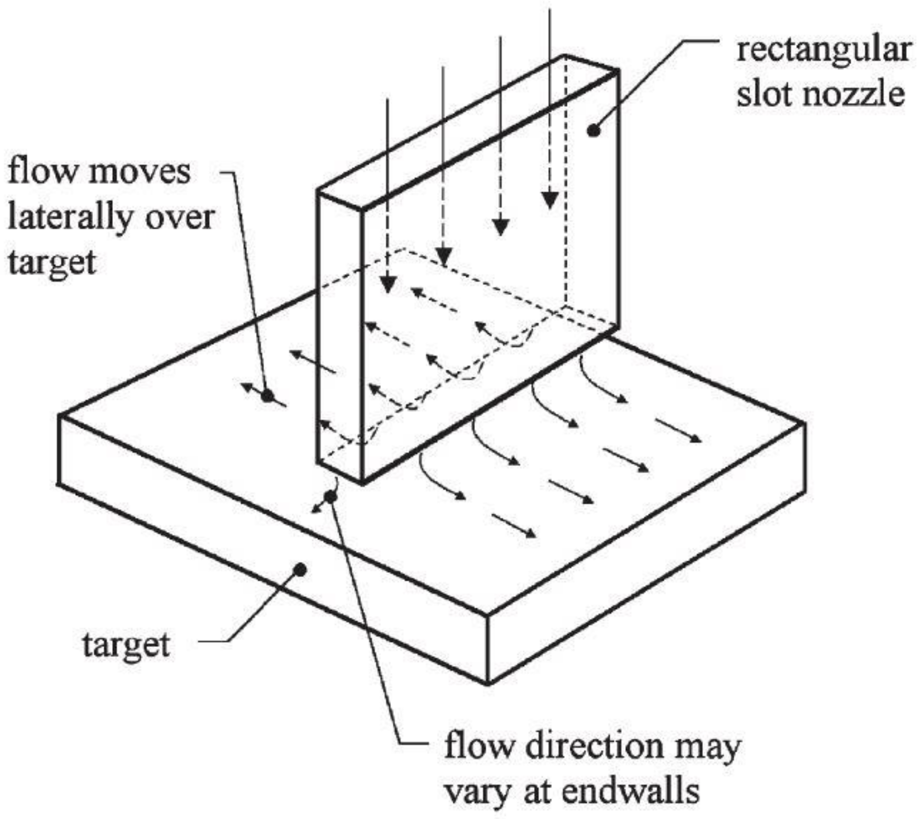


Fig. 5.7: Types of nozzles.

Test by Lee and Lee demonstrated that orifice nozzles produce higher heat transfer rates than a fully developed pipe flow at all radial positions, with local *Nu* increases of up to 65% at *H/D=2* and up to 30% at *H/D=10*. This difference between the nozzle types becomes larger at decreasing *H/D* values. In fact orifice plates confine the flow at smaller *H/D*.



rectangular slot nozzle
flow moves laterally over target
target
flow direction may vary at endwalls

Fig. 5.8: Slot jet scheme.

A slot jet provides a heat or mass transfer pattern that varies primarily in one spatial dimension on the target wall, an advantage when uniformity of transfer coefficient is desired, but presents some structural disadvantages relative to a round nozzle array orifice plate.

The use of multiple nozzles to cover a target surface offers some improvements in efficiency and uniformity of transfer properties with both two-dimensional (slot) and three-dimensional nozzle geometries. For a typical single-round impinging jet the *Nu* values can vary by a factor of 4 or 5 from *r/D=0* to *9*. The incorporation of a nozzle array can reduce this variation to a factor of 2. An array of pipe nozzles requires more effort to manufacture, but can also provide useful pathways for exiting flow.

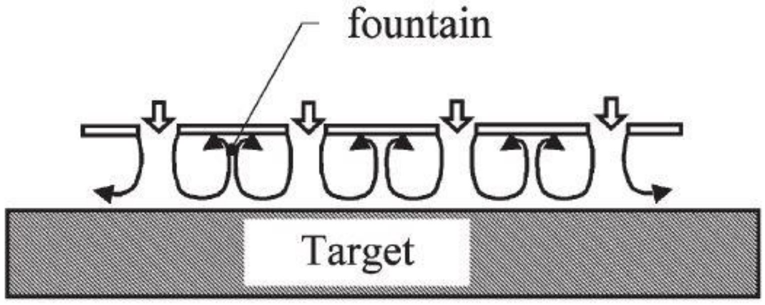


Fig. 5.9: Circulation pattern in the confined jet array.

The pitch p_{jet}, or center-to-center positioning of jets in an array, determines the degree of jet interaction. For jets spaced at pitch-to-diameter ratio $p_{jet}/D<4$, the jets show significant interaction. Some studies showed that for $H/D=2$, the interference persisted up to spacings of $p_{jet}/D=8$ or *10*, and the maximal *Nu* occured at $p_{jet}/D=8$.

At the small p_{jet}/D for medium and big jet lenghts, i.e. $p_{jet}/D<2$ and $H/D\geq4$, the growing shear-layer jer boundaries may influence each other. If the two neighboring shear layers grow and combine then the velocity gradient at the edge of the jet decreases in magnitude, reducing further turbulence generation and interfering with the generation of large-scale eddies. At the target impingement plate, the wall jets of two adjacent flows may collide, resulting in another local stagnation region or boundary layer separation, and a turning of the flow away from the wall into a "fountain" shape (Fig 5.9). The fountain can alter transfer rates in the location of colliding wall-jets, and for the highly constrained jets ($H/D<2$) it may influence the free jet-shearing layer. If the fountain flow exchanges momentum with the free jet-shearing layer, the surface transfer rates are found to decrease. For large p_{jet}/D, namely when the fountain is far from the free jet, the highly turbulent region beneath the fountain itself may have higher transfer rates then the upstream wall jet, with values of the same order of magnitude as those produced in the stagnation region.

Jet interaction is also affected by the ratio of jet nozzle diameter to target wall spacing, *H/D*. In Tab. 5.1 we have an overview of this effect as the *H/D* ratio changes. As a rule of thumb, the jet interaction plays a minor role for $p_{jet}/D>8$ and $H/D>2$, and the undesired interference will increase as p_{jet}/D and *H/D* decrease from these values. Experiments have shown that the interference is much less sensitive to *Re* than to these two geometric ratios.

Tab. 5.1: Effect of *H/D* ratio upon jet array.

H/D	**Effect upon jet array**
Up to 0.25	Highly constrained flow, may have strong crossflow and high additional backpressure (on the order of magnitude of the nozzle exit dynamic pressure). Additional flow acceleration expected to shift peak *Nu* laterally by 0.5÷1.5*D*
0.25÷1.0	Fountain flow may greatly affect heat transfer in confined arrays
1÷2	Mild fountain effects may occur. Minor turbulence generation. Flow will be affected by confinement wall, need to ensure a clear exit pathway.
2÷8	Shear layers may interact, need to mantain sufficient p_{jet}.Best performance tends to lie in this range.
8÷12	Minimal confinement effect is overshadowed by nozzle type. Need to ensure that neighboring jets remain separate.
12+	Confining wall does not influence flow, instead nozzle type and jet spacing dominate the flow field. *Nu* affected by jet energy loss approaching the wall. Need to ensure that neighboring jets remain separate.

In case of crossflow, it tends to disturb the impinging jet pattern, thicken wall boundary layers, and degrade transfer rates. Several experiments by Florschuetz *et al.* mapped out the effects of jet nozzle spacing and the resulting crossflow in jet arrays. The results showed the immediate benefits of decreasing *D* and allowing space between jets for the channeling of spent flow. They also measured downstream displacement of heat transfer maxima caused by crossflow. In some peculiar application (e.g. drying of paper) the jet impinges on a moving target, which makes the heat or mass transfer in the direction of motion more uniform. An effective speed has to be selected, depending on the jet spacing, on the nonuniformities in the flow and on a time costant associated with the rate at which heat or mass can be trasferred to or from the target. For small lateral translation speeds compared to the jet speed, i.e. target wall speed $<0.2U$, the motion has little effect on the fluid flow; when the wall velocity is higher, the effect is like that of superposing a crossflow. In a jet array this tends to decrease overall heat transfer, but with a single jet some benefit has been noted with target motion. For wall speeds greater than *U*, the turbulence production in the stagnation region is no longer dominated by wall-normal turbulent stresses but rather by wall-parallel fluid velocity component (Chattopadhyay *et al.*). Industrial processes may use target speeds up to 10 times the jet speed but it has been found that for a single jet a maximum averaged *Nu* occured for target speeds at about 1.2 times the jet speed, with maximum averaged *Nu* up to 25% higher than with a stationary target.

Constrained by strenght, space, and cost considerations, designers added features to the simple impingement designs, primarily to obtain higher heat transfer coefficients, at an acceptable pressure drop and energy consumption. Many nozzle types were tested, including cross- and star-shaped cross-sections, fluted, scarfed, tabbed, and angled nozzles. Some nozzles were structured to swirl the flow before allowing it to hit the wall, resulting in a higher flow speed at a given nozzle mass flow. For high *H/D* ratios the beneficial effects tend to be lost but, for $H/D \leq 2$, the swirl can make the *Nu* distribution more uniform, at the cost of a lower peak value. At high swirl, the stagnation region heat transfer may decrease due to the recirculation on the target immediately under the swirling jet.

The jet impingement angle has an effect on heat transfer and was studied often; it may be needed due to some unique feature of the hardware design or is motivated by desire to reduce the penalties of jet interaction or to reduce losses in the approaching or spent-flow exit pathway. The inclination distorts the heat transfer contours isolines of *Nu.* It was found that *Nu* decreases as the impingement incidence angle becomes smaller than 90°. The maximum *Nu* (Nu_0) occurs downstream of the intersection of the nozzle axis and the target. Although the transfer rate patterns is affected by the inclination, the area-averaged transfer coefficient decreased by only 15-20%.

Jet arrays with pulse jets generate large-scale eddy patterns around the exit nozzle, resulting in unsteady boundary layers on the target that may produce higher or lower heat transfer coefficients, depending on frequencies, dimensions and jet Reynolds number. Bart *et al.* demonstrated an increase of up to 20% in *Nu* for lower pulse frequencies (200-400 Hz) at moderate jet spacings ($H/D=4$-6). Göppert *et al.* investigated the effects of an unstable precessing jet over a fixed target plate. As with a pulsed jet, the variation in local fluid velocity over the target prevented the development of a steady boundary layer. This effect was counteracted by an increased tendency of the precessing jet to mix with the surrounding fluid, lose energy, and reach the target at lower velocities than would be found with a stationary jet ($H/D=25$), resulting in a 50% decrease in wall-jet *Nu* values.

Moving towards more complex nozzle structures, Hwang *et al.* altered the flow pattern in the initial shearing layer by using coaxial jets. The entry velocity in the second layer of jet flow surrounding the primary jet was varied to control vortex shedding rate and the persistence of large-scale vortices. Experimental results showed that with high flow speed in the secondary nozzle, the onset of vortex formation was delayed, which increased Nu_0 by up to 25% for higher *H/D* ($9<H/D<16$). Something similar occurs in the case of an annular jet, which produces *Nu* maxima spaced laterally outward from the center of the jet axis. It also creates a local *Nu* minimum on the target in the center of the annulus, where the flow washes upwards as a fountain and gest entrained into the surrounding annular jet. The annular jet provides a mean of widening the jet, and hence the stagnation region where *Nu* is highest, without increasing the required mass flow. The withdraw is that the higher wetted surface area brings to higher frictional losses and pressure drop. Due to the internal and external shearing layers, it promotes additional turbulence downstream of the nozzle if compared to a cylindrical nozzle.

Other alternative solutions have been dictated by the target surface or manufacturing process design.

Impinging jets can be used with other target modifications such as ribbed walls. These ribs disturb the wall jet and increase turbulence; moreover they can function also as fins, increasing the effective surface area for transfer of energy. While ribs increase the transfer rates outside the stagnation region, on the other hand the heightened drag causes the wall jet to decelerate and disperse more rapidly, decreasing *Nu* far from the stagnation region. Gau and Lee found that in the region of low jet turbulence the gap between the ribs can fill with an "air bubble" which reduces *Nu* by 20-50%. By reducing rib height to 15% of the nozzle width and raising the nozzle to *H/D>6* they were able to set up a recirculating flow in these gaps and improve local *Nu* in the stagnation region. The resulting value of Nu_0 at *H/D=10* increased by up to 30% compared to a flat target.

In the case of restrictive exit pathways, it is possible to open additional holes for the spent flow in the fluid supply plate of an orifice array. This solution proves to be valuable for *H/D<2* increasing average transfer rates: Rhee *et al.*showed a 50% improvement in jet array average transfer rate at *H/D=0.5* by adding diffusion holes, but reported minimal influence for *2<H/D<10*. The design of these holes can be challenging depending on the application; for instance in a turbine blade the preferred pathways are either through holes in the target wall itself to form a film cooling layer on the opposite surface, or through the confined flow region leading to aerodinamically favorable exit holes on or near the trailing edge of the blade.

Impinging jets are not limited to single-phase flows; impinging gas jets containing liquid droplets deliver a large increase in heat transfer and also the inclusion of small solid particles improve it as the they impact the target. Based on the same principle, we can mention also flame impingement. We will not describe it in detail being more interested to heating and cooling applications.

5.6. Reynolds analogy

The Reynolds analogy is a popular expression to relate momentum and heat transfer. The main assumption underlying is that heat flux in a turbulent system is analogous to momentum flux, which suggest that the ratio of these two quantities must be constant for all radial positions. It was developed to circumvent the difficulties of dealing with turbulence. This simple analogy enables heat transfer coefficients to be predicted from measurements of the pressure losses due to friction in flow. After the first formulation, it has been considerably amended and extended since his early work to include mass transfer as well as heat transfer.

As just said, the Reynolds analogy provides a useful tool to predict heat transfer, and this is true also when we are dealing with impinging jets.

In turbulent flow, the shear stress in the boundary layer may be written:

$$\tau = \rho(\nu + \varepsilon_m)\frac{dU}{dy} \tag{5.10}$$

where ε_m is the eddy viscosity.

The heat transfer is given by:

$$q'' = -\rho C_p\left(\alpha + \varepsilon_q\right)\frac{dT}{dy} \tag{5.11}$$

where $\alpha = \frac{k}{\rho C_p}$ is the conductive diffusivity and ε_q the eddy diffusivity.

Experiments show that:

- $\varepsilon_q = \varepsilon_m$
- velocity and temperature profiles are similar
- $\varepsilon_q \gg \alpha$ and $\varepsilon_m \gg \nu$

Reynolds assumed $\varepsilon_q = \varepsilon_m$ and identical profiles $\left[\frac{dT}{dy} = \frac{dV}{dy}\right]$, obtaining:

$$\frac{q''}{\tau C_p} = \frac{dT}{dV} = constant \;\; if \;\; \alpha = \nu$$

Integrating at the wall:

$$\frac{q''}{\tau_w} = \frac{C_p \Delta T}{V} \tag{5.12}$$

with $\Delta T = T - T_w$ and V_w=0.

Experimental data agree approximately with above equation if the Schmidt and the Prandtl numbers are near 1 and only skin fricition is present in flow past a flat plate or inside a pipe. When liquids are present and/or form drag is present, the analogy is known to be invalid. More recent studies showed that the analogy is valid for flows that are close to developed, for whom changes in the gradients of fields variables along the flow are small.

5.7. Conclusions

Cooling and heating under an impinging jet is generally superior to that achieved with typical convective heat transfer methods. With this cooling solution it is easy to adjust the location of interest and to remove a large amount of heat on the impinged surface.

For these reasons, the impinging jet technique has been widely used in many industrial systems such as gas turbine cooling, rocket launcher cooling and high-density electrical equipment cooling.

Recently, impinging cooling schemes have been applied to improve the performance in micro and nano systems. In order to improve cooling performance, researchers have performed numerous experiments to control and design single and array jets. They considered the nozzle geometry, impinging surface, and active methods such as jet vibration, secondary injection and suction flow. In addition, various factors concerned with operating conditions (crossflow, rotation, mass flow rate, working fluid, etc.) and combined techniques (rib turbulator, pin fin, dimple, effusion holes, etc.) have been investigated in order to identify the optimum conditions and geometries.

Liquid crystals method, infrared (IR) camera technique, and naphthalene sublimation method permitted to obtain detailed experimental heat transfer distributions.

Given the improvements in computational performance, many numerical calculations have been carried out to obtain local heat transfer distributions in complex geometries. Various turbulence models have been developed for use in many applications, because the numerical results still disagree with the experimental results in their local distribution; in fact the solution of this kind of problems is not easy at all, because of the high number of phenomena that affects it.

6. Impinging jet case

During this case, we have investigated an axisymmetric impinging jet flow. The surface impinged consists of a flat plate maintained at a constant temperature. The gas flowing through the jet is air while the impinged plate is made by steel. Here we can see some properties of fluid and solid domains.

Tab. 6.1: Properties of the mediums.

	Air	**Steel**
Density [kg/m3]	rho=p/RT	8030
Specific Heat (Cp) [J/kgK]	1006,43	502,48
Thermal Conductivity [W/kgK]	0,0242	16,3
Viscosity [kg/m-s]	Sutherland	\\

The jet considered has a cylindrical nozzle with a settling chamber upstream; the higher section in the settling chamber slows down the flow limiting the losses, and limits pressure oscillations at the nozzle inlet. This solution improve overall device performance but comes at the expense of great hardware volume and complexity.

Considering the problem as axisymmetric, we can use it in order to reduce the computational cost of the calculation. Hence, we can simulate only a one degree wedge: the solution will be the same of simulating the complete geometry.

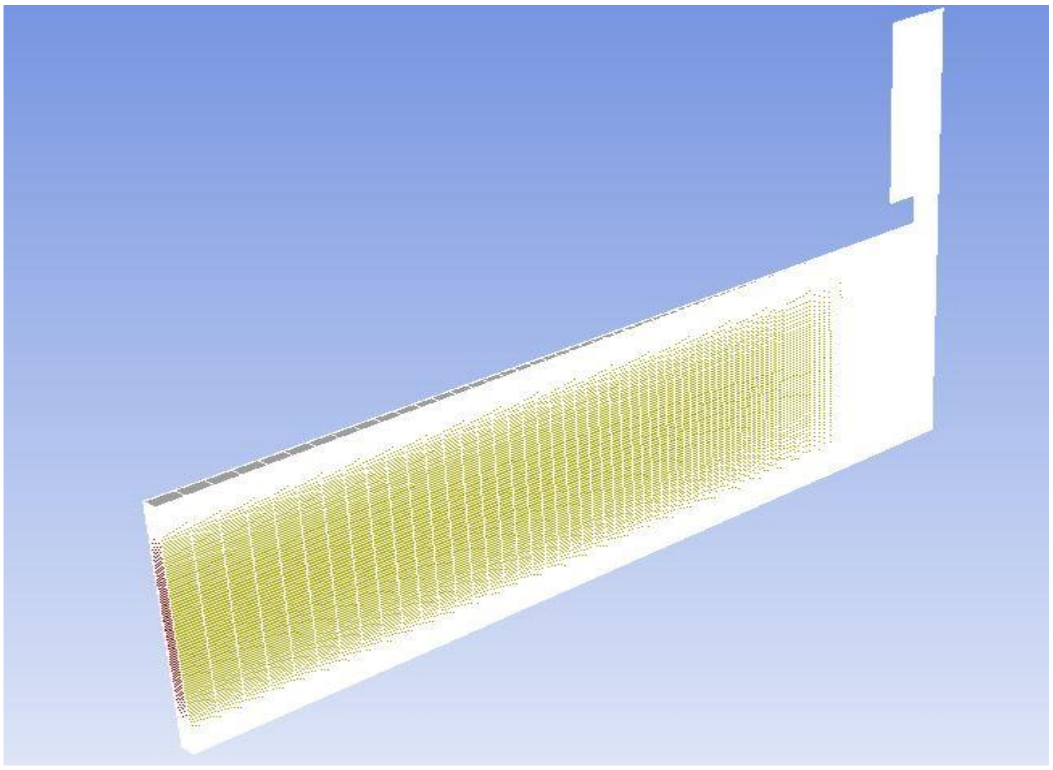


Fig. 6.1: Overview of the geometry.

6.1. Mesh

For the numerical computation of our domain we chosed a structured mesh. The mesh has been done with IGG mesher by NUMECA. We started with a 2D mesh (Fig 6.3), and then we rotated it around the axis of symmetry. We can also see in Fig 6.2 a particular of the inlet part of the mesh which is not visible in the complete mesh image.

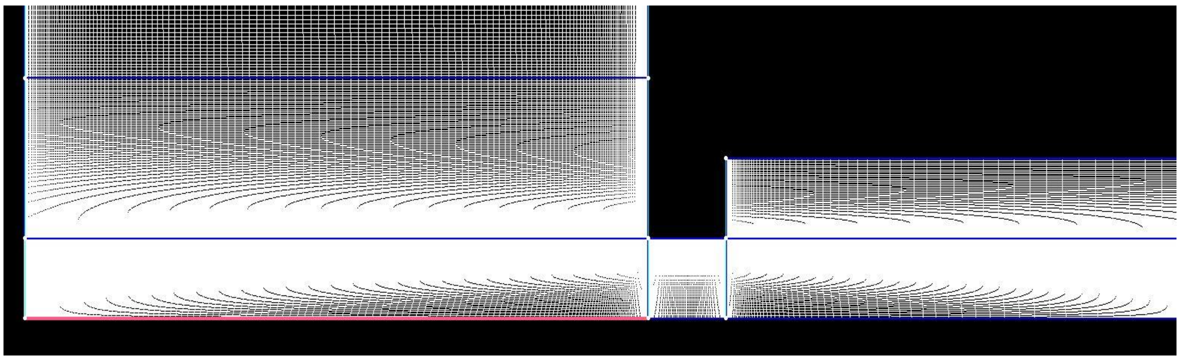


Fig. 6.2: Particular of the mesh.

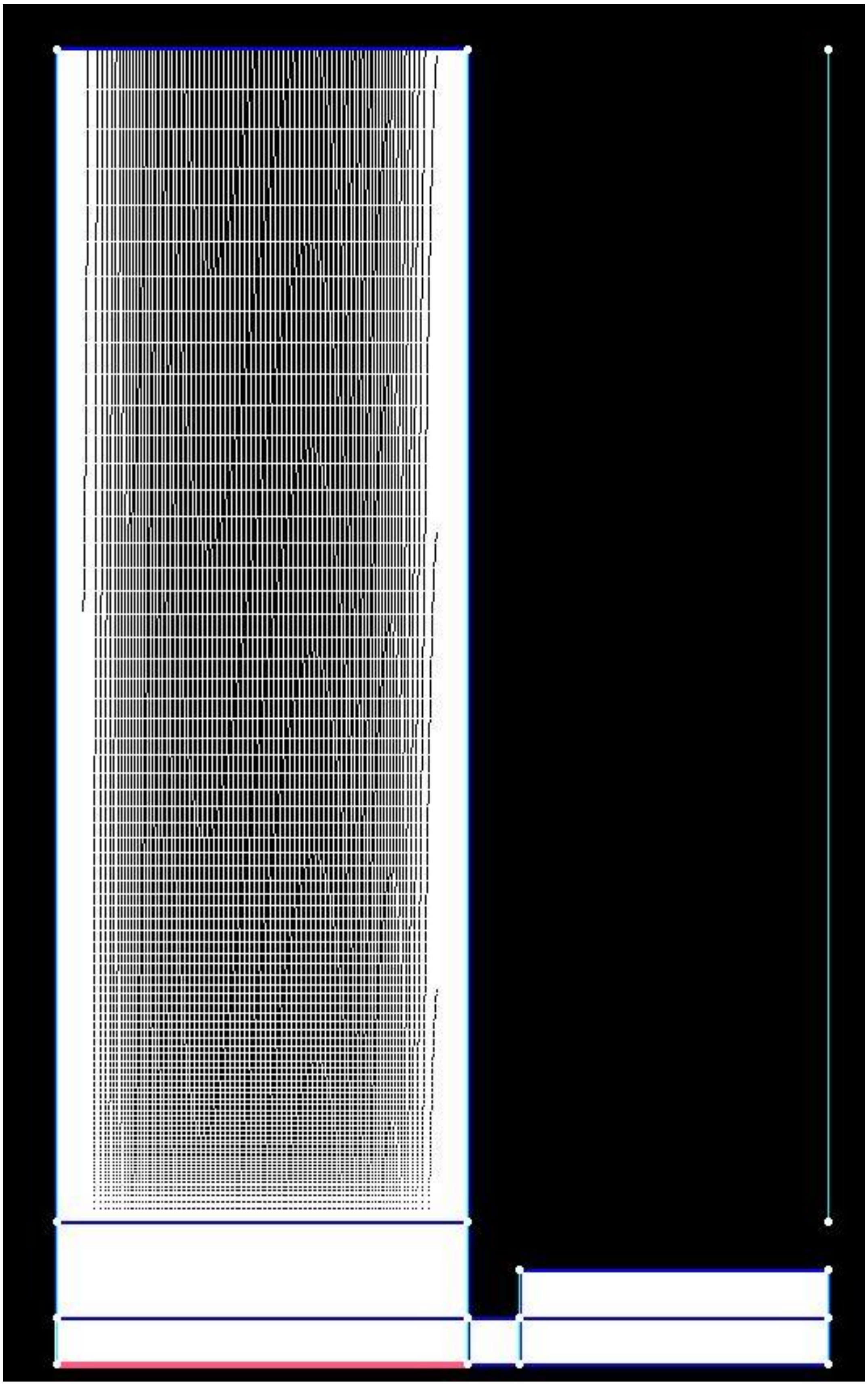
In the Tab 6.2 it is possible to see some quality aspects of the mesh. Two meshes are presented: the original one, taken from a previous Fluent case, and a coarser one, created on the same layout to check mesh dependency of the solution and to try to solve problems occurred in transient simulations.

Tab. 6.2: Mesh characteristics.

	Original Mesh	**Coarser Mesh**
No. of cells	69632	61408
No. of blocks	141812	125188
Minimum orthogonality	66.81	77.33
Maximum aspect ratio	126	126
Maximum expansion ratio	1.09	1.09
Max skewness	0.329	0.33

Both the meshes have been refined in the nozzle area, in the impinging point, and along the walls (especially the heated wall), where we are more interested in a fine solution. In the other areas the mesh can be coarser, being less interested in the solution in those zones; moreover, this allows a faster computation of the stable solution, avoiding the waste of computing resources. The choice of a structured mesh is related to the easy geometry we had to mesh and to its space efficiency: the neighborhood relationships are defined by storage arrangement, allowing to reduce the dimensions of the mesh file. It also assures better convergence. On the other hand, it obliges to have the same number of divisions along corresponding segments, and this may cause mesh refinement where it is not necessary.

