CRANFIELD UNIVERSITY

Christos Tsotskas

Comparative Analysis of Parallel Performance and Scalability of Incompressible CFD Solvers

School of Engineering
Computational and Software Techniques

MSc Thesis
Academic Year: 2009 - 2010

Supervisor: Dr Evgeniy Shapiro
August 2010

CRANFIELD UNIVERSITY

School of Engineering
Computational and Software Techniques

MSc Thesis

Academic Year 2009 - 2010

Christos Tsotskas

Comparative Analysis of Parallel Performance and Scalability of Incompressible CFD Solvers

Supervisor: Dr Evgeniy Shapiro

August 2010

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

«Ἰσχύς εν τη ενώσει»

Όμηρος από Μ.Ασία, περ. 850 π.Χ.

«United we stand»

Homer, Middle East, circa 850 B.C.

ABSTRACT

The contribution of CFD is evident in many aspects of past, present and future of scientific and industrial applications.CFD has become an indispensable tool for the aerodynamic design and analysis of aircrafts and engines. Individuals use high computational power in order to address the above issues with CFD techniques. This study attempts to explore efficiency and performance for different versions of CFD packages for a lid driven cavity case. The primary goal is to conduct simulations and analyze the above aspects of performance and scalability of incompressible CFD solvers. The results will be compared and assessed and the merits of every tool will be discussed promoting the appropriate use and highlighting pros and cons. This study enriches common knowledge related to which tool is more appropriate for each situation. To our knowledge, it is the first time commercial and open-source solvers have been compared in this level of performance running employing clusters.

In this study we will discuss the performance of two well known CFD codes in respect to the used infrastructure. For CFD evaluation lid-driven cavity problem will run on Cranfield's super computer and linux cluster. The contribution of MPI libraries, interconnection speed, and parallel architecture are of paramount importance and will be examined in relation to platforms' overall behaviour

Keywords:

High Performance Computing, cavity, MPI, FLUENT, OpenFOAM, parallel processing, Interconnection, grid, clusters, RANS, efficiency, benchmark, decomposition

ACKNOWLEDGEMENTS

First of all I would like to express my deepest gratitude to my supervisor Dr Shapiro Evgeniy for all the precious advices, guidance and constant support in helping me to carry out successfully this work. In turn, I want to thank Rentiniotis Athanasios, Antoniadis Antonios and Sourmaidou Damiani, for their active participation in this study. Special thanks to Llamas Isaac, Shi Xiaoshuang, Szikszay Fábri Anna for their great support during my simulations.

Many thanks to all the people I have met in Cranfield University, whose friendship I will always enjoy. Finally I would like to thank my family and my friends from Greece who encouraged me over these months.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1 Introduction

The quest to understand the chaotic behaviour of fluids' had been a continuous and significant issue for mankind, from Archimedes to Reynolds, Euler and, more recently, the engineering community. The flow behaviour of fluids is of paramount importance and affects processes of real world applications. Navier-Stokes (NS) equations describe physical phenomena related to fluid dynamics with accuracy and clarity. Computational Fluid Dynamics (CFD) is the apparatus able to bring a different perspective to the studying and solving of partial differential equations (PDEs) in similar cases. Computer science contributes significantly to the accomplishment of these goals.

By modelling the flow by using CFD computer software, one can simulate, assess and analyze physics related to the flow of any substance. Medical, mechanical, physical, industrial, pharmaceutical, and biomedical applications make considerable use of CFD principles to analyze flows, processes, performance, effectiveness, and interactions of their concern. The impact of these applications varies from very trivial cases such as the design of a mug to very delicate and complex structures such as aeroplanes or even city blueprints and molecule movements. Although the design scales change, the principles remain the same. CFD is a pervasive concept in many aspects of real life and its accompanying tools are more than necessary. Applications of the previously referred fields provide humanity knowledge for even greater advances. The applications to be described in this thesis have the potential of offering great advances to both the science community and the general population.

The choice of the appropriate tools is one of the most critical aspects that could affect the whole study and the final results in terms of accuracy and fidelity. Diverse metrics and needs lead software engineers to develop various scientific programs to be used on computers. The number of requirements grows rapidly and so does the number of programmes. Every user is called to select from a vast of collections of programmes to deal with his/her problem. Whether one is an experienced user or not the chances of choosing the most optimal software package are very slim. The need to simplify the choice from amongst the dominant categories of available options is the inspiration for this study. An additional motivation is to comprehend the importance of computational systems and to evaluate their current capabilities. Ultimately scientific knowledge gains from such studies as it helps researchers not to be distracted from their projects and it answers the questions related to comparisons between equivalent options.

1.1 Literature Review

In fluid mechanics there are three basic methods of analyzing the flow. The classical method includes the analytical description of the flow with NS equations. However, the complexity of mathematical models discourages the analysts from employing it. In addition, experimental methods were used or invented such as the use of huge (wind) tunnels or experimental rigs that attempted to simulate flow conditions that were as close to reality as possible. The high operating cost and difficulties to setup of problem's environment limits and discourages their usage. Thus, the need for an easy, understandable, versatile and cost effective solution has become imperative. Nowadays the most modern and most frequently applied method is the CFD. It is worth mentioning that both CFD and experimental methods are commonly part of design pipeline with the former being the method in the first stages – despite the percentage of errors and credibility – because of the ease of use. An experimental measurement is the evaluation method of high fidelity at the end phases. Generally speaking, it is estimated that in most of the recent applications 80% of the time is utilized for simulations, thus determining the importance of this method. The above described procedure is followed in the most recent projects with differences depending on the uniqueness of the case.

Computational Fluid Dynamics is the technology that makes up all the physics, mathematics and computing needed in order to approach the dynamic behaviour of flows around various objects in a very accurate way. Physics is related to fluids either in motion or rest state. Physical characteristics of fluids can be described by mathematical equations and models, often in PDEs form, the so-called govern equations. Largely speaking, all CFD concepts are based on fundamental problems such as heat or mass transfer, gas or fluid flow, compressible or incompressible fluids in various speeds, chemical reactions and generally conservation laws. Computer scientists make use of high level programming languages in order to solve these equations in a very accurate way. The ultimate goal is to achieve both minimum complexity and maximum accuracy in the shortest amount of time while capturing the essence of relevant physics.

NS equations can solve problems related to fluids but with some restrictions in their use. The main problem is that although they describe the nature of any problem very accurately, the analytical solution is very difficult and sometimes impossible to calculate. It therefore seems that the only option left is for numerical solutions to be found that can solve them as accurately and quickly as possible. This means transferring NS equations from their continuous form into the discrete one, that is, applying discretisation. In order to minimize the error, the discrete results should lie as close as possible to the continuous form. The approximations are applied to small spatial and/or time domains. Therefore, the numerical solution provides results at discrete locations in space and time. Since the solution is implemented numerically, further qualifications occur. The selection of strategy to be used lies in the nature of the problem. Usually implicit or explicit methods make use of matrices or iterative techniques and it is ideal if they are integrated with a computational system because of the excessive amount of processes and the execution time. Thereby, several techniques were advanced or developed to transfer the initial problem to a more understandable and solvable form.

Many industrial applications from different scientific fields make use of CFD principles. First the automobile and aeroplane industries come with their great inventions. Cars and planes with appropriate aero dynamical shape have become faster, more resilient, more resistant to physical deterioration and they consume less fuel. In bio-medics inhalation and exhalation affect air flow around the human face. Circulatory system simulation including arteries, veins, hears and vascular tissue is a vast area of interest as well. As in almost every huge scientific field, military use is one of the main triggers. So defence research aims to improve instrumentation and invent new and more effective equipment or improve the existing ones in intelligent ways which fulfil specific needs. This has led to faster and more flexible planes, vehicles, submarines, ships, and missiles. In electronics cooling the components is an essential aspect of efficiency and performance that is sometimes critical for the viability of chips. Pollution dissipation in rural, urban and industrial areas affects environmental issues. Space shuttles and satellites need to endure hard atmospheric conditions, which require them to be more resistant especially at high temperatures. Thus, the CFD is vitally important in every field.

All the above applications place a set of common requirements that must be met. Specifically, all the cases are related to multi-physics in a complex and dynamic environment. The geometry is also complex and its scale includes an immense amount of smaller components. The need for real time metrics also add an extra degree of difficulty to encompass the essence of physics. The most important features are laminar or turbulent, single- or double- or multi-phase flows, chemically or inert reactive materials, simple or complex fluids. Additionally, viscous or inviscid regions, uniform or non-uniform, steady or quasi-steady or unsteady, internal or external, natural or forced flows either in two- or three-dimensions create even more complicated scenarios. It is clear that the accuracy, model consistency and coherence are very important. Situations of high speed and turbulent flow present the highest interest both from the physical and computational point of view, while more complex effects occur – which at lower speeds are negligible. It is vital to remain as close to the fundamental nature of the problem.

1.1.1 Mathematical Modelling

The mathematical model that represents the above physics must be expressed as precisely so as to meet the requirements and limitations of the problem. In particular, turbulent models and conservation laws are the prime participants. Moving from averaged to more time accurate but also more computationally expensive Reynolds-Average Navier-Stokes (RANS), Large Eddy Simulation (LES) and Direct Numerical (DNS) models are the three major categories one meets. Thus, if one is aware of the pros and cons of every mathematical model one can obtain the desired level of information in respect of corresponding cost.

The above three models present particular features, and these affect the study of the problem. RANS models all the turbulent scales, but it cannot capture the complexity. LES models the “small” turbulent scales, whereas the “large” ones are resolved. Therefore, RANS is considered inaccurate, while LES remains impractical for many flows. Because of the excess of computational power a few projects change from RANS to LES come to the surface, although there are still doubts about this. DNS resolves the turbulence of every scale, even of the tiny Kolmogorov eddies, but it is not feasible for very high Reynolds Number (Re) because of the computational cost. Frequently hybrid versions emerge for specific cases which are a customisation of the above, that is, Unsteady-RANS (URANS) and hybrid-RANS-LES. These reduce cost while at the same time produce a satisfactory outcome. The quality and quantity of requested information over the amount of computational resources and time indicate which model is appropriate.

1.1.2 Numerical Modelling

Practically speaking, in most of the engineering problems, an analytical solution does not exist. Therefore, numerical solutions need to be explored and discovered in order to approximate the analytical solution as accurately as possible, while capturing the essence of any problem. Numerical model denotes the category of mathematical models that use numerical time-stepping procedures to attain the models behaviour over time. The main methods are Finite Element (FE), Finite Difference (FD), Finite Volume (FV) and Spectral Element. All the methods are appropriate for spatial discretisation. These methods are based on the tessellation of spatial domain to finite spaces. The first three are purely applied mathematical methods, whereas the last is visualisation. PDEs are transformed from continuous to finite form, and from non-linear to linear in order to accelerate and facilitate the algorithmic steps. Discretised equations are appropriate for computational systems, as they consist of primitive mathematical types (sums and products) which can be calculated very fast. By applying discretisation, the obtained algebraic systems can be solved by direct, approximate or iterative methods, which are convenient for computers of any scale. The primitive calculations are effectively resolved due to hard programmed components of computers which perform them in a short amount of time. For the time being, there are many libraries for linear systems such as Netlib[6].

While NS equations can be written in a discretised form, problems are introduced due to irregular geometry or other discontinuities – usually on boundaries. The following numerical solutions attempt to address that problem. Summarising their main features:

FE is the most common method that uses numerical techniques in order to solve Partial Differential Equations or integral equations problems. It is used frequently on structural mechanics problems both in engineering and in physics. The method simply dictates that the domain is divided into structural elements, where composing all of them together can form a more delicate geometry as in Figure 1. It is clear that it is good to be aware of the characteristics of individual structural elements and to allocate them accordingly in space. In this way the governing equations for the whole construction could be derived. Thus, simultaneous algebraic equations are produced, and this number will pose restrictions and increased complexity on the analysis if it is very high.

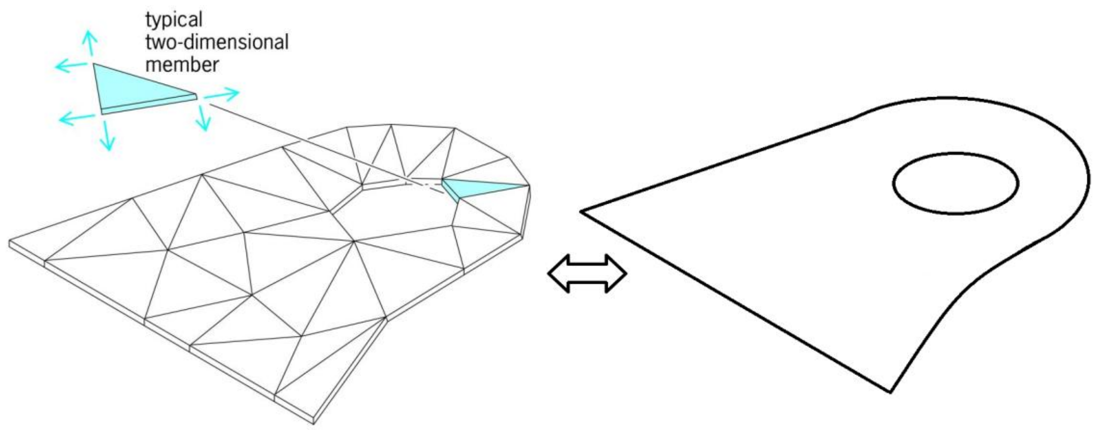


Figure 1 Finite Element Method applied on a random structure

FD is considered as a special case of FE but the quality of approximation is lower. The main concept is to approximate differential operators with difference operators as in Figure 2. Thus, 3 more cases occur, forward differencing, central differencing, backward differencing with various precision. The approximation is based on expansion series such as Fourier series, Taylor series and so forth, where truncation error needs to be specified accordingly. Various order achieves a respective degree of accuracy. This method is based on points, and once applied a system of algebraic equations is obtained. It often does not capture the essence of physics that accurately.

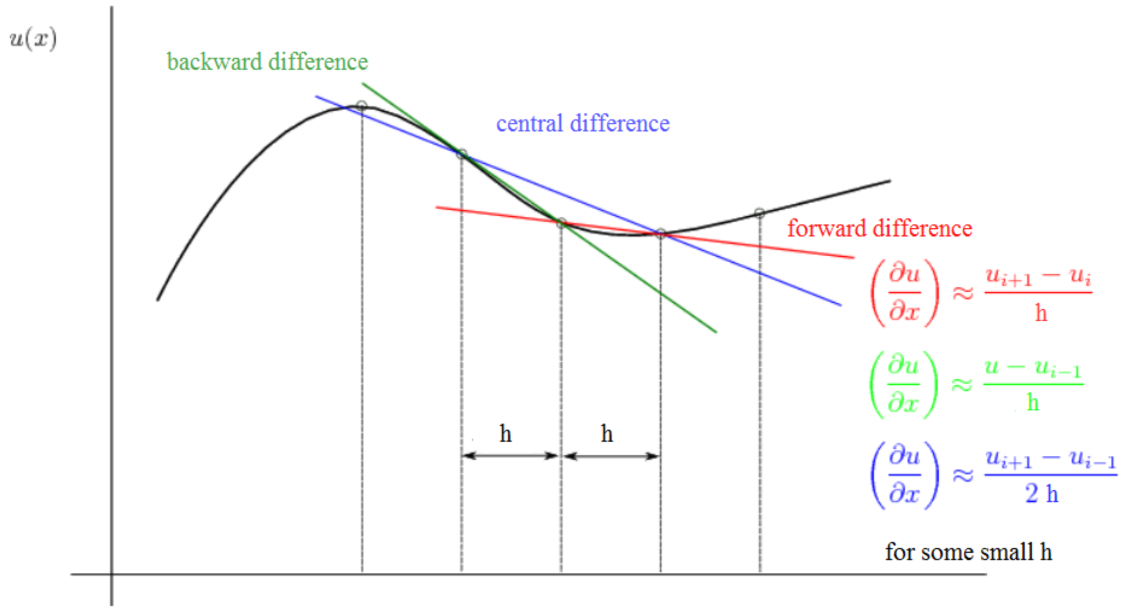

$u(x)$
backward difference
central difference
forward difference
$\left(\frac{\partial u}{\partial x}\right) \approx \frac{u_{i+1} - u_i}{h}$
$\left(\frac{\partial u}{\partial x}\right) \approx \frac{u - u_{i-1}}{h}$
$\left(\frac{\partial u}{\partial x}\right) \approx \frac{u_{i+1} - u_{i-1}}{2\,h}$
h
h
for some small h

Figure 2 Finite Differences method on a 2D plane over smooth curve

The principle of FV is the division of the domain into continuous control volumes, but considering a grid of cells as depicted in Figure 3 for a simple 2D case. Usually the boundaries of the cell are called north, east, south and east, starting from the top edge in clockwise direction. For ease of computation, the integral forms of NS are used and they are applied to each volume separately. The reason for employing integrals is that they obey exactly and accurately to conservation laws and principles – while FD cannot, and they can be implemented in parallel efficiently. Within each cell discretised NS equations are solved iteratively. For boundaries flow properties are approximated accounting the information stored in the centre of the cell. That is called reconstruction and it is implemented in various methods along with different high order schemes facilitating specific design/structure needs.

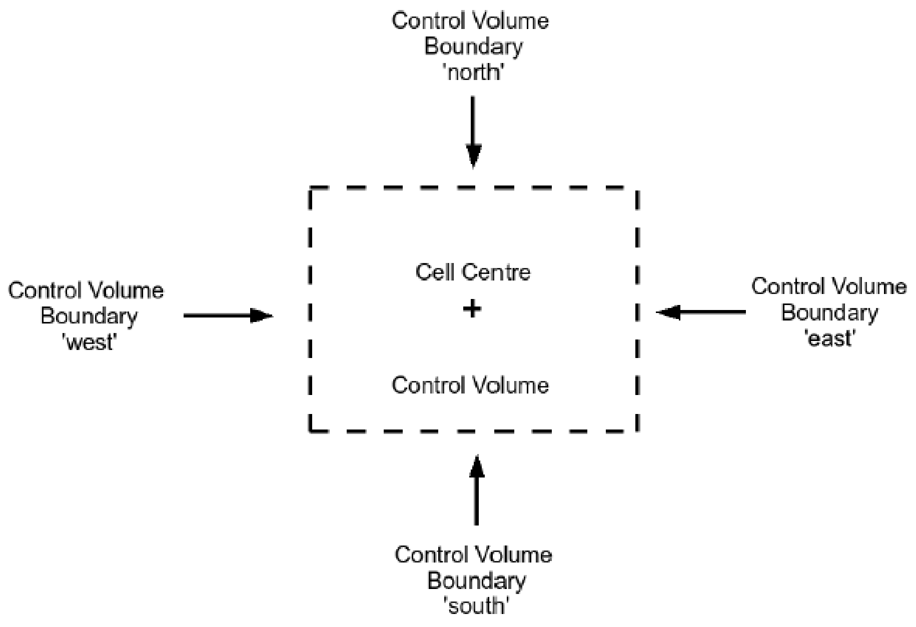


Figure 3 Finite Volume Schematic – Control Volume

FV is the most commonly used method compared to the other two. This is because FE is a too generic method and FD is not related to the physics of the problem. FD cannot cope well with unstructured grids, while most of CFD problems do include them, and it is not necessarily a conservative method. Although FE resolves shockwaves and discontinuities, it is not as good as FV for boundary layers and clustering problems. Thus, with FV more accurate and higher fidelity results could be obtained compared to their counterparts, although there is still a fraction of ambiguity over real world results.

Moreover, numerical solutions are divided into explicit and implicit. When the calculation is resolved based on previous time-steps it is called explicit, whereas when it takes values from the next time-step it is called implicit. Depending on numerical modelling explicit and implicit, solvers exist achieving various characteristics. Implicit methods are considered stable in respect to time integration, whereas explicit induce constraint on time-step. Solver packages integrate one or more of the above methods either in primitive or hybrid form, but the choice of the used method is up to the user. Some solvers offer the option of modifying and programming complementary or new functions but the kernel remains the same. For example, FLUENT[17]-users can develop UDF functions, whereas OpenFOAM[72]-users can write their own solver for specific features of flow and mesh.

1.1.3 Grid

Accumulatively, the finite spaces either in 2D or 3D form the grid that discretises any geometrical structure and substantially, the grid is the input for the solver, The better it is formulated, the better the results, in respect of time, complexity and feasibility for computational procedures. It is clear that, the higher the order of polynomial reconstruction methods, the higher the accuracy and the resolution obtained. The solution of the conservation equations requires an arrangement of a discrete set of grids or cells in the flow field; their determination for a given body is known as grid generation. Grid generation produces the mesh of any given geometry which, in turn, will be used as input for the solver. The real challenge is the creation of a grid that can cover the whole test object with that many cells in order to capture the essence of all the physics and the flow, but small enough for reduced complexity and, data- and time economy.

Grids present a variety of features that support their use in specific solvers. A grid could be either structured –organized so that indexing can directly address neighbour cells for the whole geometry – or unstructured. Frequently, the geometry is split in small spaces. Each space is called structured block and it can be out off either structured or unstructured grid Structured grids are used on boundary layers because cells can stretch and unstructured for inner regions. The knowledge of the topology and the geometry could offer insight into what type of grid will be used and what the geometric shapes will consist of. This will have a direct impact on the performance of the solver. In regard to numerical methods, FV can be applied to uniform and non-uniform meshes, while the FD requires a uniform rectangular grid. However, in all real-life problems, a uniform rectangular grid is not possible in the physical plane. Although mapping curvilinear objects to a regular plane in computational domain occurs, the mapping is not always effective or even feasible. An ideal grid, that facilitates CFD principles, should be as smooth, refined and continuous as possible.

By using FV the grid can be either staggered or collocated. The former stores the scalar variables in the centre of the control volume, while the velocity and momentum are located at the faces. Hence, staggered storage is ideal for structured grids over incompressible flows. It can generate velocities wherever they are required for scalar computations related to fluids' convection, diffusion and transport. Usually staggered grid is applied for velocity components for proper representation of pressure in the discretised momentum equations. [38]. A fuller discussion in respect to the solvers takes place in 2.4. The disadvantage is the difficulty in handling diverse control volumes for various variables while keep tracking of the metrics. In contrast, collocated grid stores all variable in the same position.

1.1.4 Solution and Analysis

The development and implementation of solvers that can determine CFD problems is a very difficult procedure because of the complexity of both the input data and the corresponding algorithm on every occasion. Data manipulation is subject to programming language so the first criterion is the appropriate coding scheme. The mathematics, behind partial differential equations, requires the creation of special data types with tight bonds and dependencies. The more complex the data structures the slower their access is and the more sensitive in algorithmic manipulation and communication (especially in parallel instances). Thus, the writing and the (re)use of libraries are critical aspects and can reduce the complexity and incorporate all the necessary functionalities in an easy-syntax way in order to be accessible to as many developers as possible. It is evident that the choice of the programming language within the appropriate platform is an important issue. For this reason FORTRAN language, for which compilers produce the quickest executable files for numerical applications, and sometimes C/C++, is chosen either in serial or parallel version while ,as demonstrated, in primitive instructions it is less time consuming-[66]. However some attempts in JAVA have been tried out with near similar performance. This is largely because of the modification of parts of the initial structure and because of compiler dependencies and libraries-[62], [67]. However, some JAVA applications produce analogous results as their competitors - while the number of nodes increases the efficiency slowly drops and the time-gap between equivalent instances remains steady.

In fact the majority of the users and programmers would only spend time on programming solvers in special cases. The reason is that there are commercial packages that can do the job instead. Commercial-off-the-shelf (COTS) CFD solvers are very useful tools because they incorporate many mathematical and physical models and aggregated experience with various graphical tools. Thus the user’s only problem is to consider what settings should be configured while using the solver. Regardless of the above comparison of languages each of them offers characteristics that are more or less appropriate to any given case. Primarily FORTRAN is used by academia because of the execution speed and its scientific nature, while some of the most common applications are written in C (e.g. FLUENT) mainly because of interfacing and coherency factors. Although the latter is generic and easy to use and much more powerful, the disadvantage is its behaviour in some delicate cases of high interest. This could make it less effective when compared to the pure algorithmic approach. Thus, it is preferable that both ways should be used in order to arrive at final decisions and assess each case study.