Fig. 6.3: 2D mesh in IGG.



6.2. Boundary conditions

In this section we describe the boundary conditions used in our simulation.
First, we have to consider the names given to the patches

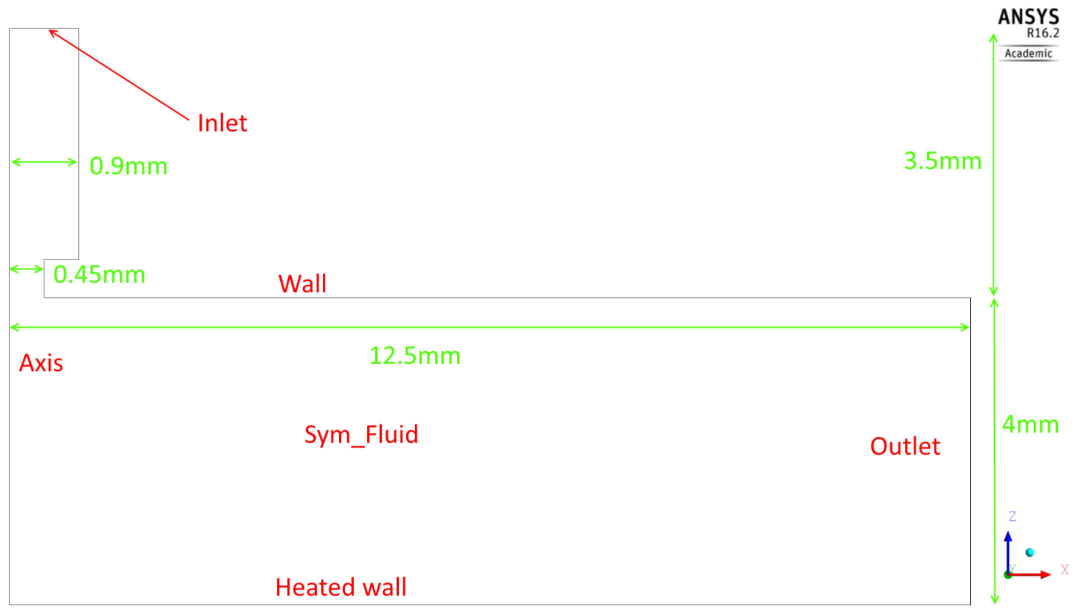


Fig. 6.4: name of the b.c. patches.

In the figure is not represented a patch named "sym_fluid1", which represents the other lateral face of the domain meshed and therefore is hidden behind the "sym_fluid" patch; this patch has exactly the same geometry of the "sym_fluid" one and share with it also the type of boundary condition. Thus, from now on we will not mention it again, but everything said about the first one will be valid for both.

The angle of extrusion adopted was 1 degree but, due to some problems while importing the mesh for the first time, in some OpenFOAM simulations this angle is equal to 2.5 degrees; after we managed to import it correctly with the same angle for both the solvers. This issue we experimented however has demonstrated that the results do not vary changing the angle, as expected in an axisymmetric problem. Existing simulations with different angles, in the OpenFOAM boundary conditions are reported different values for the quantities varying with the area surface of the "inlet" patch.

List of the b.c. used in our simulation (b.c. for both Fluent and OpenFOAM are described, due to some differences between the two softwares).

- **Inlet**
  - Fluent: *Mass Flow Rate*, with value equal to 5.10833e-08 kg/s as required. Pressure: 21720 Pa. Turbulence intensity: 1%; turbulence viscosity ratio: 10%. Temperature: 321K, costant during all the simulation.
  - OpenFOAM: *flowRateInletVelocity*, with 5.10833e-08 kg/s (1 degree mesh) and 12.770825e-08 kg/s (2.5 degrees) respectively. Temperature: 321K. Pressure: 21720 Pa.

- **Axis**
  - Fluent: *Symmetry*.
  - OpenFOAM: *symmetryPlane.*
- **Outlet**
  - Fluent: *Pressure Outlet.* Temperature: 321K. Pressure: 21720 Pa. Backflow turbulence intensity and turbulence viscosity ratio are 1% and 10% respectively.
  - OpenFOAM: velocity: *zeroGradient.* This b.c. simply extrapolates the quantity to the patch from the nearest cell value. The meaning is: the quantity in space is developed in space and its gradient is equal to zero in direction perpendicular to the patch (to the boundary). In some simulations we tried also the *inletOutlet* b.c.: it is normally the same as *zeroGradient*, but it switches to *fixedValue* if the velocity vector next to the boundary aims inside the domain (backward flow). The value of that *fixedValue* is set by the user in the entry *inletValue* of the b.c.; we tried it just to not consider the effect of the backflow in some simulations.

    Temperature: *fixedValue* 321K. Pressure: *fixedValue* 21720Pa.
- **Sym_fluid (and sym_fluid1)**
  - Fluent: *Symmetry.*
  - OpenFOAM: *wedge*. It is used in 2 dimensional axy-symmetric cases: the mesh is specified as a wedge of small angle (<5°) and 1 cell thick running along the plane of symmetry.
- **Wall**
  - Fluent: *wall.* Stationary wall with no-slip condition. Neither heat flux not wall temperature are set.
  - OpenFOAM: velocity: *fixedValue* (uniform (0 0 0)). For both pressure and temperature: *zeroGradient.*
- **Heated Wall**
  - Fluent: *wall.* Stationary wall with no-slip condition. Wall temperature: 644K.
  - OpenFOAM: velocity: *fixedValue* (uniform (0 0 0)). Pressure: *zeroGradient.* Temperature: 644K.

One of the most important parameters in a jet is the Reynolds number: as seen in the previous chapter, it appears in several empirical correlations and characterizes the flow conditions.

For jet nozzles, we use this formulation:

$$Re = \frac{uD}{\nu} \tag{6.1}$$

According to the boundary conditions above, the Reynolds number distribution at the jet outlet is represented in Fig 6.5. The value is between 1000 and 3000, so we are working exactly in the middle of the transition zone.

Only the OpenFOAM value is plotted, as the Fluent one is exactly the same. At the inlet of our domain the Reynolds number is constant and equal to 688.5 (laminar), as attested by the boundary layer development at the wall.

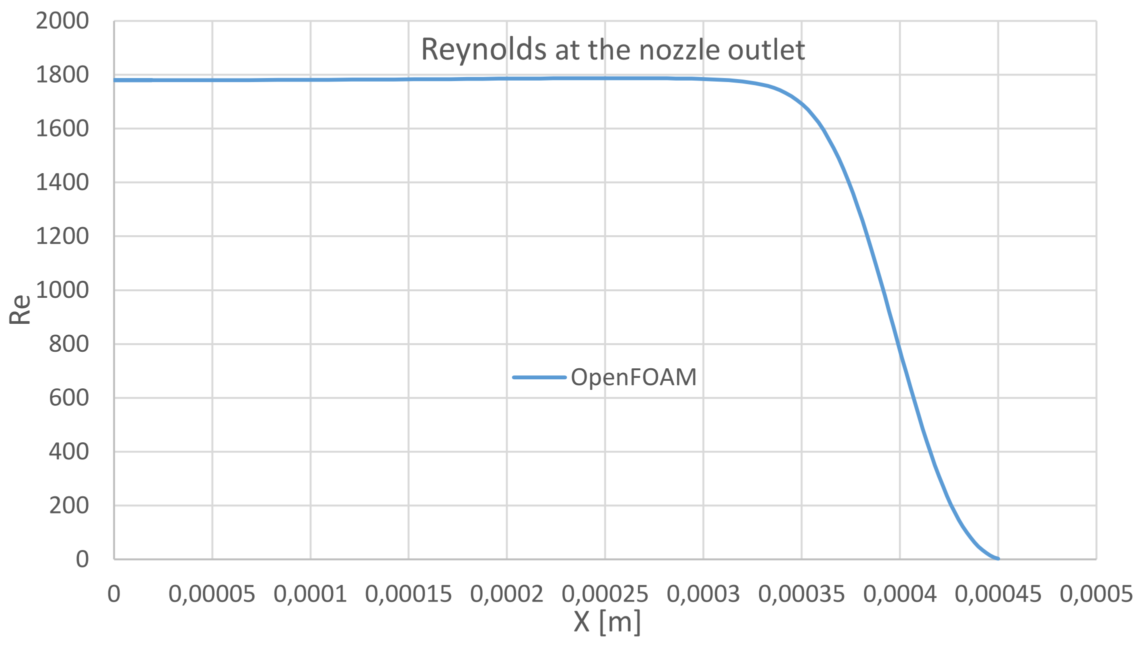


Fig. 6.5: Reynolds Number at the nozzle outlet.

As seen, there are many differences in the implementation of the same boundary conditions in the two codes. While in Fluent the turbulence quantities have to be set in the turbulence model dialog and in inlet and outlet boundaries, in OpenFOAM we add some new quantities to the '0' folder; these quantities are strictly related to the turbulence model we want to use.

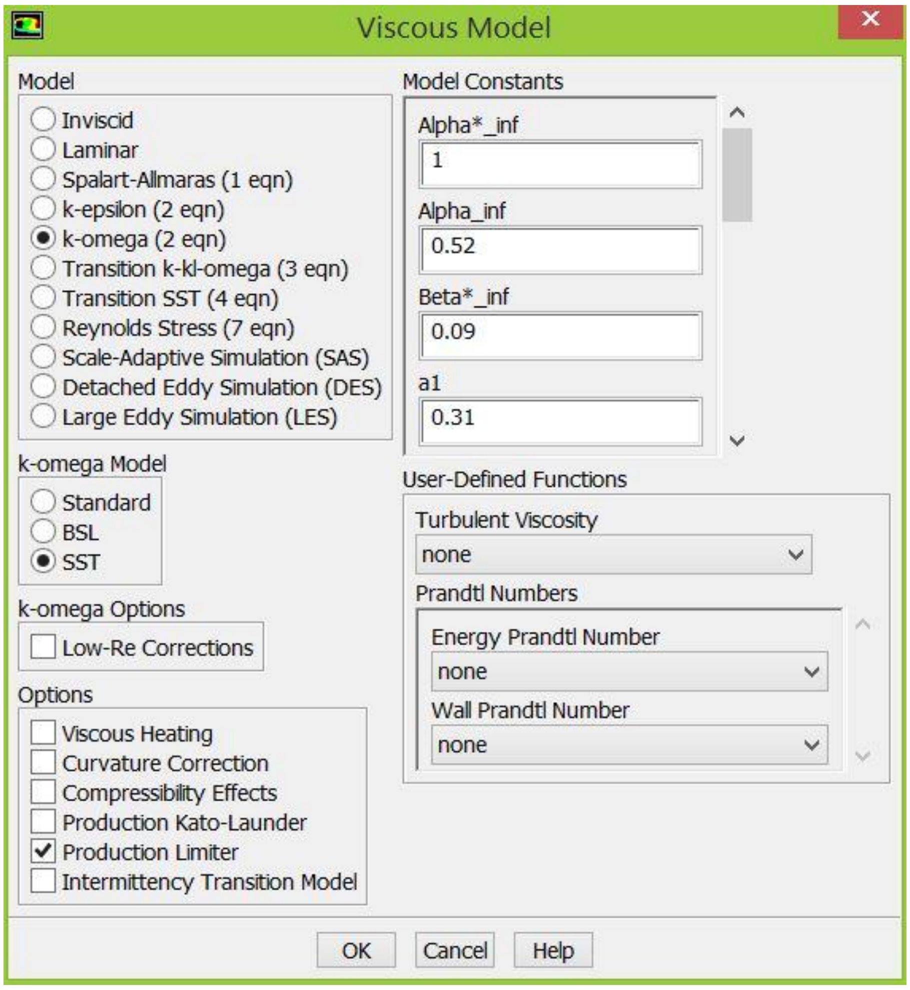
6.2.1. Turbulence quantities

Opening the drop down menu 'Models' in Fluent, we can change some options of the models utilized during the simulation and in particular we can chose the turbulence model we want to use.

In Fig 4.6, we can see the dialog 'Viscous Model' where we can choose the turbulence model, its implementation (if more than one is available), its options and the values of the constants in the model equations.

The values reported in the figure are the one used during the simulation. There are several model constants: their values are not presented in this thesis as we used the value set by default in Fluent.

In OpenFOAM the implementation of a turbulence model is not so direct as a GUI is not available.


Viscous Model

Model
- Inviscid
- Laminar
- Spalart-Allmaras (1 eqn)
- k-epsilon (2 eqn)
- [✓ k-omega (2 eqn)]
- Transition k-kl-omega (3 eqn)
- Transition SST (4 eqn)
- Reynolds Stress (7 eqn)
- Scale-Adaptive Simulation (SAS)
- Detached Eddy Simulation (DES)
- Large Eddy Simulation (LES)

k-omega Model
- Standard
- BSL
- [✓ SST]

k-omega Options
- Low-Re Corrections

Options
- Viscous Heating
- Curvature Correction
- Compressibility Effects
- Production Kato-Launder
- [✓ Production Limiter]
- Intermittency Transition Model

Model Constants

Alpha*_inf	1
Alpha_inf	0.52
Beta*_inf	0.09
a1	0.31

User-Defined Functions

Turbulent Viscosity	none
Energy Prandtl Number	none
Wall Prandtl Number	none

Fig. 6.6: Viscous Model dialog.

Here we have the list of quantities we had to add in order to use the same model seen in Fluent.

The κ-ω *SST* model requires the specification of the following variables: κ e ω. The first one is the turbulent energy, which can be computed as:

$$\kappa = \frac{3}{2}(UI)^2 \qquad (6.2)$$

where U is the mean flow velocity and I is the turbulence intensity ($I = u'/U$, where $u'$is the root-mean-square of the turbulent velocity fluctuations and U is the mean velocity).

Omega (ω) is the specific turbulent dissipation rate. It can be computed with the following formulas:

$$\omega = \frac{\sqrt{\kappa}}{l} \qquad (6.3)$$

where l is the turbulent length scale (physical quantity describing the size of the large energy-containing eddies in a turbulent flow) and κ is the quantity defined before. It is also possible to define ω from the eddy viscosity ratio:

$$\omega = \frac{\rho\kappa}{\mu}\left(\frac{\mu_t}{\mu}\right)^{-1} \qquad (6.4)$$

where μ is the molecular dynamic viscosity and $\frac{\mu_t}{\mu}$ is the eddy viscosity ratio.
Here is attached the code used to define κ e ω.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.140584;

boundaryField
{
    Inlet
    {
      type fixedValue;
      value uniform 0.140584;
    }
    sym_fluid
    {
        type            wedge;
    }
    sym_fluid1
    {
        type            wedge;
    }
    Axis
    {
        type            symmetryPlane;
    }
    Heated Wall
    {
       type            compressible::kqRWallFunction;
        value           uniform 0.140584;
    }
    Wall
    {
        type            compressible::kqRWallFunction;
        value           uniform 0.140584;
    }
    Outlet
    {
        type            zeroGradient
    }
}
```

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      omega;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 184.3784;

boundaryField
{
    inlet
    {
      type fixedValue;
      value uniform 184.3784;
    }
    sym_fluid
    {
        type            wedge;
    }
    sym_fluid1
    {
        type            wedge;
    }
    Axis
    {
        type            symmetryPlane;
    }
    Heated Wall
    {
        type            compressible::omegaWallFunction;
        value           uniform 184.3784;
    }
    Wall
    {
        type            compressible::omegaWallFunction;
        value           uniform 184.3784;
    }
    Outlet
    {
        type            zeroGradient;
    }
```

The omegawallfunction and krQwallfunctions are two boundary conditions for wall patches and they work properly also for meshes with y^+=1 (not only with coarse meshes); they move from a starting value which is modified in order to suit better for the problem investigated and then it behaves very similar to a *zeroGradient* boundary condition. Adding the new quantities is not enough: the turbulence model has to be “switched on” in the *RASProperties* file contained in the “*constant*” directory. Here we have an example of such a file:

```
//************************************************************//

RASModel         kOmegaSST;
//RASModel laminar;
//RASModel SpalartAllmaras;

turbulence       on;

printCoeffs      on;


//*********************************************************** //
```

In the example also SpalartAllmaras model and laminar model are present. The turbulence entry has to be ‘on’, otherwise the solver solves by default the laminar equation.

6.3. Fluent

In this subchapter settings and results for the Fluent case are reported. Further results and plots are available in the comparison between Fluent and OpenFOAM solutions.

6.3.1. Settings

In Fluent we ran a steady-state case of the problem with the following settings:
Pressure-Velocity Coupling Scheme: SIMPLE;
Spatial Discretization:

Gradient: *Least Squares Cell Based;*
Pressure: *Second Order;*
Density: *Third-Order MUSCL;*
Momentum: *Third-Order MUSCL;*
Turbulent Kinetic Energy: *Third-Order MUSCL;*
Energy: *Third-Order MUSCL;*

6.3.2. Results

In this chapter results are presented.

Considering the total temperature field, the first thing we can see is the backflow at the outlet; in fact, we set 321K as outlet temperature and doing so, the backflow is clearly visible because of the lower temperature, if compared with the outflow heated from the lower wall.

It is also noticeable the decreasing cooling effect of the jet moving away from the stagnation point. The thickness of the layer at higher temperature increases along the plate till a maximum, then it decreases again because of the cold air backflow.

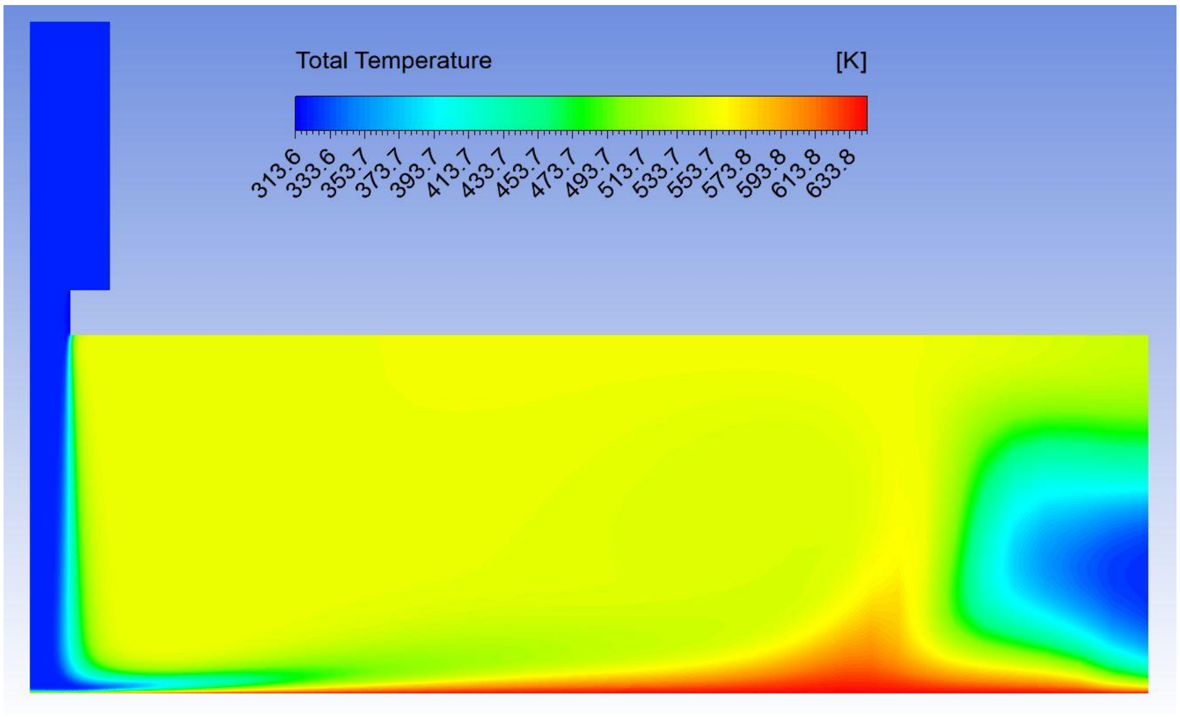


Fig. 6.7: Total Temperature Field.

Now, let us have a look of the absolute pressure field.

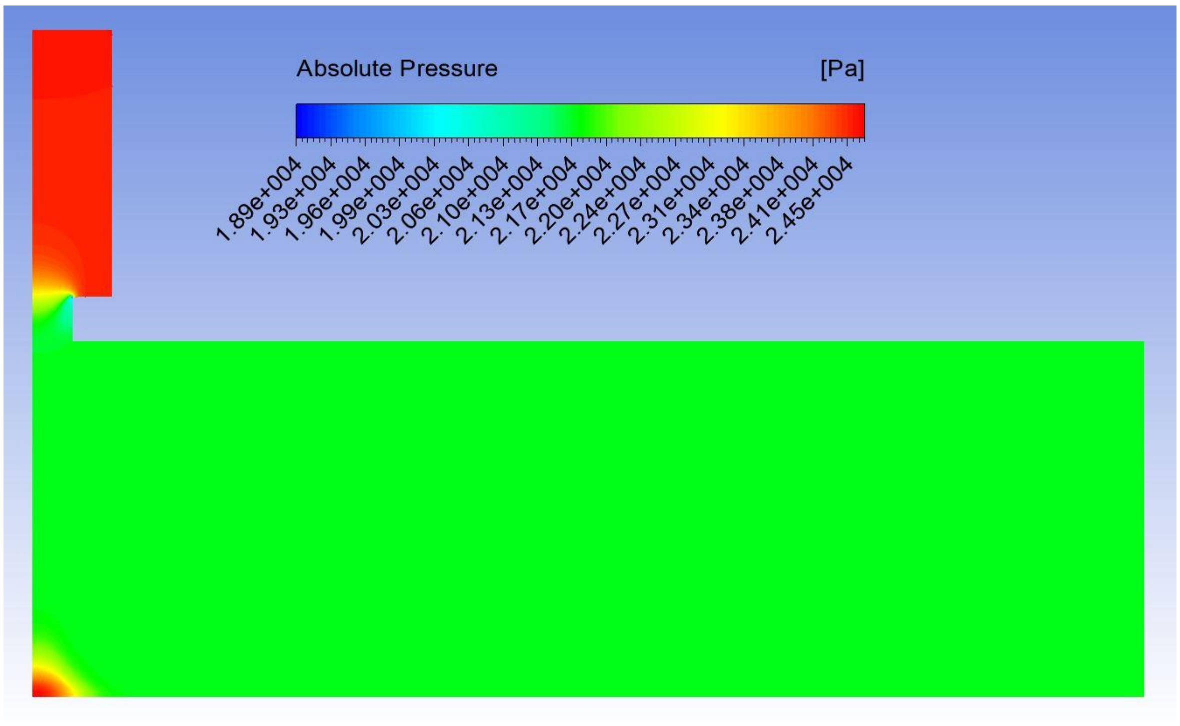


Fig. 6.8: Pressure Field.

Before the jet, the value is almost uniform: in this simulation the pressure value at the inlet is constant, but in any case the settling chamber before the nozzle levels every possible oscillation. Therefore, it can operate with almost constant input conditions and so prediction of the performances is possible; it helps also in the design phase: the designer can project the best jet possible for those conditions, while he must find a compromise when dealing with variable operating conditions. We can see another location with maximum value for pressure in the stagnation point: as we will see in the velocity field (Fig. 6.10), in this point the fluid velocity is zero and all kinetic energy has been converted into pressure energy. It coincides with the impact point on the surface of the plate.

The lowest values in the pressure field, represented in blue, are barely visible in Fig. 6.8.

They are settled in the corner at the inlet of the nozzle; the sharp corner does not allow an optimal fluid flow, creating a separation bubble where we have a decay in the absolute pressure.

In Fig. 6.9 we can see a particular of this area.

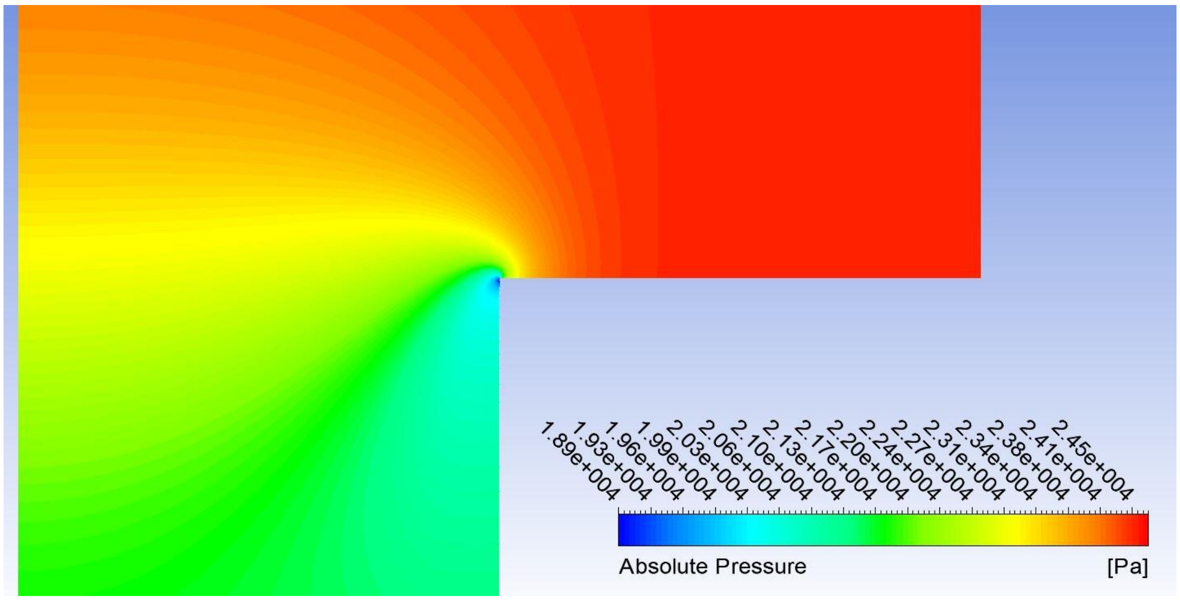


Fig. 6.9: Pressure Field at Nozzle Inlet.

The strong separation we have in this location does not seem to affect Fluent solution, but this does not apply to OpenFOAM, as we will see later on.