1.1.5 High Performance Computing

As the methods and technology advance more needs emerge either for time or for efficiency. In the case of simulations the goal is to include as many details of reality as possible, while achieving as much accuracy as possible at the same time. All numerical methods divide and analyze the initial space into smaller sub-domains. The scale of particles or cells that contribute to the final behaviour of the fluid is so small and also the computations (applied set of NS discretised equations) for each of cells are so complex and massive that it is necessary to make use of immense computer power for calculations. Obviously the number of all the entities that are part of the system is so big that the use of high end computing infrastructure is the only feasible way to have an accurate approach in time-efficient intervals. In general most of the applications are implemented in distributed memory systems. This is mainly due to portability and interconnection issues, I/O imposed overhead and memory dependencies. It is crystal clear that by this kind of work cannot run on any personal computer.

The community of High Performance users demands high frequency processors in order to address their problems quicker. However, physical layout problems, difficulty in integrating specific features onto the chip and applications' requirements related to memory referencing resulted in any research being done on such projects ceasing. For a single general purpose (GP) processor (Complete-Instruction-Set-Computer, i.e. CISC architecture integrated in silicon chips) the upper bound is the frequency that CPUs can perform calculations. Since nowadays the frequency of one core of processor cannot exceed ~4GHz due to physical/material constraints, a decision was made to use multiple entities (either inside the same CPU or not) to achieve the former scope in a coordinated and managed way. This induced the meaning of parallelisation, i.e. splitting the initial task into smaller chunks, addressing each part to individual core and running every part simultaneously. So, the next step was to bundle many machines in small networks under a shared interconnection. However the above do not meet the requirements for the medium and big order of magnitude of CFD applications. The evolution of CPU architecture had led to the integration of multiple cores into one die. Finally, because of the need for even more power many machines each with multi core (dual- or quad-core) processors were employed to solve big problems. Also recent advances of processors have come close to addressing referencing problems by integrating controllers within the die minimizing communication time. This has also led to the creation of new type of machines that handle multiple instructions e.g. supercomputers.

Depending on the configuration of the available infrastructure the applications function accordingly. If the machines are located in a “close” range, this arrangement is called a cluster, otherwise the term Grid is used. This is not to be confused with structure-grid applied in CFD methods. The definition of “close” is arbitrary, but usually it ranges from one room to one building. Also depending on the granularity of the application some versions are recommended to operate in specific machines because their architecture supports better correspondence on system calls. Very often gridified or clusterised versions derive from original application as unique features of the infrastructure have been taken into account. Thus, overall performance levels are increased in terms of speed and efficiency, as demonstrated in performance plots.

From the very beginning, HPC has set high standard for cutting edge technologies. The top500 is a project that ranks and details the 500 most powerful (non-distributed) computer systems in the world. It actually aims at maximum performance metrics and its list is updated twice a year and includes the latest achievements in HPC. It also presents peak performance, but this is only a theoretical value in ideal situations of zero-delay communications and flawless runs. Watching the magnitudes in market shares could offer a first understanding of HPC trends. As show in top500 the majority of HPC applications running on supercomputers are related to Research, Finance, and Geophysics-[94]. The former had risen the most since 1993-[95]. Accumulatively, since its start, HPC has increased in linear fashion year by year as more computational power is needed-[9]. Nowadays the greatest market share of vendors is held by computer electronics manufacturer (e.g. ALTERA, INTEL. NVIDIA), computer network manufacturer (CISCO), computer system manufacturers (e.g. DELL, HP, IBM), and CFD enterprise (ANSYS). Search engines are excluded from comparison as they are used by any user, while the scientific/industrial community is a subset of the former-[2]. In terms of segments owing huge HPC infrastructure, industry dominates the charts (60%), followed by research (18%) and academia (~16%)-[11]. Although, more interest is gathered around HPC only two supercomputers, listed at the top 2 places of the top500 list, host CPUs within 1025-2048, whereas the range of 4k-8k holds the greatest share and the category 2049-4096 comes second-[7]. The need for HPC points a bright future for even more computational power.

The nature of the problem and the used algorithms affect the solution procedure even more. CFD problems include many tiny components and/or demand very accurate numerical results that prompt the use of special computer architecture instead of traditional tools. In numerical cases the final solution is obtained after many complex algorithmic iterations. Accumulatively the number of computations is large enough that single-chip – single, dual or quad core (in the best case scenario) – computers would need an immense amount of time and sometimes even resources in order to produce any solution. This situation created the need for High Performance Computing (HPC). It is the recruitment and/or creation of supercomputers to solve highly computationally demanding cases that are either too large or too slow for traditional computers. Therefore simulations are executed in a very significant fraction of time and challenging problems are solved easily.

Aside from the computers aspect of the problem there is another issue. The lack of insight and perception between engineers and developers is so great that intervention should be considered essential. Despite the extra complexity of the problems that involve many processors, knowing and using appropriately the existing/available tools could provide quick answer to the final solution-[60].

1.1.6 Scalability

Over time many different programs, algorithms and machines have been developed and employed in order to address the above issues. Because of ubiquitous Multiple-Instructions-Multiple-Data (**MIMD**) systems, parallelization is highly advisable and it can be achieved in various ways both on hardware and software form. Connecting diverse systems can also be a good solution, but dependencies should be specified and determined so that the weakest machine carries the lightest load and only used on rare occasions so as not to stall or slow down the rest of the system. Also bandwidth starvation both in memory and network often restrain the speed-up (which is explained below). Nevertheless, the workload should be spread accordingly in cases of heterogeneity. Current capabilities of CFD are limited due to complex geometry and/or physics but the technology also sets the upper limits of time/speed gain. Thus, parallelization saves time and the results are calculated quicker. The most powerful computer infrastructures demand an adequate amount of space for reasons of cooling/overheating, maintenance, power supply, monitoring and so on… One of the first attempts of the parallel machine era linear scaling was achieved up to 128 cores, thus improving numerical methodology along with accompanying CADs-[65] to the very recent with 15,000 cores

Obviously the more concentrated these resources, the more the cost rises exponentially so it is wiser to distribute the resources. On the other hand, all of the available resources in one place could never solve problems of this magnitude, in comparison with distributed machines. Therefore, it is necessary to collaborate with appropriate resources scattered around the world. The level of distribution determines whether these resources are tightly coupled in a close area, which is called a cluster or they are spread in larger interconnected areas, in other words, a Grid. It is clear that there are also intermediate cases. All of them can manipulate heavy computational situations assuming that the programming part can support them.

Although parallelism is the way, into how many parallel parts could any job be divided? There are two models for parallel computations, these are, Equal Duration Model and Parallel Computation with Serial Sections Model. Because the first has been proven to be unreliasable, thus the latter is the one applied in every study. In numbers language:

Let's consider the statements below for any program that it can be parallelized for N processors. If the data dependency is high and sequential there is no point arguing about parallelism.

Therefore, satisfying the previous condition, there is always s serial part and a p parallel part such as s+p=1

T_o be the overall execution time

T_s be the execution time for the serial code

T_p be the execution time for the parallel code $N \cdot T_p = T_s$

Thus, Speed-up S denotes the ratio of time gain of serial execution time over the parallel. Including the previous sentences:

$$S \leq \frac{T_s}{s \cdot T_s + p \cdot T_p} = \frac{1}{s + p / N} \tag{1-1}$$

The inequality exists in order to be strictly mathematically consistent. There are external factors such as communicator speed, network throughput, CPU suspend time etc , that they can't be predicted since their value varies dynamically, but only a very small number might cause irregularities.

The former formula is not totally right because the factor of communication time T_c among the resources is considered negligible comparing to T_s. If it was comparable with either serial of parallel time, or even higher than them, then there would not be any point discussing efficiency since its effect would be deceleration which is meaningless. For the sake of simplicity it is omitted. In the very extreme cases where the initial job is split into so many chunks that its execution time is comparable to T_t the above formula should be revised. This situation usually occurs when the simulation is conducted in very big systems. Then, the algorithmic procedure should take into account complementary parameters. In these cases everything depends on network interconnection, as discussed in a later section. Thus the above formula changes to:

$$S \le \frac{N}{s \cdot (N-1) + 1 + N \cdot (T_c / T_s)} \qquad \textbf{(1-2)}$$

In the very end since there is an upper limit, the problem transforms into an optimization case where the right hand side must be the as large as possible. That means that denominator should be as small as possible, which means that, for very a large number, N should be infinite. In a more realistic approach minimizing T_c is achievable nowadays thanks to interconnection hardware. Obviously, it is not feasible to have infinite processing nodes, but a very big number is necessary. Implicitly, at the same time the serial part must be also very small. Generally the idea is to write up a program in such a way that the parallel part should be the largest possible and provide a huge amount of computational resources so as to achieve the ideal speed-up. Usually the more resources provided the more speed-up is expected. But, sometimes there are implications in the behaviour of the initial program i.e. algorithmic logical errors or misconceptions, the infrastructure’s current state, failures in computers’ components (partial or overall), unpredicted memory failures and/or leakages etc that could potentially affect concepts that were not taken into account from the start.

In 1940 Grosch axiomatically stated that the computational power of any system rises exponentially with its manufacture cost.

$$P = K \cdot C^S \qquad \textbf{(1-3)}$$

Where S usually lies between 2 and 3.

According to Amdahl’s law there is also an upper limit in the amount of parallelization one can achieve in any application. Thus:

$$S \le \frac{N}{1 + s \cdot (N-1)} \qquad \textbf{(1-4)}$$

In practice, it has been observed that actually the value of s depends on the number of N. In other words:

$$S \le \frac{N}{1 + s(\mathrm{N}) \cdot (N - 1)} \tag{1-5}$$

On the other hand Gustafson – Barsis, by observing systems with millions of processors (Massive parallel Processing/ors - MPP), concluded that significant size problems could be parallelized effectively and including Amdahl's law they produced an alternative formula for speed-up which is:

$$\mathrm{SS} \le \mathrm{N} - (\mathrm{N} - 1) \cdot \mathrm{s} \tag{1-6}$$

However, the above formula has some restrictions such as the existence of available memory for the CPU. It dictates that there is an upper limit on the number of tasks to be fulfilled within a given time interval. That was a milestone for parallel computing researchers.

Both Amdal's and Gustafson-Barsis's laws are very important but not mutually exclusive! In fact these are called strong and weak scalability respectively- [55]. Also it is important to distinguish between **scale-up** which is the "upgrade" to a bigger and more powerful server, whereas **scale-out** is the addition of more computational nodes on system, usually for clusters.

Some important notations follow. **Strong scalability** denotes how the solution time varies according to the number of processors for a fixed *total* problem size, whereas **weak scalability**, describes how the solution time varies according to the number of processors for a fixed problem size *per processor*. **Efficiency** is the fraction of speed-up over the number of used processors. **Throughput** denotes how many jobs are executed in a given time interval. **Turnaround time** denotes the clear time needed starting from submission of a job until its completion, excluding pending time. Nowadays, all the CPUs are MIMD type. MIMD systems are categorized according to the way they use CPUs and memory. Because memory is the bottleneck, higher interest is attracted to **shared memory** and **distributed memory** systems. The former are the machines that have access to a common pool of memory which is fast but often suffers from race conditions. The latter are individual machines interconnected to each other in various configurations. Alternatively, considering the CPUs arrangement as criterion for taxonomy, one individuates **symmetric** and **asymmetric multiprocessing**. In order to measure parallel applications' performance, the metric Floating Point Operations Per Second (**FLOPs**) is used. It is equal to the product of the frequency of processing node times the number of nodes times the number of floating point units per processing node. Obviously for optimal efficiency all the processing nodes should be identical.

Alternatively the summation is calculated but in the real integration of the resources variety may cause relapses. Bearing in mind all the above the efficiency plane is set. Therefore all the scientific attempts aim to obtain the largest fraction of that plane to capture all the merits of computational systems.

The execution of heavy computations that characterize any numerical problem, within short term is the main principle of HPC. Basic components – which are called nodes – consist of technologically cutting edge computational resources such as high frequency CPUs, large and quick memory bundles and huge storage servers. Usually these components along with a few vestigial front-end nodes are integrated into racks dedicated for HPC applications that require persistent maintenance, continuous environmental provisions (such as power and cooling) and permanent support. Improving the response rate, powerful front-end nodes that can serve many users are necessary. It is also critical to install a fast network interconnection among the nodes in order to achieve the lowest turnaround timings. It is obvious that the more the resources are used, the more difficult their preservation and maintenance is, as the system's complexity rises in an exponential way. The same concept applies in diverse resource arrangements from local clusters to supercomputers, and from grid- to cloud computing. Thus, if the system behaves more optimally it could handle all the requirements for running any demanding simulation.

While programming in parallel, the primary goal is the reduction of execution timings and complexity, i.e. computation, idle and communication time between the nodes, successful conversion of sequential code to parallel and ideal load balancing. Ideal parallelization is achieved through dynamic load balancing according to resources workload taking into account already running processes and minimizing the interactions with outer components e.g. network drivers or instrumental interlinked equipment. Also message exchange reduction can accelerate job completion. Thus, minimizing communication, whenever it is possible, either in terms of messages sent or data exchange and overlapping communications and computations (with aggregated functions/routines) could lead to optimal scaling. Compilers, profilers and coding are the main contributors used for the development of parallel applications. One of the first attempts to use parallel techniques in solving FEA problems indicate that the future of scalability is promising and it is leading concern for bigger advances exploiting the massive computational power-[49]. The more processors that are gathered to solve a 3D NS equations problem, the quicker a solution will be produced. Solver's efficiency with more than 16 nodes is higher than 95% providing by order of magnitude better timings-[75].

The desired property in every parallel application is the super linearity, which denotes higher rate in scalability plot than linear for each core(s) added. Generally it is the attainment of better system performance as problem size increases. It has been proven that Message Passing libraries can achieve this effectively. When applications run on a cluster infrastructure with appropriate architecture that serves algorithmic demands and the nature of the problem is designed in a way that the more processors added, the smaller the sub-problem becomes, impressive results are produced. Both near-linear and super-linear behaviour for CFD cases has been illustrated in-[90], using supercomputers. Moreover, as will be discussed below, appropriate programming techniques and coherent infrastructure deliver linear scalability on a very large number of CPUs-[64]. Memory architecture issues should usually be taken care of in advance because that could lead to a bottleneck in the system and to failures or crash-[68]. In rare cases, linear behaviour cannot be achieved due to flawed programming or high complexity and data dependencies of the solver.

In any parallel application the most important factors to be specified – on the hardware side – are the compute nodes arrangement and the interconnection among them. In any case CPU architecture and compilers – along with memory bandwidth – can improve performance. Therefore, achieving the lowest feasible turnaround time produces a better performance with peak efficiency 1.05 in some cases. Conducted simulations in mixed network configuration clusters depict linear and sometimes superlinear scalability on Euler and NS equations achieving simultaneously high FLOPS measurements-[86].

Recent studies on the same cases were conducted with even more important results. Conducting even more simulations and tests under various and more advanced infrastructure configurations the efficiency reached 1.3 with linear scalability. Two different instances of METIS library (introduced in the next section) were used achieving similar results up to 256 nodes. Crucial factors were memory bandwidth (which significantly affects overall performance over multi-core processors) and communication (that demands quicker message exchange). This kind of supporting equipment is very expensive – suggesting that more advanced network technologies need to be developed as they suppress the potential of multi-cores' potential-[85].

In [69] developers aimed at writing a sequential code, distributing data accordingly and decreasing message exchanges for 3 CFD applications-benchmarks. Therefore, scalability at software level was achieved. Modifications and optimization of the code improved significantly overall flops metrics. Additionally, implementing kernels for the architecture of specific machine produced results that are more than satisfactory-[14].

It is important to mention that the frequency and cache memory of the cores is not the same for dual and quad. Specifications of dual cores list higher rates and larger capacity respectively. The more the computing nodes are gathered together the higher the emission of heat affects the performances of CPUs' frequency as electrons collide with each other. CPU's architecture undoubtedly influences the performance, because of hard programmed components and operation either as single core or as a whole-[30]. So far big manufacturers have managed to handle their resource arrangements to scale almost linearly over CFD applications while keeping the efficiency as high as possible-[89]. As stated below in certain situations the use of dual cores is preferable as it affects power consumption and environmental issues. However, quad cores present better price-to-perform metric and they are ideal for applications where time is the most critical factor. Even in smaller scale simulations the authors of [79] suggest that the right blend of hardware and software resources along with the right strategy can derive equivalent results highlighting the importance of choice of various architectures on single- and dual-core-based supercomputers.

In contrast, software parallelization can be very frustrating and hard. In order to address that issue parallel libraries and standards have been instituted in collaboration with dedicated tools for parallelism, such as parallel debuggers, profilers and workflow designers. Nowadays there are many message passing protocols such as Theoretical Chemistry Message Passing Tool Kit (TCMSG), Parasoft Express (EXPRESS), Network Linda (LINDA), and Parallel Virtual Machine (PVM). The mainstream in parallel application implementation is the use either of MPI or OpenMP. The former performs well on Symmetric MultiProcessing (SMP – categorization based on memory usage by CPUs) and distributed memory systems, whereas the latter is appropriate only for SMP. Although these two methods provide very good results in high performance applications, their structure differs and for this reason they are more or less appropriate for various problems. Alternatively a few programming languages provide their own flavour of parallelism, although they might already be compatible with the above standards, such as High Performance Fortran (HPF).

Even if the resources are available, the algorithm is possible the bottleneck according to [26]. Newton-Krylov-Scharz (NKS) CFD solver cases run both on Cray T3E and IBM SP super computers delivered near linear efficiency with the former achieving slightly better measurements. Analysing the results from simulations, trends for CFD solvers intensively move towards latency tolerant algorithms since parallel infrastructures are pervasive. So far algorithms are I/O bounded and affected by dynamic distribution, although some hybrid implementations attempt to overcome the above obstacles, for example [99].

On the one hand, OpenMP is easier to program and debug and as a result the code is more understandable and more maintained while it also offers gradual parallelization. Nevertheless, it is mainly used for loop parallelization and runs only on SMP while complementary OpenMP compatible compiler is needed. On the other hand MPI runs on both shared and distributed memory systems. Thus, it applies to more cases and is more affordable (in the case of distributed systems, because of their price) and provides data locality. Drawbacks are the exclusive use in SMPs, choice of compatible OpenMP compiler and mainly loop parallelization. Therefore it is not used to program CFD applications in OpenMP.

On the other hand, MPI is a standard that comes in many flavours either specific or general. Some vendors or institutes have created their own versions in order to serve their needs according to the projects for implementation. For example, HP-MPI collection, powered by Hewlett-Packard, presents better overall performance-[43]. OpenMPI offers the “mpi_leave_pinned” trigger that improves large message exchange-[93]. Also tools that increase parallelism detect wait cycles that stall synchronization. These wait states occur often and sometimes they introduce very big overhead as the problem scales. However, in its first stages, authors in [34] developed an architecture that can cope with this situation and indicate symptoms and pointing points of wait, performing traces analysis in parallel that can support up to 65536 processes. In [19], a prediction tool about performance in parallel applications was presented taking into account parallel efficiency loss measurements. This tool is ideal for large datasets achieving good performance and indicating improvements for any geometrical cases. Nonetheless there are difficulties in transferring from serial to parallel versions, as well as problems in debugging. Frequently, the network’s limitations are the main bottleneck in performance and communication between the nodes. However most of CFD packages are based on MPI such as the ones used in this study.

As an alternative way to improve efficacy a double layer MPI/MPI and a hybrid MPI/OpenMP implementations were tested. It was proven that both ways provided notable results in scalability and the applications scale more smoothly and steadily. The pure MPI way achieves better parallelism under the same configuration-[28]. Another hybrid comparative simulation illustrated that both MPI and OpenMP scale near linearly for single- and multi-grid simulations. The latter has better performance because of the shared memory architecture. However the reduction of overhead in MPI implementations in order to achieve higher efficiency is imperative-[21]. Similar results were presented in [54] whereas for small-medium enterprises OpenMP is a viable solution because of the small number of resources owned. Beyond 32 processors the efficiency drops, basically because of the number of threads (smallest processing unit) used and communication between the nodes, suggesting the need for an alternative solution such as MPI or hybrid methods. Nonetheless, hybrid implementations are not as profitable as single ones. Depending on grid properties (single- or multi- grid) speed up and efficiency single-grid clearly excels either for small Re=400, or Re=3200. For the latter case the gaps are smaller. In spite of this, the multi-grid method presents better results on convergence and the percentage of total simulation is faster and less time consuming respectively, it is the communication that slackens the procedure as a whole-[91]. Therefore, the choice of either original or hybrid methods depends on the problem's unique features.

In fact, MPI standard is used for the majority of parallel applications either as back-end (commercial CFD solvers employ it for inter-communication use, which is specified upon the installation), or directly (in-house codes) and it is supported by communication protocols. These protocols play a substantial role in the performance as they provide features that affect MPI's functionality. Usually MPI operates via TCP at transport layer, but there are also alternative/dedicated counterparts. Stream Control Transmission Protocol (SCTP) is one of them. It operates again in the same layer and it is implemented in different structure that of TCP, where in certain circumstances if it performs more effectively. It incorporates multi-streaming and, thus, provides direct mapping between streams and MPI-tags along with increased concurrency. It uses aggregated system calls and scatter socket transmissions. Through multihoming it is also fault tolerant and versatile when network routes are blocked. Tested in real world applications, if packages larger than 22Kbytes are used, it will perform quicker and with fewer losses resulting in higher throughput. In general it is recommended that it be used for shared memory environment where, considering the above, it attains higher scalability-[50].

Irrespective of the case the user cannot choose which type of library the software package will use because the kernel is hard programmed and compiled with one of the two methods. However, depending on the given infrastructure, MPI or OpenMP perform smoothly. Therefore, the right package choice is due to the nature of the problem and computational resources.

All the software packages follow the former dogmas about scalability, supposing that the user has or at least can provide the appropriate amount of resources. Hierarchical architecture is applied in allocation of resources. Since there is always a serial part that it should remain sequential that bit should run on a master node and the rest runs on worker nodes. This procedure had been proven very efficient in gridified (conversion to run on grid environment) scientific applications.

Making the assumption that are of the above issues have been resolved, there is an ultimate enhancement for an efficient parallel application. Wait states due to synchronization could often be the bottleneck. Diagnostic tools could lead to this phenomenon. The approach proposed in [34] could handle 2^{16} or even more processes exploiting parallelism of massive configurations. Running XNS CFD application near 50% efficiency was achieved concerning number of time-steps over the number of processes. Thus detecting wait states achieves satisfactory performance. Nevertheless, applications are I/O bounded and memory limitations affect orchestration inducing selective or sporadic tracing.

Frequently physical properties affect simulations over numerical schemata. For this reason explicit solvers are employed. Also, they are easily parallelised on data decomposition. Although LES is computationally expensive, it is a viable alternative providing more accuracy. During the simulation, data exchange, based on Message Passing Interface (MPI), of big sets of halo cells takes place. High-order methods (9th order) employed on HIRECOM software affect scalability levels, as more data exchange is needed among nodes. Normalising as 100% the efficiency of a simulation running on 96 nodes, the efficiency drops to 80% for 512 nodes on HPCx infrastructure at large Re-[29].

First and foremost, discretisation techniques contribute significantly to scalability and accuracy. High order methods are based on polynomials which offer extra local extreme values. In order to restrain caused oscillations in regions of high gradients, limiters are applied – while they obey the concept of Total Variation Diminishing (TVD). Usually, these limiters might locally reduce the order of discretisation in regions of high interest. The Weighted Essentially Non-Oscillatory (WENO) scheme, developed by [61], overcomes the previous weakness. Especially in specific types of flow and types of grid, its performance is significant-[20]. Also, in WENO, polynomials are used for the approximation of unknown values, such as high order schemes. For example, FD is able to produce some good results in specific circumstances. Compact-WENO is one of them. By using explicit methods and 5th or 6th order accuracy, numerical errors are negligible despite any parallel approach. It avoids global dependencies, too. Thus, scale-out is very close to theoretical values (because of communication overhead) both for coarse or fine grids, and for various numbers of participating nodes. Also efficiency is near 90% even for 1246 nodes, which represents an excellent achievement. Good overall performance makes compact-WENO suitable for more complex geometries that can even include elaborate scenarios-[25].