Now we can consider the velocity field (Fig. 6.10). In the part just before the jet, we can notice the boundary layer development due to the no-slip condition set at the wall. When the flow meets the nozzle, it accelerates until the maximum speed. Due to the sharp corner, the flow cannot follow the geometry and that is why we have the separation bubble we already noticed looking at the previous contour plot. Within the free jet zone, which is largely unaffected by the presence of the impingement surface, we can notice a potential core: here the jet velocity is conserved and the turbulence intensity level is relatively low. Approaching the wall, the shear layer existing between the potential core and the ambient fluid entrains the latter and causes the jet to spread radially. Moving away from the impinged zone, the flow becomes parallel to the plate and loses strength, creating a big recirculation vortex. This vortex changes the fluid flow at the outlet and is primarily responsible of the backflow reported in the temperature field. In fact it creates a secondary vortex that entrains external fluid at the outlet.

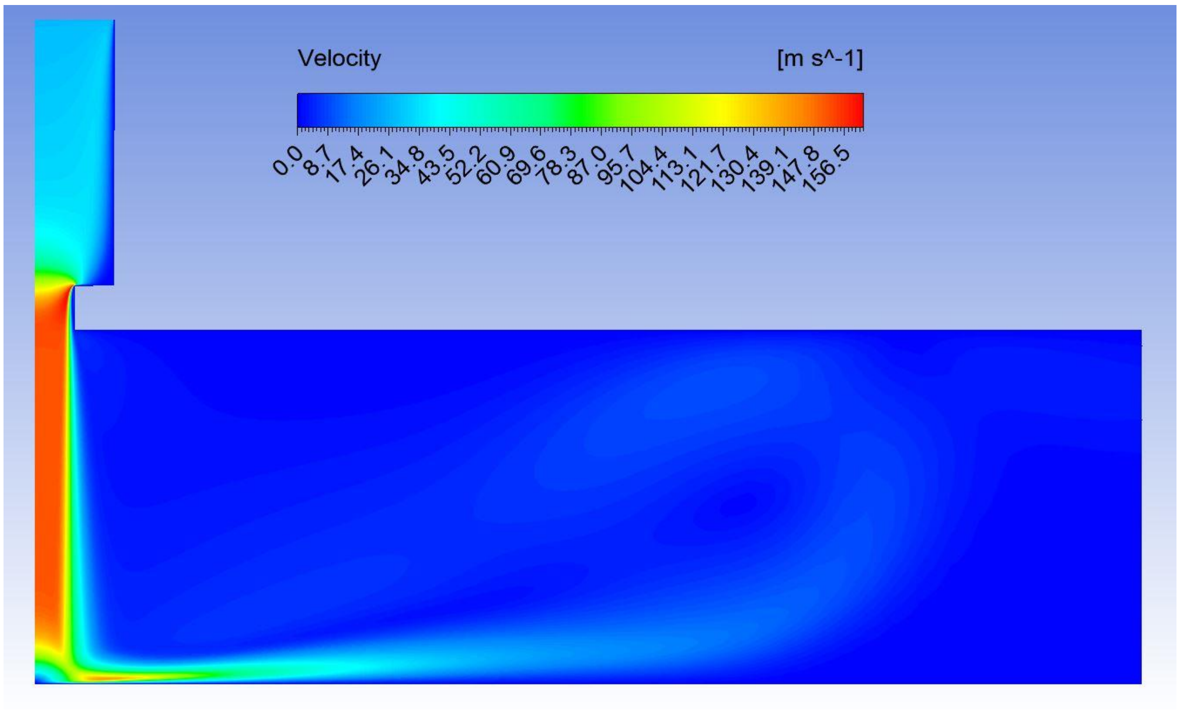


Fig. 6.10: Velocity Field.

6.4. OpenFOAM

6.4.1. Settings

In OpenFOAM we used the same SIMPLE algorithm; however the implementation may present some minor differences from the original one. Not all the other settings are presented here in order to not burden the discussion. For further infos, have a look of *fvSchemes* and *fvSolution* files provided in the Appendix.

6.4.2. Results

Starting from the temperature field, we can notice a quite different behavior if compared to the Fluent one we have previously discussed. As we can see in Fig 6.11, the temperature field presents an overshoot next to the outlet. This overshoot is completely nonphysical, raising the temperature till 677K, 33K more the maximum temperature set as boundary condition (644K at the heated wall). It can be explained considering numerical production due to recirculation at the outlet. Indeed, this difference should not affect the good result obtained next to the heated wall, which is the region of interest while studying impinging jets. According to some opinions I gathered from advanced Fluent users, the simulation with the Ansys software does not show this behavior thanks to some numerical tricks implemented in this commercial code in case of backflow.

Notice that we have slightly different color maps even if we have used the same color scale for both software, because of OpenFOAM overshoot.

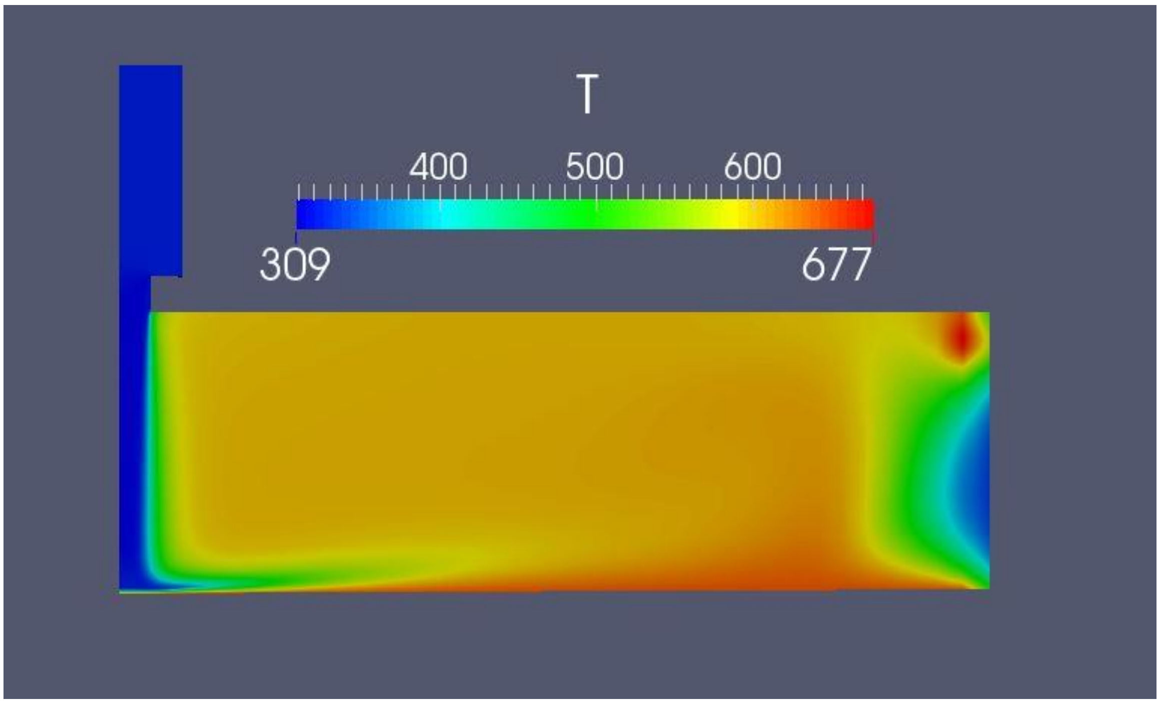


Fig. 6.11: OpenFOAM Temperature Field.

Considering OpenFOAM fields for *p* and *u* (Fig. 6.12; Fig. 6.13), we did not notice any major difference if compared to Fluent ones.

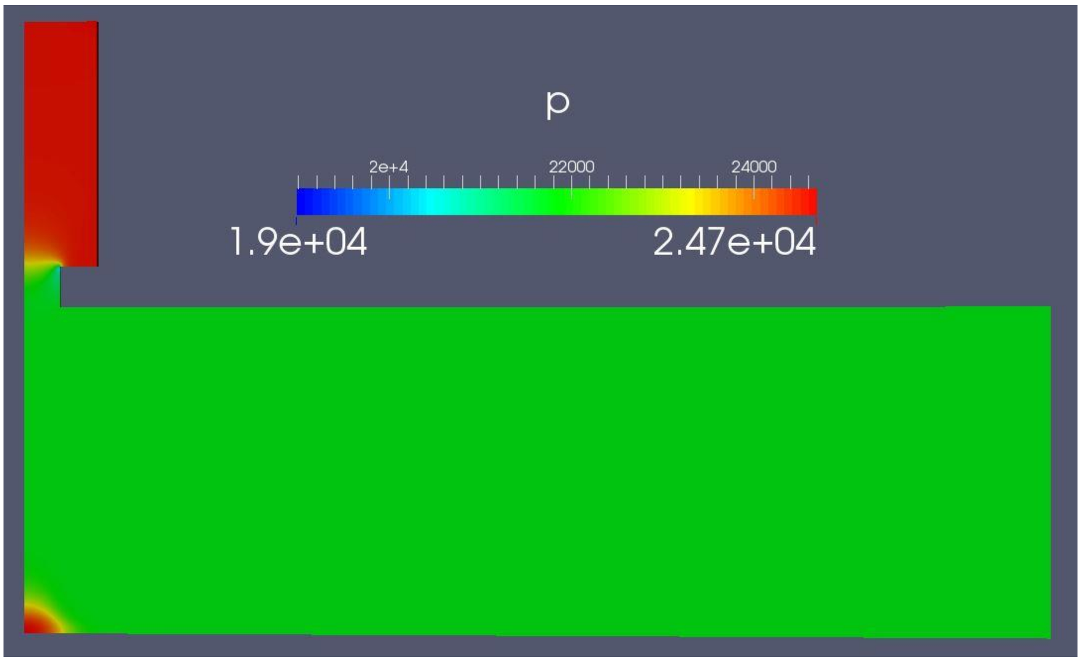


Fig. 6.12: OpenFOAM Pressure Field.

At the inlet of the nozzle we have the same behavior of the Fluent simulation, and also the lowest pressure value, located at the corning point of the flow, is almost the same.

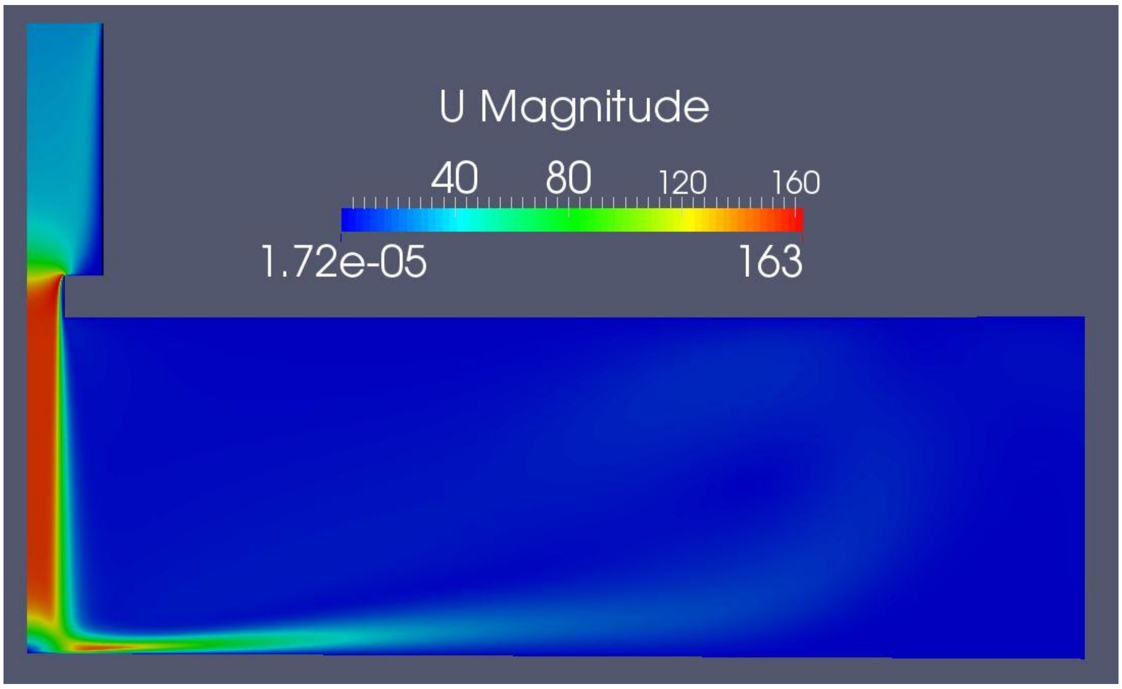


Fig. 6.13: OpenFOAM Velocity Field.

In order to solve the overshoot problem, we have tried to change some settings in the solver; first, we moved from *linearUpwind* to *MUSCL* scheme, also to solve the problem with settings as similar as possible to the Fluent case. With these new settings the overshoot disappeared, while *T* e *p* fields remained unchanged. In Fig. 6.14 the new temperature field is reported.

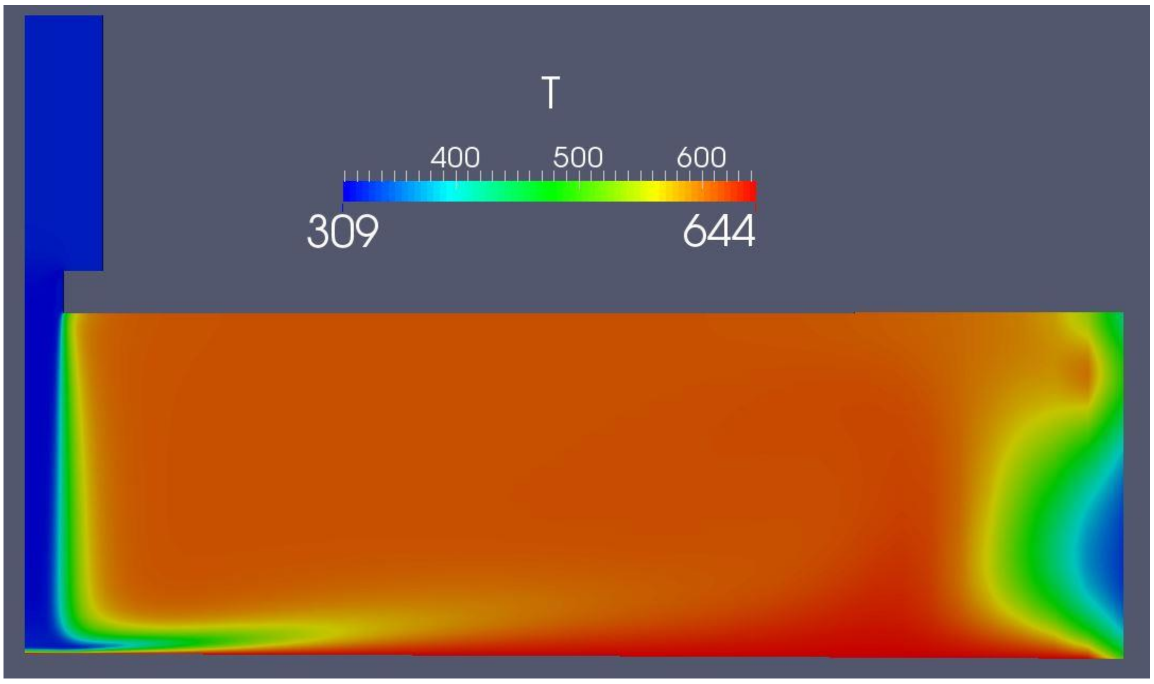


Fig. 6.14: *MUSCL* simulation temperature field.

The solution, if compared to the Fluent field, has higher values for *T* in almost all the domain far off the wall; on the other hand we continue to have very similar values next to the plate.

The behavior reproduced at the outlet is way better than the previous OpenFOAM case (with *linearUpwind* scheme), if compared to the Fluent one.

Then, we have also changed relaxation factors, lowering their values. Lower values mean it takes longer to converge, but they should also lead to a more stable solution. The chosen values were extremely low if compared to default ones. In this case the change did not bring to the hoped results: after a longer simulation we got exactly the same solution. Even the residuals were not less fluctuating once achieved a converged value.

6.5. OpenFOAM transient case

Even if in OpenFOAM and in Fluent residuals are calculated in a different way, during our simulation we noticed some differences in their behavior.

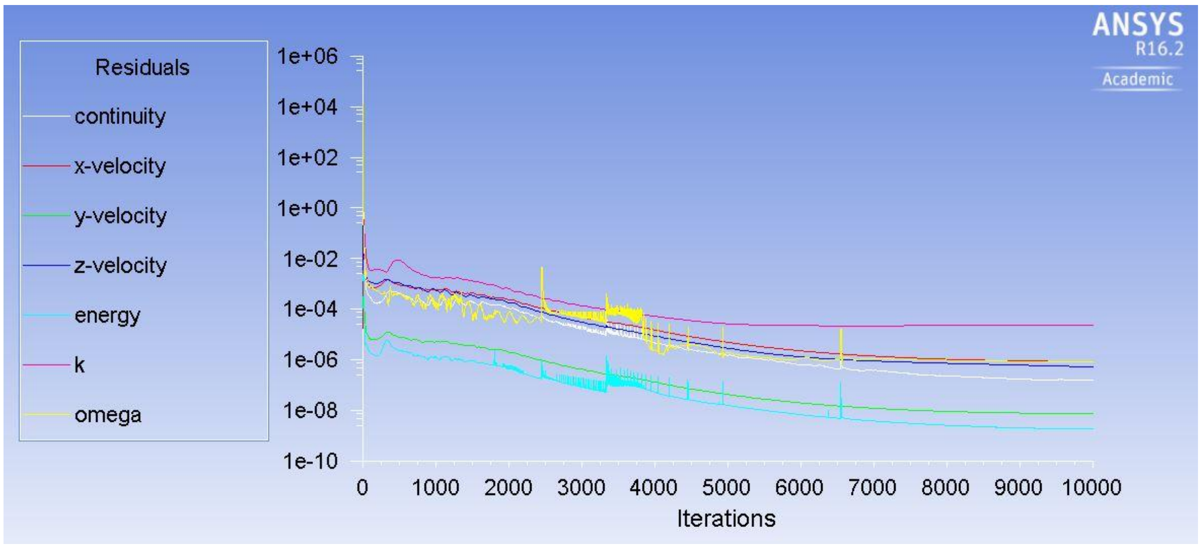


Fig. 6.15: Fluent Residuals.

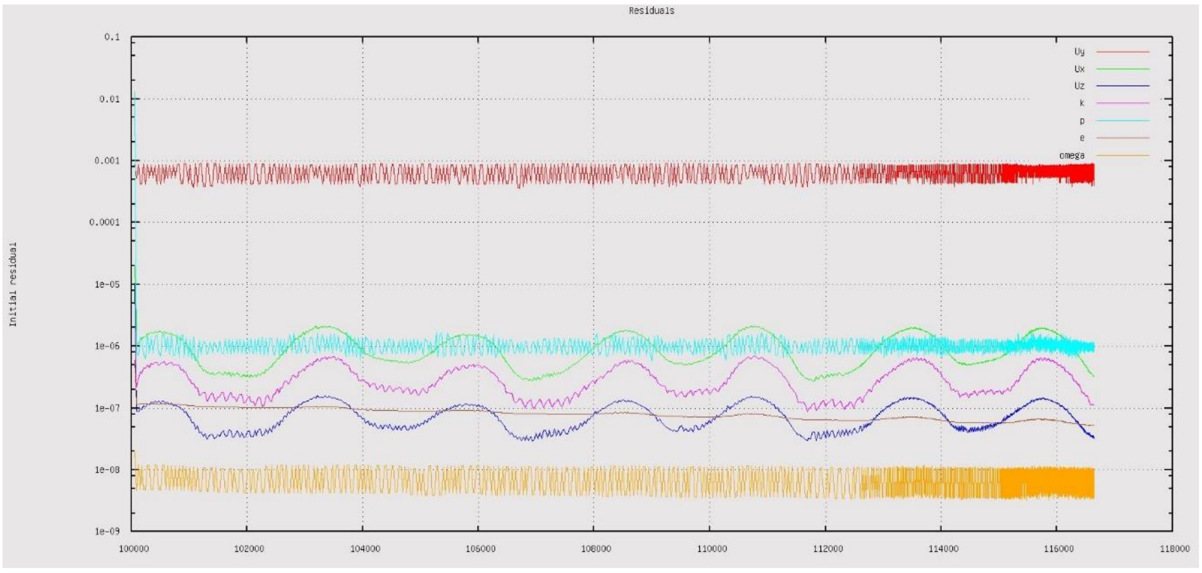


Fig. 6.16: Last 50000it. OpenFOAM.

In particular the Fluent ones (Fig 6.15) converged to a value without any oscillation, while in the OpenFOAM ones (Fig 6.16) a periodic oscillation was clearly visible, also after a very high number of iterations.

Given the periodic behavior, we thought it was due to the vortices effect in the fluid domain; so we decided to run also a transient simulation, considering as unsteady the flow through our domain. The unsteady simulation has been run with both *linearUpwind* and *MUSCL* schemes.

6.5.1 Settings

For transient simulations we utilized *rhoPimpleFoam* solver, which is a transient pressure-based RANS solver. We used both Euler and CrankNicolson time discretization schemes.

We had some problems in the time step setting with all the transient cases with which we have been dealing.

The choice of the time step is possible in the *controlDict* file. We were not able to set any value higher than $2x10^{-8}$, value ridiculously low for this kind of simulation (more suitable for a DNS simulation than for a RANS one). With such a small value the simulation took a very long time. With higher values, the simulation stopped after few iterations due to the explosion of the Courant number.

The Courant number is defined as:

$$Co = \frac{u\Delta t}{\Delta x} \leq C_{max} \tag{6.5}$$

where C_{max} changes with the method used to solve the discretized equation. In general, C_{max}=1 in case of explicit method, while higher values can be tolerated using implicit solvers.

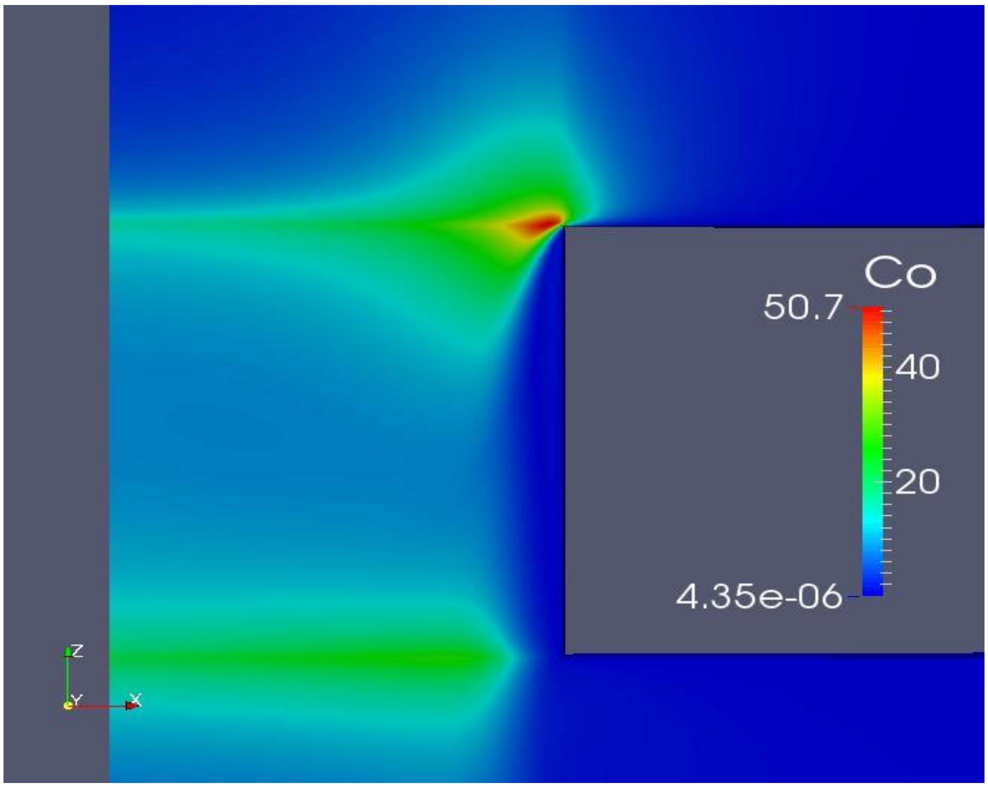


Fig 6.17: Co field at the nozzle.

In Fig. 6.17, we can see the Courant field for our simulation at the nozzle, which is the most critical location, just before the explosion of the solver.

Especially in the Co field, the value is generally very low (order of magnitude 10e-6) but there are some locations with higher values exploding the simulation. The critical location coincides with the minimum value of the pressure field we reported before.

As we can see in the particular of the Co field, the value of this quantity is related to the mesh; in fact we have higher Co values where the mesh is finer (Δx is at the denominator in the Co definition). This has later pushed us to slightly change the mesh itself.

Plotting the vorticity field in the same location (Fig. 6.18), we get a similar behavior.

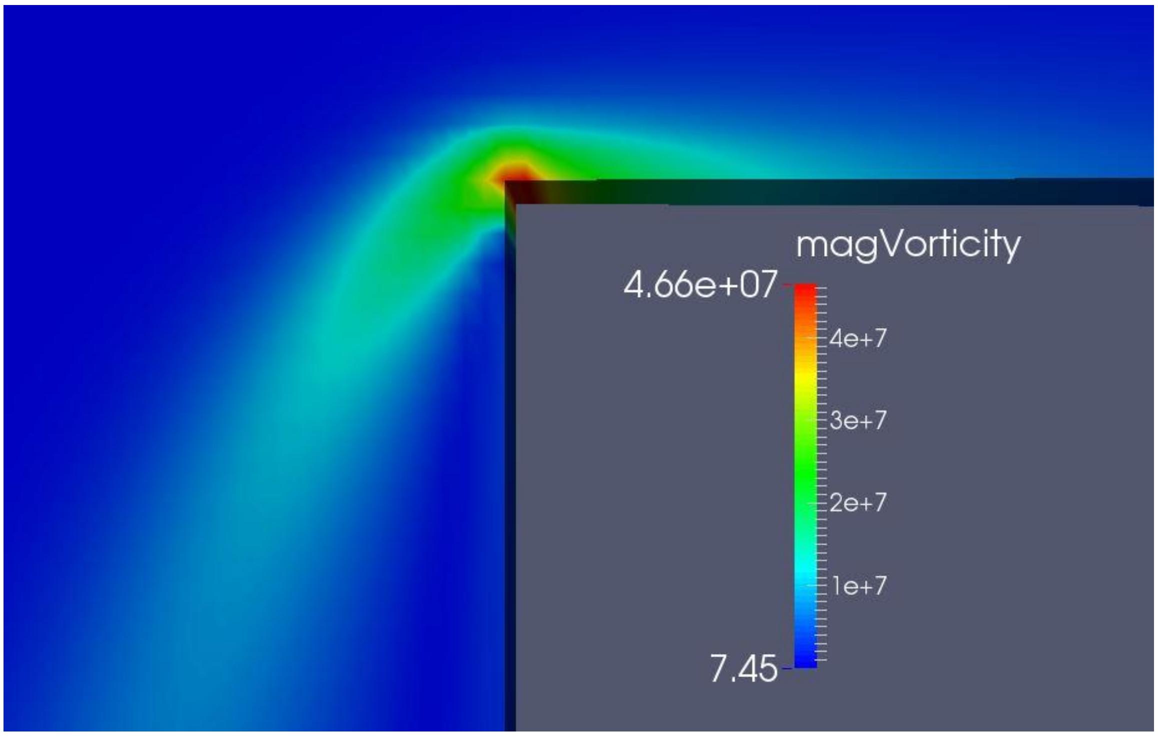


Fig. 6.18: Vorticity field at the nozzle inlet.

Also for this case all the other settings are provided in the Appendix.

6.5.2. Time averaging

Now we are going to present time-averaged plots for velocity. As the unsteady solution shows a periodic behavior when it gets to convergence, the time-average is done on last 5 periods computed. We did the same also for 10 periods, as usually recommended in order to have a more trustable averaging, but we could not notice any difference in this particular case. This means that at the time we averaged the solution is perfectly periodic and it is repeated equal to itself for each period. Time-averaging on 5 periods allowed us to save some time, as the time-average mechanism can take a long time, especially when a high number of variables have to be averaged, as in this case.

Considering velocity U, in Fig. 6.19 the velocity field for a certain time is represented.

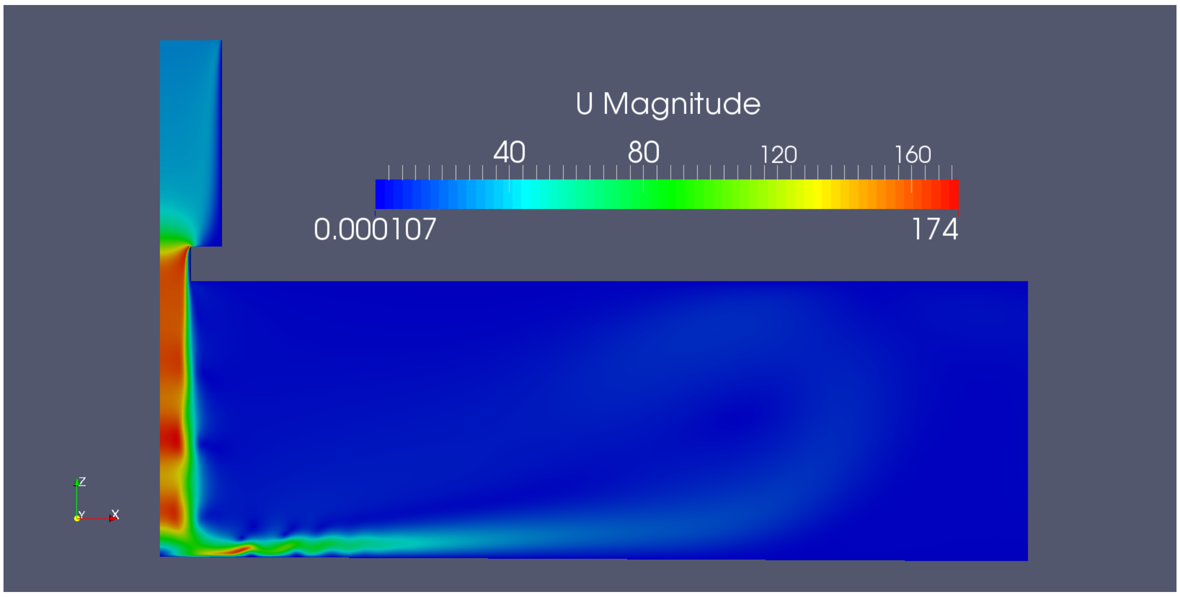


Fig. 6.19: Velocity field for a generic time step.

In the figure above is clear how the steady velocity field is deformed in the unsteady case. Shear stresses between the jet and the surrounding fluid form vortices that move to the exit becoming bigger and bigger and affecting the velocity behavior; this is clear if we put in a video the velocity field for *n* subsequent time steps: it is possible to catch the behavior of vortices along the jet flow and the periodic oscillations that they create. Averaging in time, it is possible to smooth the oscillation and we obtain the following plot.

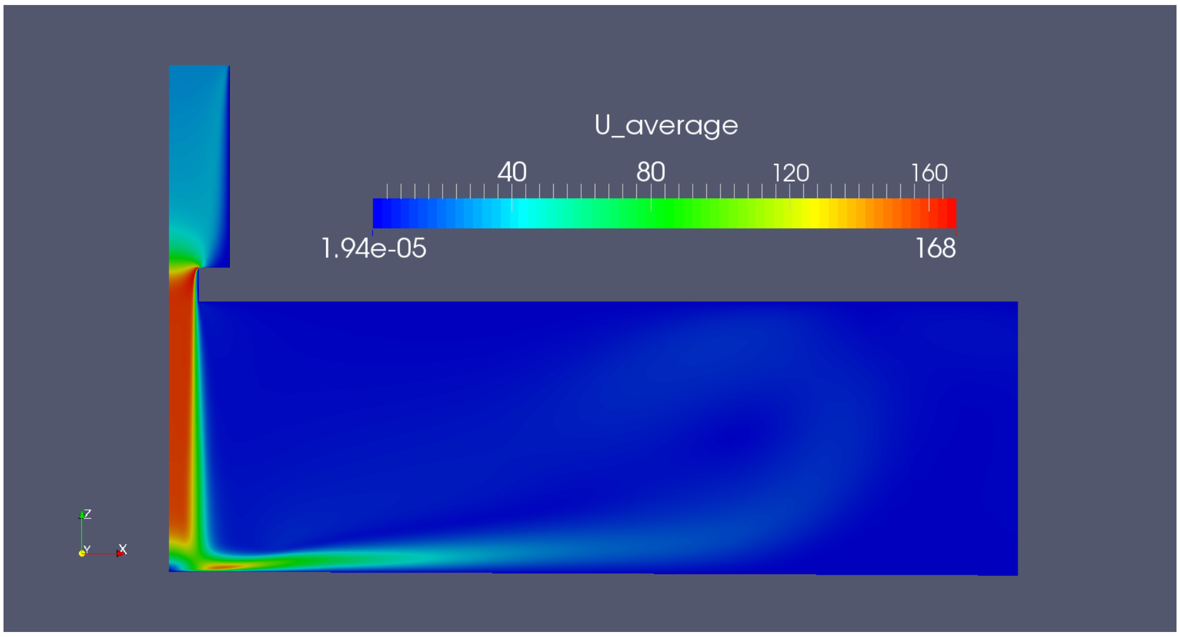


Fig. 6.20: Time-averaged Velocity field.

It appears clear that time-averaging the unsteady velocity field we obtain the same behavior we got during the steady simulation; this result was expected, in fact averaging on some periods we obtain the flow that assure the same mass flow, without the oscillations which are evened out by the average itself. As the mass flow imposed at the outlet is the same for both the steady and the unsteady case, the result obtained is physically justified.

The small difference we can notice is in the highest value of the field, which is higher than in the steady case. We can explain this small difference (few % points) with the shrinking of the passage section due to the evolution of the vortices; in particular this effect is clearly visible after the impinging area.

To represent the deviation of the actual velocity from this averaged one, we calculated the standard deviation of the velocity (Fig. 6.21).

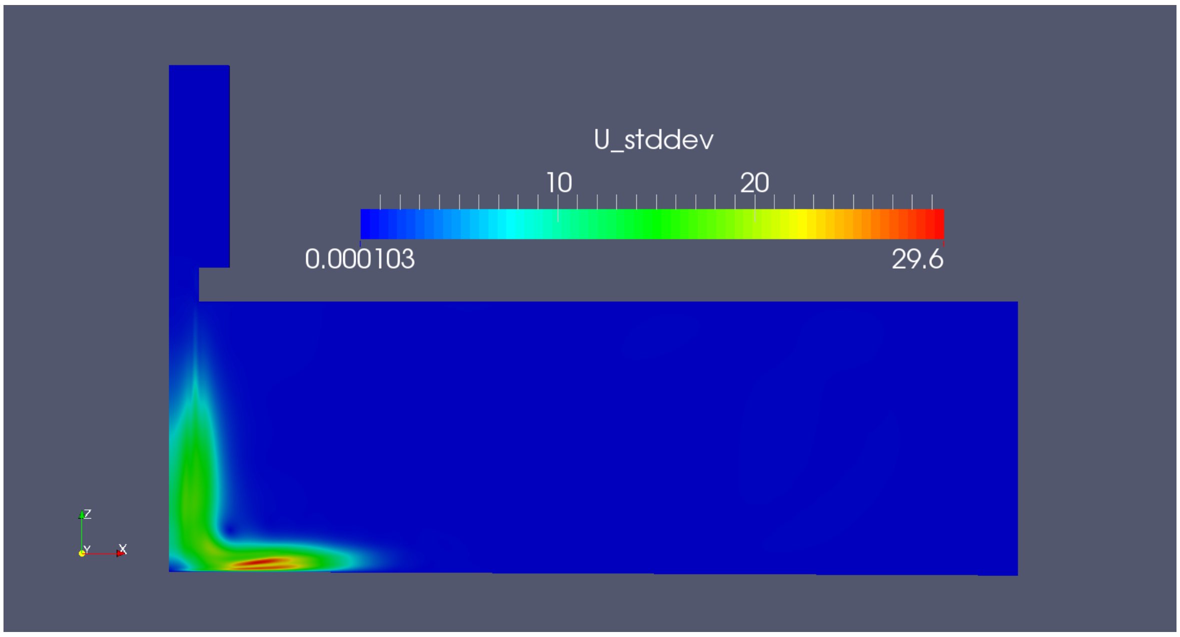


Fig. 6.21: Velocity standard deviation from the time-averaged value.

The maximum variation is located just after the impinging zone, where the flow is more afflicted by the vortices. We do not have any variation in the potential flow zone and this is in agreement with the theory presented in the previous chapter; in fact, the effect of shear stresses needs some space to influence the jet flow.

Now we consider the turbulence kinetic energy *k*. This quantity represents the mean kinetic energy per unit mass associated with eddies in turbulent flow. From the physical point of view it is characterized by measured root-mean-square velocity fluctuations u_i'.

It can be written as:

$$k \overset{\text{def}}{=} \frac{1}{2}\overline{u_i'u_i'} = \frac{1}{2}\left(\overline{u_x'^2} + \overline{u_y'^2} + \overline{u_z'^2}\right) = \frac{3}{2}\overline{u'^2} \tag{6.6}$$

As we are using RANS, the turbulence kinetic energy can be calculated based on the turbulence model adopted (*k*-*ω* SST in our case).

Considering the averaged value, we can see how *k* increases moving towards the wall because of shear stresses, confirming somehow what we said before commenting the velocity behavior. The maximum is located just after the impinged portion of the wall and also this behavior is expected; in fact, it is the location where we have also the maximum standard deviation from the average velocity and, being *k* the RMS of velocity fluctuations, that is the location where this mean reaches the maximum value. We can also notice how the *k* value is almost zero in the internal part of the jet and next to the wall; as the turbulence kinetic energy can be produced by fluid shear, friction or buoyancy, or again through external forcing at low-frequency eddie scales, its value will be higher at the surface of the jet while tends to zero in the internal part, where the flow remains almost laminar.

The same applies for the flow after the impinging region: *k* is very close to zero in the laminar sublayer, while achieves the maximum value just off this region, where stresses are relevant and we have big eddies in development.

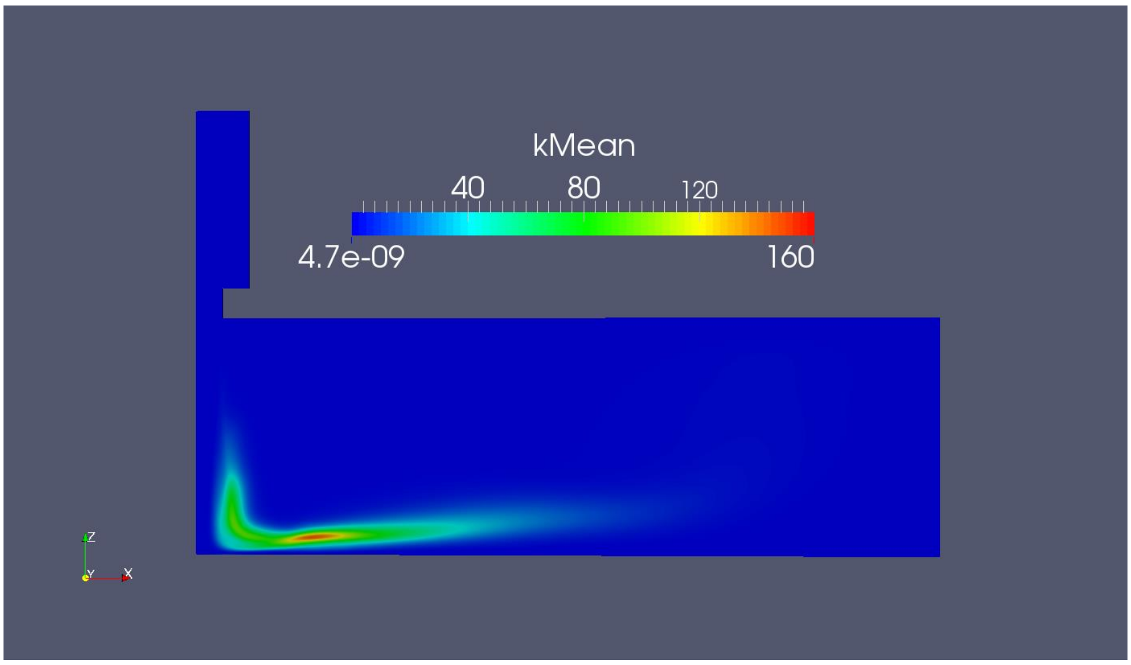


Fig. 6.22: Time-averaged K field.

6.5.3. New Inlet Mesh

In order to solve the problem of the time step for the unsteady simulation and after having tried all possible changes in solver settings, we decided to further change the mesh; as the biggest problems were located at the sharp corner at the inlet of the jet, we chose to avoid any problem eliminating it. Farther, as the dimension of the cells influences the Courant number, we slightly changed it, coarsening the mesh. This expedient should allow the use of bigger time steps, lowering the Courant number value without affecting the accuracy of the solution. At first, we have run the steady simulation. We obtained exactly the same results of the previous mesh, and also the overshoot was still present. Then, we ran the unsteady simulation but again some problems with the time step occurred. This time the problem was not at the jet inlet, having taken off the corner, but in other locations, especially where the mesh was finer because of the clustering adopted. Although the modifications carried allowed us to run more iterations with a bigger time step, we did not managed to run the complete simulation as hoped. Changes only retarded the explosion of the solver but did not solve the issue.

As we could not obtain the solution with an adequate time step, we came back to our original mesh.

6.6. Heat Transfer Comparison

When studying an impinging jet, the most important parameter is the effect we want to obtain from this device. In our case we want to check the cooling effect on a flat plate and so the most important parameter is the wall heat flux at the heated plate.

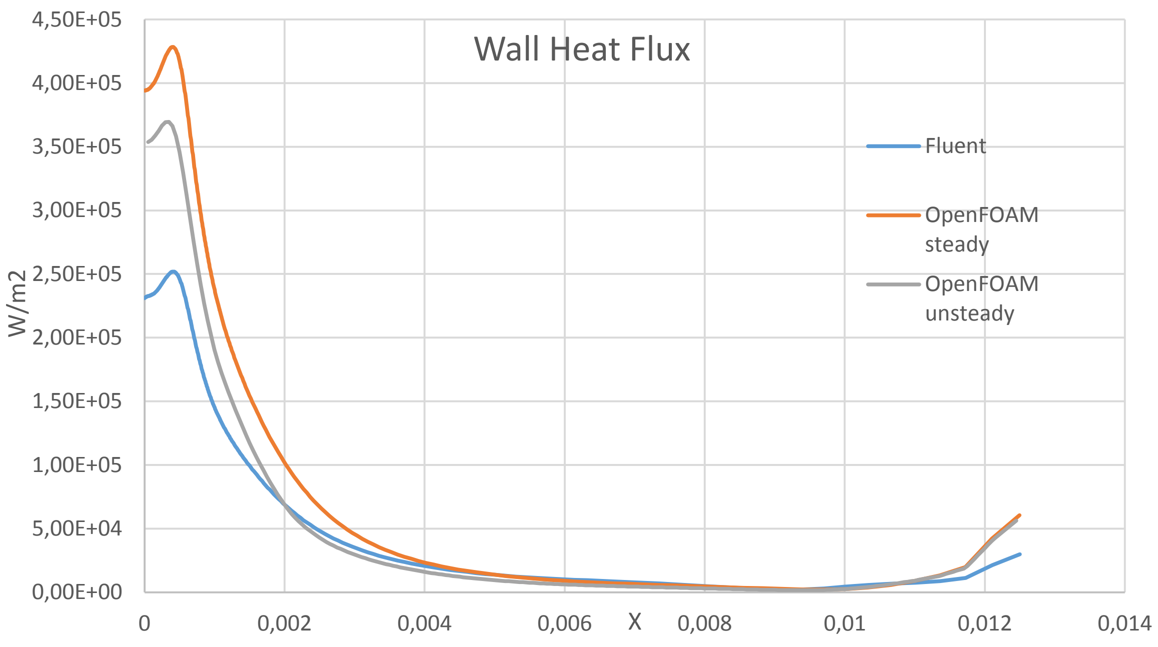


Fig. 6.23: Wall Heat Flux at the heated wall.

The results found show quite a big differences between the two software. While the overall trend is similar, with the maximum located at the same distance from the axis, the wall heat flux maximum value differs, between the two steady solvers, by more than 40%. The difference is lower when considering time averaged wall heat flux of the transient case: nevertheless, the difference is still about 30%, which is a very high value. Moreover, in the flat plate we found a very small difference and in that case the Fluent value was higher. As we are using an unsteady solution for OpenFOAM, we could expect higher heat transfer rates: in fact unsteadiness in the flow promotes mixing and should help heat removal. But again it is not surely easy to explain also the different behavior in the two OpenFOAM solutions: as seen in the theoretical chapter, usually we have better heat transfers in unsteady solutions but not in this case. We can try to justify this remembering that our steady solution has never come to convergence so it is not really steady, as we have always some oscillations that raise heat transfer coefficient. Furthermore, moving from steady to unsteady, we had to use a different solver in order to have solution in time, and some differences can be related also to this.

However, apart in the impinging region, we have similar behaviors, especially for the unsteady solution. There is another small deviation close to the outlet, due to the different recirculation treatment we have already considered while comparing the temperature field. Both OpenFOAM case have the same identical behavior at the outlet as they have the same treatment over there.

In order to conclude the comparison in terms of heat transfer performances we should compare the results achieved by the two software with experimental data. Unfortunately, due to problems in the manufacturing department of the institute, we have not been able to get experimental data by the end of this thesis. The comparison will be added when data will be available.

6.7. Traverses Investigated

During our work, we have plotted data along several traverses, in order to follow the evolution of the flow and compare Fluent and OpenFOAM behaviors. First, we have considered 5 horizontal traverses, including the jet outlet.

In Fig. 6.24 we can see the location of these traverses:

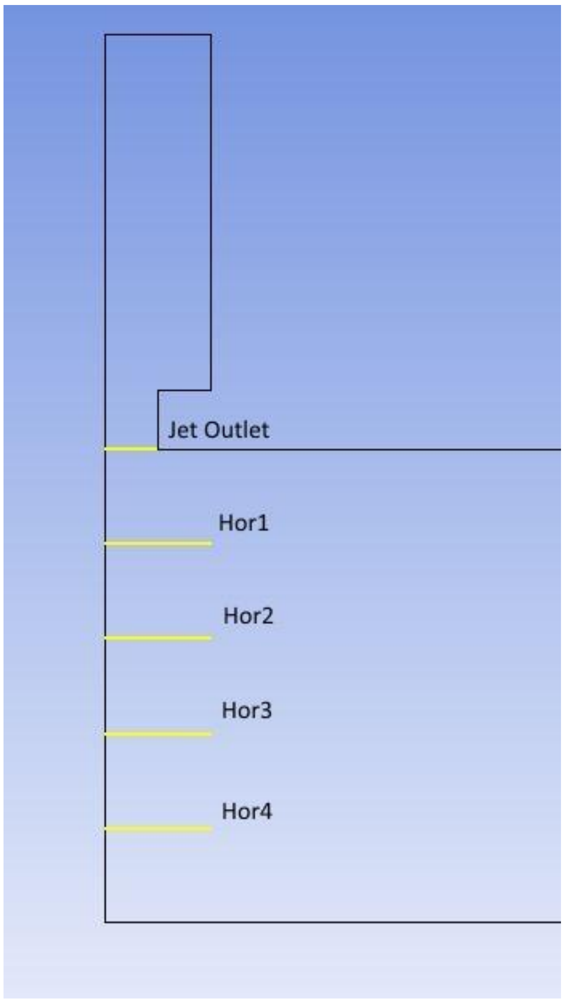
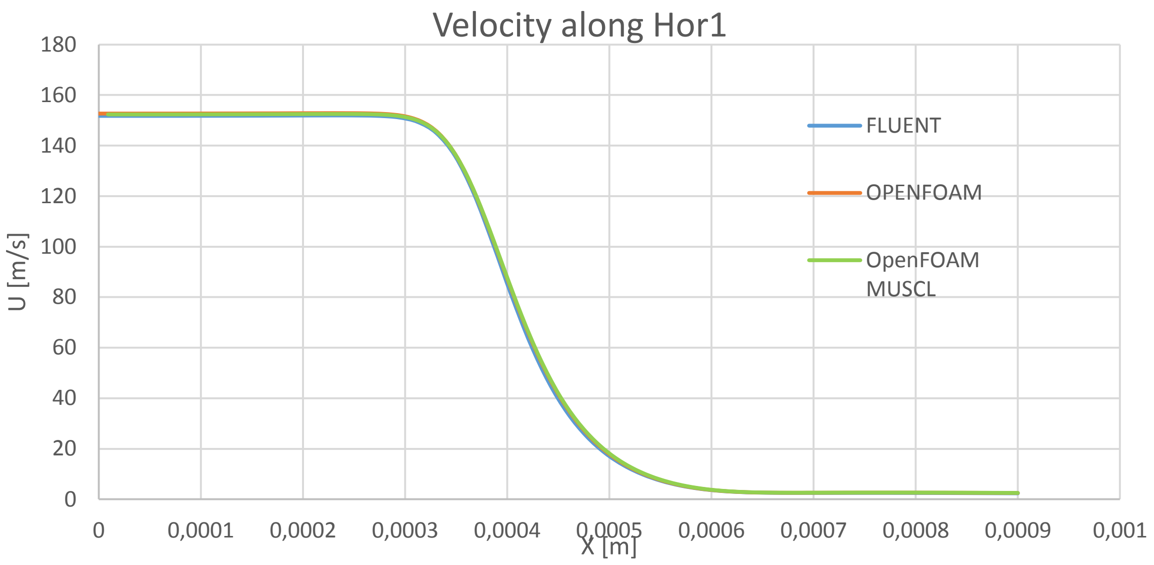
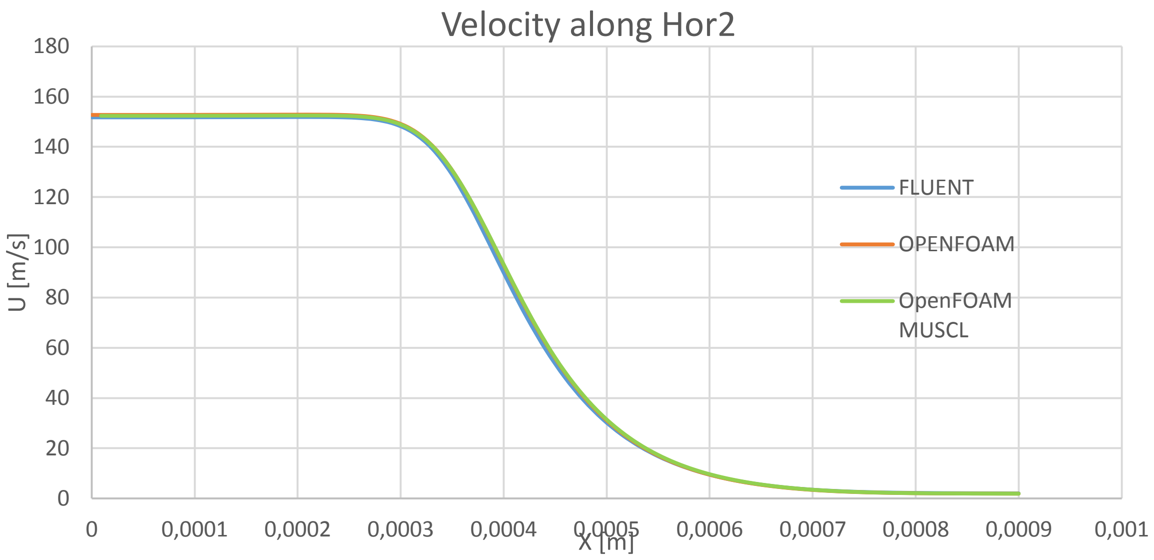


Fig. 6.24: Horizontal Traverses Location.

Moving from the jet outlet to the plate, we have divided this length in equidistant parts (0.8 mm in between each one). Apart the jet outlet, all the other traverses have a length of 0.9 mm, equal to the diameter of the settling region before the nozzle.

The quantities plotted along these traverses are: velocity, temperature and turbulence kinetic energy.