1.1.7 Decomposition

So far the above statements regarded the software part as a black box ignoring the type of input data. The type of data directly affects the quality of the simulation in terms of how computations proceed and final results are derived. The complexity of the structures to be tested can cause various implications. In CFD the basic principle is to analyze any given structure applying a geometrical grid, as indicated in numerical methods. Thus various primitive cells will be created. The choice of the shape and the size of these are subject to user's needs and idiosyncrasy, assuming that programs can manipulate any combination. Sometimes the nature of the problem dictates the use of very specific geometrical schemata in order to achieve desired properties and results. Following the tessellation, geometric graph partitioning algorithms are ideal to decompose the initial structure into smaller chunks either with multi-objective criteria or not. Partitioning is not affected by graphs distribution and is an on-going concept, despite its great progress in recent years-[80]. After the formation of cells these should be grouped in sets. Thereafter, these sets are assigned to the distributed resources, where calculations take place, and by the end all the results are reduced to the origin. So, proper pre-configuration is critical so as to achieve better parallelisation and higher scaling.

The procedure that concerns splitting the prime data set is called domain/data decomposition and often is the bottleneck of all parallel applications because of the dependency of data. Besides data decomposition, functional decomposition also exists when manipulating the algorithm as a function. The algorithmic functionalities are considered a function's sub-sections. The separation is subject to the number of chunks that the function can be divided into and data apply to one chunk according to set conditions. In rare cases, hybrid forms of both exist but in real engineering projects the first one is the most common. This is due to the volume of input data which is much bigger than functions. So, dividing initial data into smaller sets achieves less workload for computational resources. Thus the choice of the method is due to the application's requirements.

Partitioning data is due to the ease of computational workload and the minimum possible network load. Thus, results will be derived rapidly through computational procedures. Via the discretization method PDEs turn into big linear systems that involve huge matrices. Matrices do not offer very good properties and manipulation. So, they are mapped into hybrid schemes and form equivalent graphs which is a purely discrete structure and easier for computers to handle it faster. The transformation is a sensitive procedure and unless it is done carefully, it will have implications for later procedures-[82]. Then, optimal partition cutting applies on derived graphs subject to the restriction that discontinuity is kept as low as possible. Many structures and algorithms have been developed with various pros and cons that address this issue. One common goal is to achieve the highest parallelism, the quickest runtime and the best quality of data (minimizing errors). Geometrical based methods largely fulfil the above such as Geometric Sphere-cutting, Geometric Sphere-cutting-KL-[83].

There are several libraries and packages that implement domain decomposition for scientific applications such as PETSc[10], Aztec[96], Geo-FEM[78], PLS[23] and ADVENTURE[13]. However, in our work the library METIS[51] and more specifically the version ParMETIS – which is its parallel form – is used by all the software packages that we are using. METIS library works in a multi-constrain way splitting geometrical data represented in graphs form. In CFD field ParMETIS is an MPI library about graph and mesh partitioning, graph repartitioning, partition refinement and matrix reordering based on numerical kernel developed at Karypis Lab, University of Minnesota. The initial data set scales smoothly in reciprocal with the use of graph nodes and load balance test in beta and real cases-[81]. A unique drawback is that data partitioning is limited to graphs' properties and the number of applied constrains. Hence, all the merits of domain decomposition are addressed achieving higher efficiency and balanced workload among all the nodes taking into account data aspects such as geometry information, data dependencies and computational resources formation-[51],[81]. Although scaling is not always ideal, many scientific applications use MeTiS with very good results in terms of speed-up. Alternatively whenever the nature of data introduces implications, the library is flexible enough to satisfy specific criteria subject to special design needs with multi level functionality and parameter settings. Thus, more dynamic situations(with trade off between vertex migration and edge cut off) may occur either partitioning from scratch or re-partitioning with the former being the dominant choice-[82], [91].

1.1.8 Infrastructure

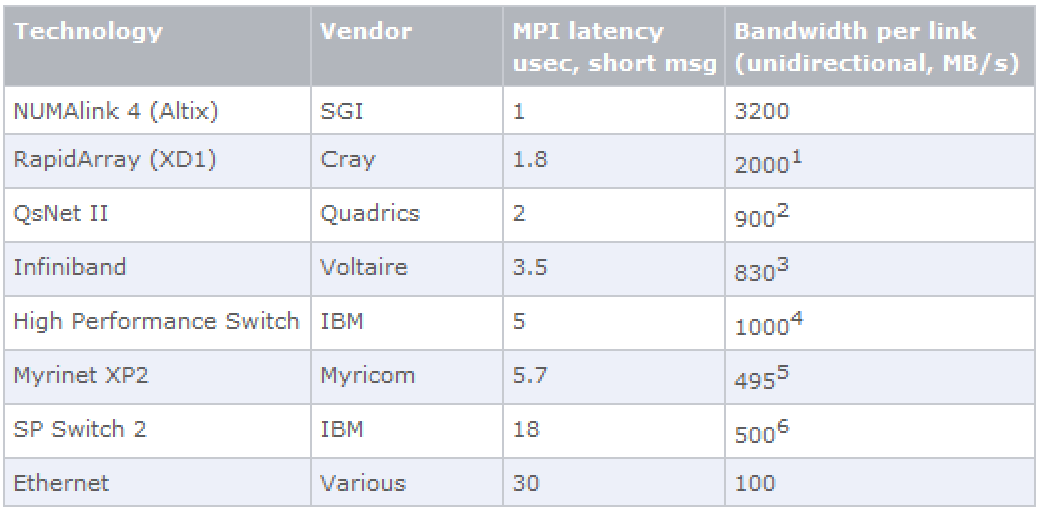
In total there are four are the main components that comprise any computational infrastructure. These are CPU, memory, I/O and interconnection. Different combinations of those orchestrate various architectures to address specific problems. Massive computations of CFD demand high end combinations of these for greater results and more applications. For compatibility reasons, many manufacturers develop CFD applications dedicated to unique configurations. By far, the biggest fraction of architectures is of cluster type. The main reasons are CFD problems are tightly coupled and quick network connection is imperative. Every bit of hardware progress is automatically integrated with new systems augmenting the level of HPC even more. At the moment CPUs and memory have already reached their ceiling. Therefore analysis follows for the rest remaining two parts, as a high level interest is concentrated around them. There is always incentive for better equipment, more accurate and quicker results over reduced cost.

Although many systems are optimised for MPI schemata another important factor is the manipulation of resources. The authors of [98] developed SCALASCA toolset, which quantifies and isolates a variety of important performance aspects and it was tested on 3 large multiprocessor system. Good knowledge and experience of an application provide insight for its use. Nonetheless, this can often be might be incomplete and unreliable. The situation becomes more complicated for new and large systems, where trivial and simple cases for small systems can cope with success. SCALASCA could provide a scalable mechanism for diverse length and complexity, and thus it is appropriate for primary study, analyses and profiling. On their comparative plots CFD application present good but low scalability up to 4096 cores. After that threshold the scalability is almost linear achieving the ideal curve. It was also been demonstrated that the revised version of a CFD solver produced better exponential behaviour on time-steps per hour against its ancestor. Generally speaking procedures of measuring and analysing performance on various platforms were assessed. Despite scalability limitations which could be either unique – depending on hardware configuration – or wide –because of used tools – flexibility, automation and integration for ease of use would always be an issue under investigation and research.

1.1.9 Network Interconnection

Mainly the integration of computing elements as close as possible is due to the optimization of communications speed. Running in parallel requires a very robust and quick interconnection between the nodes that is actually the bottleneck of performance in terms of infrastructure. Table 1 presents, in order of magnitude, the dominant network technologies. The most common infrastructures used in this kind of scientific applications are Gigabit Ethernet (GigE)-[1], Myrinet-[5] and Infiniband(IB)-[3], which provide normal, fast and very-fast rates of communication respectively. Obviously the higher the speed, the more expensive it is to use. Some systems incorporate only one of the above, whereas others apply hybrid or proprietary implementations. For the latter case, the choice of the specific level of speed in the communication is critical and it should be resolved carefully. In scalability up to 16 nodes the right choice is vague. Usually for HPC applications IB is the highly indicated joint both for reduced latency and increased throughput terms. Most of the greatest supercomputers are based on GigE and IB-[4]. As illustrated, its big advantages are a significantly low cost and quicker execution for the same workload – compared to its competitors – thus achieving a higher order of magnitude performance or even super-linearity. At the same time it pushes CPUs to their limits achieving in overall more FLOPs by increasing the number of sent jobs-[44], [47]. In general, the more cores employed, the more important the interconnection is.

Table 1 Comparison of Interconnections-[84]



Technology	Vendor	MPI latency usec, short msg	Bandwidth per link (unidirectional, MB/s)
NUMAlink 4 (Altix)	SGI	1	3200
RapidArray (XD1)	Cray	1.8	2000[1]
QsNet II	Quadrics	2	900[2]
Infiniband	Voltaire	3.5	830[3]
High Performance Switch	IBM	5	1000[4]
Myrinet XP2	Myricom	5.7	495[5]
SP Switch 2	IBM	18	500[6]
Ethernet	Various	30	100

Elder infrastructures are based on GigE and in most cases it is cost-prohibitive to upgrade to Myrinet or even to IB. It is also a cheap solution for young organizations that would prefer not to incur high costs in IT department. In these cases dual core processors outperform over quad up to 16 cores. In addition, up to 32 cores the quad cores certainly outperform. Of course both cases are bounded by Amdahl's law until a peak point. After that performance drops. Complementary dual processors' cost in terms cost/core is higher than quad, while they also demand more volume compared to their counterpart. In conducted benchmarks small to average workload quad cores present better performance either partially- or fully- populated, whereas for average to large workload partially-populated perform better than the fully-populated ones. Generally dual cores scale smoothly and ultimately for huge data sets they beat their competitors-[32].

Recent comparison tests have shown that concerning pure communication speed Myrinet outperforms over GigE in diverse message exchange with various packet size and number of nodes. Using benchmark tests based and running on MPI in Linux clusters. the contention effect is the most crucial factor of communication for either point-to-point or collective instances among nodes. Analytically, on the one hand, in GigE the most occurrences lay between 5 and 10ms. This range increases as the package size augments with a few peaks near the main lobe. On the other hand Myrinet presents an average of 10ms for the largest package. Generally speaking in terms of scalability of Myrinet in time plots, the gradient of scalability is very slow and linear. This means that even more nodes could be incorporated to reach MPI_Barriers and the rest of MPI calls without causing any congestion, whereas GigE's plots augment exponentially as the resources pool gets larger. The architecture of Myrinet performs better for various sizes of exchanged packages sent through interconnection. Also, a disadvantage of GigE is the need for configuration of TCPRTO (protocol setting). Thus, as expected Myrinet is better especially when network saturation is reached-[37].

The above comparisons included commercial codes, while on open-source instances interesting results emerge. On OpenFOAM linear scaling is achieved by up to two cores on a total 16 cores cluster. The time difference between GigE and IB is very small, it is only 6% for IB for 16 CPUs. The most impressive result was the wall clock time for 8 cores on four nodes with IB, which outperforms the two nodes because of the contribution of cache in CPUs' computations and IB's stable communication speed. It is worth mentioning that no I/O was included, so spreading the workload to as many CPUs as possible delivers better results-[70].

1.1.10 Parallel I/O

Benchmarks usually measure purely computational performance and ignore the execution of the algorithm as a whole in terms of data manipulation, file operations and on-the-fly resource handling, etc. The largest part of these I/O (non-computational) operations is dominated by file operations which tend to be a bottleneck and it is critical that this is improved by increasing provided data rates-[88]. This implies setting parallel I/O as a standard for every high performance application. COOLFluid is a memory parallelization framework either for serial or parallel implementations and hides architectural details from end user while achieving significant scalability in data throughput. PVFS2 file system can provide very good efficiency as equivalent commercial solutions (Lustre) and MPI-IO are suggested for contiguous and not- access method respectively providing parallel cache functionality. This results in quicker file operations and improved overall system performance in I/O terms-[53]. Many scientific applications take advantage of MPI-I/O combined with HDF5[41] which provide important and stable results in terms of scalability in parallel file operations supporting the purely computational part of the application and improving the overall timings-[39], [74].

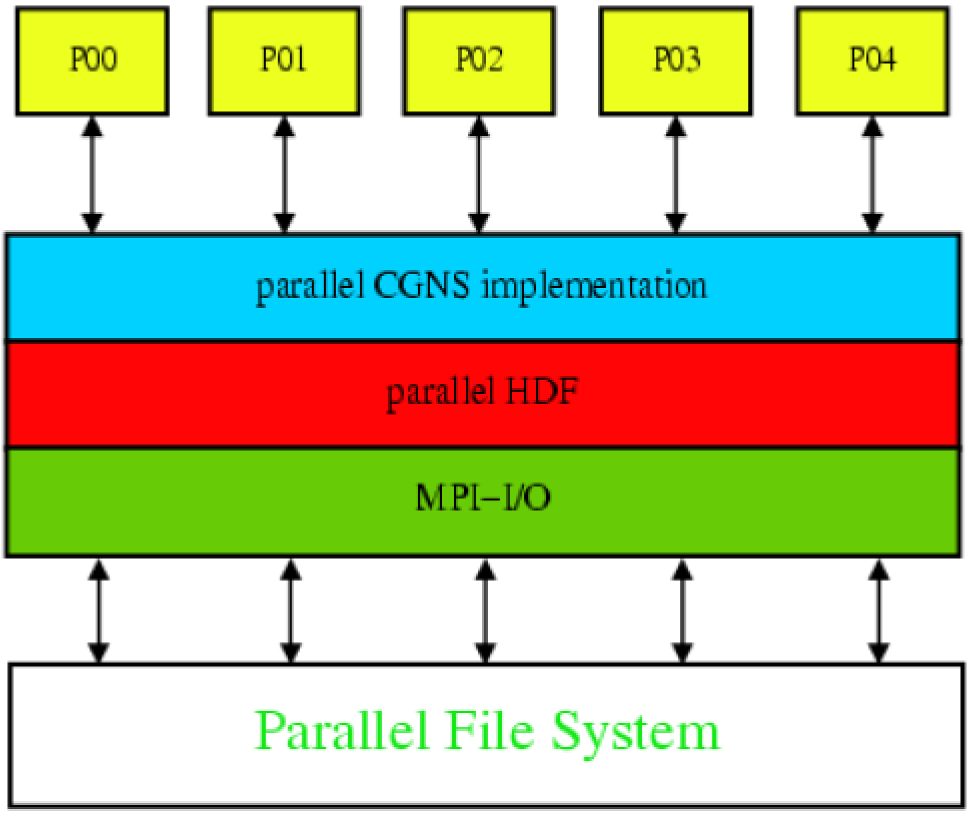


Figure 4 Data access using parallel CGNS API based on HDF and MPI-I/O [39]

In practice, large numbers of employed CPUs also make frequent use of I/O operations. Using parallel I/O libraries and indexing data on storage phase significantly improves the performance. Unfortunately that is true up to a fixed amount of processes, while beyond that amount, performance drops dramatically. Additionally, the form of the overset grid contributed positively by providing ideal load balance, with an impact on the communication. If the grid was composed of a few component grids, the performance would be higher. Despite the improved version of ADPDIS3D solver, unique features of the structure affect efficiency for more than 1024 processors. Judicious programming resolving data dependencies, hardware specifications and network conditions makes good use of I/O without under-utilising computational resources-[87].

1.1.11 Platforms and Solvers

As discussed in most of the above cases LINUX-based operating systems (OS) dominate the test cases and implementations-[8]. LINUX flavours provide a good test-bed with many features and functionalities specified for parallel applications and infrastructure. If communication channels among platforms are quick enough, this will provide more efficient parallelisation for a greater number of nodes. The former stands because the slowest node limits the upper performance level. Assistant mechanisms such as checkpoints, redundant disks etc can improve a system's overall performance in terms of safety, maintenance, availability, reliability, confidentiality and integrity. Competitors based on Windows OS are still in their early stages but provide potential for perfection [40]. Undoubtedly, most of the compilations run on a LINUX based operating system because it not only supports all the high performance features discussed before and frequently upgrades, but it is also cost-saving basis for multiple development purposes as well as expanding the existing ones.

In terms of updating on-the-fly LINUX kernels for parallel machines forming a cluster the authors of [63] developed DynAMOS. It is an operating system that exploits features of parallel architectures and kernel programming without interrupting its operations and continuously provides services with very small overheads. It also adapts its behaviour dynamically according to a system's needs, thus attaining the highest possible peak performance. Therefore, uninterrupted simulation run continuously without ever crashing.

1.1.11.1 STARCD

Despite the high rate of data exchange, the impact of high communication overhead – due to cluster's size – affects turnaround time. It was noticed that messages do not exceed 4KB size. Aggregated MPI functions are more effective in application's behaviour, whereas point-to-point communication can significantly stall the performance as the size of cluster rises. Profiling benchmarks evaluate this statement in performance plots. The largest number of sent messages is between 1K and 8K, while the kernel of the software uses aggregated functions. Moreover slightly better performance has been achieved with a custom version of MPI (namely HP MPI) for workstations-[42]. Performance is unaffected by power management tools, while the power consumption and cost savings are slightly improved-[45].

1.1.11.2 CFX

CFX presents the same behaviour with the main difference being that it handles smaller messages. So the main volume of exchanged messages is up to 64B mostly for synchronization, whereas for higher ranges the number of messages is negligible. The communication included mainly block communication functions-[46],[30]

1.1.11.3 FLUENT

Nowadays FLUENT comes in two versions 6.3 and 12, with the latter endeavouring to be the mainstream choice. The former version is comparatively better in terms of efficiency than the above solvers but the 12th is by far the best. The 6th version of FLUENT scales gradually using blocking communications. Besides the synchronization messages, most of them stand within the range of 256B and 1KB-[27]. Moreover, the 12th version of FLUENT introduces a few significant changes compared to its predecessor, as it was optimized for this reason. Although most of the messages (except the synchronization ones) yield within the same range, the number of messages is reduced. The reason is the increased non-blocking and aggregated functions. Thus, impressive performance and scalability are achieved reducing the communication and operating costs, as well. In both versions as the cluster's size increases, so does the number of exchanged messages among the nodes, but it also allows more jobs to be submitted, and consequently increasing productivity-[43].

1.1.11.4 OpenFOAM

It is an alternative open-source CFD solver that, despite its unique nature, it is highly competitive with major CFD codes. Only a few benchmarks are available as most of the users avoid using it, mostly because its interface is challenging to handle. In [71] it was proven equivalent to another well known solver (CFX) presenting near identical results in plots of energy and pressure along with experimental data. Similar behaviour was observed in [16], where OpenFOAM was compared against FLUENT for use in combustion models. Extending the 2D case of lid driven cavity flow, OpenFOAM run on HECTOR (supercomputer located in UK) delivered linear behaviour for the various grid sizes tested-[77].

1.1.11.5 Common Features

Many enterprises adopt the idea of distributed systems and require equivalent products. Distributed ANSYS (DANSYS) solvers utilize the latter feature benefiting from scaling and multi-core processors capabilities. This functionality is addressed automatically in FLUENT and CFX commercial solvers, but in the case of StarCD, OpenFOAM and custom codes the user should use specific commands and arguments. Especially ANSYS has not only taken into account all the above software and hardware individual aspects, but also collaborates with Operating Systems and Computer manufactures in order to produce packages that operate ideally regardless of the diversity of resources. It is clear that commercial solutions are preferred because they provide more capabilities in an easy-to-handle way.

1.1.12 Solver's Verdict

So far no single solver is ideal for every possible case. One may speculate that the reason is that every code implements and corresponds to respective needs subject to the nature of the problem and the user's demands and interest. Also the supporting mathematical models, compatibility libraries for different architectures and fluid physics allow infinite combinations which cannot meet all the standards that correspond to every scenario accurately. It seems that for general and abstract cases commercial packages can cope well whereas for high interest purposes it is better to use pure code solutions. The basic criterion when choosing from all the available options is the suitability of the package along with required physics. Sometimes it is hard to distinguish which category any project belongs to, and only experience can help and indicate the right way.

In terms of purchase cost two packages are dominant. FLUENT and OpenFOAM are the commercial and open-source packages respectively with the largest users community and support. FLUENT is the most commonly used package in industries as a complete suite for CFD, while the latter despite the zero-cost of purchase, is expensive in terms of training by the provider enterprise. These two instances present high interest in CFD field (in use and supporting features) and will be discussed below.

1.2 Aims and Objectives

Simulations conducted for this thesis are to explore performance trends related to CFD solvers running on high end clusters under various configurations. In order to achieve the former goal various combinations of software and hardware resources will be employed. Specifically one representative from each category of commercial- and open-source packages will be tested. Those are FLUENT 12, and OpenFOAM 1.6 respectively. As for the hardware part two clusters located in Cranfield University will run all the simulations. Technical specifications are summarised in Table 2 and Table 3. This study will provide new insight into solver performance and solution applicability, and will also benefit researchers by providing with knowledge that will serve their needs more efficiently. Further objective is to relate physical metrics of the geometry size (in our case grid size per edge) to the execution time under specific circumstances. Another similar study had been conducted in [15] on a Grid environment.

Breaking down the initial goal, the pipeline consists of the following steps:

Configure the test bed properly for comparison. This is to prepare accordingly the case as input for the software resources and run it on computational resources. Creating and meshing the geometry, choosing the mathematical model and setting parameters of the model and the solver are of paramount importance.

Derive simulation results in agreement with experimental data. That is to align simulation curves as close as possible with experimental ones. The above settings will indicate to the user which settings should be used in order to achieve coherency between the two phases of design. Therefore simulation is considered reliable and deeper analysis will advance.

Compare execution timings for equivalent computational power. After the end of each simulation and since it is assessed as reliable the results are gathered and form curves of performance in plots.

Appraise the reasons for diverse behaviour and provide adequate discussion and final conclusions. This is subject to computations and configurations referred to above. Interpreting and understanding the differences, if any, is the ultimate goal.

From a reverse point of view, in order to limit the amount of time, more work needs to take place in parallel. By increasing the computational power and splitting tasks accordingly, the execution time decreases. Therefore time results are derived quicker. This has a dual effect because the results are produced quicker but there is also the possibility for further improvement and revision of the solution. In addition, more tasks could take place within the same time limit. Academia is benefited for scientific purposes being able to produce even better tools, while industry's is the raise of throughput. Academia gains through the development of even better tools, while industry gains through the increase of throughput.

1.3 Thesis Outline

The structure of the thesis follows in summary. It begins with the followed methodology which describes the lid driven cavity case used for our study along with the resources employed for the simulations. The respective mathematical modelling and used solvers are presented, too. The simulations settings chapter illustrates all the steps needed for measuring, ultimately, performance and scalability. These are all the standard procedures of CFD pipeline from setting the case until plotting the flow for FLUENT and OpenFOAM. In turn, the results of conducted CFD simulations are presented and discussed. Timings for each case are presented in a comparative way, as well, in relation to the used infrastructure, while commenting any ambiguity or vagueness. Lastly, the simulation results are revised and future work is suggested.

2 Methodology

2.1 Overview

Measuring performance and scalability means to run the same scenario that produce identical results in diverse platforms under variant configurations for the same workload. Respective time until completion is recorded and complementary measurements are inspected and discussed. In order to achieve maximum coherency similar settings are chosen wherever it is possible. Since different software packages are employed, their structure will be different as well. Because not all the settings could be mapped, their impacts are due to be investigated and discussed and will provide interesting annotations.

Our study uses a real world application of high engineering interest and applicability, instead of benchmarks used in other studies. The test bed includes the 3D lid-driven cavity (LDC) case. The reason for choosing LDC is the simplicity of building and setting the scenario in every software package and, historically, most comparative simulations have tested this object in every new study-[24], [35]. LDC imports negligible amount of complexity due to its geometry and used grid, while obeying fluid dynamics principles. Flows in rectangular spaces are also important because they can be studied systematically by numerical and experimental methods. Because the flow is entirely enclosed within the cavity, the controlled problem is affected by fewer external factors relating to the state of the flow. So, purely computational deductions can be extracted. The most well known candidates were chosen from each of the categories in CFD field. So the commercial code of FLUENT and the open-source OpenFOAM will be tested. As for the resources, Galileo cluster from Cranfield's Aeronautics Dept and ASTRAL[12] super computer will run every simulation exploiting their unique features. Extensive details for each clusters are presented in Table 2 and Table 3.

All the simulations were conducted in local clusters with message passing libraries. As reported above MPI is ideal for this type of problems as long as interconnection and network speed are quick enough and flawless and its importance in communication and integration with CFD solvers is substantial as it is the basis of parallel applications. Galileo makes use of OpenMPI, while HP-MPI was provided by the manufacturer (HP) for ASTRAL, because of its unique structure. All the transactions are as short as possible in order to be more reliable and fast whenever they occur. A unique side-effect is that the number of processors will not be that high compared to a Grid infrastructure, largely because the cost of purchase, support and maintenance is extremely high. At the same time the tightly coupled nature of the application points to the use of clusters because Grid computing does not cope particularly well with this kind of situation, as its scope is primarily for coarse-granularity cases.

In every CFD application there is a sequence of steps to be followed. These are pre-processing, simulations and post-processing as depicted in figure below.

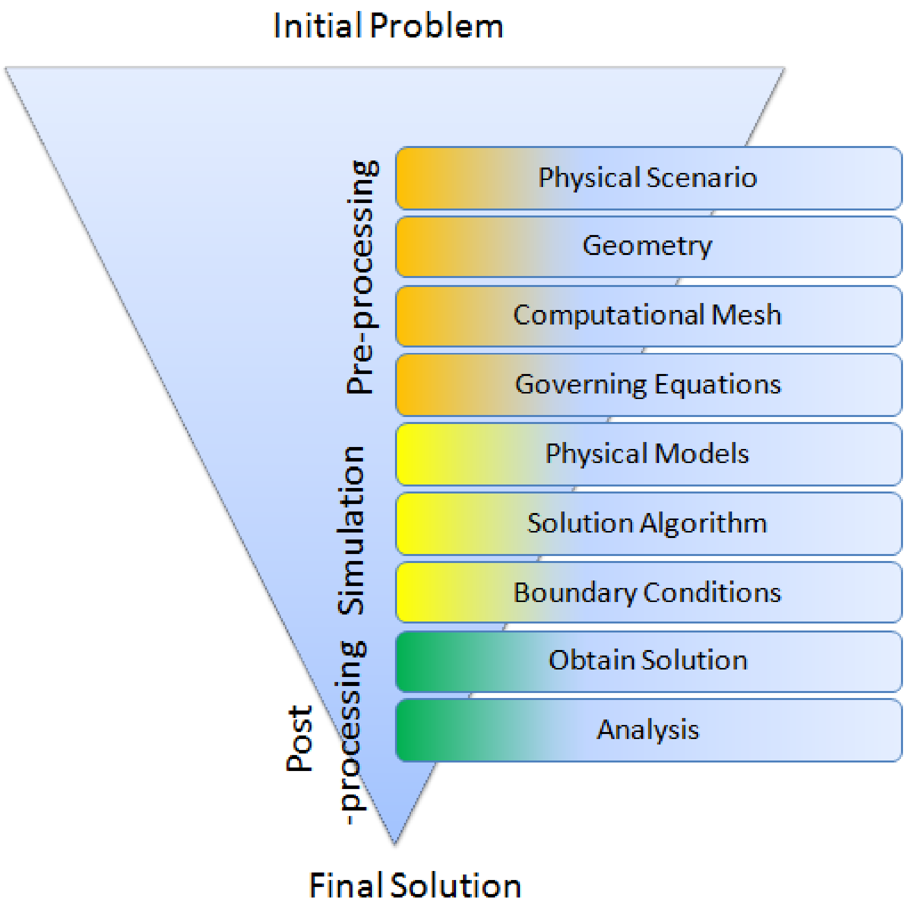


Figure 5 Modelling Process Pipeline

Whatever the software used, the steps described in Figure 1 are identical and in the same specific order. In order to present a uniform study, whenever possible, the same pre- and post-processing tools were used. Thus, actual differences are limited to the core of CFD-packages, that is, the simulation code, which is the prime interest. The choice of RANS models is due to the smaller execution time in comparison with LES, as the computational resources cannot be reserved for very long intervals. The simulation results reported below in this study were run on Astral and Galileo. Both clusters are located in Cranfield University and up to 16 cores were used for each case.

Table 2 Computational Resources Specifications – Master Nodes

	Galileo	Astral				
Vendor:	***OCF***	HP				
Master Node		**2 front-end nodes**	**3 admin nodes**	**2 export nodes**	**4 File Server nodes**	**Interconnect**
Motherboard/basic node	IBM eServer 326m	HP DL585 G4	HP DL380G5	HP DL140G3	HP DL380G5	**Voltaire 2012 Infiniband switch**
Processor	2xAMD Opteron 275 dual-core (2.2GHz per core)	4 X Intel Xeon 7130 dual-core (3.2GHz)	2 X Intel Xeon 5160 dual-core 3.0 GHz	2 X Intel Xeon 5160 dual-core 3.0 GHz	2 X Intel Xeon 5160 dual-core 3.0 GHz	
RAM	4x1Gb	64GB	8GB	8GB	8GB	
RAID	IBM Ultras320 SCSI Card (note: SCSI card interfaces with external SATA RAID array)	HP P400 SCSI/SATA	HP P400 SCSI/SATA	NONE	HP P400 SCSI/SATA + HP P6400 SAS/SATA	
Storage	80GB 7.2k Seagate SATA	3 X Seagate SAS 72GB SAS	8 X Seagate SAS 72GB SAS	80GB 7.2k Seagate SATA	2 X Seagate SAS 72GB SAS	
LAN ports	3	4	2	2	2	288
Management	IPMI over separate port	HP ILO2 Port	HP ILO2 Port	HP ILO100i Port	HP ILO2 Port	

Table 3 Computational Resources Specifications – Slave Nodes

	Galileo		Astral				
Slaves Number		***Upgrade***					
	15	7	214				
Slave Nodes							
Processor	2xAMD Opteron 275 dual-core (2.2GHz per core)	2x AMD Opteron 2214	2 x Intel(R) Xeon(R) 5160 (3.00GHz per core)				
RAM	4x1Gb	4x1Gb	8x1Gb (34 nodes with 8 X 2GB)				
Networking							
	1xHP2848 switch		7 X HP 2650 switches; 7 X HP 2848 switches; 7 X HP 3700 switches; 1 X HP 3406 switch; 1 X HP 2626 switch			10 X HP SFS20 Array enclosures each with 12 X Seagate 250 GB 7.2K SATA disks	
Parallel I/O	NFS v3		HP Scaleable File Share (LUSTRE) Filesystem				
Networking	Gigabit Ethernet		Infiniband				
Performance (max/peak)	240/290 Gflops		7.25 / 10.27 Tflops				
Operating System	CENTOS 4.3		RedHat EL 4 update 3 built by HP				

Final results and timings along with each underlying coding issue will be discussed and compared. Because irregularities very often intervene – especially in distributed environments – measuring timings is a trivial and repetitive procedure. The results are gathered in a matrix and the average is calculated. This is because heavy I/O periods/seasons, diverse network traffic, component failures, system calls and so forth occur even while the application is running. In the final analysis hardware components are the original Achilles' heel of parallel performance because every crash collapses the harmony and balance of the system.

Comparative Analysis over famous commercial solutions is a very useful report that studies advantages and disadvantages as well as suggesting improvements and new research directions. Although it is difficult to gather all this information and run the same test case on multiple occasions because of the waste of time, power and resources, this is in fact the only way for data acquisition. Analysis at this level has never been conducted before. Despite the simple geometry, physical phenomena are always complex. As in every simulation initial data such as velocity, pressure, Reynolds number, mathematical model, boundary conditions should be set from scratch. Moreover because the study focuses on performance, parallelization and execution parameters should be set, too. This is of vital importance since the combination of multiple fields of engineering and industrial problems have an impact on real life environment. Finally as pointed out in [33] the execution time might seriously affect later decisions. So, to produce the optimal results, time proves to be the ultimate criterion.

2.2 LDC Case Description

The test object is a 3D model LDC cube structure as depicted in Figure 6. Its dimensions are 1m x 1m x 1m and only the upper face - the lid - is moving horizontally, while being firmly attached to the side of the other vertical faces. The geometry is analysed in coarse and fine resolution non-uniform structured mesh grids as illustrated below. The grid is denser towards the walls because these areas are of higher interest as important physical phenomena take place. As referred to in [59]smaller secondary eddies are formed which affect the primary flow in the centre of the cavity. The Reynolds number will take the values 3200 and 10000, laminar and turbulent flow respectively. As observed in [59], internal flow presents turbulent features between Re=6000 and Re=8000. The moving lid moves towards the X-axis direction. The speed is fixed in order to create laminar and turbulent flow in the interior of the cavity. The rest of the walls are stationary and the whole volume is initialized with zero values. The interior of the cavity is filled with incompressible liquid (water). Thus, in the purpose of having two types of flow, consider characteristic length=edge length, and water kinematic viscosity. So the respective speed values are obtained as:

$$u = \frac{\mu \cdot \mathrm{Re}}{\rho \cdot L} = \begin{bmatrix} 0.003215 \\ 0.010048 \end{bmatrix} \text{m/s} \qquad \textbf{(2-1)}$$

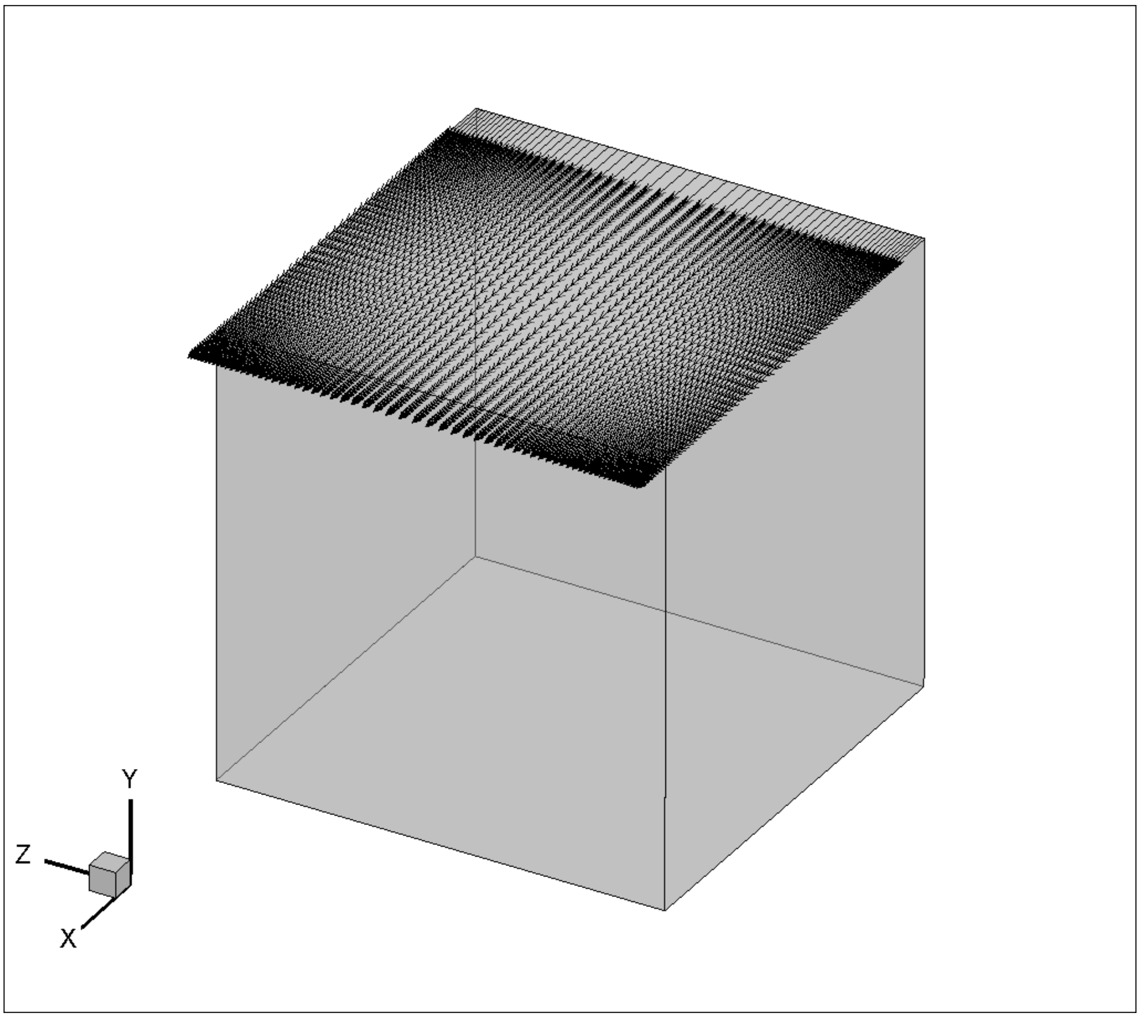


Figure 6 cavity container

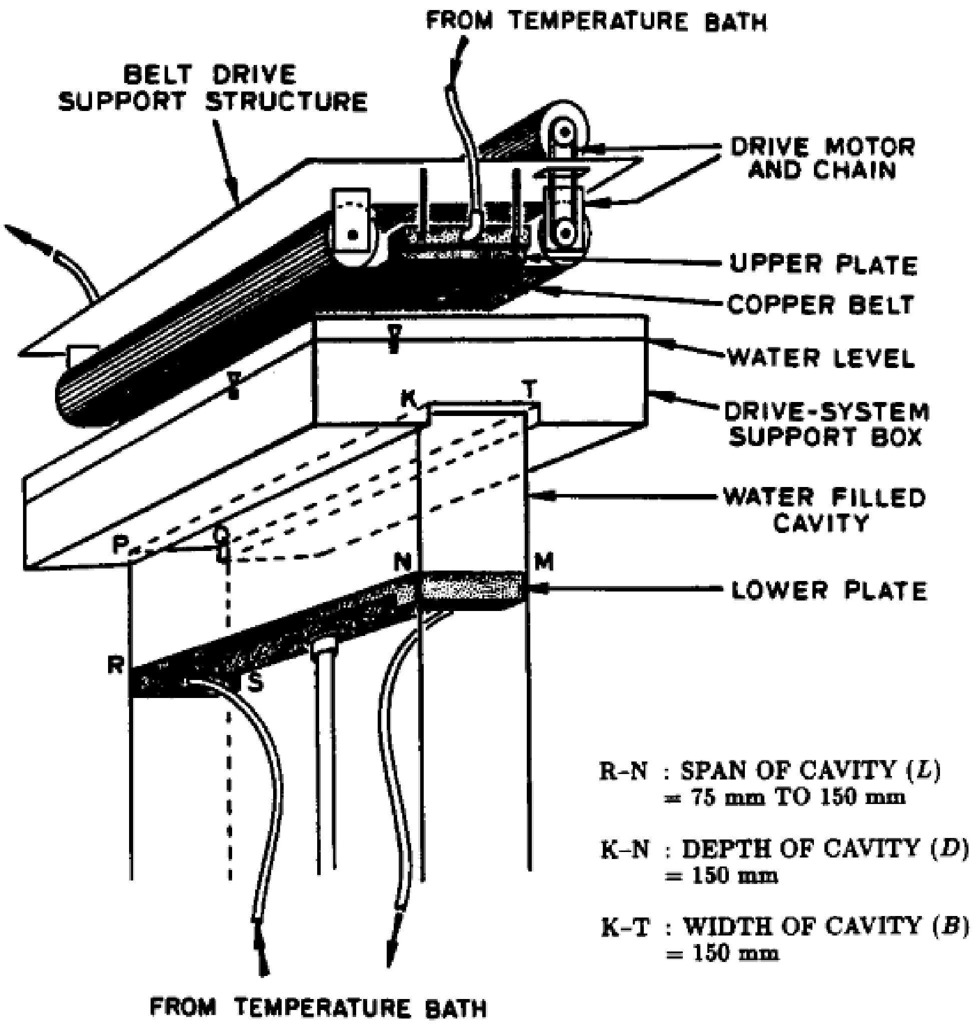


Figure 7 Lid-driven cavity experimental rig [57]

The experimental device in Figure 7 was used to retrieve experimental data. These will be compared to the simulation results for evaluation purposes.

Regarding the reliability of simulations for further comparisons, employing robust resources is crucial. In order to secure a significant speed-up without any drawbacks the platforms will be the local clusters that referred to above. Thus transfer time would be as negligible as possible. At the same time, each arrangement of resources has GigE and IB as the interconnection among the nodes and the impact of this will also be commented on. Various numbers of processing nodes will, of course, be used on every infrastructure for scalability issues. The number of processors employed will be to the power of 2 for balanced decomposition of the problem. Also maximum performance without underutilising the available resources will be reached.

2.3 Mathematical Modelling

The model presented below is one of the most prominent developments for general purpose software packages (commercial or not). Researchers aim to develop and optimize for specific classes of flow and to create a completely general purpose turbulence model. In their general form NS equations can describe every flow. In our case the water flow does not change over time and retains the initial temperature. Therefore, it is characterised as incompressible, steady-state and isothermal, which produces a simpler form of NS equations. Thus, the mathematical model is less complex and computations are performed more quickly.

2.3.1 Reynolds-Average Navier-Stokes

In our study turbulent flow is approached with RANS model. That is done so as to obtain either time- or ensemble-averaging of the NS equations. Normally, the solutions of NS equations are subject to random time and space. By averaging the time, mean quantities present a smooth spatial and time function. If the average time is greater than all the time scales of the turbulent flow, then RANS equations can be derived. Applying RANS modelling an extra term appears, called Reynolds stress, which is the effect of turbulent fluctuations on the flow. This stress quantity is to be approximated via turbulent model. RANS is ideal for LDC since only a few quantitative properties of turbulent flow need to be calculated.

The main concept in RANS is variable decomposition of scalar quantities into a sum of mean and a fluctuation term, such as:

$$\phi = \bar{\phi} + \phi' \quad \textbf{(2-2)}$$

,where ϕ denotes any given scalar, for instance velocity and pressure, and the average quantity is given by:

$$\bar{\phi}(x_i) = \lim_{T \to \infty} \frac{1}{T} \int_0^T \phi(x_i, t) dt \quad \textbf{(2-3)}$$

,where t represents time and T the averaging interval respectively. As stated above T should be large enough compared to the typical time scale of the fluctuations. The first equation models velocity in terms of mean and fluctuating components respectively. The same strategy is applied also to the second equation, where p stands for for pressure. Therefore:

$$\vec{u} = \vec{\overline{u}} + \vec{u}' \qquad \textbf{(2-4)}$$

$$p = \overline{p} + p' \qquad \textbf{(2-5)}$$

Physical properties of any case affect the use of RANS. The large eddies are noticeably affected by boundary conditions. Diverse turbulent features occur due to various combinations of geometric structures and boundary conditions. Thus, there is not a universal model for all the problems. Also, it is not possible to capture unsteady phenomena accurately.

Therefore RANS equations are:

Mass conservation

$$\frac{\partial \vec{u}}{\partial x_i} = 0 \qquad \textbf{(2-6)}$$

Momentum

$$\frac{\partial}{\partial x_j}\left(u_i u_j\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_i}{\partial x_i}\right)\right] + \frac{\partial}{\partial x_i}\left(-\rho\overline{u_i' u_j'}\right) \qquad \textbf{(2-7)}$$

,where

i and j are the indices for x,y,z axes,

u, p, μ denote velocity, pressure and dynamic viscosity respectively,

δ_{ij} is the Kronecker delta, defined as 1 if i=j, and 0 elsewhere,

ρ represents density,

the last term, $(-\rho u_i' u_j')$, is called Reynolds Stress.

2.3.1.1 Turbulence Modelling

Turbulence model closes the system of mean flow equations. It is worth mentioning that not all the flow details need to be resolved, since the effect of turbulence on mean flow is our only concern. Basically, turbulent model is divided into Reynolds stress (or second momentum closure model) and eddy viscosity models. The latter splits to linear eddy viscosity and non-linear viscosity models. Zero equation or algebraic models, one equation model and two equations models consist of the linear eddy viscosity model. Among the previous stated models, the k-ω will be used. Its advantage is to have a production term for k- and ω-equations, and it can predict circulation regions in turbulent flow. Moreover, the k-ω SST accounts shear stress calculations near the wall.

2.3.1.1.1 Wilcox k-ω model

Standard k-ω is an empirical model based on turbulence kinetic energy k and the specific dissipation rate ω. It is applicable to both wall-bounded flows and free shear flows, increasing – thereby – the range of applications with witch it can be used. The main concept is that the rate of change of k/ω plus the transport of k/ω by convection are equal to transport of k/ω by turbulent diffusion plus the rate of production of k/ω minus the rate of dissipation of k/ω.

The disadvantage of k-ω is that problems occur when k and ω tend to zero (since eddy viscosity, $\mu_t=\rho k/\omega$, where ρ is density). This happens because of ω in free stream cases and could lead to numerical errors (or even registers overflow) since μ_t is indeterminate or infinite. That is due to the regular use of stream boundary conditions. This situation occurs exclusively in external aerodynamics and aerospace applications, where LDC does not belong to these categories-[97].

Turbulence kinetic energy k is given by

$$\frac{\partial}{\partial x_i}(ku_i)=\frac{\partial}{\partial x_j}\left(\Gamma_k\frac{\partial k}{\partial x_j}\right)+G_k-Y_k+S_k \quad \text{(2-8)}$$

and the specific dissipation rate ω is given by:

$$\frac{\partial}{\partial x_i}(\omega u_i)=\frac{\partial}{\partial x_j}\left(\Gamma_\omega\frac{\partial \omega}{\partial x_j}\right)+G_\omega-Y_\omega+S_\omega \quad \text{(2-9)}$$

Where for each subscripted term k and ω:

G is the generation of k due to mean velocity gradients and the generation of ω, in that order,

Γ represents effective diffusivity,

Y denote dissipation due to turbulence,

S symbolizes of user-defined source terms

2.3.1.1.2 k-ω SST model

A variation of Wilcox's k-ω model, the Shear Stress Transform (SST) k-ω model, was proposed by Menter. This version is a hybrid composed of a transformation of k-ε to k-ω close to the vicinity of the wall and an implementation of original k-ε in large distance of the wall, within fully turbulent regions. Later modifications by the same author were the revised model constants, blending functions and limiters. The latter two led to improved numerical stability – k-ω's better near wall behaviour along with k-ε's robustness in the far field are combined – and limitation of eddy viscosity which derived higher performance for specific flow properties (wake regions and pressure gradients). In external aerodynamics and General-purpose CFD behaves better than its competitors, being more preferable and general-[97].

Regarding our study, k-ω SST was chosen for the reasons stated below:

Because of the combination of k-ε and original k-ω previously referred to, the prediction of separations is more accurate while the dissipation remains in low level.

It makes use of modified turbulent viscosity formulation to explain the transport effects of the principal turbulent shear stress-[18]

Although, Menter's model uses k-ω's Reynolds stress computation and k-equation, it transforms k-ε's ε-equation according to the relation ε=ωk. The latter is expanded by the cross-diffusion term derived from the transformation.

2.4 Solvers

In order to solve the NS equations, two new problems arise. The first is the non-linearity of some terms – such as the derivative of pu^2. All the equations are intricately coupled due to the velocity component in the equations of momentum and continuity. Additionally, pressure appears only in momentum equations. None of the quantities referred to before are known a priori. Writing the transportation equations for incompressible, isothermal and steady-state flow, the velocity is only a function of pressure. By definition, the density component is constant and, thus, it is not linked to the pressure. Applying appropriate pressure field in the momentum equations produces satisfactory results on continuity in velocity field. It would seem that an elegant method needs to be employed in order to solve the system of equations.

The above type of problems is addressed through pressure-velocity coupling solvers, about which a brief description follows Using collocated grids might lead to odd-even decoupling (discretisation error) and, consequently, to checker-board patterns in the solutions. Therefore utilizing a staggered grid between velocity and pressure tackles the previous problem. Staggered grid on velocity field achieves realistic behaviour of discretised momentum equations avoiding spatially oscillating pressures. As discussed previously, the solvers implemented FV methods. Assuming that input grid is appropriate the two solvers will be employed. Each is used for one type of flow only. Pressure type solvers were chosen because they could operate in a coupled way, which corresponds ideally with the concept of using clustered servers. The solvers are SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) and PISO (Pressure Implicit with Splitting of Operators) for FLUENT code. The combination of icoFoam and pisoFoam are respectively the equivalent versions provided by OpenFOAM. The description of each solver is identical on both software packages.