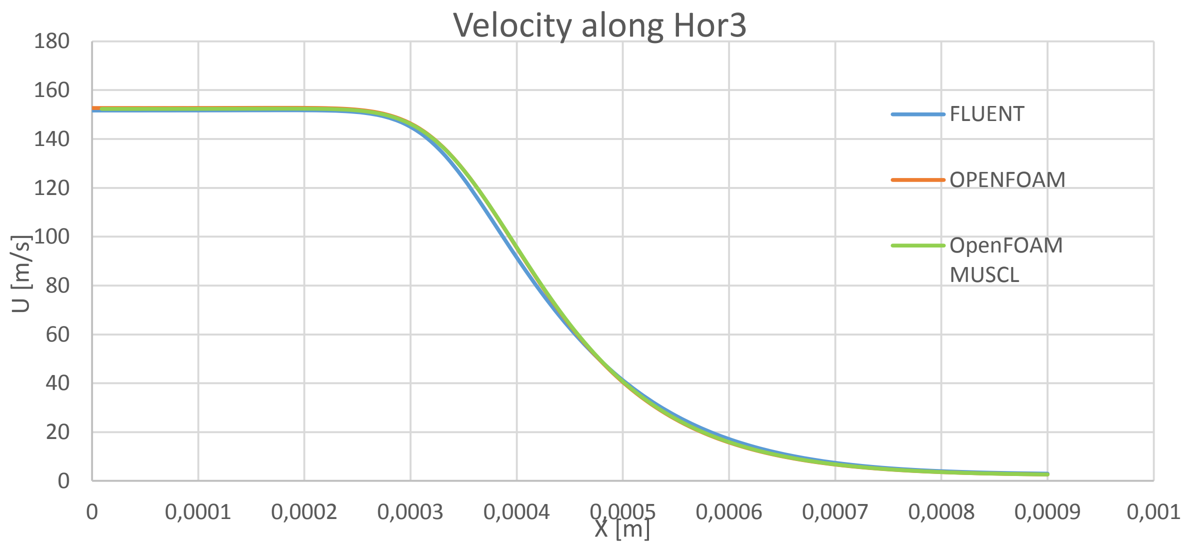


Fig. 6.25: Velocity profiles along Hor1, Hor2 and Hor3.

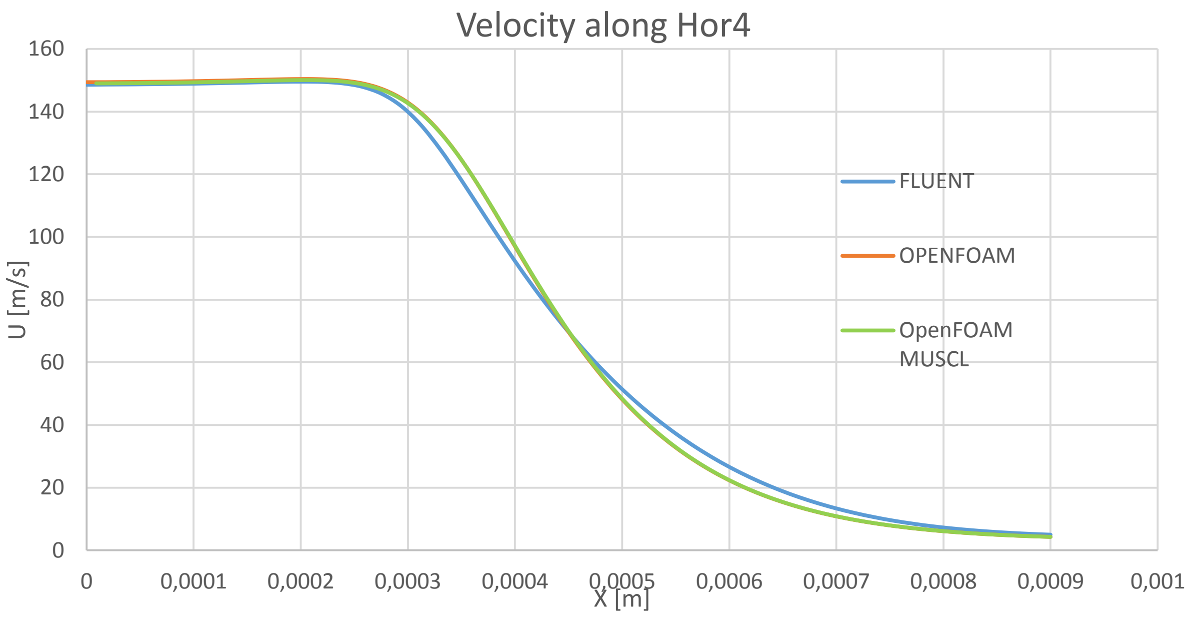


Fig. 6.26: Velocity profile along Hor4.

As clearly noticeable considering the plots above, velocity does not differ that much between the two software. Farther off the jet outlet, some differences start to be present (Hor3 and Hor4 plots); although this differences do not seem to be big, we have to consider that in some equation involved in the solution of our problem second derivative (?) terms of velocity are present: small differences in the value of velocity lead to bigger differences in the derivative terms and this can heavily affect results. On the other hand, behaviors for the two OpenFOAM simulations are identical, so we can say that the choice of a different discretization model does not affect velocity solution.
Now we consider the turbulence kinetic energy profile.

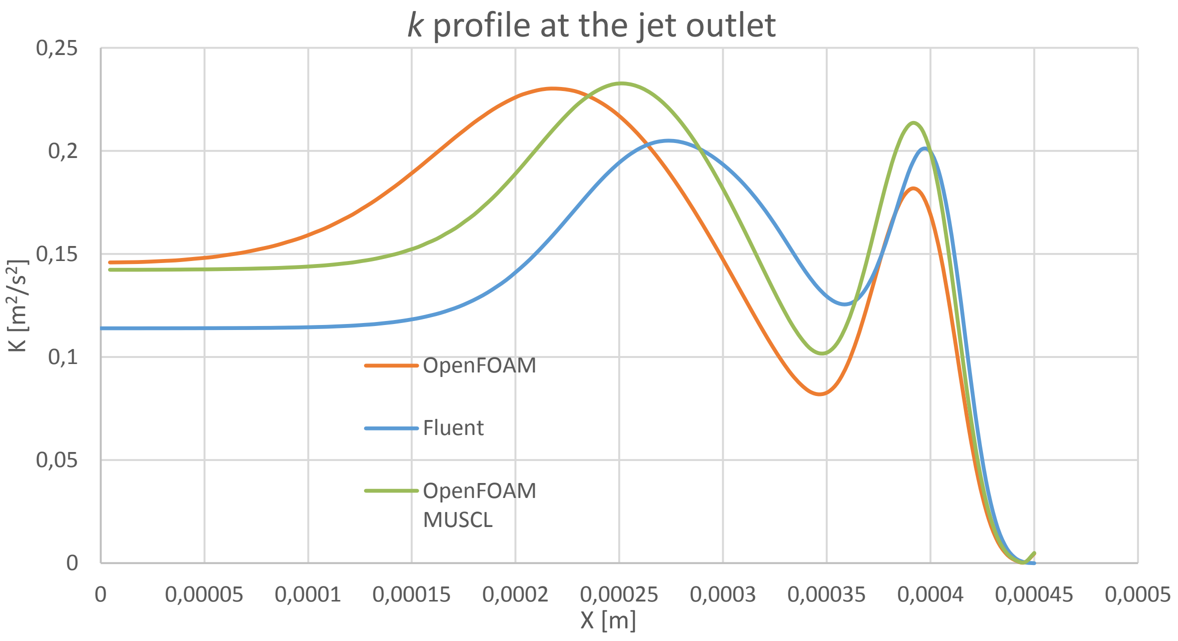
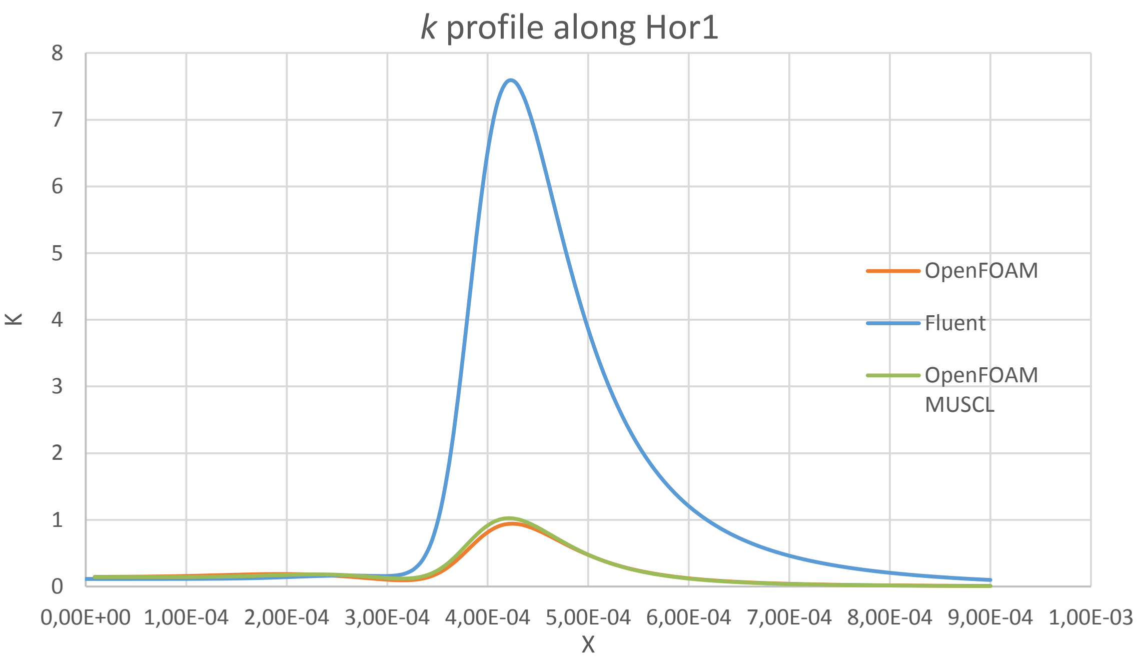


Fig. 6.27: *k* profile at the jet outlet.

At the jet outlet we have different behaviors but values are comparable. For *k*, we have a different trend using MUSCL scheme and we can say that it is more similar to the Fluent one. However, after the jet outlet, we have significant differences between Fluent and OpenFOAM (both cases), as we can see in the following plots:



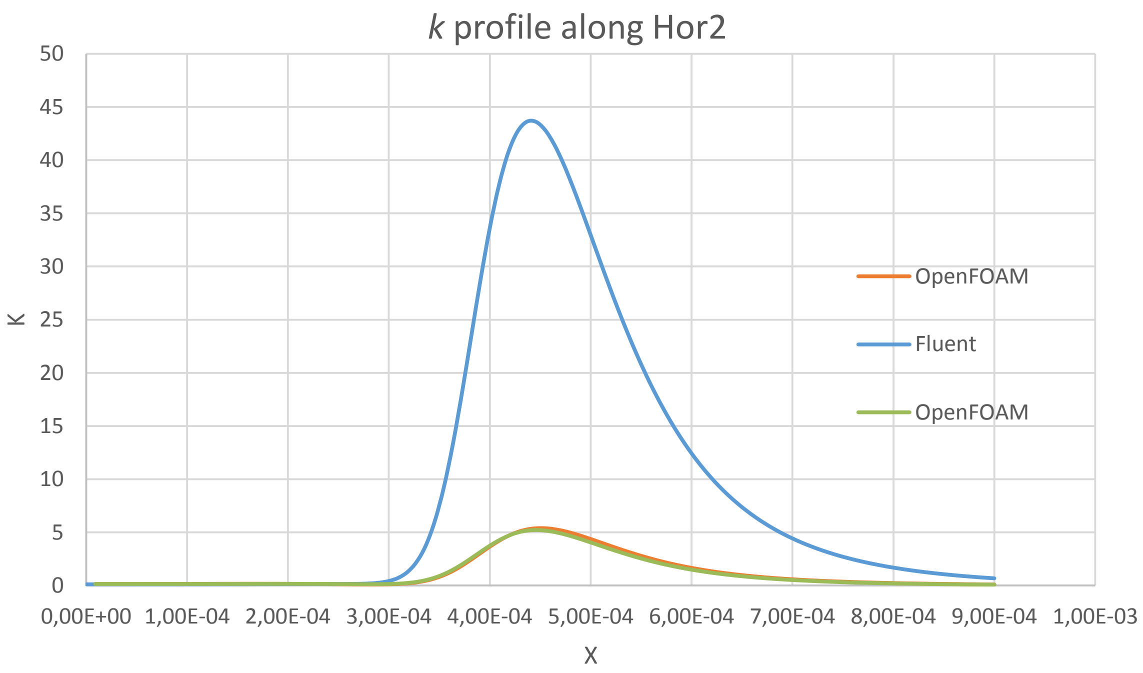
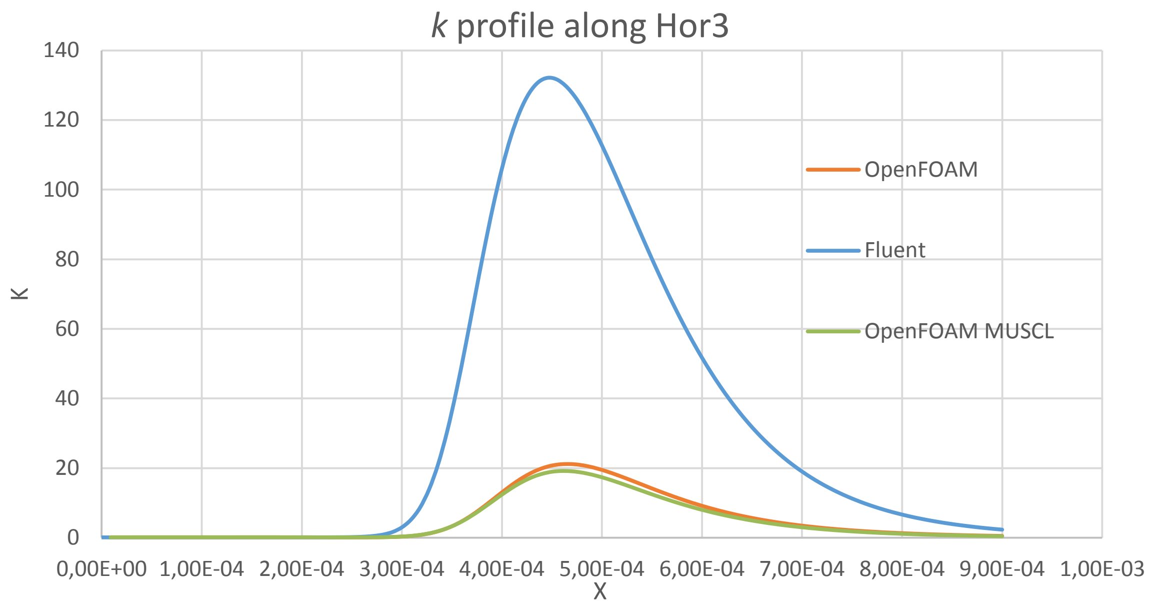


Fig. 6.28: *k* profile along Hor1 and Hor2.



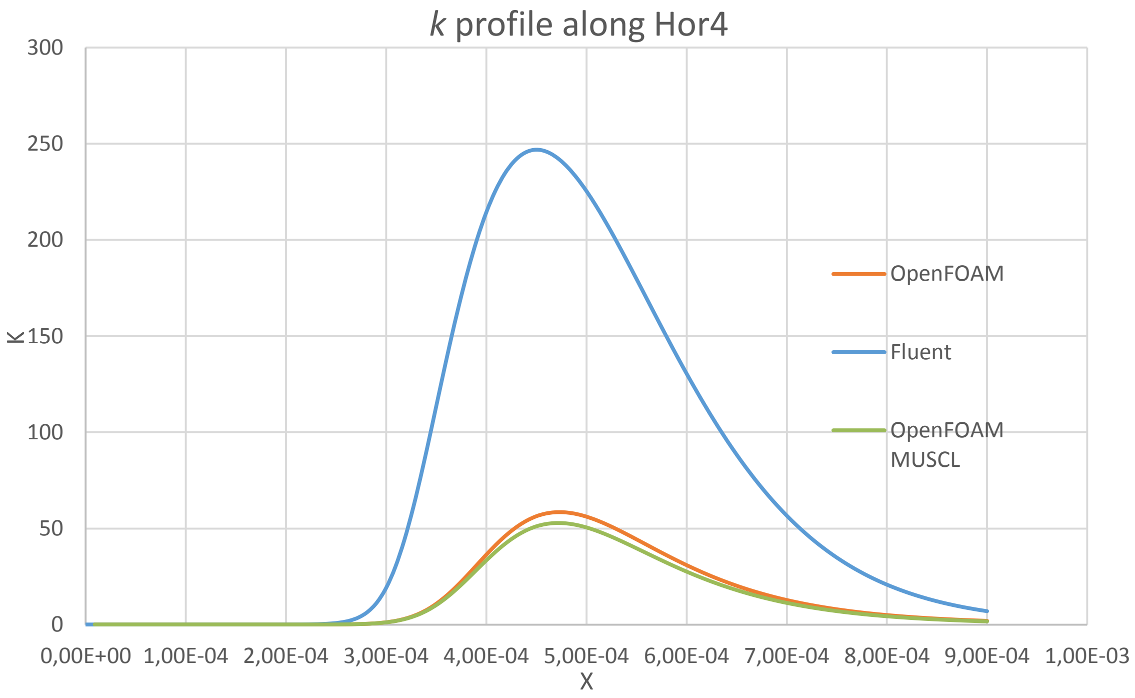


Fig. 6.29: *k* profile along Hor3and Hor4.

Plots along the four traverses have similar trends, with the difference between Fluent and OpenFOAM that becomes smaller moving towards the plate but remains in the order of 80%. The difference is quite big and can be explained with a different implementation of the turbulence model in the two software; in particular it is likely that *k* has a slightly different definition in Fluent. The value of the turbulence kinetic energy becomes bigger moving towards the plate; this is in accord with the theory, as the effect of the shear stresses and of the eddies in development increase in that direction.

It is interesting to plot *k* against velocity *U* for one of these traverses, for instance along Hor2 (Fig. 6.30):

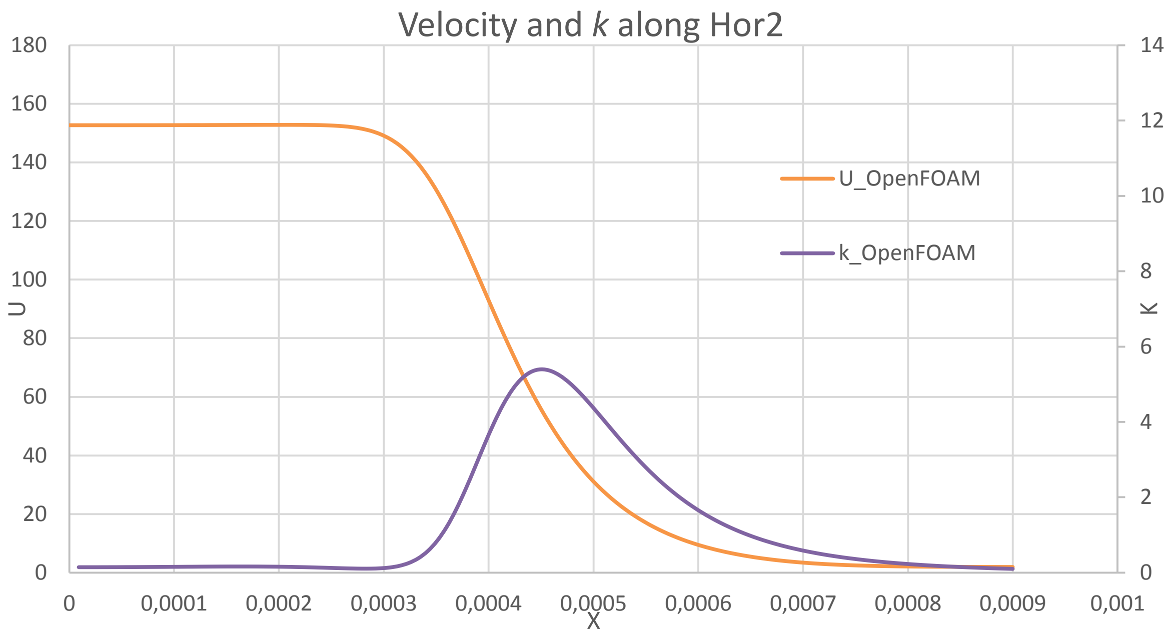


Fig. 6.30: Velocity vs. *k* plot along Hor2.

Considering the plot above, we can notice how the variation in the k trend is related to the variation of velocity due to the contact between the jet and the ambient fluid. In this shear layer, eddies start to detach and raise the value of the turbulence kinetic energy to its maximum; then the value decrease again because moving off the shear layer the interaction between the jet and ambient fluid becomes lower and lower. Again, the same applies for all the other traverses.

Now, let us have a look at the temperature profiles along the same traverses:

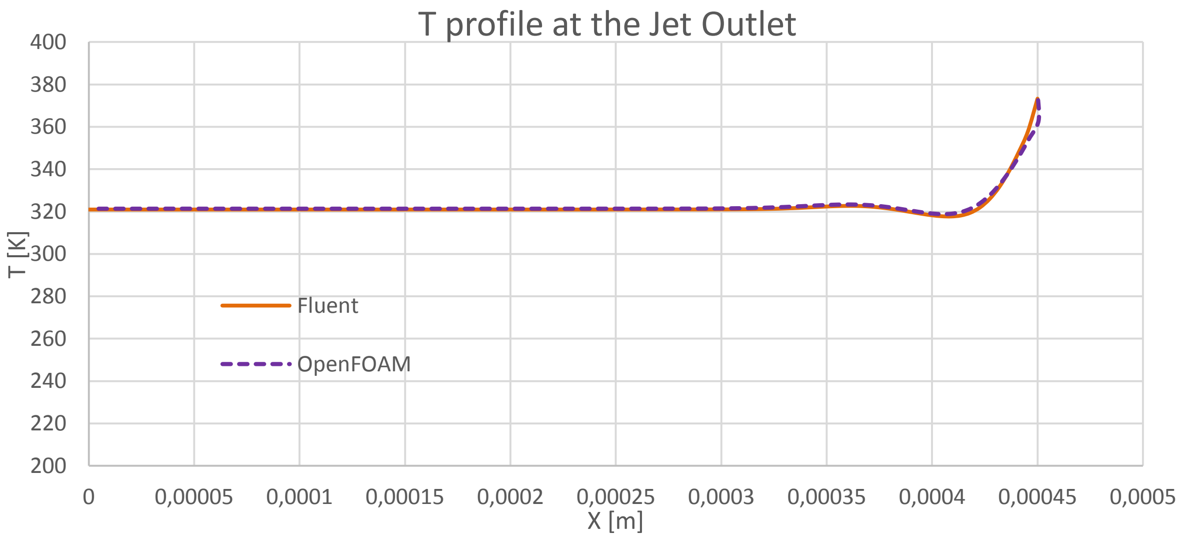
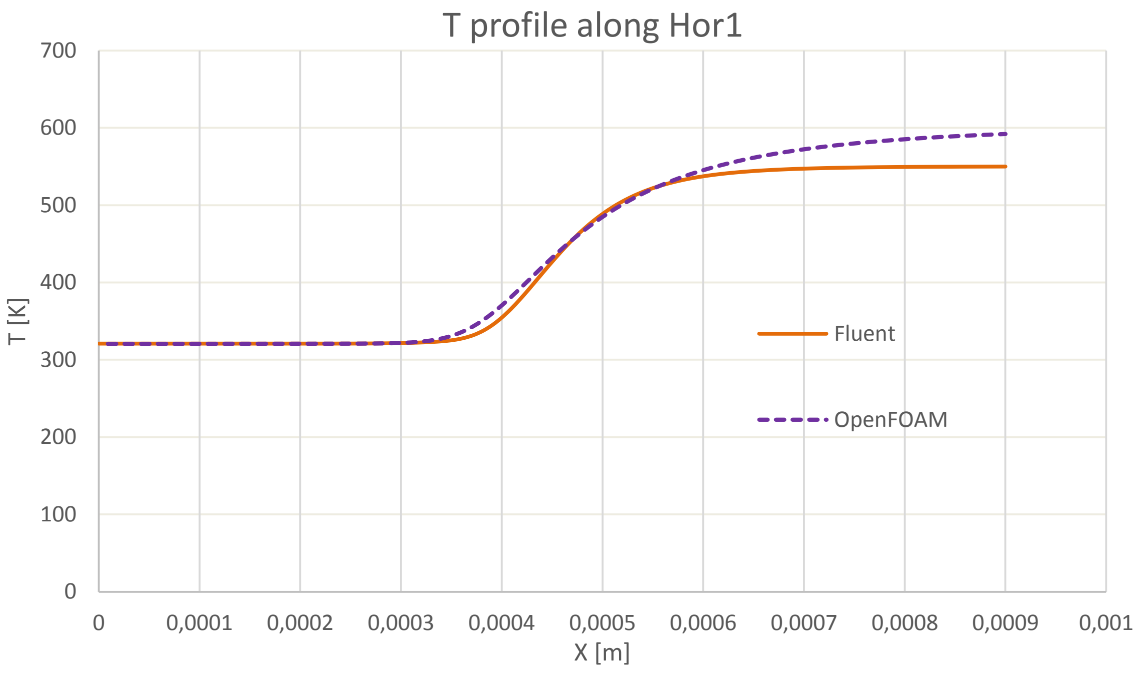


Fig. 6.31: Temperature profile at the jet outlet.

While at the jet outlet the agreement between OpenFOAM and Fluent is complete, off the jet the two software present some differences. The main difference is in the value of temperature far off the axis, where we have a $\Delta T \approx 25K$ between the two trends, which remains almost constant approaching the heated plate. It seems like in OpenFOAM the mixing along the radial direction is less intense and this leads to a higher temperature of the fluid. This can be explained also with the turbulence kinetic energy plots we showed above: a lower value of *k* means lower mixing at the interface.