For laminar flow, SIMPLE calculates velocity and pressure fields. It is a guess-and-correct procedure that couples iteratively the NS equations (set boundary conditions, compute velocity, mass fluxes, solve pressure equations, apply corrections, update boundary conditions etc). The PISO algorithm is another approach in solving NS equations in turbulent flow problems based on SIMPLE but it applies more corrections. Hence, it requires more storage and time. Although PISO-loops are longer in execution, the algorithm as a whole presents increased performance per iteration. PISO is quick, because further corrections on each loop reduce the number of unnecessary iterations, in comparison with SIMPLE. It is important to notice that both solvers use an under-relaxation method to stabilise their operation. The algorithms of SIMPLE and PISO are depicted in Figure 8. While the data flow is linear, the whole domain is its input. So, by applying data decomposition, satisfactory speed-up due to parallelism is achieved. The previous set of solvers will be used on FLUENT. -[18], [73]. Despite their features, the choice of which solver is more appropriate as a general purpose procedure is unclear.

In the discretisation phase of each solver different schemes are applied for each term presented in NS equations. Each user makes individual choices when selecting the proper scheme depending on major features of the geometry and the flow. This results in speed and effectiveness of the solver. However, as expected, there is a trade-off between speed and accuracy. The used discretisation schemes of our simulations are presented in 6Appendix B and 6Appendix C had been chosen in order to achieve maximum performance for the given structure.

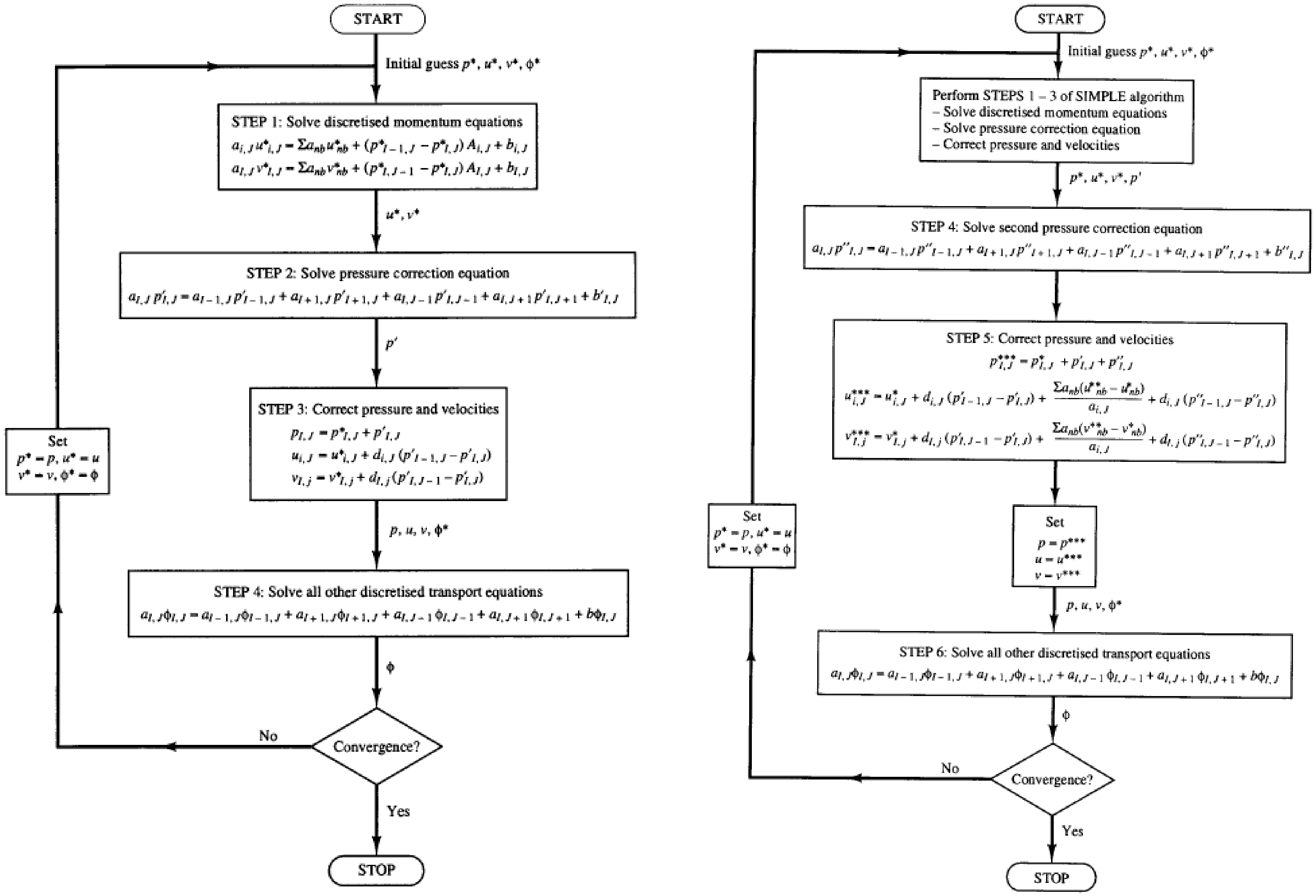

START
Initial guess p^*, u^*, v^*, ϕ^*
STEP 1: Solve discretised momentum equations
$a_{i,J}u^*_{i,J} = \Sigma a_{nb}u^*_{nb} + (p^*_{I-1,J} - p^*_{I,J})A_{i,J} + b_{i,J}$
$a_{I,j}v^*_{I,j} = \Sigma a_{nb}v^*_{nb} + (p^*_{I,J-1} - p^*_{I,J})A_{I,j} + b_{I,j}$
u^*, v^*
STEP 2: Solve pressure correction equation
$a_{I,J}p'_{I,J} = a_{I-1,J}p'_{I-1,J} + a_{I+1,J}p'_{I+1,J} + a_{I,J-1}p'_{I,J-1} + a_{I,J+1}p'_{I,J+1} + b'_{I,J}$
p'
STEP 3: Correct pressure and velocities
$p_{I,J} = p^*_{I,J} + p'_{I,J}$
$u_{i,J} = u^*_{i,J} + d_{i,J}(p'_{I-1,J} - p'_{I,J})$
$v_{I,j} = v^*_{I,j} + d_{I,j}(p'_{I,J-1} - p'_{I,J})$
p, u, v, ϕ^*
STEP 4: Solve all other discretised transport equations
$a_{I,J}\phi_{I,J} = a_{I-1,J}\phi_{I-1,J} + a_{I+1,J}\phi_{I+1,J} + a_{I,J-1}\phi_{I,J-1} + a_{I,J+1}\phi_{I,J+1} + b_{\phi I,J}$
ϕ
Convergence?
No
Yes
STOP
Set
$p^* = p, u^* = u$
$v^* = v, \phi^* = \phi$
START
Initial guess p^*, u^*, v^*, ϕ^*
Perform STEPS 1 – 3 of SIMPLE algorithm
– Solve discretised momentum equations
– Solve pressure correction equation
– Correct pressure and velocities
p^*, u^*, v^*, p'
STEP 4: Solve second pressure correction equation
$a_{I,J}p''_{I,J} = a_{I-1,J}p''_{I-1,J} + a_{I+1,J}p''_{I+1,J} + a_{I,J-1}p''_{I,J-1} + a_{I,J+1}p''_{I,J+1} + b''_{I,J}$
STEP 5: Correct pressure and velocities
$p^{***}_{I,J} = p^*_{I,J} + p'_{I,J} + p''_{I,J}$
$u^{***}_{i,J} = u^*_{i,J} + d_{i,J}(p'_{I-1,J} - p'_{I,J}) + \frac{\Sigma a_{nb}(u^{**}_{nb} - u^*_{nb})}{a_{i,J}} + d_{i,J}(p''_{I-1,J} - p''_{I,J})$
$v^{***}_{I,j} = v^*_{I,j} + d_{I,j}(p'_{I,J-1} - p'_{I,J}) + \frac{\Sigma a_{nb}(v^{**}_{nb} - v^*_{nb})}{a_{I,j}} + d_{I,j}(p''_{I,J-1} - p''_{I,J})$
Set
$p = p^{***}$
$u = u^{***}$
$v = v^{***}$
p, u, v, ϕ^*
STEP 6: Solve all other discretised transport equations
$a_{I,J}\phi_{I,J} = a_{I-1,J}\phi_{I-1,J} + a_{I+1,J}\phi_{I+1,J} + a_{I,J-1}\phi_{I,J-1} + a_{I,J+1}\phi_{I,J+1} + b_{\phi I,J}$
ϕ
Convergence?
No
Yes
STOP
Set
$p^* = p, u^* = u$
$v^* = v, \phi^* = \phi$

Figure 8 SIMPLE (left) and PISO (right) data flow[97]

The use of each solver depends on the properties of the flow. For steady-state calculations, SIMPLE is recommended for laminar flow, while PISO behaves better in transitional flow. Also PISO might be used in steady-state and transient cases on highly skewed meshes. As depicted in Figure 8 the data flow is very similiar.The main difference in comparison with SIMPLE is the multiple corrections of the momentum computation step, which makes PISO more time consuming and additionally increases its complexity (because of the internal iterative corrections applied on every loop while running). Therefore, it overcomes the limitation of SIMPLE as well as satisfying the momentum balance after the solution of pressure-correction equation at the cost of additional computational power.

It is also worthy studying the complexity of each algorithm. Usually, the big-O notation is used, which denotes the worst case scenario complexity of any code. This means the maximum number of loops is resolved on every iteration. In the end there is a correlation between the size of input data and the respective execution time until completion. In the case of incompressible flow, velocity is only a function of pressure, otherwise the density is also taken into account.

3 Simulation Settings

3.1 CFD Codes

Below follows a short description highlighting the differences between OpenFOAM 1.6 and FLUENT 12.

OpenFOAM is open source, more flexible as you can edit and add some code snippet(s) into the solver, but it lacks of an easy-to-comprehend grid generator and the support is paid for. It is good for research and tends to evolve like open source operating systems but at a slower pace. So far, it does not support many models, but it is constantly being enriched.

Former's disadvantages are FLUENT's pros. Therefore, FLUENT is an industrial code which is more user-friendly, has better support, and more experience (since 1998). The CAD design is common for FLUENT and CFX as ANSYS integrated the feature on their new versions. A recent version of ANSYS Workbench provides the feature of amending structure specification without setting and solving the problem from scratch, but just updating the results. Thus, saves both time and effort from the viewpoint of both the user and resources. Also ANSYS in collaboration with AMD had evolved its products together in order to derive better performance.

On the other hand, apart from COTS, in their attempt to improve the effectiveness and accuracy of the former products, universities have developed their own in-house codes that are configured for specific cases and have achieved competitive results. Thus merits of programming and geometric structure are exploited with most delivering better performance because of their unique features.

3.2 Pre-Processing

Two different non-uniform structured grid files are the ones depicted in Figure 9 and Figure 10. It is obvious that the second grid is finer. A closer look at both grids is presented. Because the finer grid contains more cells, the computational load and execution timings are proportional to the number of the cells. The grid is adjusted in order to provide adequate y^+ results. Adequacy is discussed later. The progressive aspect ratio is set to 1.05, that means that the size of any cell compared to its adjacent is 5% different in size. By moving from centre to the outer end the size increases. Starting from the centre of the cube towards the walls, the size of the cells is reduced, but their number increases. As will be explained the different use of aspect ratio is to capture physical phenomena discussed in section 4.1. Wherever geometric coordinates are used, the origin is at the centre of the cavity.

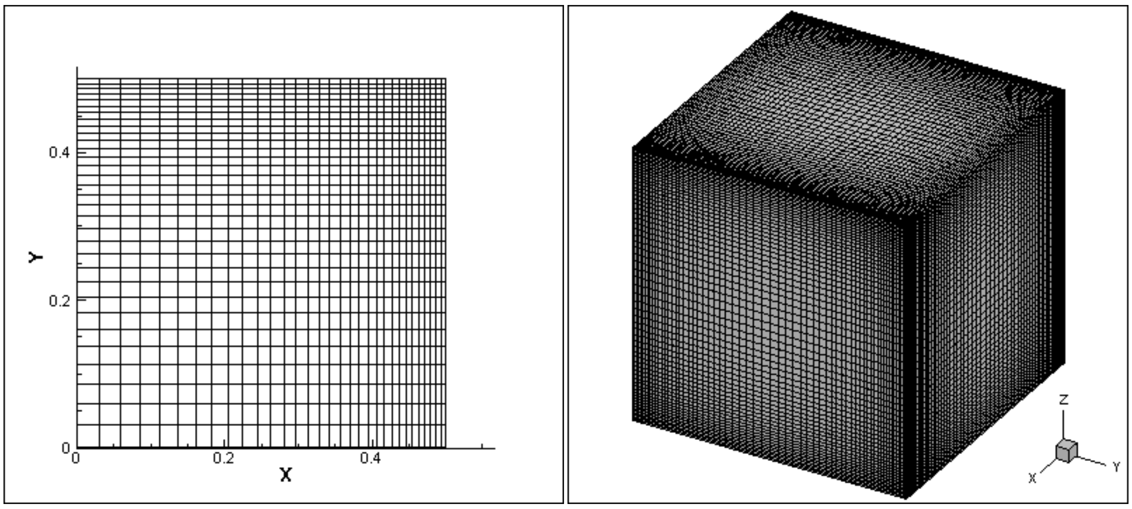


Figure 9 left) X-Y quarter of a slice of 64 edge grid with progressive aspect ratio, right) perspective view of 64^3 cells cube

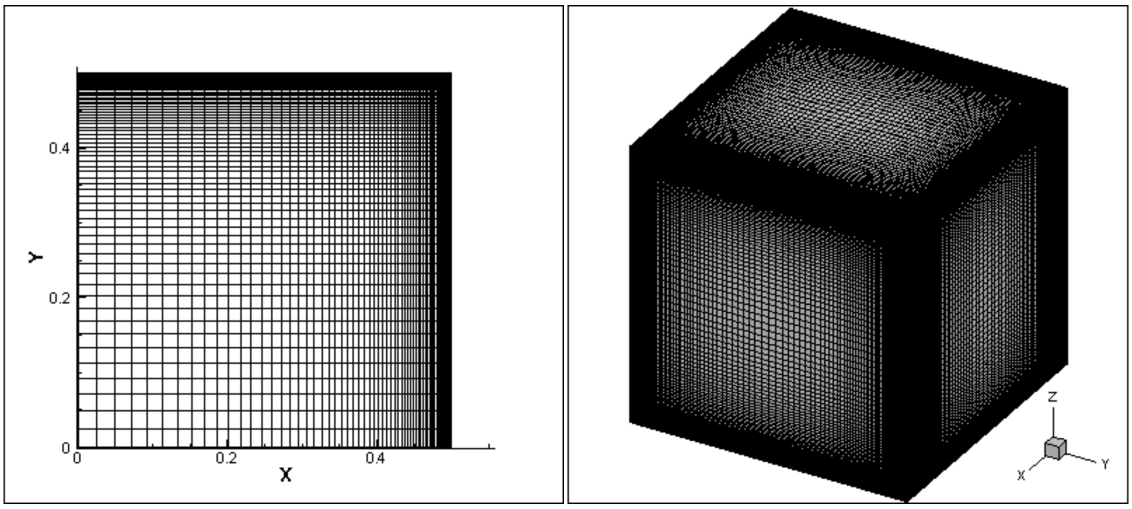


Figure 10 left) X-Y quarter of a slice of 128 edge grid with progressive aspect ratio, right) perspective view of 128^3 cells cube

The next step is to set boundary conditions. So far the geometry has been meshed in order to fit properly as input to the solvers. GAMBIT (structure grid generator program that comes with FLUENT and is produced by the same company) was used for the previous task. The initial and boundary conditions, the options related to the solvers such as lid speed, stationary walls, fluid properties, mathematical and numerical models along with their respective parameters, convergence criteria, decomposition settings are set for each software separately. This procedure is very straight forward for FLUENT. However, OpenFOAM has a utility, named fluent3DMeshToFoam, which can convert GAMBIT’s files into a compatible format. Because of its special interface, settings are set by file operations/editing. Thereby, identical input will be used resulting in a more optimal comparison.

For the time being all the simulations related to CFD features and properties have been set accordingly. The last and key point for scalability is to define decomposition settings. As it was explained above, METIS library presents very good degree of efficient decomposition. The number of partitions applied is equivalent to the number of employed cores on each case. Figure 11 and Figure 12 demonstrate how the geometry is organized in respect to the number of cells for both grid sizes. Comparing the way that domain is partitioned up to 8 sub-spaces are alike, while for 16 cores the decomposition chunks are different. Due to the number of cells needed to form sub-spaces the chunks for 64^3 are rougher, whereas for 128^3 they are smoother. Smooth subspaces facilitate data exchange especially for boundary conditions, as fewer transactions occur. In turn communication and, thus, computations among the computational nodes are more optimal.

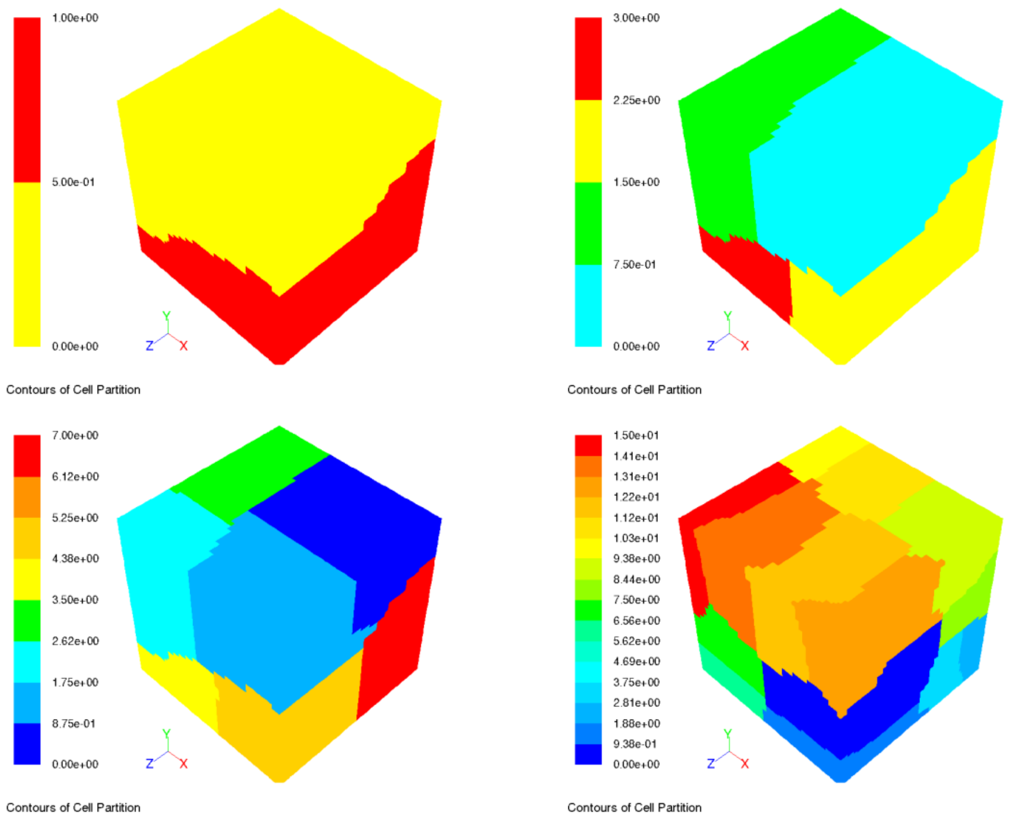


Figure 11 (top-to-bottom and left-to-right) decomposition of the geometry for 2, 4, 8 and 16 partitions via METIS algorithm for 64^3 cells

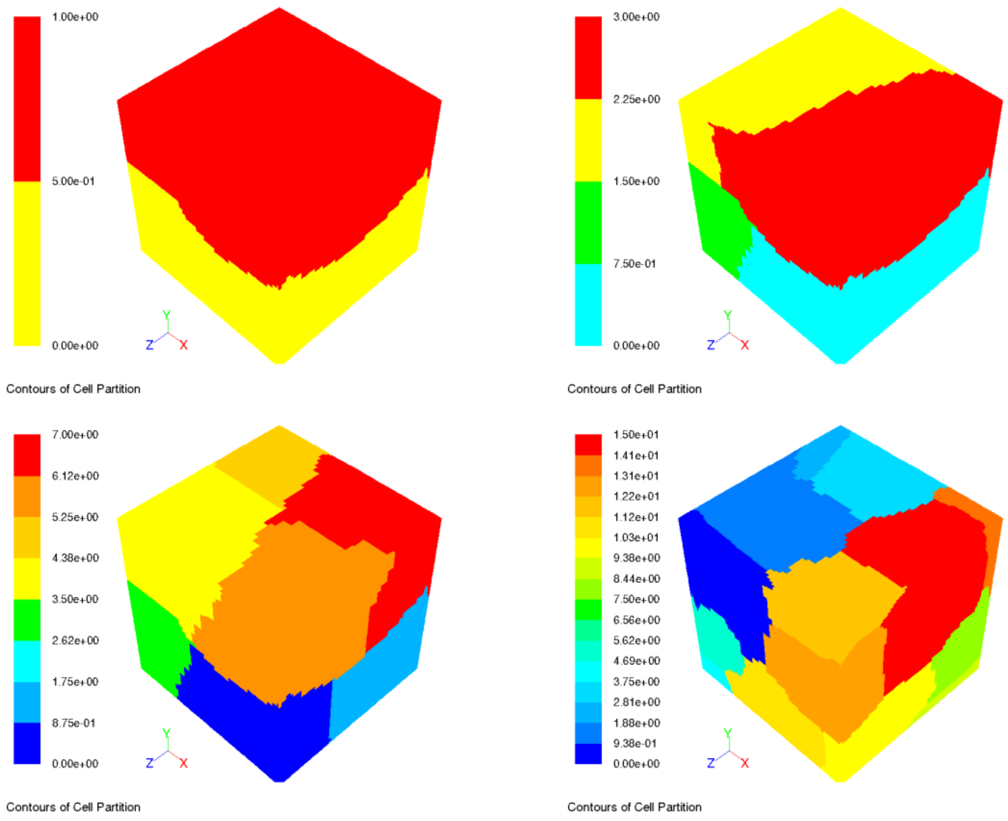


Figure 12 (top-to-bottom and left-to-right) decomposition of the geometry for 2, 4, 8 and 16 partitions via METIS algorithm for 128^3 cells

3.2.1 Settings

This section describes the parameters used for the simulations. FLUENT provides a graphical user interface (GUI) for this reason, while OpenFOAM interacts with the user through command line and editing files. LDC is subject to the following settings:

At time t=0 the liquid inside the cavity and the lid are at rest. Thereafter, at $t=t_a$, $t_a>0$, the lid starts moving to positive X direction, drifting the layers of liquid underneath, while all the other walls are stationary. The top speed of the lid is reached instantly and remains steady until the end of simulation. The value of speed depends on the type of flow, laminar or turbulent, which is fixed for Re=3200 and 10000 respectively. The characteristic length is the edge length of the cubic cavity, which is equal to 1, and kinematic viscosity value is set to water value.

RANS mathematical modelling with k-ω SST turbulence model was used for the turbulent flow. The model constraints were set to their default values. At the same time, for FLUENT the option “Low-Re Corrections” was used. In order to secure a fixed number of iterations high convergence value is set. Thus, the simulation will run for a specific number of steps. Because the computational nodes are identical and consist of the same number of processors; decomposition will take place equally in every core. Therefore METIS weight coefficients are all set accordingly to the number of employed processors in each case.

3.3 Post-processing

The Post-processing part of our study includes the comparison of the normalised velocity magnitude over the lid’s speed within the cavity. For the flow description part a set of data needs to be separated and studied individually in every dimension. That is a 3D instance of streamlines within the cavity in order to have an insight of the flow – Figure 15. Having 2D slices of data within the dimensions of the cube facilitates the alignment with experimental velocity profiles in qualitative level – Figure 13. By plotting y^+ of the cavity surfaces the use of a clustered grid is justified. Finally, the normalised velocity magnitude that lies on the axes depicted in Figure 22, Figure 23, Figure 24 and Figure 25 will be compared with the respective values of experiments. The above multiple procedures enhance the correctness and robustness of the simulations.

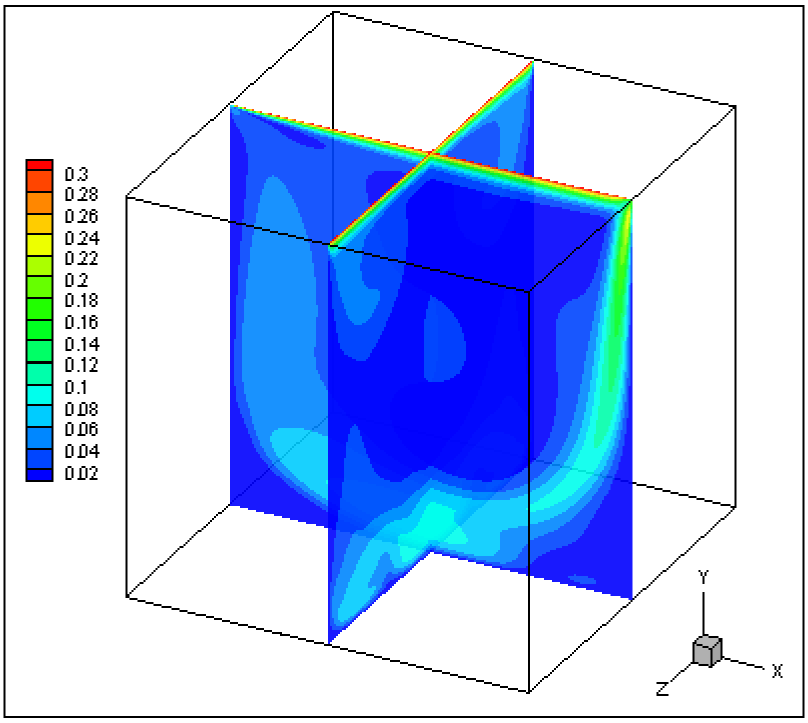


Figure 13 Slices of data in X-Y and Y-Z plane

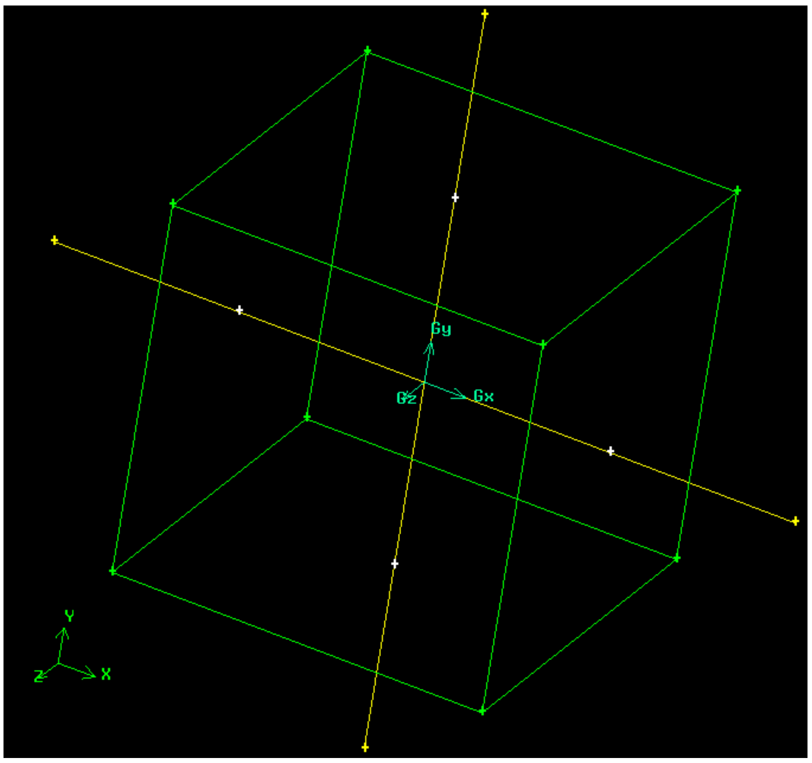


Figure 14 Centrelines at X-Y and Y-Z plane, within cube's geometry

Pathlines of simulation data that lie on the previously defined slice will be collected. The value of velocity points that lay on X and Y centrelines will be normalised with lids speed and will form curves for each plane separately. In every case the respective simulation data will be aligned and compared to the experimental values.

A combination of software will be used to visualise the obtained data. The experimental data had been extracted by [76] with the tool Plot Digitizer 2.5.0, [48]. Aliasing the numerical plane with the simulation one, the curves should follow the same patterns. That is part of the evaluation process. All the presented plots were derived by using either Tecplot 2009 [48] and FLUENT.

3.4 Performance Measurements

Regarding the computational resources, every node handles up to 4 threads because of their architecture, which hosts 2 dual cores on the same board. The communication time between these cores is the lowest possible, since internal bus interface is used. As referred to 1.1.6 weak and strong scalability are the most important performance metrics. During simulations multiple measurements were recorded for each case, where the same case ran for diverse number of iterations. It has been observed that the longer the simulation runs the shorter is the time interval per loop for the respective iteration. The reason is that initially memory has data that is not related to the simulation. While the simulation runs spatial and temporal locality along with page replacement algorithms fetch data inside the memory which improve hit searches in cache resulting in quicker turnaround time. However, its variation is usually less than 5%. Hence, it is safe to consider that weak scalability is negligible. On the other hand the speed-up depends on dynamic conditions such as free available memory, network congestion and so forth. Therefore simulations are conducted again until the timings stabilize with variation less than 5%. Fulfilling the previous statements further analysis will be more accurate and reliable.

Having considered data acquisition successful for every case, data related to performance then need to be studied. That is execution timings and decomposition statistics for each case individually. The runs depend on computational resources' current state. These clusters are used daily by many users and run various applications. Network's, processor's, memory's and disk's state depends heavily on the previous use in terms of traffic congestion, temperature, space and time locality, hardware and software availability and other operating features. Therefore, running a simulation and recording time once is not reliable. Multiple instances in various times should be run. Average timing could give a reliable result for later deductions. For each case the best results are chosen in terms of the lowest execution time which presents the lowest variation. Finally, gathering metadata related to each simulation could throw light on unpredictable behaviour.

Turbulent flow is expected to be more time consuming than laminar because of the larger number of calculations due to turbulence modelling equations as discussed in the previous chapter. Apparently the larger the grid size, the longer the simulation will be because of the increased amount of data to be processed. The cases will be presented by ascending turnaround time; laminar 64, turbulent 64, laminar 128 and turbulent 128.

Knowing how software works is important in terms of focusing on the target timings. Breaking down the solver procedure; mainly consisted of data decomposition, applying boundary conditions, calculating chunks numerically and reducing data to the origin and reordering results before the next iteration started. On FLUENT these parts are automated, whereas on OpenFOAM further interaction by the user is needed. In order to have a fair overall comparison, OpenFOAM’s steps are bundled in a batched way and overall timings are recorded. Obviously, for FLUENT the execution time provided by OS time-wrapper is resolved. Since OpenFOAM is more versatile in its execution the timings may vary depending on the time needed to decompose and reconstruct data. The first depends on the number of used cores and decomposition library, whereas the time needed for the latter is due to the frequency of data storage, and, in turn, the number of files to be merged together. In comparison with the most time consuming part, which is the used solver (as discussed above), the cumulative time for the rest of procedures is smaller than 0.1%. So the recorded time for OpenFOAM is purely the wall clock time provided by the same application.

The previous reported way of recording timings is not suitable for in-house codes because of their unique nature. It depends on the intrinsic features of the code and the developer’s way of programming. On in-house codes of [100] analysing the geometry with smaller grid size in 3D LDC the computation part is separated into three phases – integration, Right Hand Side (RHS) and solver – present different level of efficiency. The highest one is achieved for RHS, whereas the solver is the least efficient. The bottleneck is the cache size because within it the problem size is fitted effectively. Thus, more reliable timings could be compared and deductions obtained safely.

3.5 Scalability Measurements

The prime interest in this section is the turnaround time over a fixed number of iterations rather than measuring the time needed until the convergence is reached. That means that convergence is set to a very low value, so that for the respective number of iterations, it will never be obtained. The reason for choosing this strategy is that we would like to present results independent of the availability of resources or strictly bound by the number of iterations. It is the execution speed that is our concern, which denotes the number of iterations over the time until completion. In fact, if measuring time until desired convergence is recorded, it should be divided by the number of iterations done. So that would result in the same metric.

The number of iterations is big enough to eliminate statistical error from our population. Also the variation of the results is checked before the final plotting and their average value is calculated. If variation is higher than 5%, then the simulation is conducted again.

Regarding the speed-up measurements, communication speed is very important, which is very difficult to compute due to network congestion and lack of knowledge of the exact quantity and type of MPI calls. Population statistical results will be used for the calculation of the efficiency Therefore, efficiency in terms of frequency will be derived as in:

$$Eff = \frac{\frac{f_p}{f_s}}{N} = \frac{\frac{t_s}{t_p}}{N} = \frac{t_s}{N \cdot t_p} \qquad \textbf{(3-1)}$$

Where f is the frequency metric and s and p indices denote serial and parallel instances respectively. N is the number of used processors.

In order to have a clear picture, all the values in plots were normalised at their lowest value, which is the one for one processor. Finally, the desired properties are (super) linearity in speed-up plots and efficiency above 1 for every case and any number of employed cores.

4 Results

4.1 Agreement with Physics

The number or simulations used for this chapter is revised in 6Appendix A. The initial step is to check whether the simulation data agree with the experimental data. This can be done in qualitative and quantitative way by analysing flow features and patterns. In order to cross-check the fidelity multiple aspects will be compared thus ensuring the reliability of the data.

The description of the flow starts from general features to more specific in 3D, 2D and 1D as follows. Initially the flow is at rest. Through the movement of the lid the molecules will follow a cyclic course until they reach the lid again, since the geometric structure of the cavity forces them to move in that way. Then the recirculation of the flow continues as the lid keeps moving. Figure 15 demonstrates the "recirculation" cycle of the flow. In comparison with experimental sketches of the flow the upper secondary eddy is not depicted. This is related to y^+, as explained below.

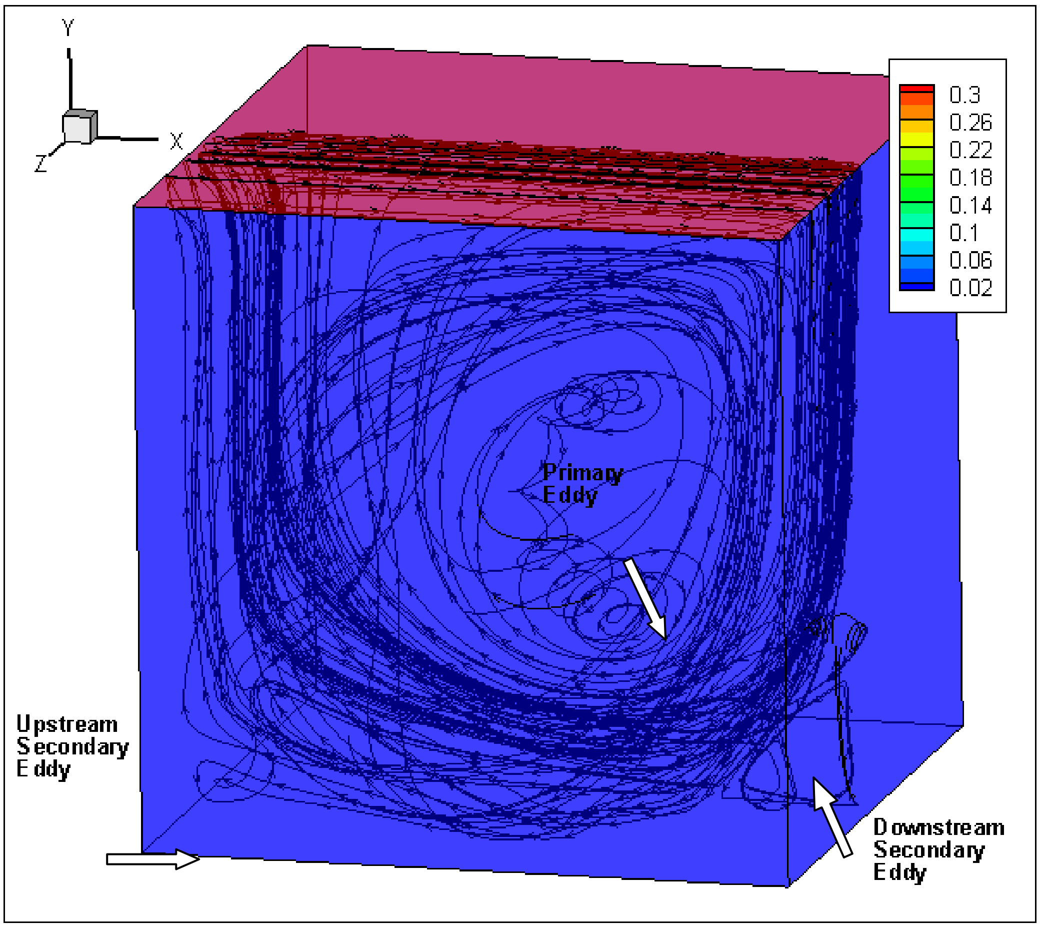


Figure 15 3D flow definition sketch

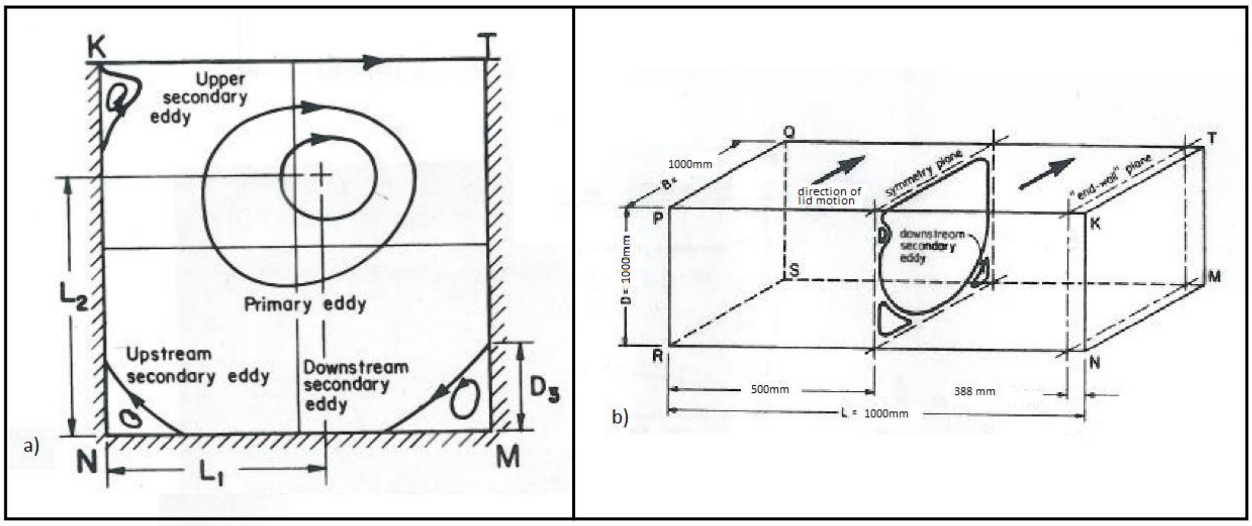


Figure 16 LDC flow definitions in a) slice-[59]**, and b) geometry** [56]

On every slice of the cube parallel to the movement of the lid, there is a primary eddy in the centre of the cavity with the same direction as the movement of the lid. While the lid moves in X direction, it forces the primary eddy to circulate. Because of primary eddy's movement some secondary eddies are formed on three corners edges. These are named downstream-, upstream- and upper-secondary eddy with counter direction of the primary eddy and their size depends on the speed of the lid. These are the so called Taylor-Görtler-like (TGL) vortices in the vicinity of downstream secondary eddy and corner vortices, and they do not disappear as Re increases. As expected because of the shear and pressure forces acting on the fluid, vortices are produced near the ends. Various case of spanwise-aspect-ratio (SAR) on rectangular cavities have shown that the eddies should be significantly smaller compared to the primary eddy at the centre of the cavity-[58]. That is also the case on our cubic LDC. The hypothetical fourth eddy that is perpendicular to the lid's movement, vanishes because of the momentum of the water. Since the fluid is at rest and final speed is applied instantly, a finite transient period occurs before the eddies are formed. Those phenomena are present both on experimental and simulations as a first comparison-[56], [59].

Quantitatively, Authors of [76] provided experimental values on the centrelines of the cavity. Specifically, the values of velocity on X and Y axis are normalized with speed of the lid for laminar and turbulent flow respectively. Thus, the curves in Figure 22, Figure 23, Figure 24 and Figure 25 are derived. If the circulation patterns within symmetry – slice at the centre of the cavity – and “end-wall” differ, the velocity curves for Re=3200 and Re=10000 will be very similar. This is true in our case. Furthermore, in turbulent flow fluctuations have 25 times more energy. The number of vertical fluctuations over horizontal ones verifies the existence of TGL-[56]. The results are very similar to the ones obtained on 2D study in [35]. For laminar flow the curves are near identical, while for turbulent flow they are very close. That is due to the complex computations for turbulent cases. Convergence is slower when we use the same number of iterations for both types of flow. Aligning the curves would be possible, if more iterations were run. However, obtaining exact similarity is out of the scope of this study. It is obvious that the patterns of the curves are very similar and, thus, the input is reliable and accurate.

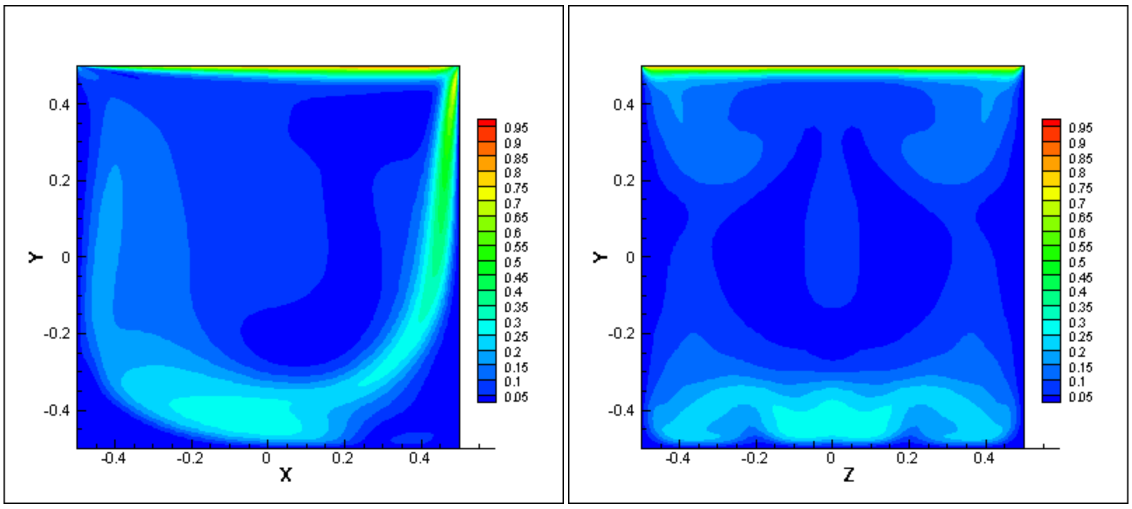


Figure 17 2D visualization of slice of X-Y (left) and Y-Z(right) plane of normalized velocity magnitude over lid speed for laminar flow on FLUENT

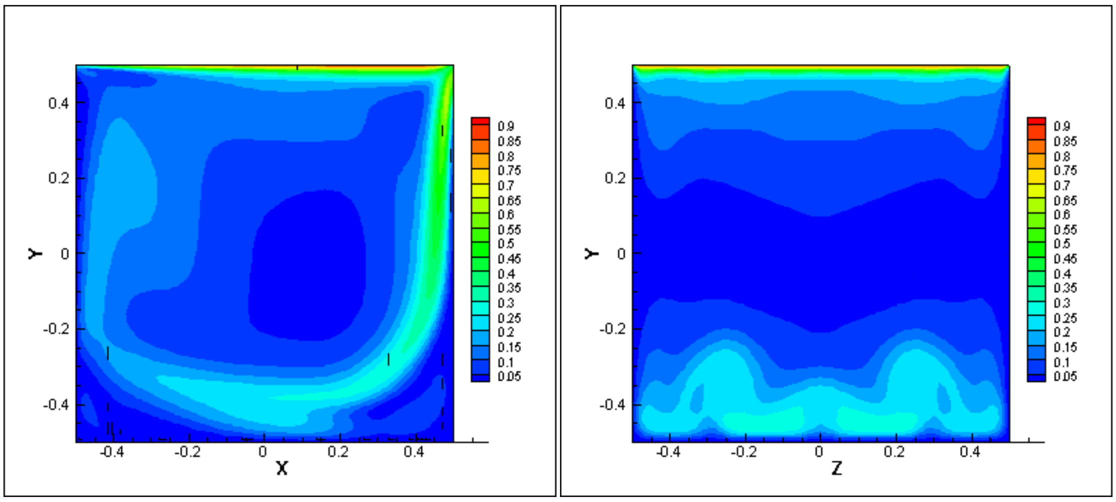


Figure 18 2D visualization of slice of X-Y (left) and Y-Z(right) plane of normalized velocity magnitude over lid speed for laminar flow on OpenFOAM

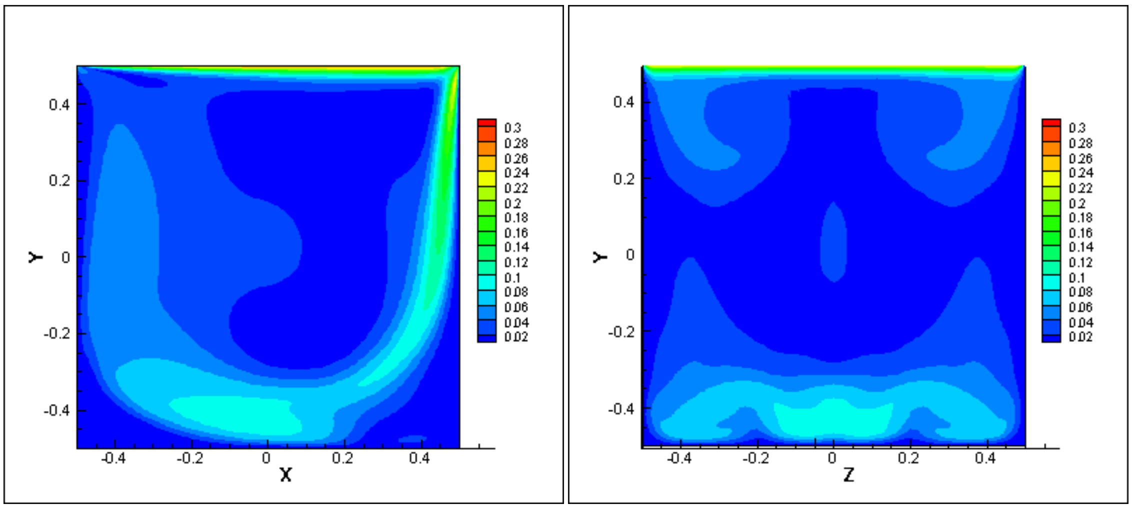


Figure 19 2D visualization of slice of X-Y (left) and Y-Z(right) plane of normalized velocity magnitude over lid speed for turbulent flow on FLUENT

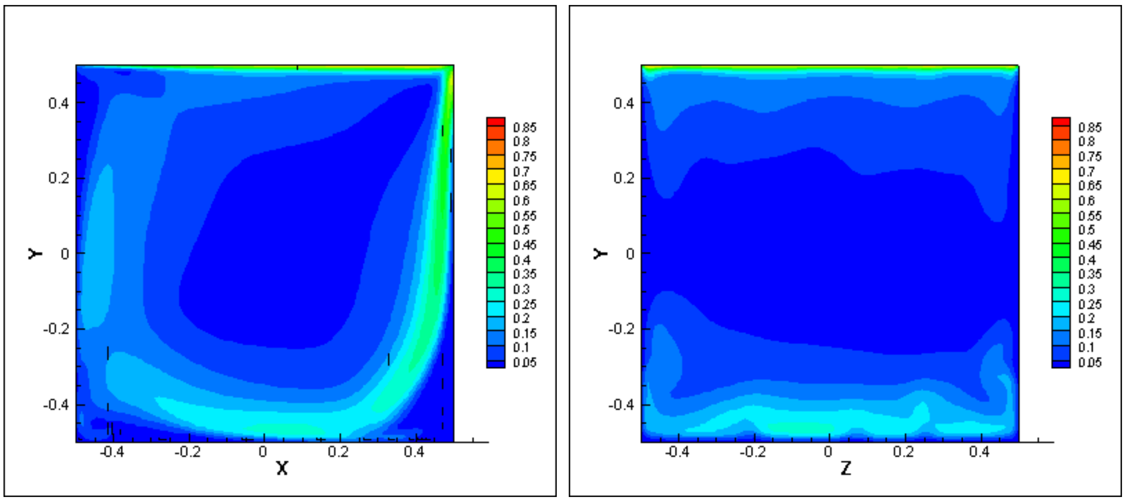


Figure 20 visualization of slice of X-Y (left) and Y-Z(right) plane of normalized velocity magnitude over lid speed for turbulent flow on OpenFOAM

The quality of grid also needs to be assessed. The y^+ metric is used for this reason. Figure 21 presents the plots for every surface of the cavity. It is worth to noting that the majority of the points lie below 1. Apart from the lid, the rest of the surfaces present y^+ value lower than 2, while the lid surpasses it slightly near the edges of the cavity. The existence of higher level of y^+ near the edges explains why the upper secondary eddy was not identified. Nonetheless, it could be detected, if the grid was even finer or at least denser over that area. In theory, if y^+ is lower than 5, it is acceptable. Hence, the quality of the grid is good enough to capture effectively all the vortices of viscous layer. According to experimental and simulation data the upper vortex has not been captured clearly. That is the reason for the higher y^+ next to edge of the lid. If the grid was finer to that vicinity, the existence of vortex would be apparent and y^+ value would be even lower. Therefore, the grid is satisfactory as far as the needs of the study are concerned since an appropriate number of cells will be produced and processed.