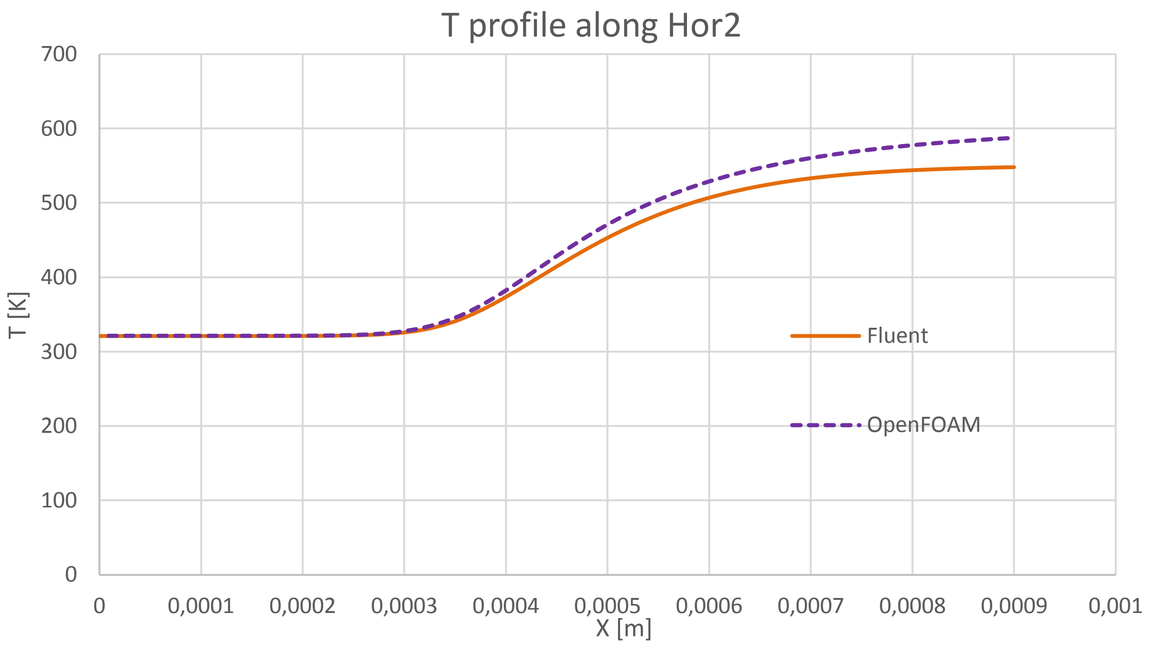
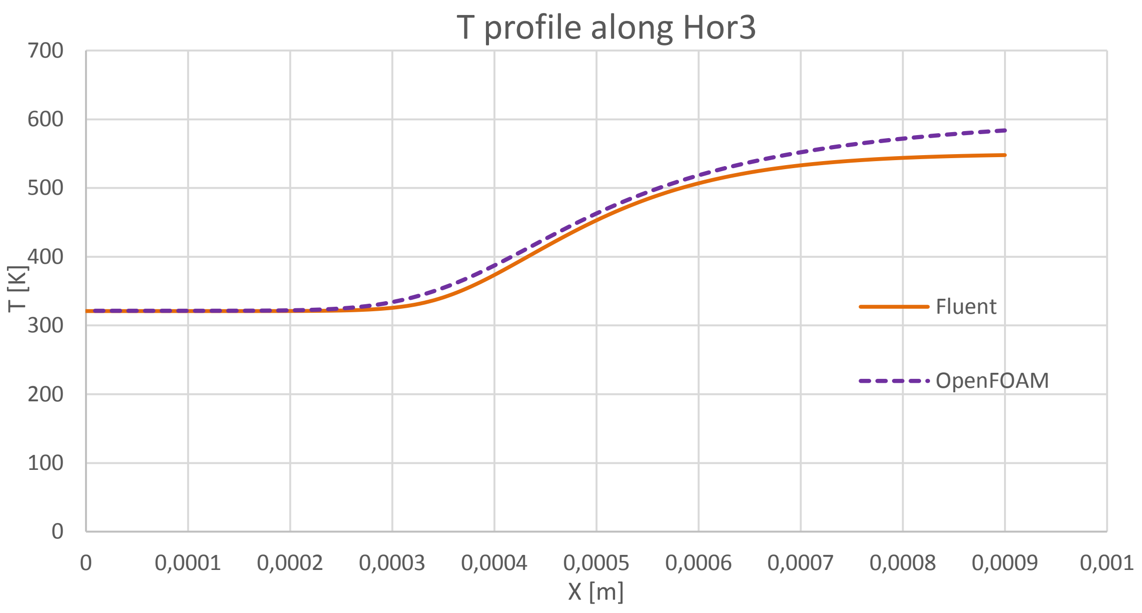


Fig. 6.32: Temperature field along Hor1 and Hor2



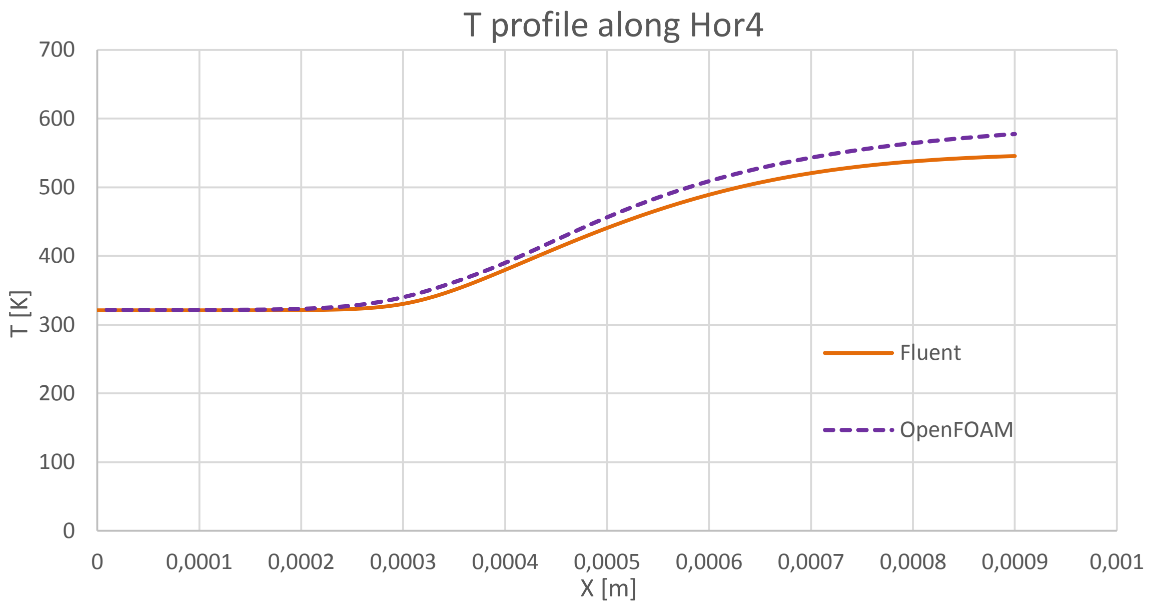


Fig. 6.33: Temperature profile along Hor3 and Hor4.

As we opted for *k-ω* SST turbulence model, another quantity of interest is the specific dissipation rate *ω* that is the rate at which turbulence kinetic energy is converted into thermal internal energy per unit volume and time; sometimes it is also referred to as the mean frequency of the turbulence (in fact the SI unit is *1/s*). We are interested in plotting this quantity along the heated wall (Fig. 6.34).

OpenFOAM and Fluent show a very similar trend, especially in the impinging region; moving to the outlet, the two solvers settle to different absolute value but another time the trend is similar, with a deviation of only 3%. The specific dissipation rate remains the same in the MUSCL case, therefore only one OpenFOAM case is reported.

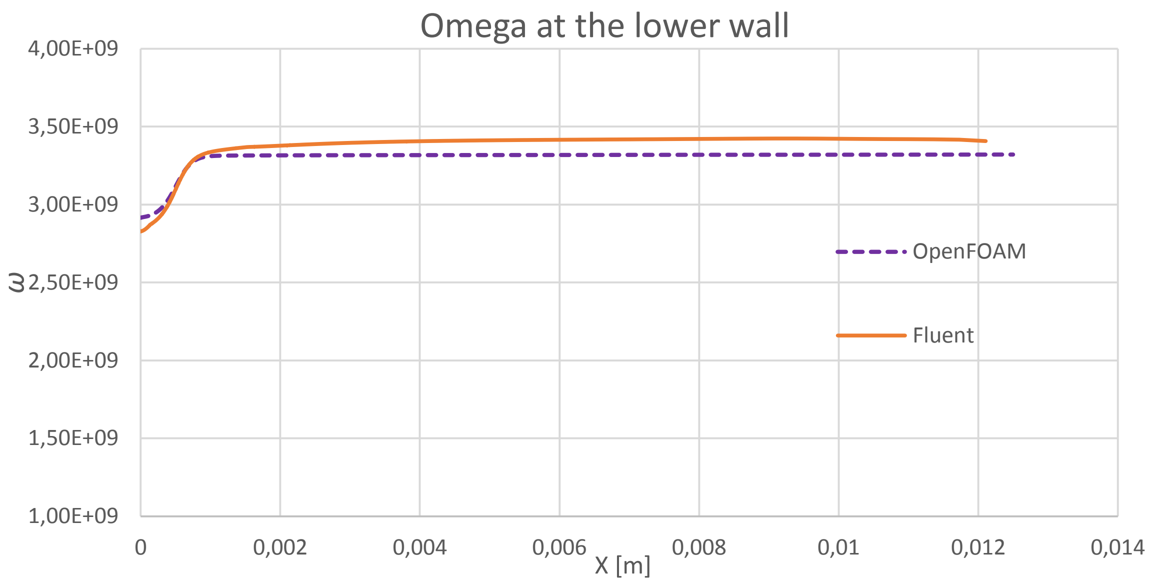


Fig. 6.34: Specific Dissipation Rate at the lower wall.

After the horizontal traverses, we have investigated some vertical traverses, primarily to check the differences in the flow evolution after the impinged area. In Fig. 6.35 there is a schematic representation of the traverses investigated.

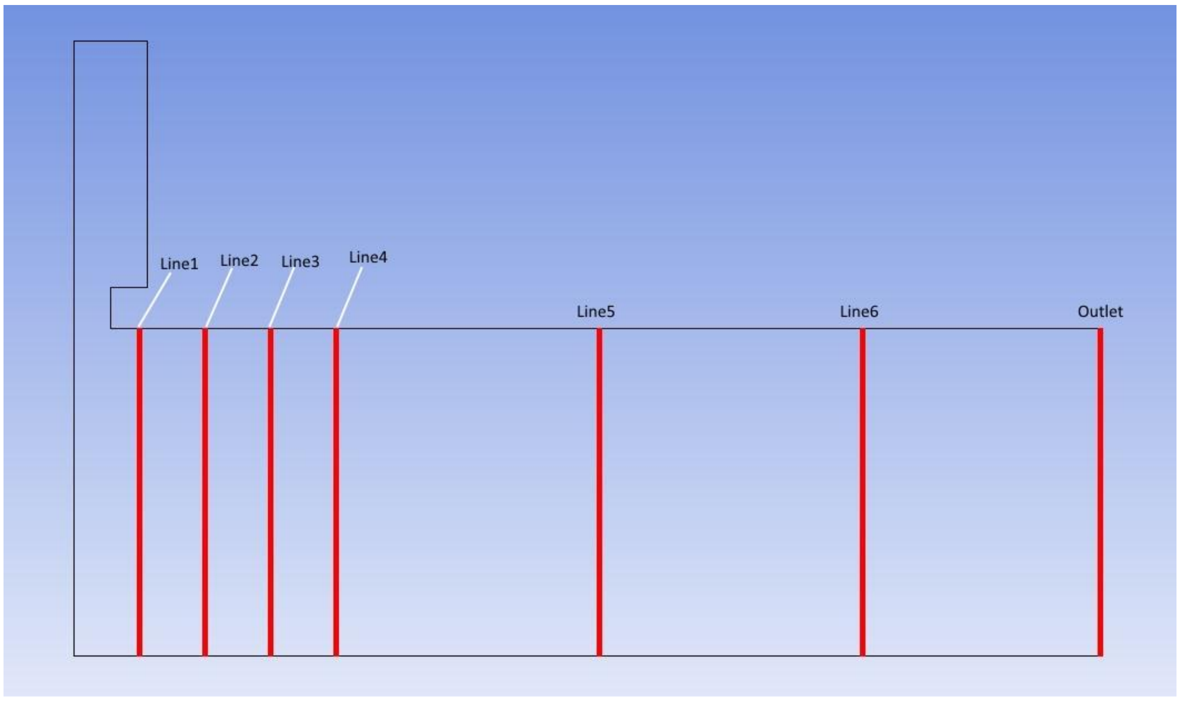
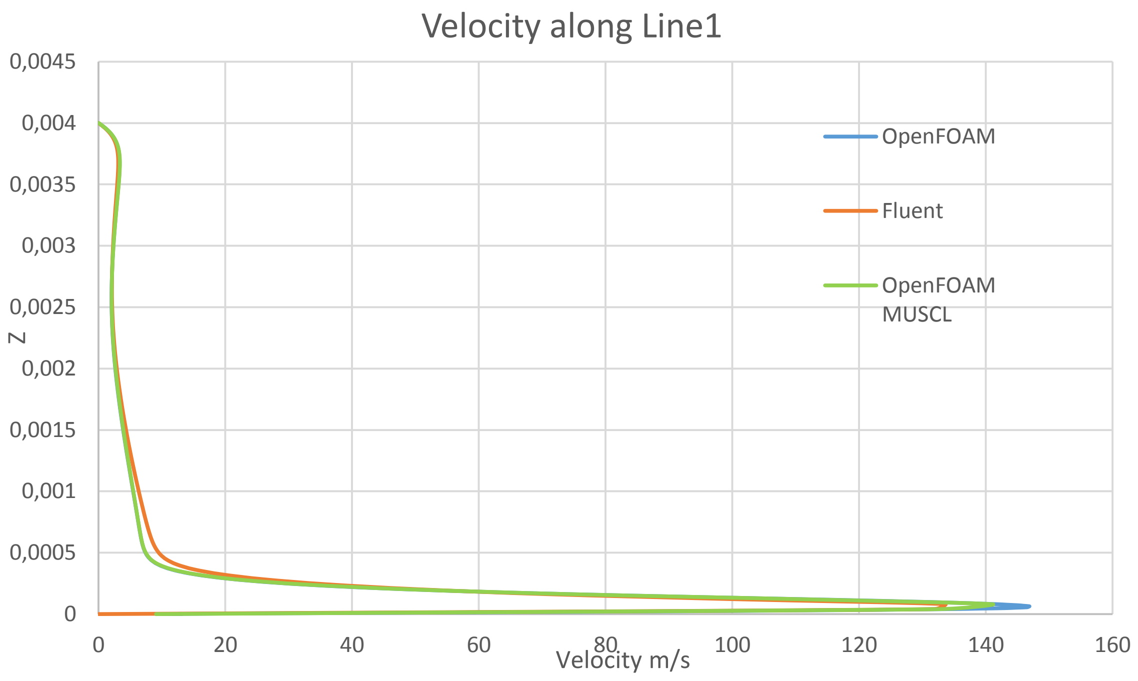


Fig. 6.35: vertical traverses layout.

Also for this traverses we investigated the same quantities.

The first four lines from the axis are spaced 0.8 mm from each other, while Line5 and Line6 are located 6.4 and 9.6 mm far, respectively. The last traverse is located at the outlet and allows us to understand a little bit better how the two solvers behave regarding recirculation at this location.

As done before, we start presenting velocity plots for Fluent and the two OpenFOAM cases.



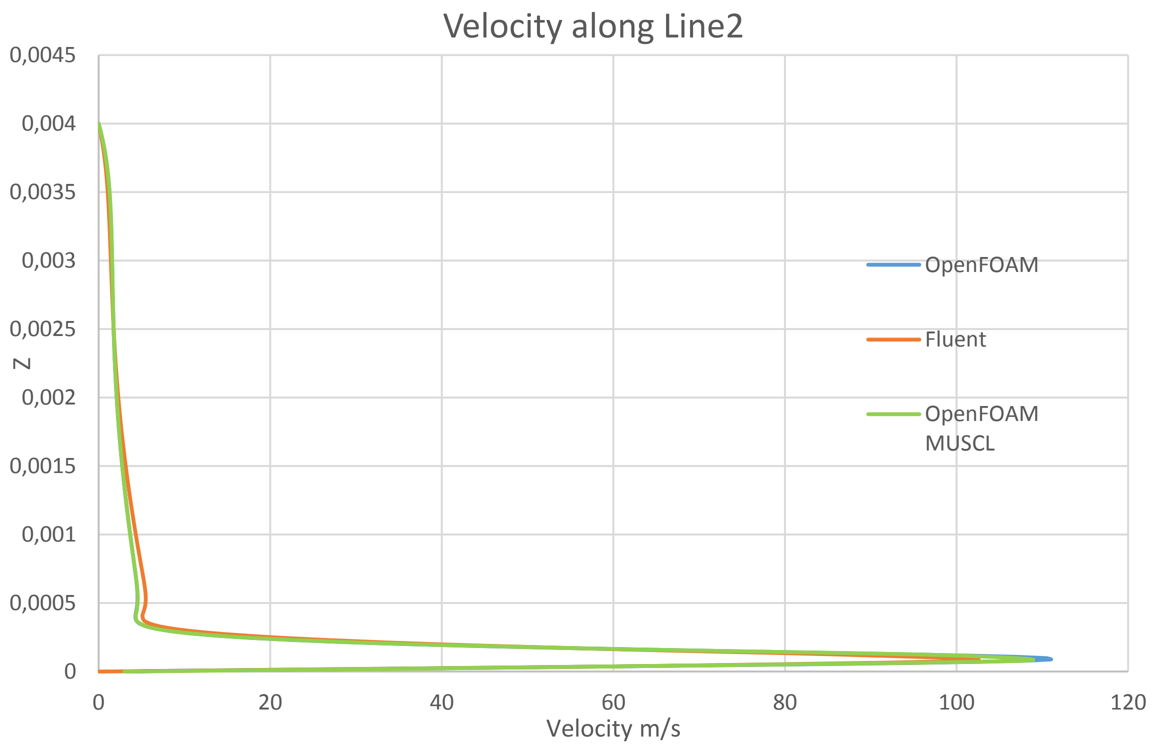
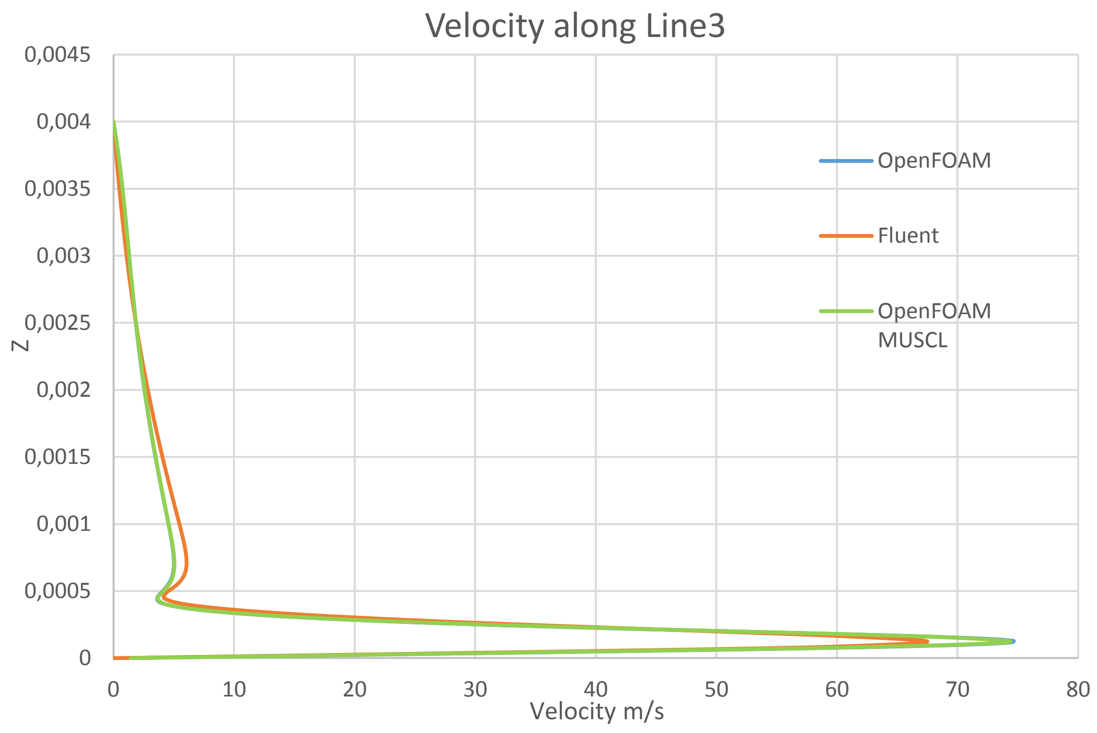


Fig. 6.36: Velocity profiles along Line1 and Line2.



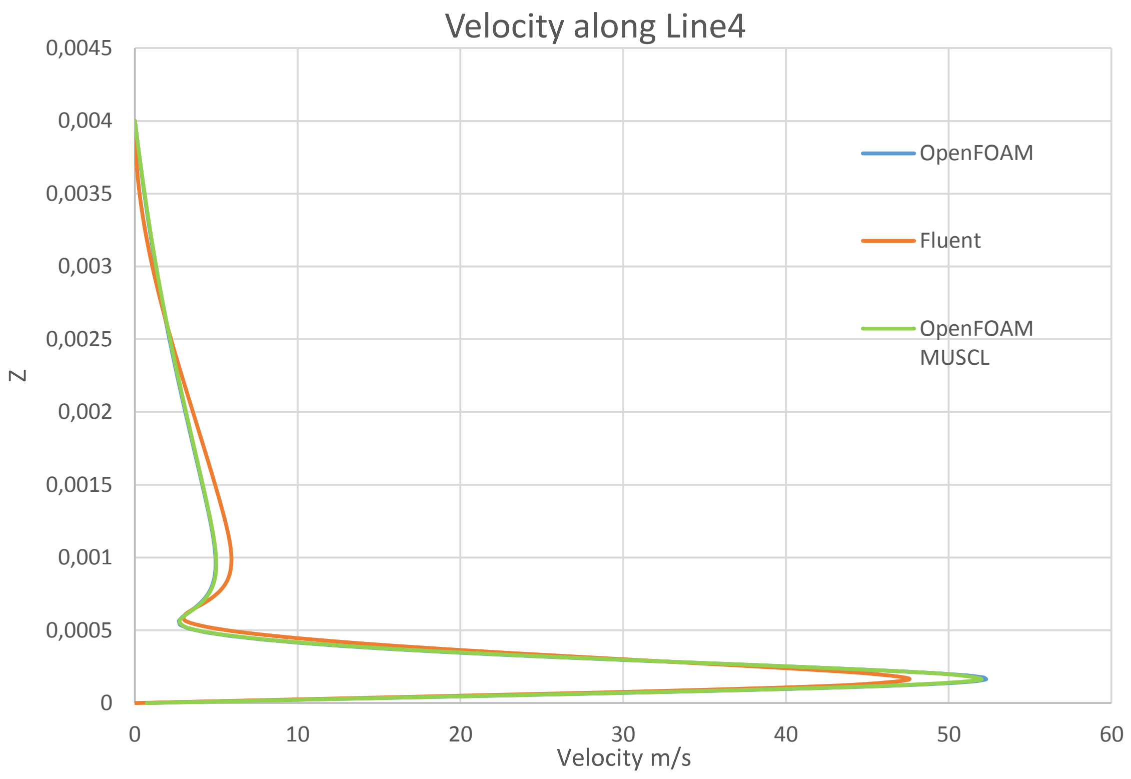
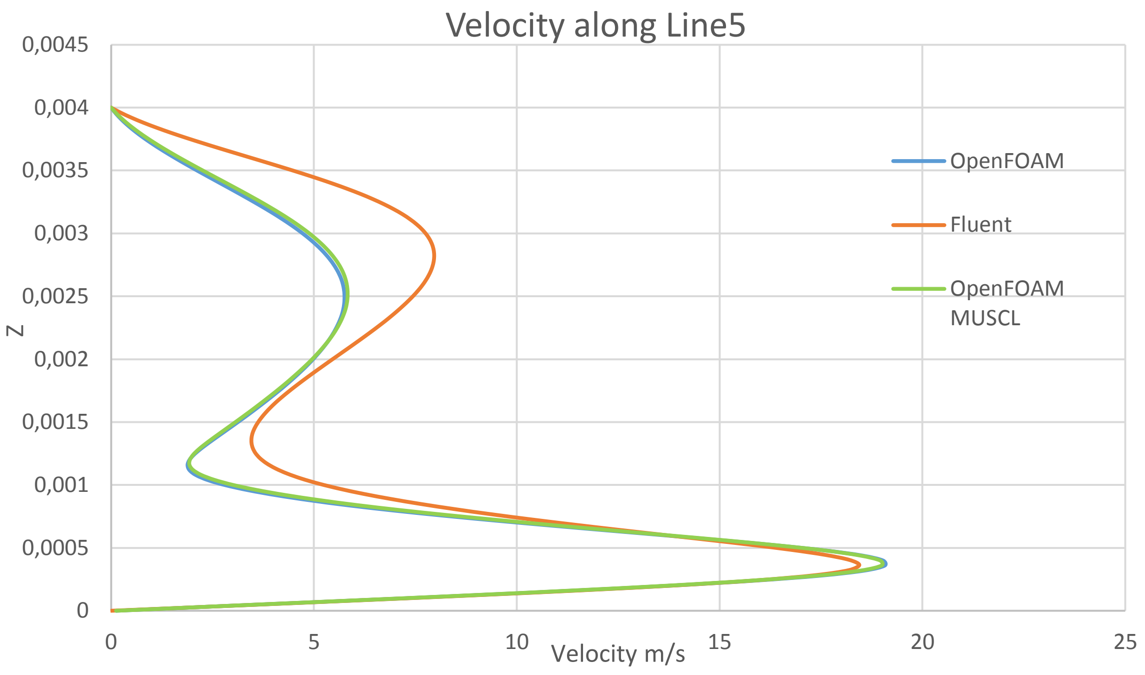


Fig. 6.37: Velocity profiles along Line3 and Line4.