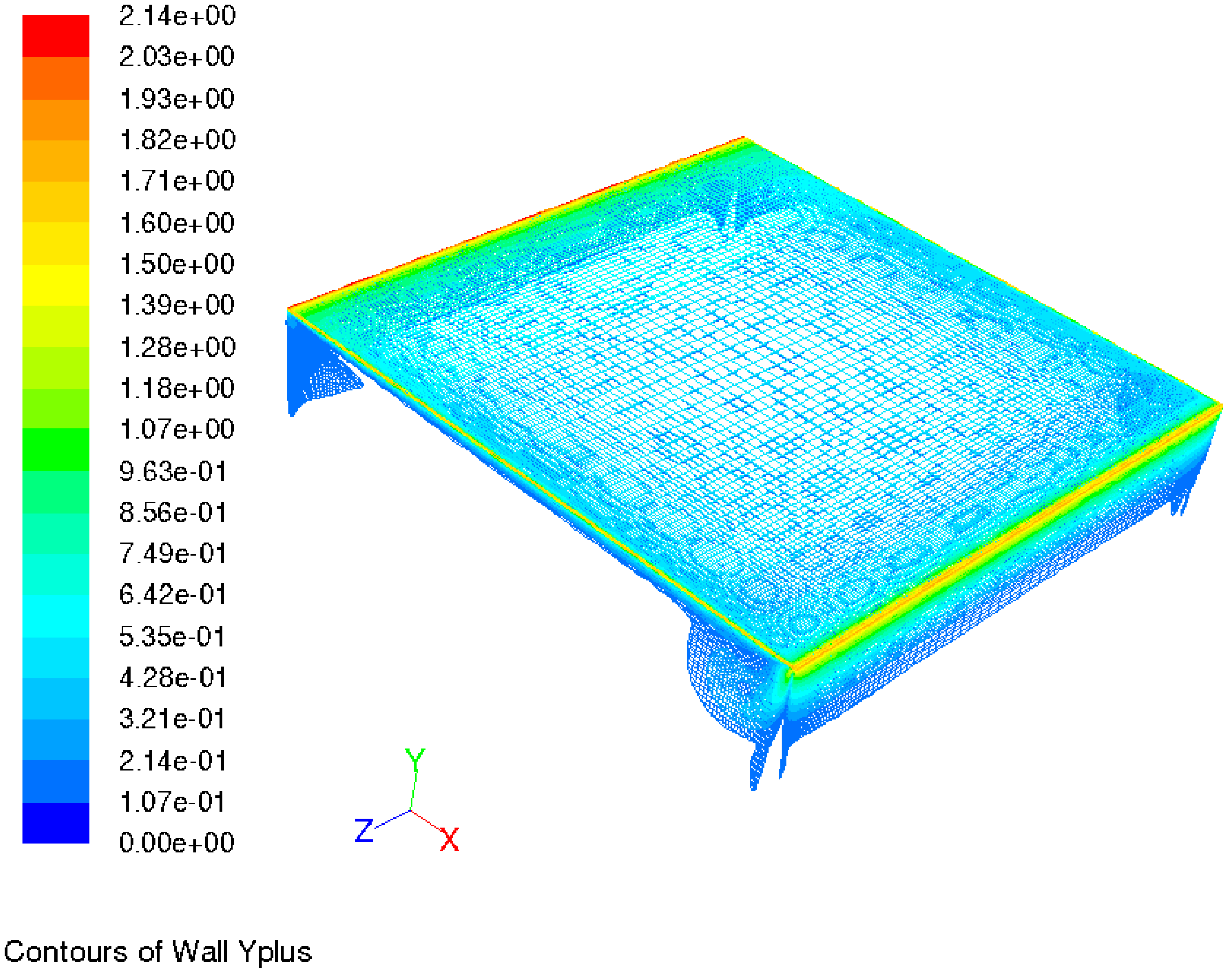


Figure 21 y^+ plots for each surface of the cavity

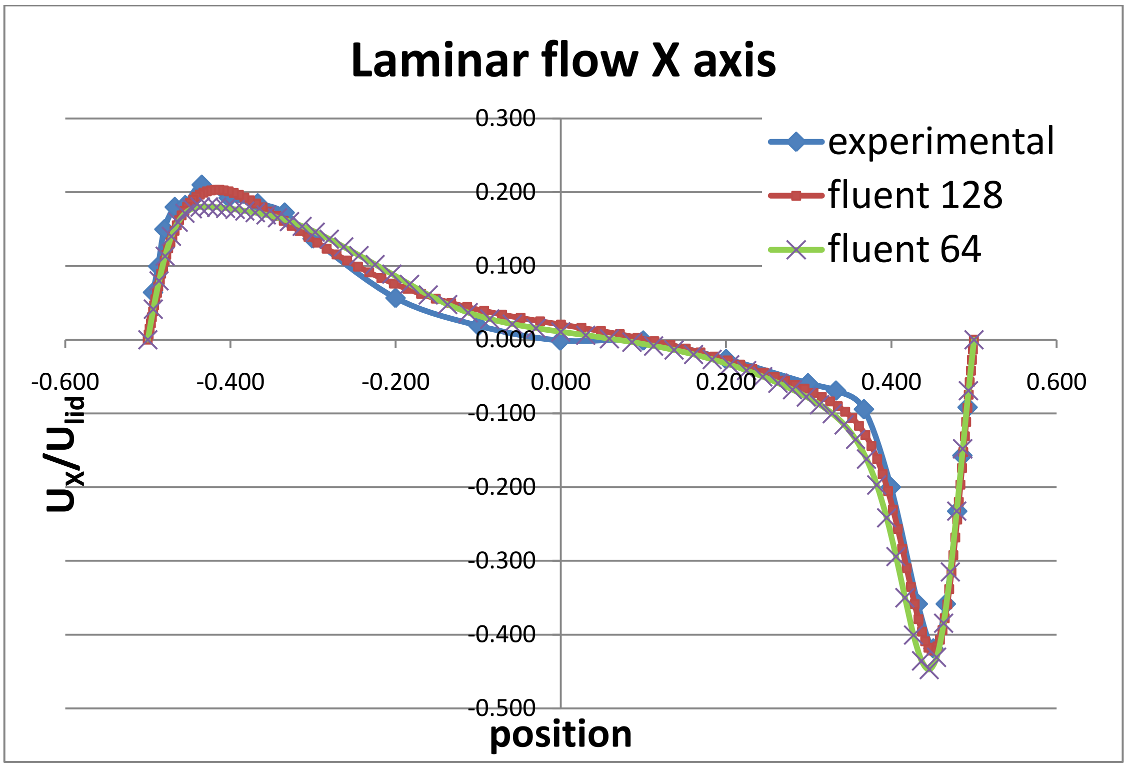


Figure 22 Laminar Flow on X axis for experimental and simulation data

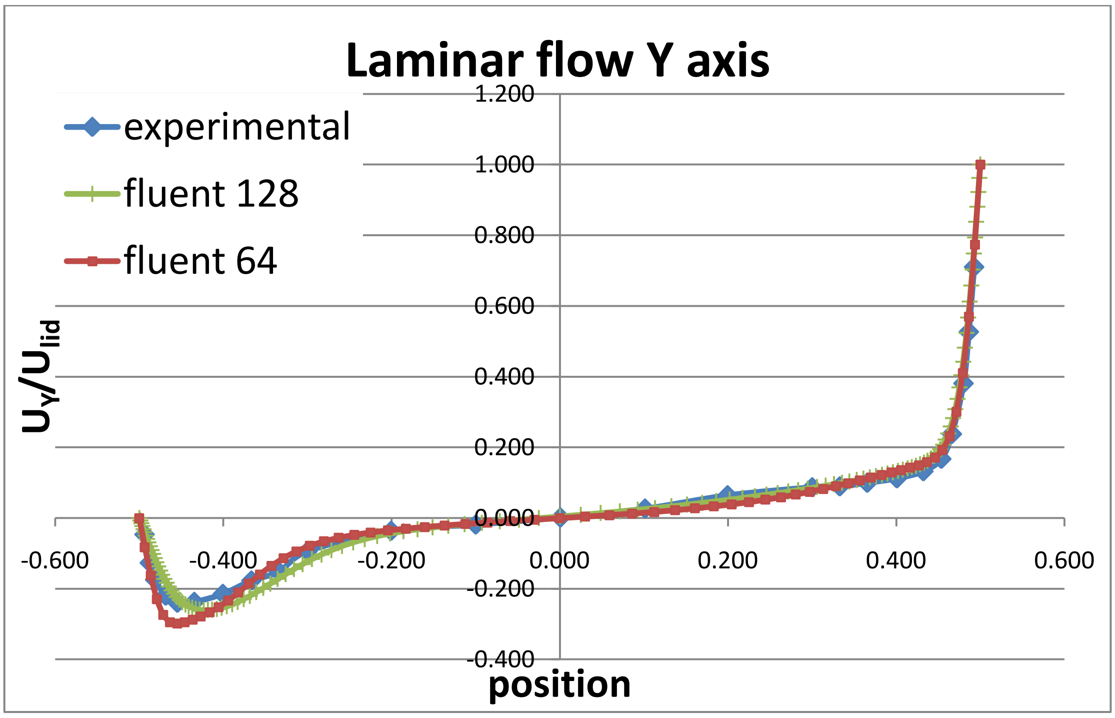


Figure 23 Laminar Flow on Y axis for experimental and simulation data

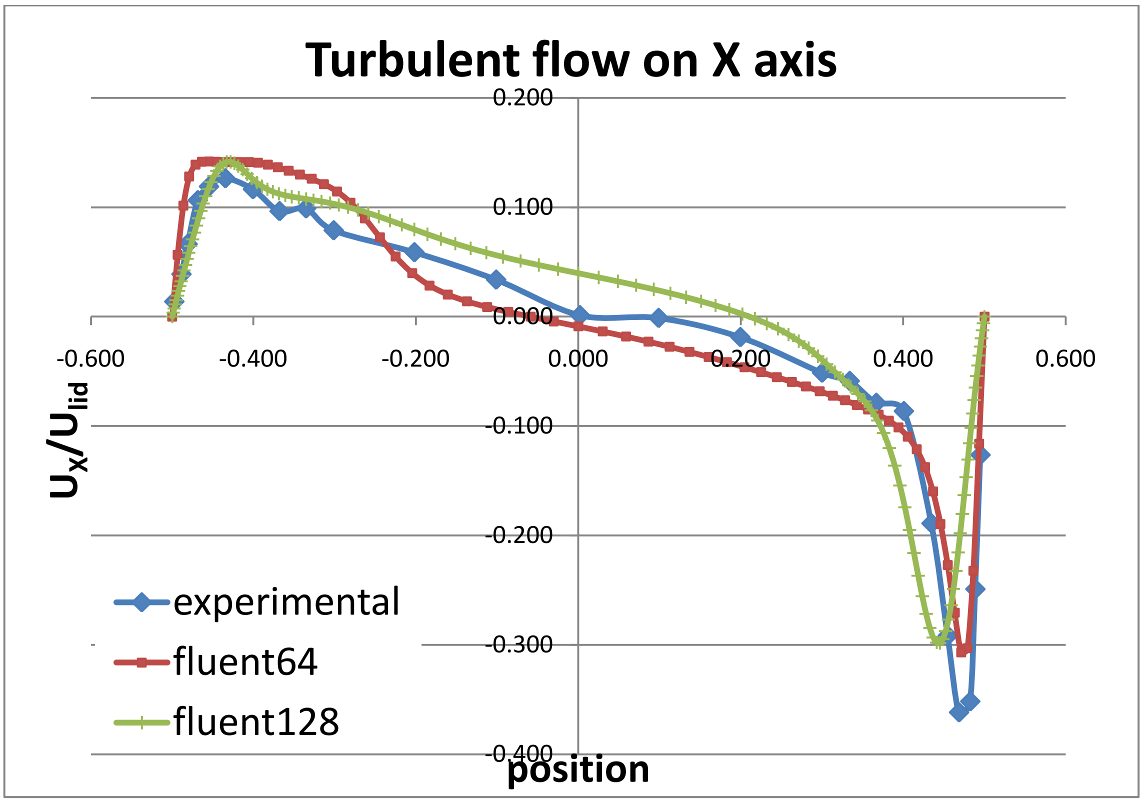


Figure 24 Laminar Flow on X axis for experimental and simulation data

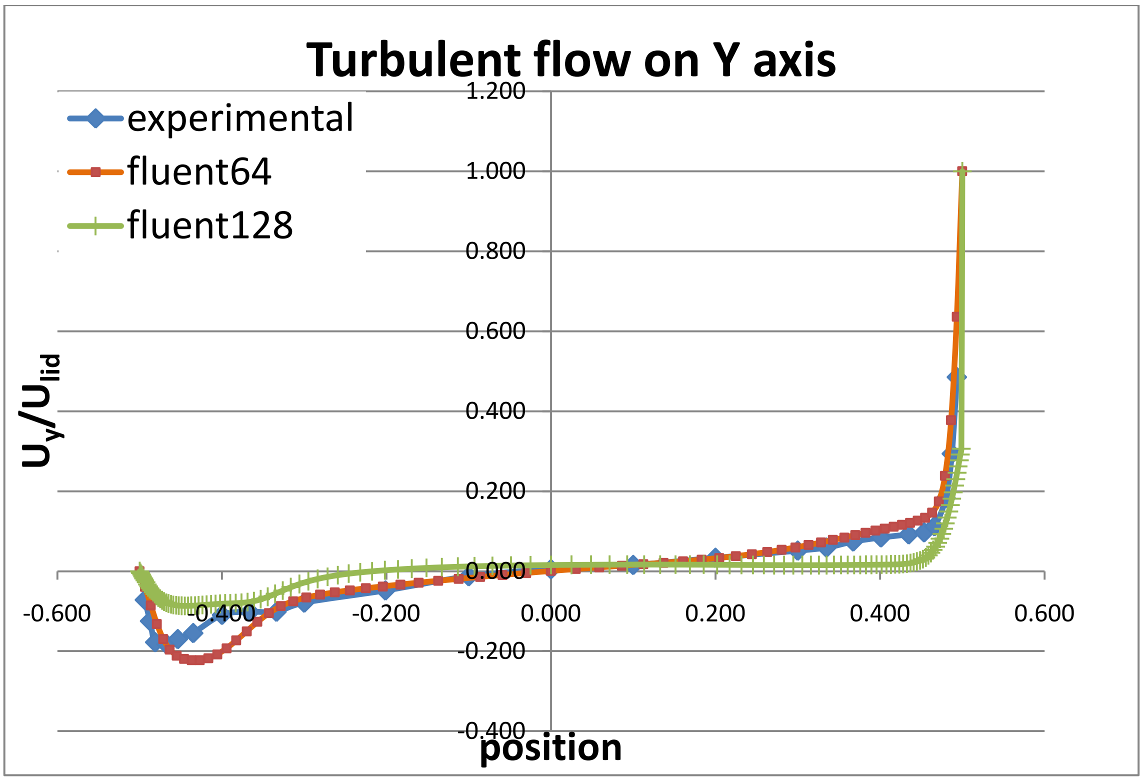


Figure 25 Laminar Flow on Y axis for experimental and simulation data

The experimental data are in agreement with the simulation data. So, the simulation is considered robust and the main comparison plots follow. Overall execution time plots from each solver are aligned on the same picture for ease of comparison.

4.2 Performance Results

In each of the four following sections simulation results will be discussed individually focusing attention on the particular scenarios. The plots are clustered in groups for ease of evaluation. According to clusters' specifications, both of them use 2 dual core processors. That means that each node can handle up to 4 threads concurrently, and, in turn parallelization up to 4 cores takes place within the same chip. The reason is that communication channels between cores are the most optimal routes and, by order of magnitude, the quickest way to exchange messages. By nature, the network channels are slower, because of the kernel material of the cable (copper or optic fibre) and the distance between two ends. Therefore, above 4 cores the tasks are clustered in groups of 4 (cores) and are spread accordingly. This is the only feasible way for the highest performance to be achieved.

4.2.1 Commercial – ASTRAL

Looking at Figure 26 and Figure 27, one can understand that FLUENT presents a stable behaviour. Employing a second core achieves linear speed up, whereas beyond that it presents satisfactory speed-up on every scale. So, the underlying hardware is adequate for application requirements and the overall speed up is the best among all the conducted simulations. A small "delay" in terms of speed up is observed from 2 to 4 cores. That is due to memory size that imposed more miss and fetch calls to the memory, resulting in idle execution cycles. From 8 to 16 cores the speed-up is linear.

A drop in efficiency is due to non perfect applicability of decomposed data into the processors' memory. So, more cache instructions slacken the performance by approximately 40%. For more than 4 cores the impact of interconnection partially contributes in the decrease of performance. However, speed up is linear from 4 to 16 cores which means that either few network transactions occur or interconnection speed is so high that communication time is negligible. As discussed above the small gap (which remains nearly constant throughout all the cases) is subject to the quantity of data to be processed. In general, FLUENT is very balanced on astral and the time ratio between two grid sizes is nearly 1:8 for all the cases; turnaround time is analogous to problem size, without any irregularities. In order to achieve ideal results processor replacement is the only way because of the need for larger memory to secure better spatial and temporal locality and probably the use of optimized cache replacement algorithms for CFD cases.

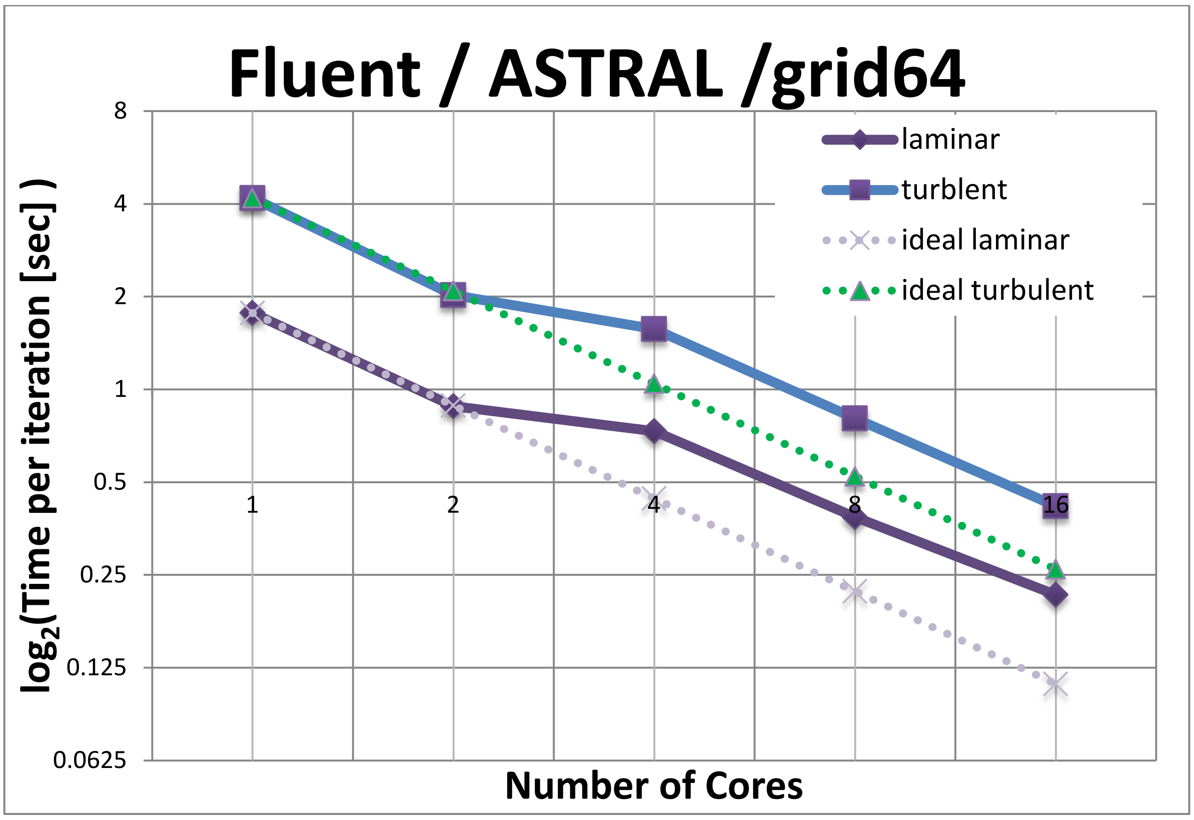


Figure 26 Turnaround time of FLUENT on ASTRAL for coarse grid

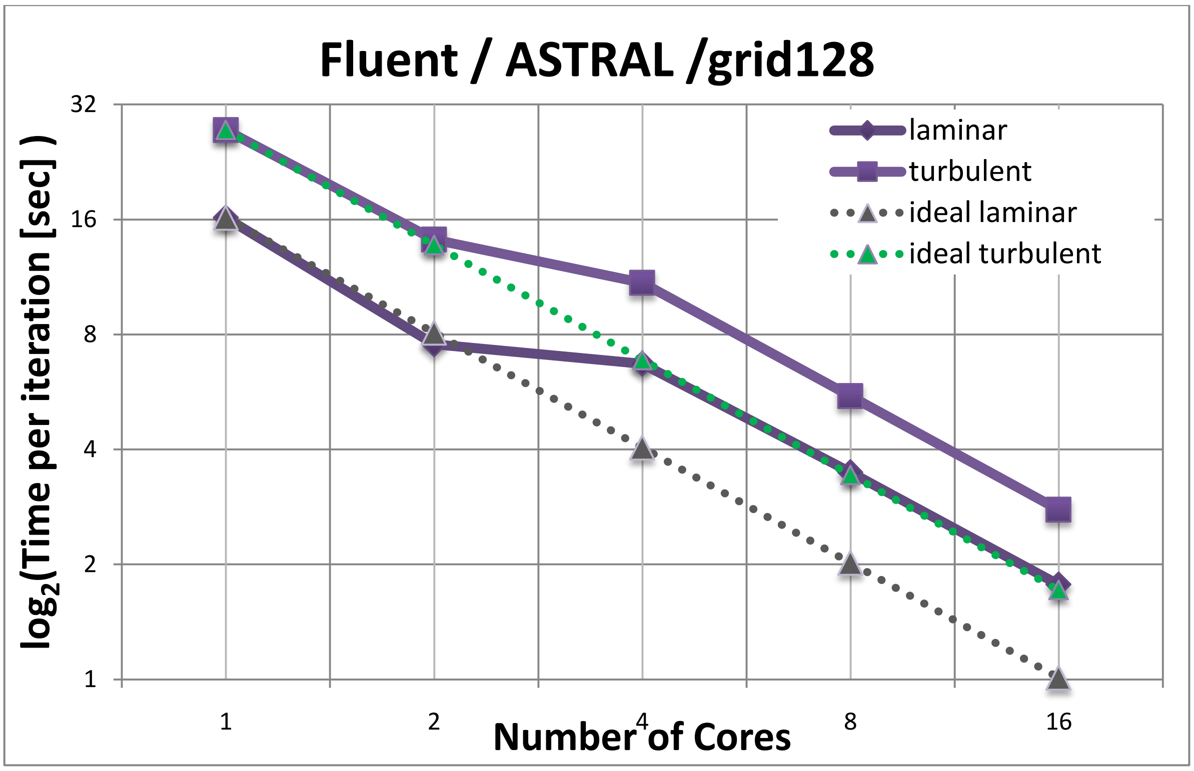


Figure 27 Turnaround time of FLUENT on ASTRAL for fine grid

Generally speaking, the efficiency of FLUENT is nearly identical for all the cases. All the lines follow the same trend; starting from 1 core a small peak in efficiency exists for 2 cores, except for turbulent of 128 gridsize. Beyond that, from 4 cores to 16 the efficiency lies in the range from 50 to 65%. The most efficient is the turbulent flow of 64 grid size presenting slightly higher trends than the rest of the cases. That means that FLUENT makes better use of available memory resulting in better time per iteration.