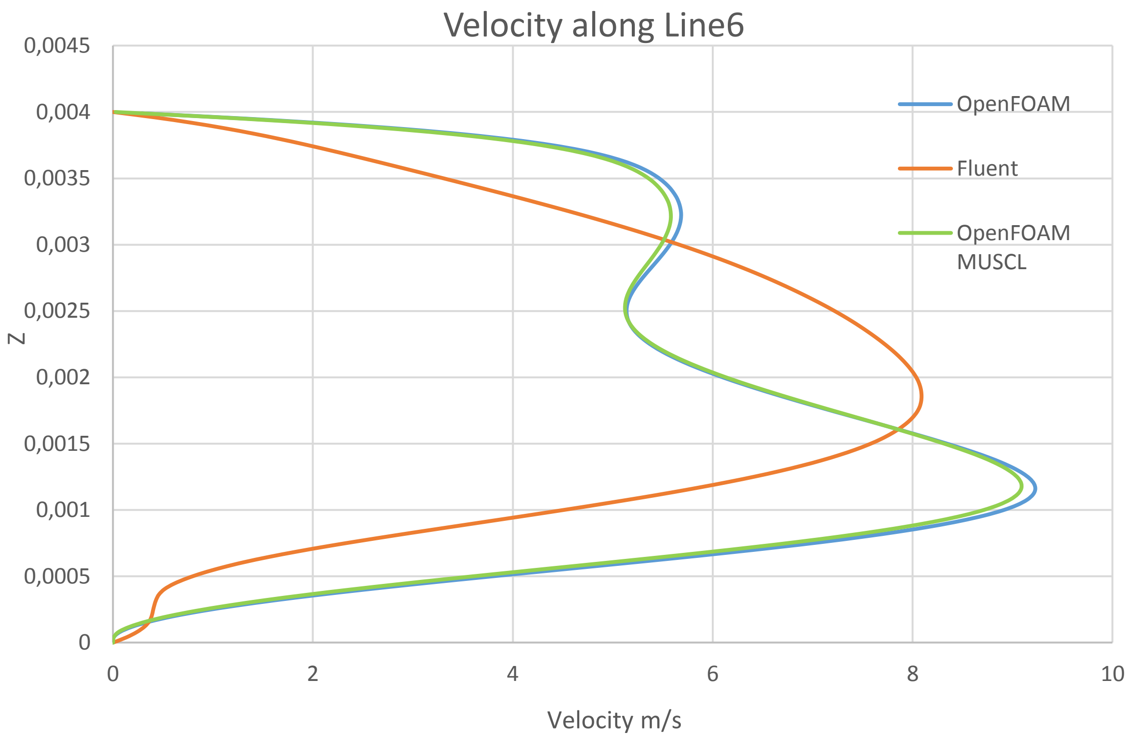


Fig. 6.38: Velocity profiles along Line5 and Line6.

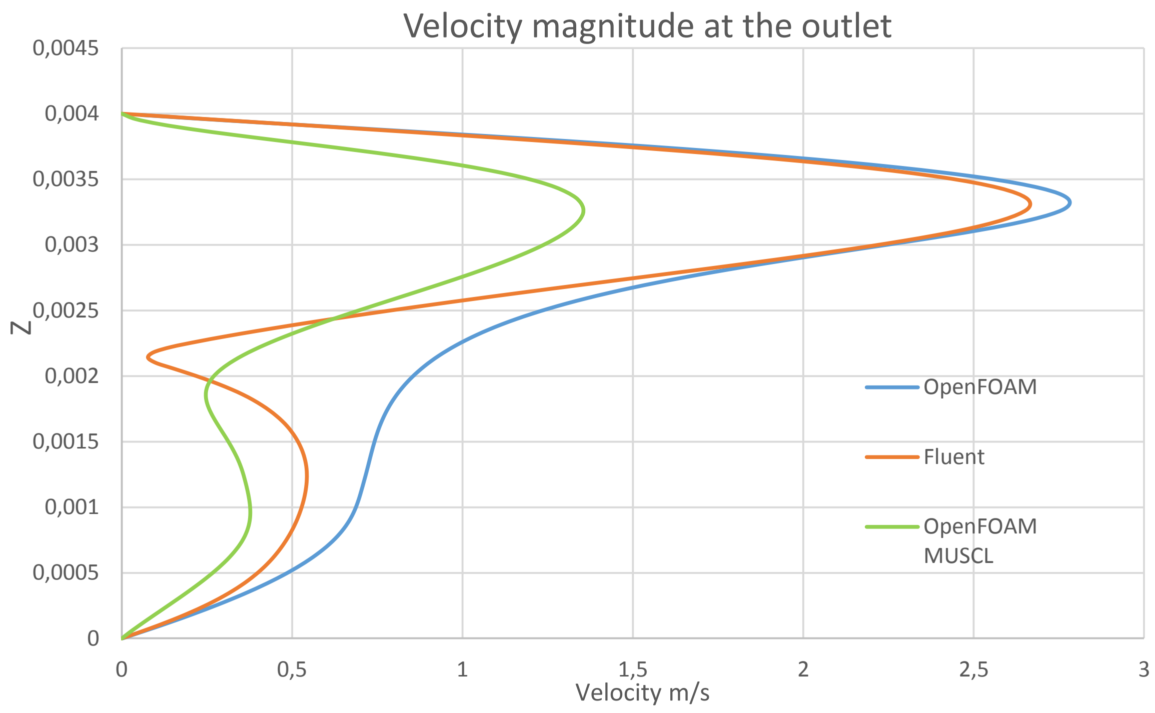


Fig. 6.39: Velocity profiles at the outlet.

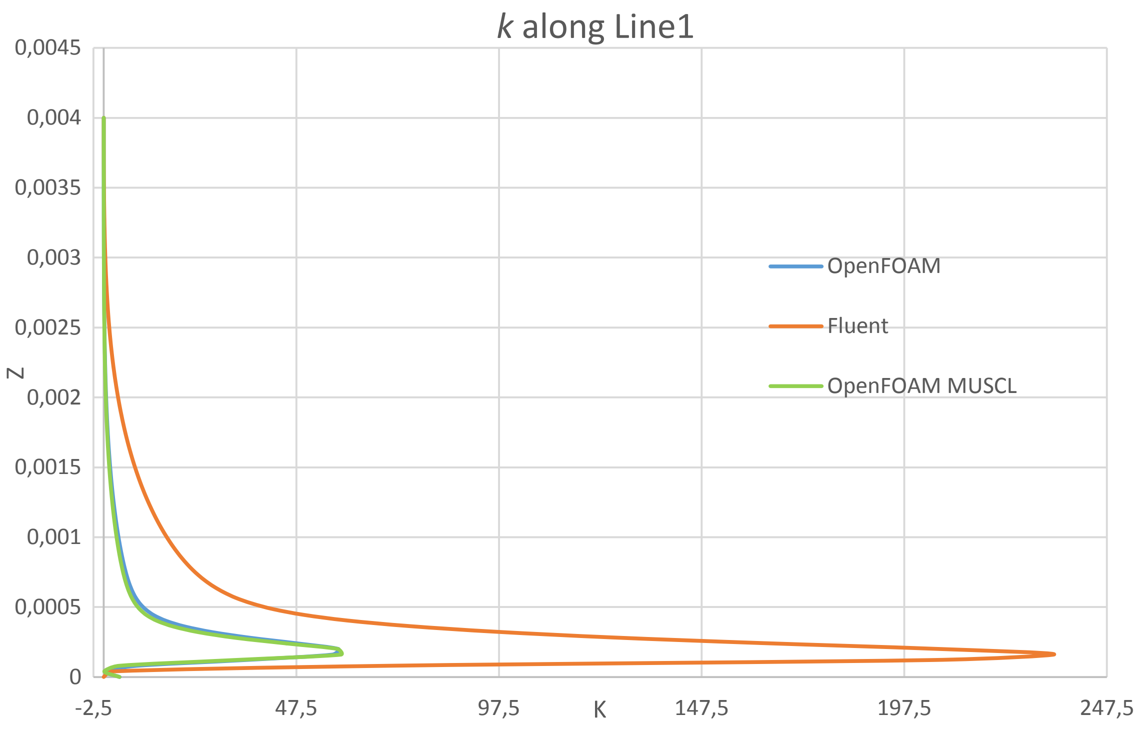
In the first four traverses we have obtained very similar results for all the simulations. Fluent presents a lower velocity peak next to the wall but a thicker profile in the median part, as the continuity equation has to be respected. The deviation between Fluent and OpenFOAM is about 7% at Line1 and tends to increase moving towards the outlet; at Line4 its value is ~10%. At Line5 the behaviors are still similar but values of the velocity peaks start to present big differences. Getting closer to the outlet we have very different trends; it appears clear that the different treatment of the incoming back flow modifies velocity profiles in a different way.

Considering the two different OpenFOAM cases, in the first four traverses they behave in a very similar way, with the MUSCL one presenting a smaller velocity peak at the wall; along Line5 and Line6 the profiles are almost equivalent. At the outlet, the trends are much more different, and the OpenFOAM case seems to suffer less the back flow, having a higher velocity profile. As nothing but the FVM scheme has been changed in the two OpenFOAM cases, the differences are due only to this.

Moving to the turbulence kinetic energy *k*, also along the vertical traverses it is somehow confirmed what we said about commenting horizontal profiles. Especially in the first four traverses the deviation from Fluent to OpenFOAM is about 80%, which is a really high value. Moving off the axis, the difference becomes smaller and smaller, and at the outlet OpenFOAM value is even higher; however at this location the value of *k* is really small and we do not have to forget the differences already remarked of the two software in case of recirculation. Considering the general evoution of the *k* profiles plotted, Fluent has way bigger values in the impinged area and in the nearby zone and then decreases in a faster way than OpenFOAM, until they assume similar value, next to the outlet.

OpenFOAM simulations have identical behaviors next to the jet but assume distinct trends closer to the outlet, in a similar fashion to what we have seen with velocity profiles.

All the plots are reported below.



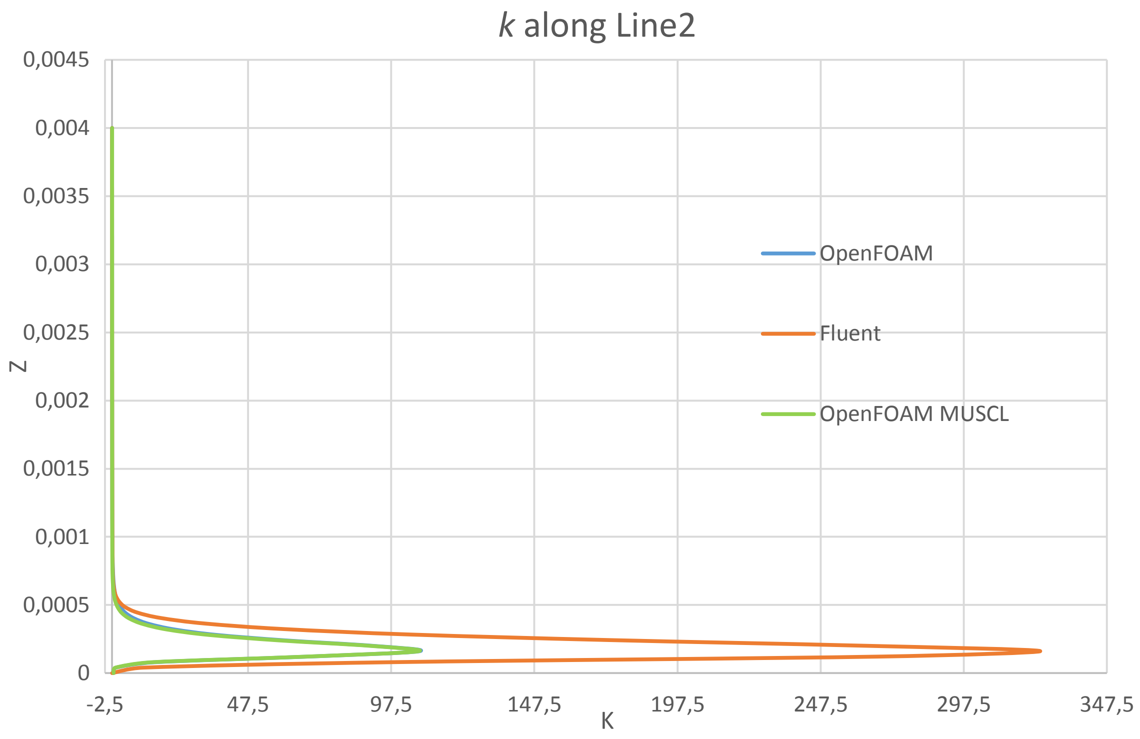
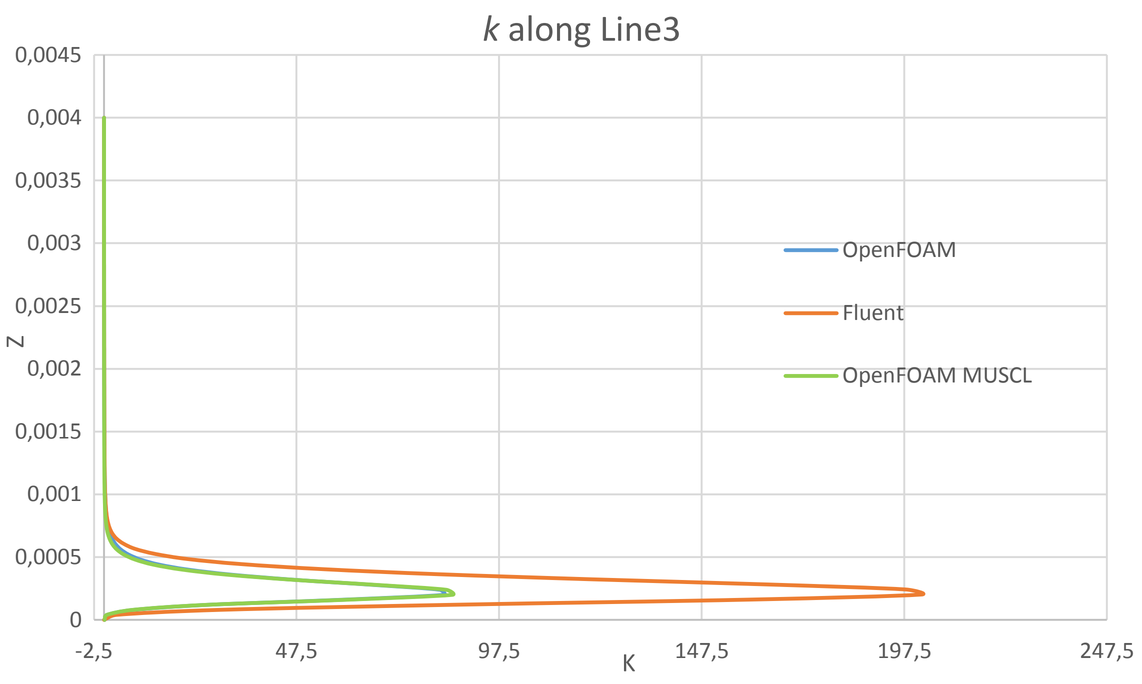


Fig. 6.40: *k* profiles along Line1 and Line2.



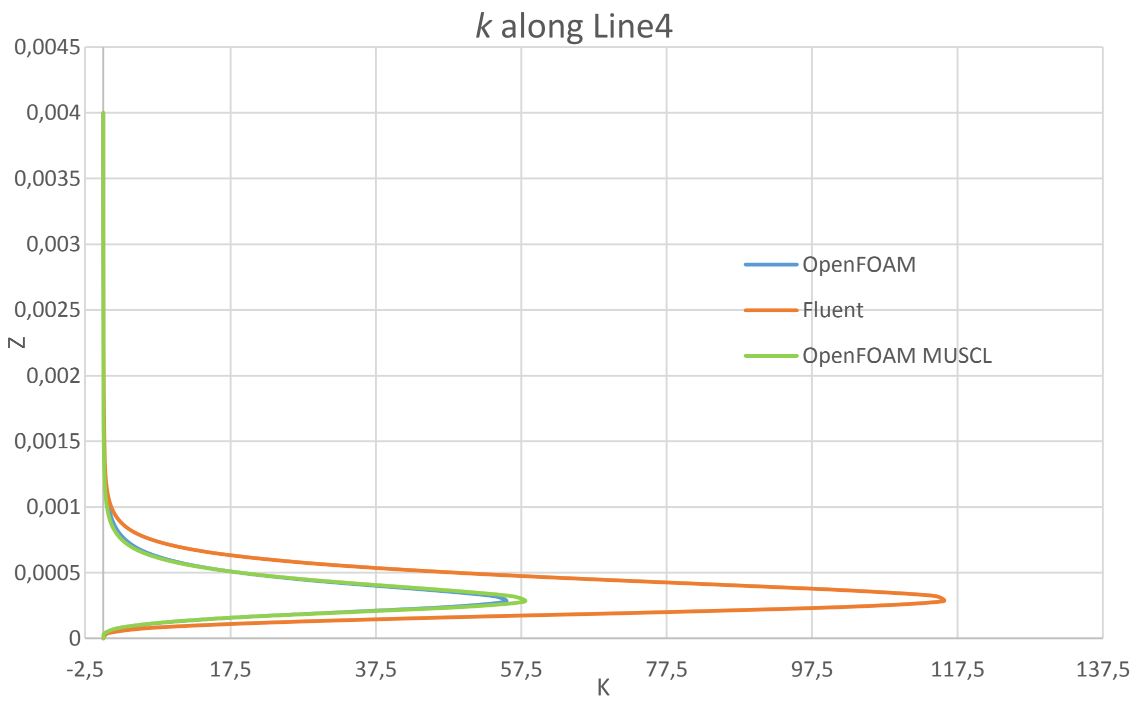
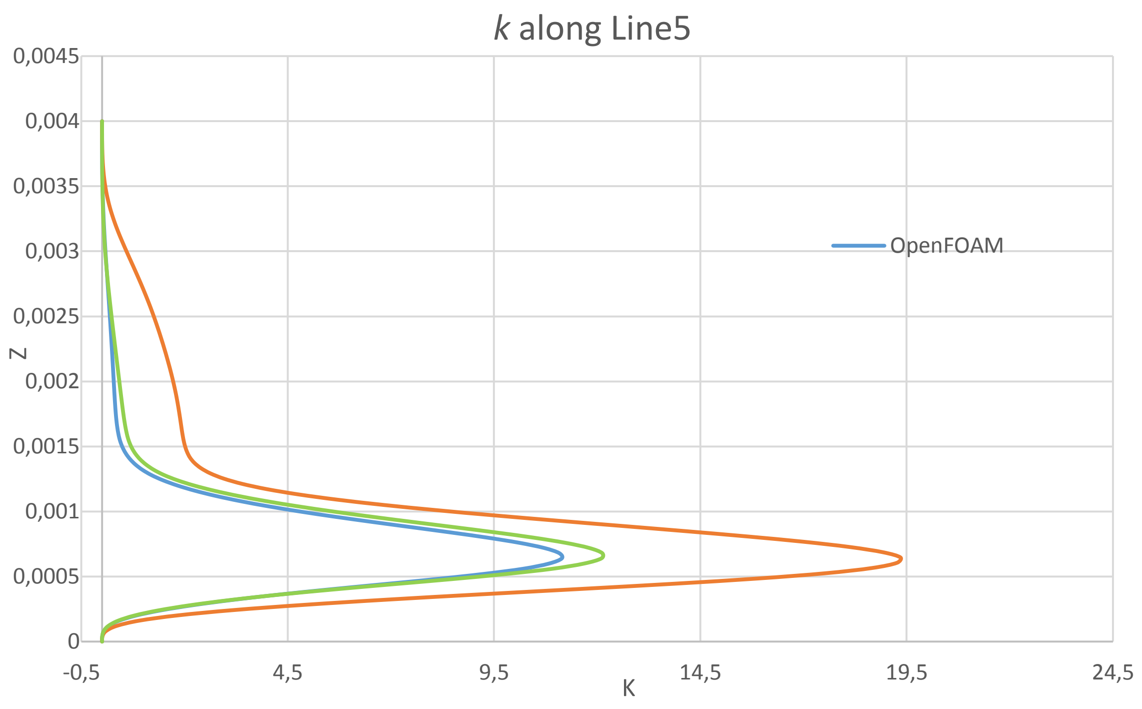


Fig. 6.41: *k* profiles along Line3 and Line4.



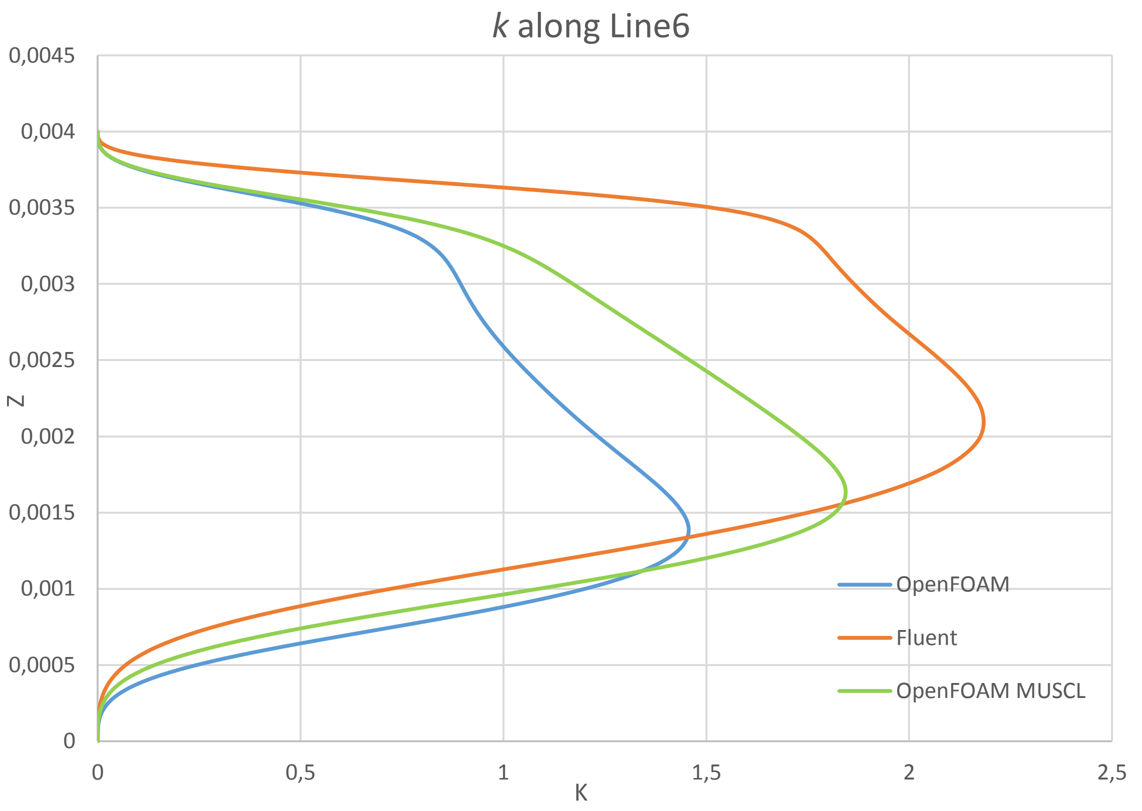


Fig. 6.42: *k* profiles along Line5 and Line6.

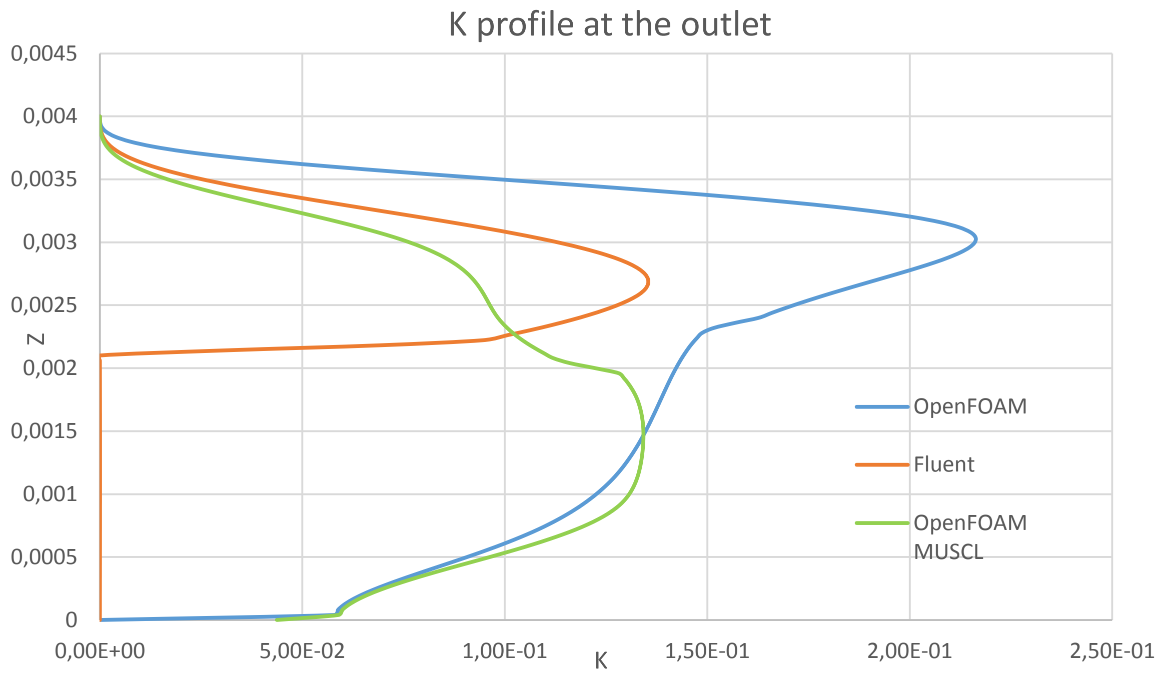


Fig. 6.43: *k* profiles at the outlet.

Lastly, we can have a look at the temperature profiles. As the temperature at the lower wall is given and equal to 644K for both Fluent and OpenFOAM, all this profiles should start from the same point. We can see below that this is verified.

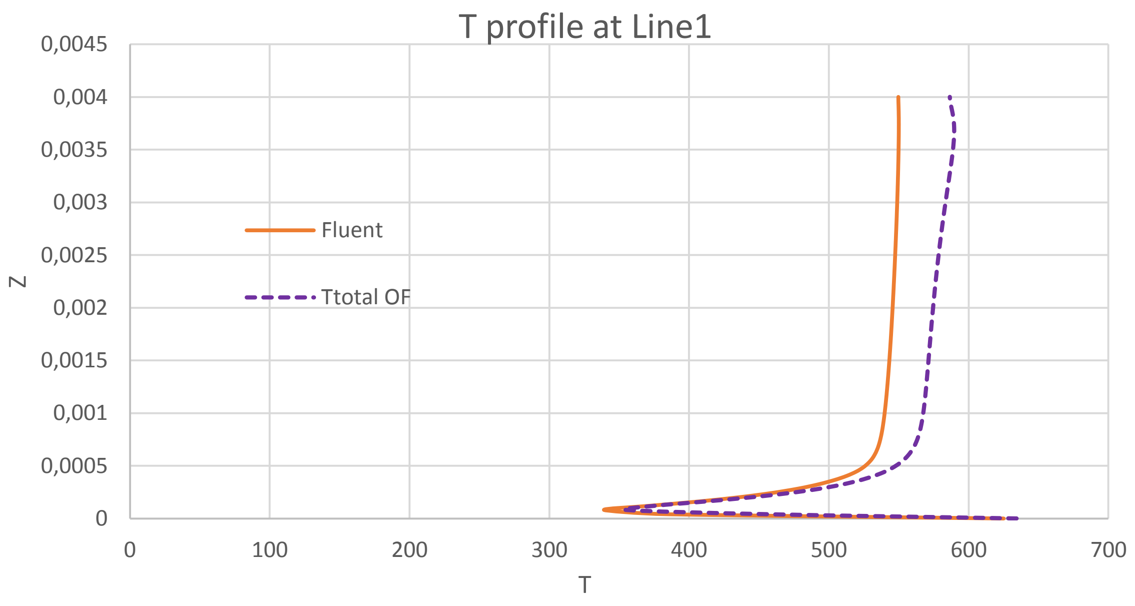
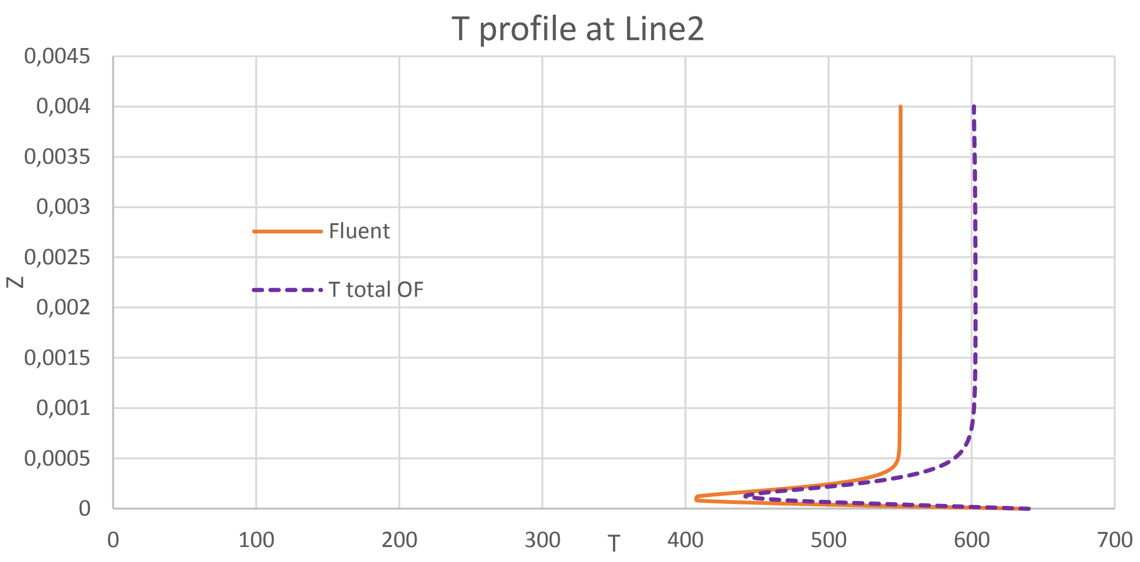
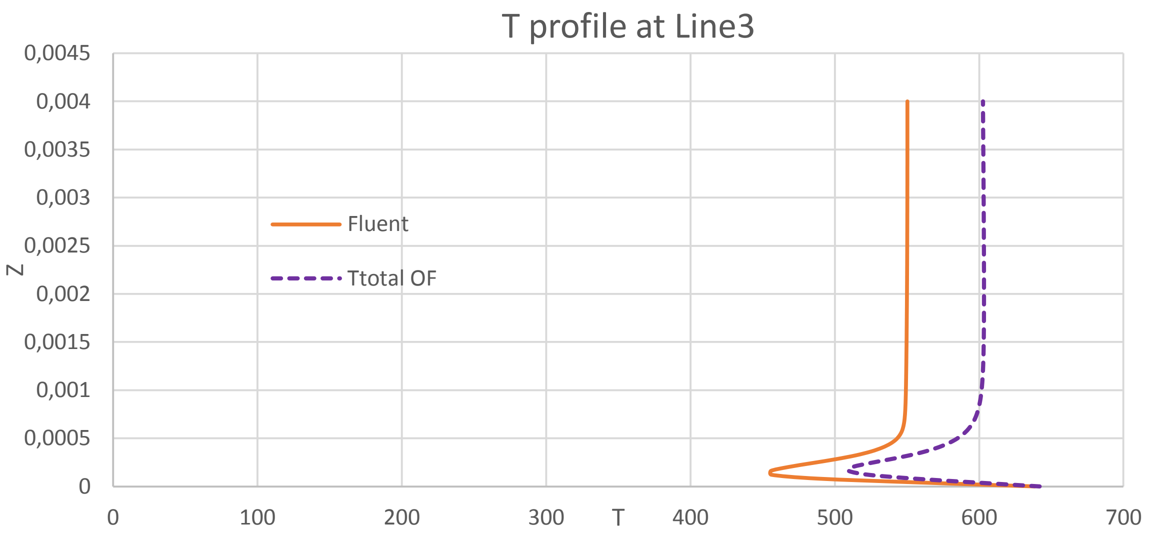


Fig. 6.44: T profiles along Line1.





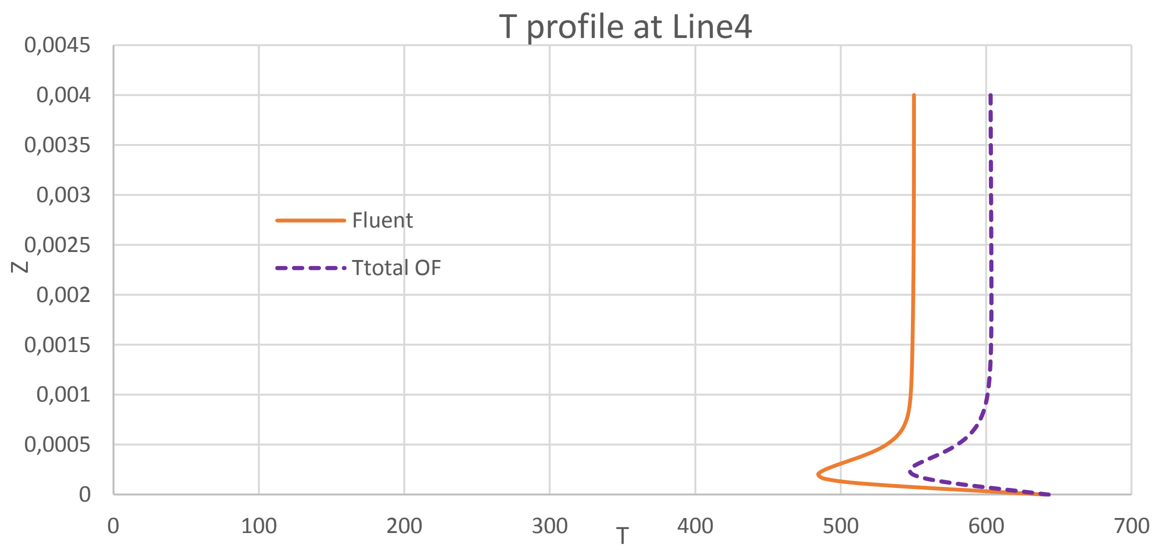


Fig. 6.45: T profiles along Line4, Line5 and Line6.

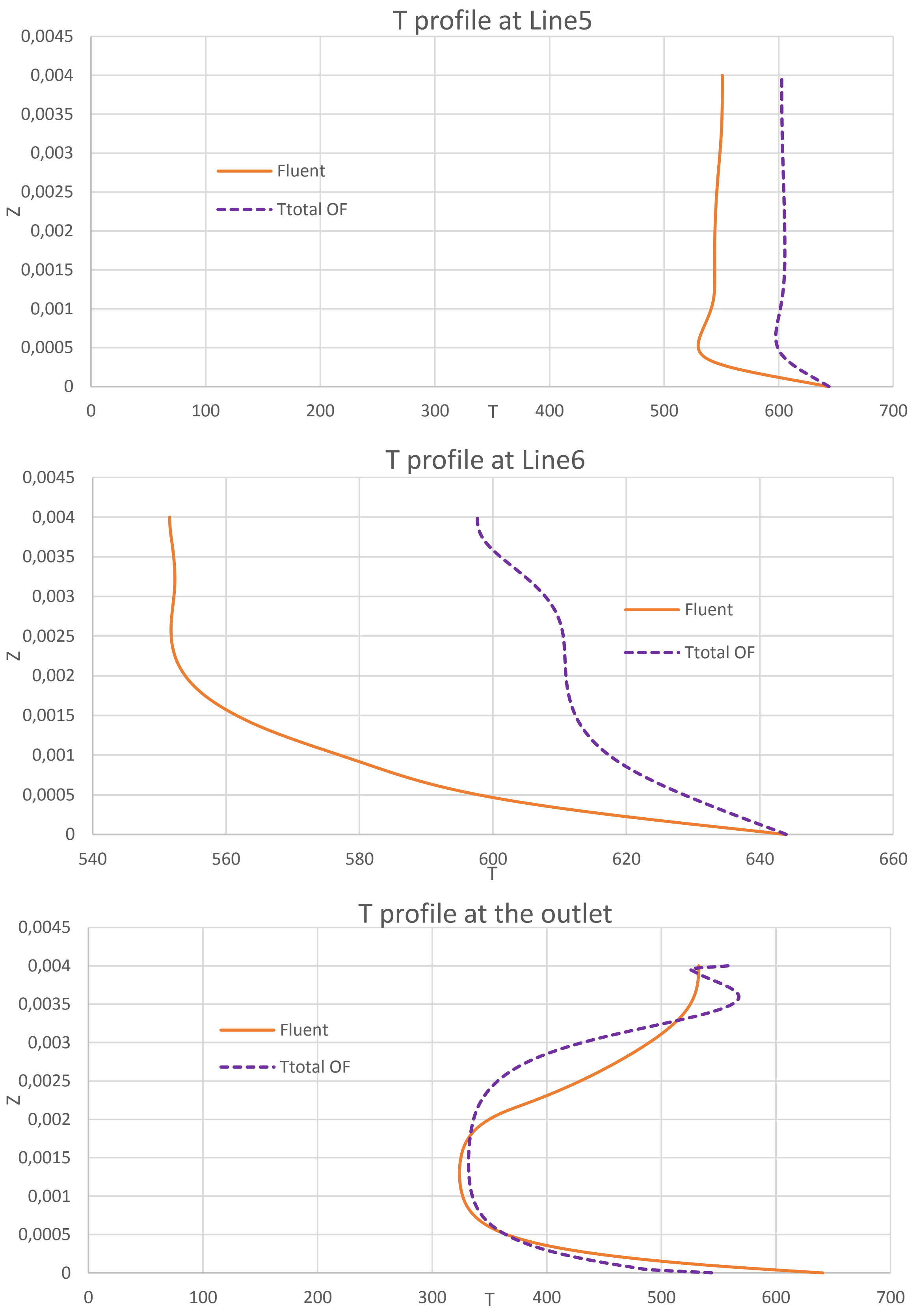


Fig. 6.46: Temperature profiles along Line5, Line6 and at the outlet.

The most important part of the profiles is the one close to the wall as we want to know the cooling performances of the jet. In the first traverse, the agreement between Fluent and OpenFOAM next to the wall is very good and we have deviation only for relatively high *Z* values, in a portion of the domain where we are less interested; here OpenFOAM presents higher values, as we had a higher thermal diffusivity. This can be noticed also considering the temperature contours we reported at the very beginning of the chapter.

Moving towards the outlet, the profiles start to differ also next to the wall, although the profiles always have the same starting points: casting a deeper look on the evolution of the profiles, it seems that in Fluent the temperature field is affected from the jet for a longer portion of the plate. While the second-last profiles have very different trends, as they lie in the recirculation area, the profiles at the outlet are quite similar but in this case the starting point is not the same; OpenFOAM wall temperature is affected by the back flow at a lower temperature, also if it should be fixed at 644K.

7. Conclusions

Here we are at the end of this thesis. What can we say about OpenFOAM? Well, when you have to use it for the first time, you can easily feel lost, also if you are a CFD user. The reason is easy to be found: OpenFOAM runs only in Linux and the most of the commands are given by terminal, without a real Graphical User Interface. Add to this the peculiar folders structure, which requires a specific layout depending on the solver or on the type of problem investigated, and we should understand why commercial codes are still so much used all over the world and somehow preferred. Anyway, evaluate OpenFOAM just considering these few aspects would be superficial, as what matters are results. We remember that the part relative to the meshing in OpenFOAM has not been investigated in this work, as we used IGG mesher by Numeca to create all the meshes utilized.

First of all, we have to underline that the learning curve is pretty steep, especially at the very beginning, but after some time the user becomes practical with the different way of setting up the problem. Moreover, it is possible to refer to previous cases or to tutorials on the internet. In fact, it is very uncommon to start the analysis of a case from zero and, in order to speed up the set-up of the case, we take the folders needed from other simulations; the more similar are the two cases, the less modifications will be required later. In any case, compared to Fluent, the set-up of a case is less intuitive and takes longer, especially in complex problems. The workflow itself is not as linear as in the Ansys software. On the other hand, OpenFOAM gives us the possibility to set in more variables (e.g. in turbulence models); this can scare an unconscious user but allows expert ones to manipulate every small parameter in order to get the set-up wanted. This comes along with the main feature we have in OpenFOAM, if compared to commercial software: the possibility to see whenever we want inside the code and to know how the problem will be solved. This is an extremely helpful feature, especially when we do not obtain wanted results and we want to investigate the reason of that. According to what we have just said, OpenFOAM represents a really powerful tool especially in the research field and its modular structure gives the possibility to everybody to develop his own solvers and utilities. Although it is a private company to release new software versions, the most of the features added and of the debugging done is work of the users' community.

The freeware license of OpenFOAM allows to use how many licenses we need and this is awesome, especially when we a have the possibility to run on an HPC with several CPUs; each CPU requires its own license and this can raise costs in an exponential way.

After this general comparison, we consider the cases investigated. Regarding the flat plate, we do not have that much to say: the results are almost equal for every variable and it is good the agreement with the analytical solution. The deviation between numerical and analytical solutions may be related not to the incapacity of the two software of simulating the correct flow evolution but most probably to the simplifying assumptions made by Blasius. Another reason can be the slightly different boundary conditions. In any case, these deviations are small.

The very small differences we have in the wall heat fluxes values numerically calculated can be explained considering different implementations of the same solver that may be present in the two codes. It is difficult to check this as we cannot see the implementation adopted in

Fluent; on the internet are available some web pages where Fluent implementation of the most used solvers is presented but the documentation is deliberately incomplete.
Considering the second case investigated, we can see bigger deviation. The problem is more difficult and the instability of the OpenFOAM steady solution does not allow a fair comparison. In particular, the back flow is managed in different ways and this leads to different behaviors of all the variables of interest close to the outlet. While velocity and pressure fields are more similar, temperature fields are quite different in both simulation. The first OpenFOAM case, which make use of the linear upwind interpolation scheme, shows unphysical values in the temperature field due to the scheme itself (it is stable but not limited); the second one shows physical results for temperature but the global distribution is still different, especially far from the heated wall.
Another remarkable difference is in the turbulence model. Although we have chosen the $k-\omega\,SST$ model for both Fluent and OpenFOAM, in turbulence kinetic energy plots it is noticeable a big deviation, especially in the peak value, while the trends are similar. This pushes us to think that Fluent utilizes a slightly different version of the turbulence model, with a higher production term for the quantity k. Once again, the implementation is not known, at least not completely, and so these are only suppositions made by analyzing results.
However, the most important difference is remarkable in the wall heat flux at the heated wall. This is certainly due to the difference in the values of the other fundamental variables described above. As the wall heat flux at the plate is one of the most important parameter considered in the project of the cooling system, big differences lead to different project designs. Unfortunately, we could not compare our numerical results with experimental data, as they are not available yet. So, we can not say which code is closer to the experimental value. When the experimental data will be available, this review will be updated with a much more complete comparison.

Final thoughts
However, after this experience, we can say that OpenFOAM provides a powerful CFD tools and its modularity and the open source license are a plus, especially in the academic and research fields. Moreover, the set-up of a case requires a deeper knowledge of the physics of the problem and of the theory: while in Fluent the workflow is automatically driven by the software itself and it does not allow the combination of wrong settings, in OpenFOAM it is necessary to pay attention, otherwise you will encounter problems while running the solver. This obliges the user to know in details what he is doing and it can be only positive in the academic field, obliging to gather the necessary information on the web and on the guide. While the community support available for OpenFOAM can somehow stand Ansys assistance, it is slower in providing the needed help. The guide is a sore point: in fact, it is far to be complete and many information are missing. Fortunately, we can find almost everything we need in some forums, but this is, in any case, a shortcoming.
OpenFOAM is an open source software but paid courses are available, either for beginners or advanced user. It is possible even to attend programming courses, where it is possible to practice with the code and the structure of the software in order develop your own utilities and software.
OpenFOAM is on the right track but, at the moment, does not provide the immediacy of ANSYS Fluent, which looks more mature. The capabilities of the software are almost

unlimited, being possible to modify and add new features in a quite straightforward way. As said above, the major shortcoming is the lack of a complete guide. If the missing GUI is a minor problem, as it is easy to get used to a different way of input, the difficulties we can find while looking for information can push a beginner to other codes. The huge amount of tutorials available on the Web try to mitigate this problem but it is not uncommon the need to simulate something different. Moreover, often we are looking for utilities implemented by third parties and not natively available in the code. Finding these third-party utilities can be tricky and many versions are available, as more people developed their own utilities facing the same problem; at times, it is possible to run into compatibility issue, especially when they have been written for previous versions of the software. In other cases, we need some feature already implemented in the original code but we do not know that, as it is not mentioned in any guide. It appears clear how experience is the most important thing if we want to do a profitable use of OpenFOAM. Regarding the post-processing tool, Paraview, I have found it really complete and, with the new release, also the GUI has improved. Compared to CFD Post (Ansys post-processing tool), Paraview is more complete but less immediate. In any case, it is just a matter of time to take confidence with the interface.
It is also expected an improvement in the interconnection with other software; for instance we could not import directly the mesh from IGG with the dedicated utility, but we had to import it before in Fluent and then in OpenFOAM. We had a similar experience trying to export data to Tecplot for a faster post processing, with non-readable or lost data.
The implementation of new solvers is usually slower if compared to commercial codes but this is understandable, being the software open source. The number of solvers available increases in every release and it is possible to tackle a great variety of problems, dealing not only with fluid dynamics. As the users’ number is constantly increasing, we can expect a higher interest in OpenFOAM, hence a faster growth of the code in terms of completeness and reliability. The creation of more complete third party GUI, already in development, could further increase the number of potential users.

References

[1] Dale A. Anderson, John C. Tannehill, Richard H. Pletcher, “Computational Fluid Mechanics and Heat Transfer”, McGraw-Hill, New York, 1984.

[2] J. Ferziger, M. Peric, Computational Methods for Fluid Dynamics, 3rd Edition, Springer, Berlin, 2002.

[3] H. Versteeg, W. Malalasekera, “An Introduction to Computational Fluid Dynamics”, 2nd Edition, Pearson Education Limited, Harlow, England, 2007.

[4] OpenFOAM Foundation (2013), OpenFOAM user guide, downloaded: (9-11-2015). URL http://www.openfoam.org/docs/user/

[5] P. R. Spalart, S. R. Allmaras, “A One-Equation Turbulence Model for Aerodynamic Flows”, Paper 92-0439, AIAA (1992).

[6] F.R. Menter, M. Kuntz, R. Langtry, “Ten Years of Industrial Experience with the SST Turbulence Model”, Otterfing, Germany, 2003.

[7] D.C. Wilcox, “Turbulence Modeling for CFD”, DCW Industries, Inc., La Canada, CA, 1993.

[8] N. Zuckerman, N. Lior, “Jet Impingement Heat Transfer: Physics, Correlations and Numerical Modeling”, Advances in Heat Transfer, Volume 39, 2006.

[9] H. Martin, “Heat and mass transfer between impinging gas jets and solid surfaces”, Advanced Heat Transfer, 1977.

[10] R. J. Goldenstein, A. I. Behbahani, K. K. Heppelmann, “Streamwise distribution of the recovery factor and the local heat transfer coefficient to an impinging circular air jet”, 1986.

[11] C. J. Hoogendorn, “The effect of turbulence on heat transfer at a stagnation point”, 1977.

[12] Y. M. Chung, K. H. Luo, “Unsteady heat transfer analysis of an impinging jet”, ASME J. Heat Transfer 124, 2002.

[13] L. W. Florschuetz, C. R. Truman, D. E. Metzger, “Streamwise flow and heat transfer distributions for jet array impingement with crossflow”, J. Heat Transfer 103, 1981.

[14] H. Chattopadhyay, S. K. Saha, “Turbulent flow and heat transfer from a slot jet impinging on a moving plate” Int. J. Heat Fluid Flow 24, 2003.

[15] G. C. J. Bart, , A. J. van Ijzerloo, L. F. G. Geers, L. Hoek, K. Hanjalic, “Heat transfer of phase-locked modulated impinging-jet arrays”, Exp. Thermal Fluid Sci. 25, 2002.

[16] S. Göppert, T. Gürtler, H. Mocikat, H. Herwig, “Heat transfer under a precessing jet: Effects of unsteady jet impingement”, Int. J. Heat Mass Transfer 47, 2004.

[17] C. Cau, C. C. Lee, “Impingement cooling flow structure and heat transfer along rib-roughened walls”, Int. J. Heat Mass Transfer 35, 1992.

[18] D. Rhee, P. Yoon, H. H. Cho, “Local heat/mass transfer and flow characteristics of array impinging jets with effusion holes ejecting spent air”, Int. J. Heat Mass Transfer 46, 2003.

Appendix

Impinging jet simulation: steady case boundary conditions and solver settings.

Pressure Boundary conditions

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 21720;

boundaryField
{
    inlet
    {
      /*  type           fixedValue;
      value uniform    110000; */
      type zeroGradient;

    }
    outlet
    {
        //type           zeroGradient;
        type           fixedValue;
        value          21720;
        /*value            uniform 0;*/
    }

    wall_heated
    {
        type           zeroGradient;
    }
    walls
    {
        type           zeroGradient;
        //type symmetry;
    }
    symmetry
    { type           symmetryPlane;
    }
    front
    {
        type           wedge;
    }
    back
    {
        type           wedge;
    }
}
```

Velocity boundary conditions

```
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
```

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
           type  flowRateInletVelocity;
      massFlowRate constant 12.770825e-08;
      value uniform (-1 0 0);
      rhoInlet  0.2358;
    }

    outlet
    {

        type zeroGradient;

    }

    wall_heated
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    walls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    symmetry
    {
        type            symmetryPlane;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

    empties
    {
        type            empty;
    }
}
```

Temperature boundary conditions

```
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions      [0 0 0 1 0 0 0];
```

```
internalField   uniform 300;

boundaryField
{
    wall_heated
    {
       type fixedValue;
       value uniform 644;
    }

    outlet
    {
        type fixedValue;
      value uniform 321;
    }

    inlet
    {
        type            fixedValue;
        value           uniform 321;
    }

    walls
    {
        type            zeroGradient;
    }

    symmetry
    {
        type            symmetryPlane;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

thermophysicalProperties file

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  //

thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       sutherland;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        nMoles      1;
        molWeight   28.9;
    }
    thermodynamics
    {
        Cp          1005;
        Hf          1;
    }
    transport
    {
        As          1.4792e-06;
        Ts          116;
    }
}
```

fvSolution

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.2;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }



    "U|e|omega|k|h"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }

}

SIMPLE
{
    nNonOrthogonalCorrectors 2;
    pRefCell        0;
    pRefValue       0;
    pMin pMin [1 -1 2 0 0 0 0] 1000;
    rhoMax rhoMax [1 -3 0 0 0] 2;
    rhoMin rhoMin [1 -3 0 0 0] 0.001;

    residualControl
    {
        p               1e-5;
        U               1e-5;
        nuTilda         1e-5;
    }
}

relaxationFactors
{
    fields
    {
        p               0.2;
      rho               0.01;

    equations
    {
        U              0.7;
        nuTilda        0.7;
        k              0.5;
        omega          0.5;
        e              0.5;
        R              0.7;
    }
}
```

fvSchemes

```
ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    div(phi,U)      bounded Gauss linearUpwind grad(U);
    div(phi,nuTilda) bounded Gauss linearUpwind grad(nuTilda);
    div((nuEff*dev(T(grad(U))))) Gauss linear;
    div(phi,omega)    bounded Gauss upwind;
    div((muEff*dev2(T(grad(U))))) Gauss linear;
    div(phi,Ekp) bounded Gauss upwind;
    div(phi,e) bounded Gauss upwind;
    div(phi,k) bounded Gauss upwind;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

Impinging jet simulation: unsteady case solver settings.

controlDict file

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     rhosimpleFoam;

startFrom       startTime;

startTime       90000;

stopAt          endTime;
//stopAt          writeNow

endTime         90350;

deltaT          0.00000001;

writeControl   timeStep;
//writeControl runTime;

writeInterval   100;

purgeWrite      0;

writeFormat     ascii;

writePrecision  15;

writeCompression off;

timeFormat      general;

timePrecision   15;

runTimeModifiable true;
```

fvSChemes: d/dt schemes

```
ddtSchemes
{
    default         Euler;
    //default         CrankNicolson 0.5;
    //default         backward;
}
```

fvSolution (relaxation factors not reported, as they are equal to the previous case)

```
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.2;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }

    pFinal
    {
        $p;
        tolerance       1e-06;
        relTol          0;
    }


    "rho|U|e|omega|k|h"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }

    "(rho|e|U|h|k|epsilon|omega)Final"
    {
        $U;
        tolerance       1e-05;
        relTol          0;
    }


  PIMPLE
  {
      //If you set nOuterCorrectors to 1 is equivalent to piso
      //nOuterCorrectors 1;
        nOuterCorrectors 2;

      nCorrectors 2;
      nNonOrthogonalCorrectors 1;
        pRefCell        0;
        pRefValue       0;
      rhoMax rhoMax [1 -3 0 0 0] 2;
        rhoMin rhoMin [1 -3 0 0 0] 0.001;

  }
```