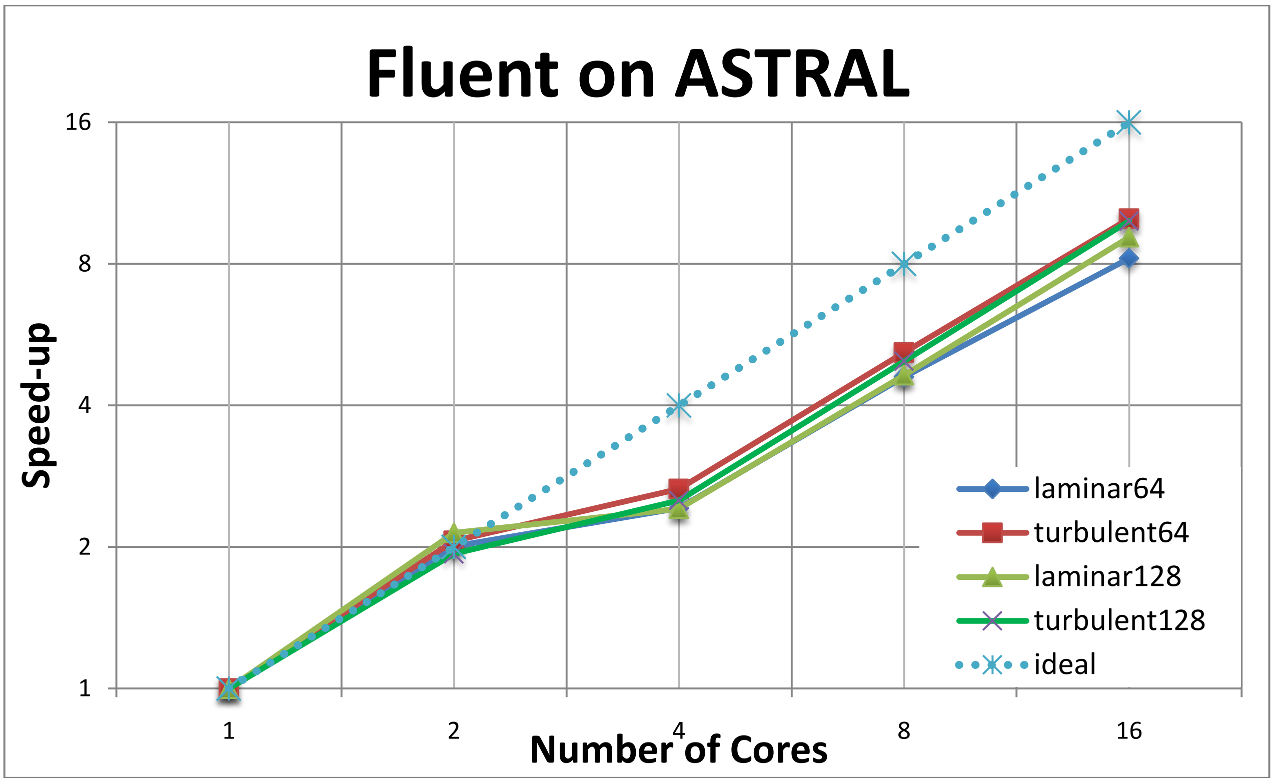


Figure 28 Speed-up of FLUENT on ASTRAL

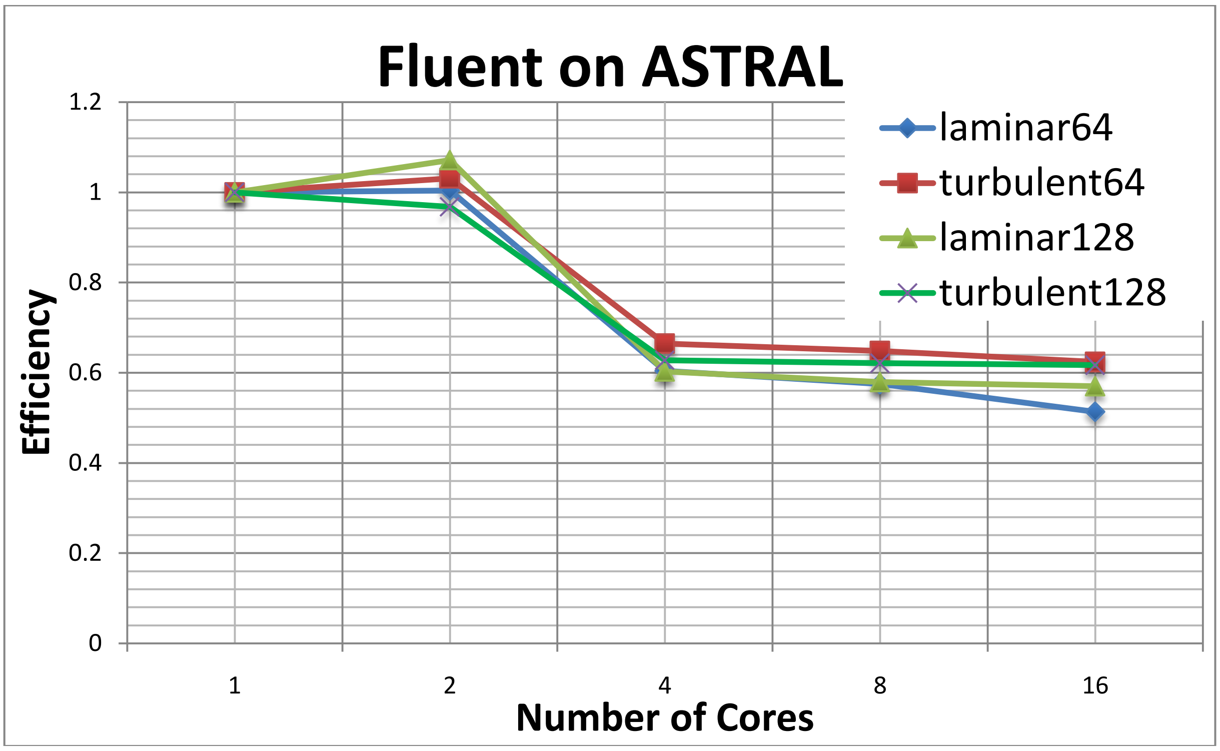


Figure 29 Efficiency of FLUENT on ASTRAL

4.2.2 Non-Commercial – ASTRAL

The overall behaviour of OpenFOAM is different. It is clear that the cases of 64 cell grid are approximately 40 times quicker than the 128 cell grid, in terms of 16 cores. The reason is the large amount of computational data due to grid size. The speed-up is fairly good for 2 processors, but beyond that it either stabilizes to a small range or it is very slow. Among the cases, only laminar flow of 64 grid reduces, even at a low pace. Turbulent flow presents minimum for 8 cores, while beyond that point it augments. In contrast, laminar flow of 128 grid presents worse behaviour for 8 cores compared to the time per iteration for 4 cores. While CPU's architecture can support up to 4 concurrent computational threads, by watching the plots one can say that the memory gets full for 4 cores. To some extend, the frequency of processing data slackens. Above 4 cores the contribution of network traffic affects overall performance as communication time is comparable with processing time. The above 3 cases that do not scale well are subject to the size of transmitted data and the amount of network transactions. Finally running the simulation in parallel is more efficient that serial form.

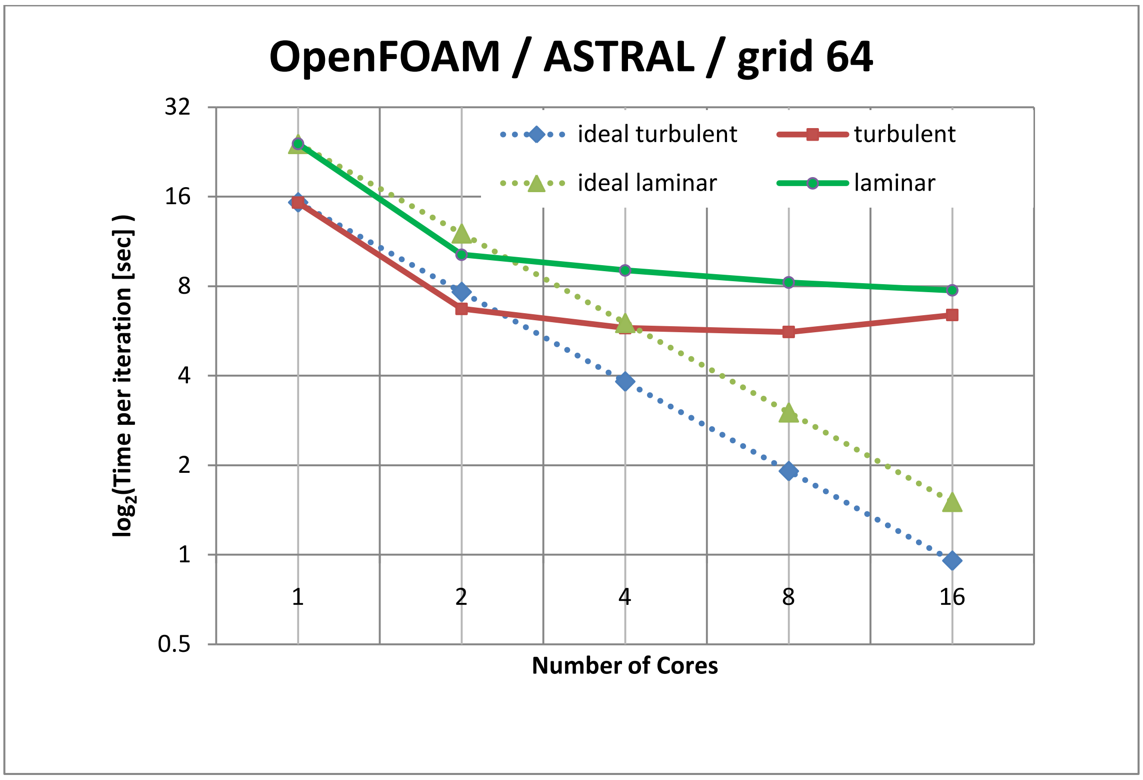


Figure 30 Turnaround time of OpenFOAM on ASTRAL for coarse grid

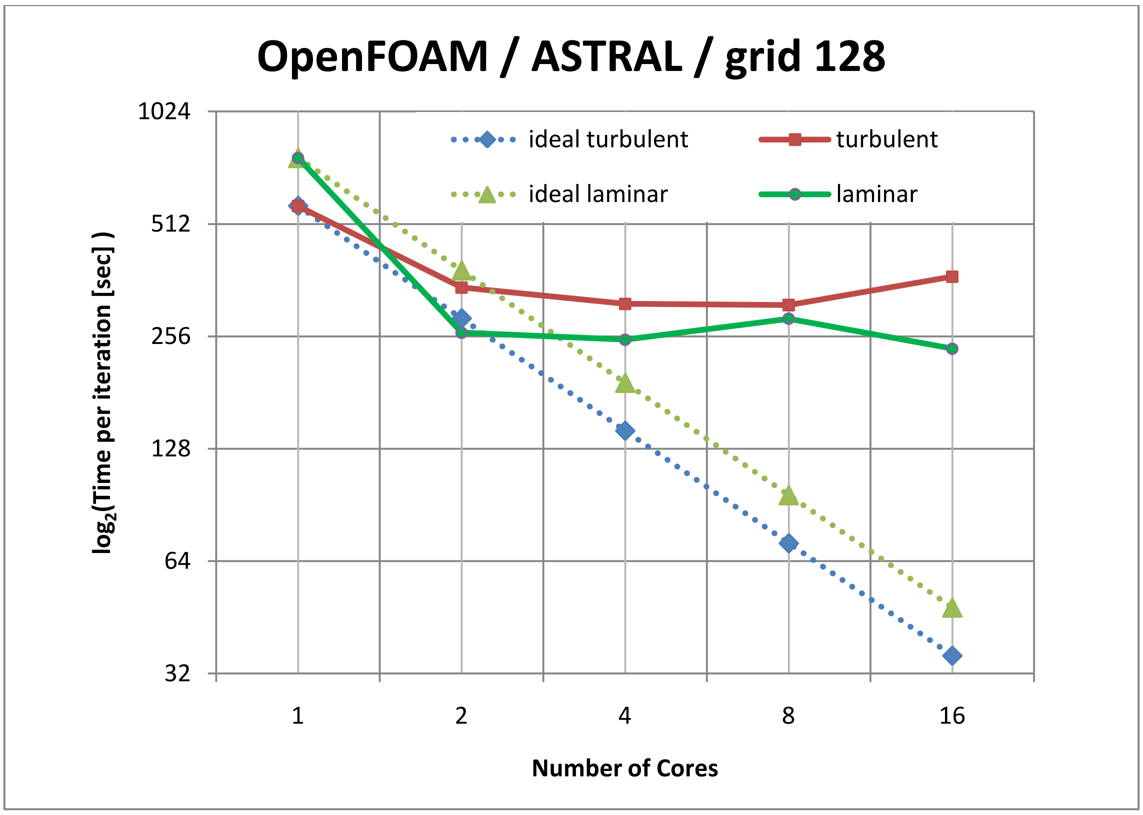


Figure 31 Turnaround time of OpenFOAM on ASTRAL for fine grid

The efficiency is presented in Figure 33 for both types of flow and both grids. All the lines follow the same trend. The peak of the plot is achieved while using two processors for all the cases except the turbulent flow of fine grid. The laminar flow in a fine grid is the most efficient by a margin of approximately 50% compared to the flow in a coarse grid.

Simulations of turbulent flow are quicker than laminar for 64 gridsize for the whole range of core. That is also true for 1 core with 128 gridsize, but running in parallel “returns” to the expected behaviour for the rest of the cases. From 1 to 2 processors super-linear scalability is observed for both cases of grid 64 and for laminar flow of grid 128. In contrast, running turbulent of 128 gridsize is not efficient at all. The efficiency of laminar 128 for 2 cores is the highest achieved among all the measurements. For grid 64 both flows present similar behaviour; up to 2 cores good scalability is achieved, while the speed up is very slow beyond that and ends at only 20% efficiency for 16 cores. According to Figure 32, the platform reached maximum performance for turbulent flow of grid 64. Also it is evident that the use of 16 cores would achieve the same performance as if 4 cores were used. So that is clearly a waste of resources both for the current user and the rest of the users of ASTRAL since large network congestion would decelerate other jobs running in parallel. For grid 128 the efficiency is even worst.

Speed-up curves stray from the ideal behaviour early and the maximum speed up is achieved for fine grid for laminar flow with 16 cores, which is very close to 2 cores. The time ratio of coarse over fine grid is approximately 1:32. Another irregularity is present for 8 cores for grid 128, Figure 31; The turnaround time for 8 cores is longer. The only obvious cause is that if data decomposition for the specific workload is off power of 4, then the simulation will be carried out in a shorter amount of time. It is undoubtedly poor collaboration between solver and provided infrastructure that is the explanation for the above performance. Compilation of OpenFOAM with HP-MPI from scratch or use of a different compiler could possibly improve the situation.

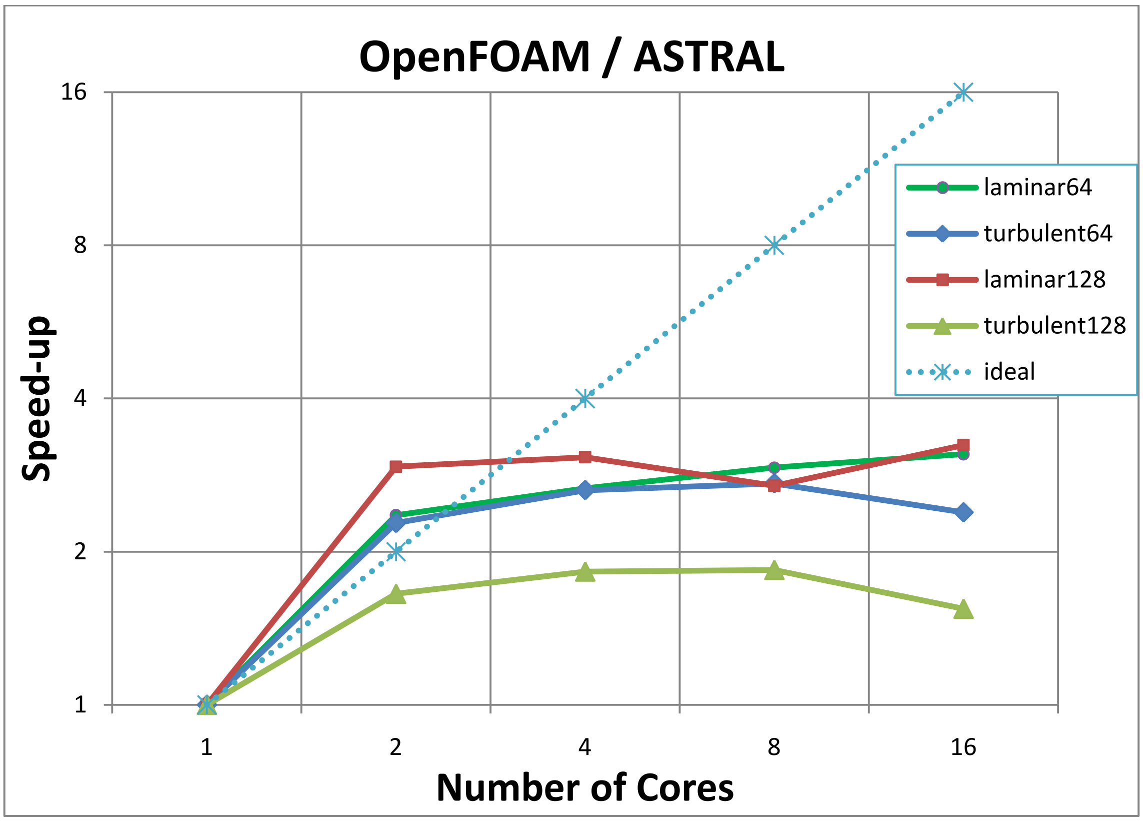


Figure 32 Speed-up of OpenFOAM on ASTRAL

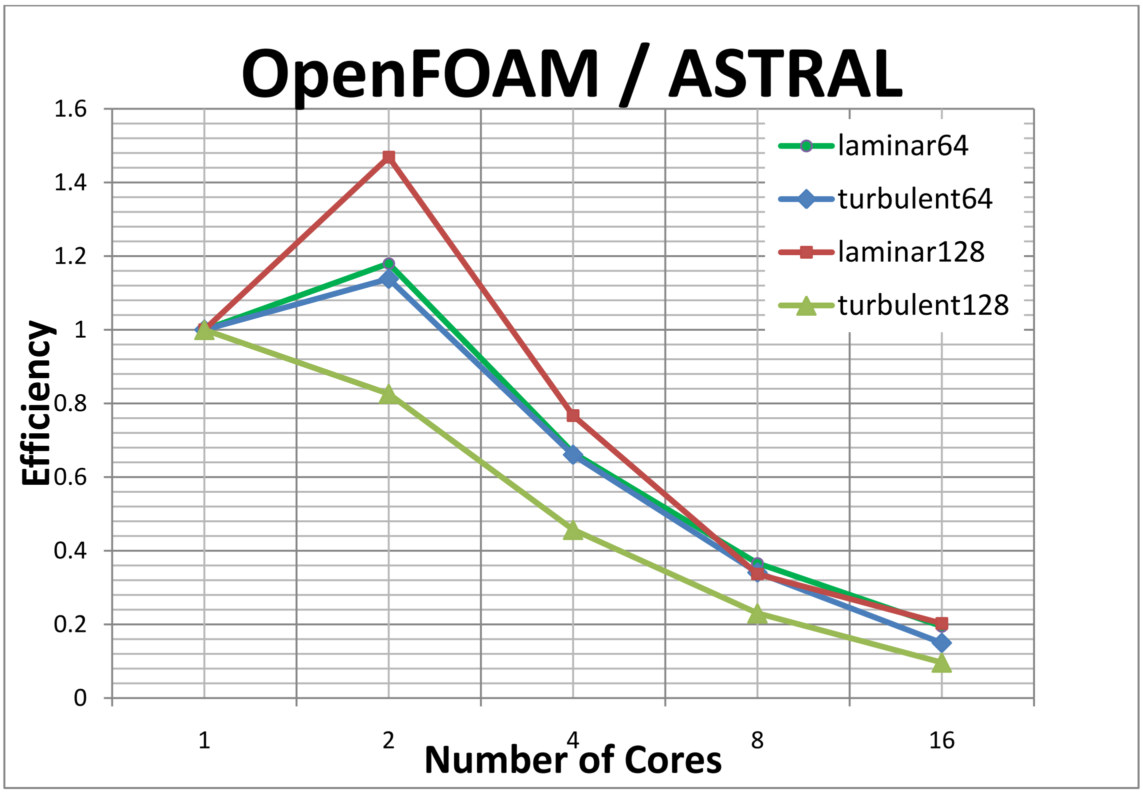


Figure 33 Efficiency of OpenFOAM on ASTRAL

One would expect even higher efficiency for 4 cores. However, this is not true because the number of equations to be solved is so large that it fits sufficiently in common CPU cache (normally Level2). For 4 cores, more miss calls and fetches occur within circuits so that the gain of 4 parallel threads is suppressed because of race conditions and idle times. Despite the big gap from 1 to 2 processors, for more than 2 processors the efficiency drops quickly. For 16 processors all the cases are around 20% efficient

.

4.2.3 Commercial – Galileo

FLUENT again presents almost similar behaviour for all cases; the speed-up is near linear up to 2 cores and beyond that it attenuates slowly. In particular, for turbulent flow for fine grid slightly superlinear efficiency is attained. Efficiency curves damp gradually, while that of laminar flow for 128 gridsize stays closer to ideal behaviour and its efficiency drops significantly only for 16 cores. Performance is satisfactory up to 4 cores, while further employment of cores results in less than 60% efficiency for more than 8 cores, with the exception stated. above.

For 128 gridsize efficiency drops slower than for coarse grid. The only problematic combination is of 8 cores for fine grid. The fluctuations of the curves imply the substantial role of interconnection for parallel runs for more than 4 cores. The turnaround ratio is again 1:8. In order to obtain better performance up to 4 cores better processors should be employed (as described in section 4.2.1). Further efficiency would be attained by using quicker interconnection.

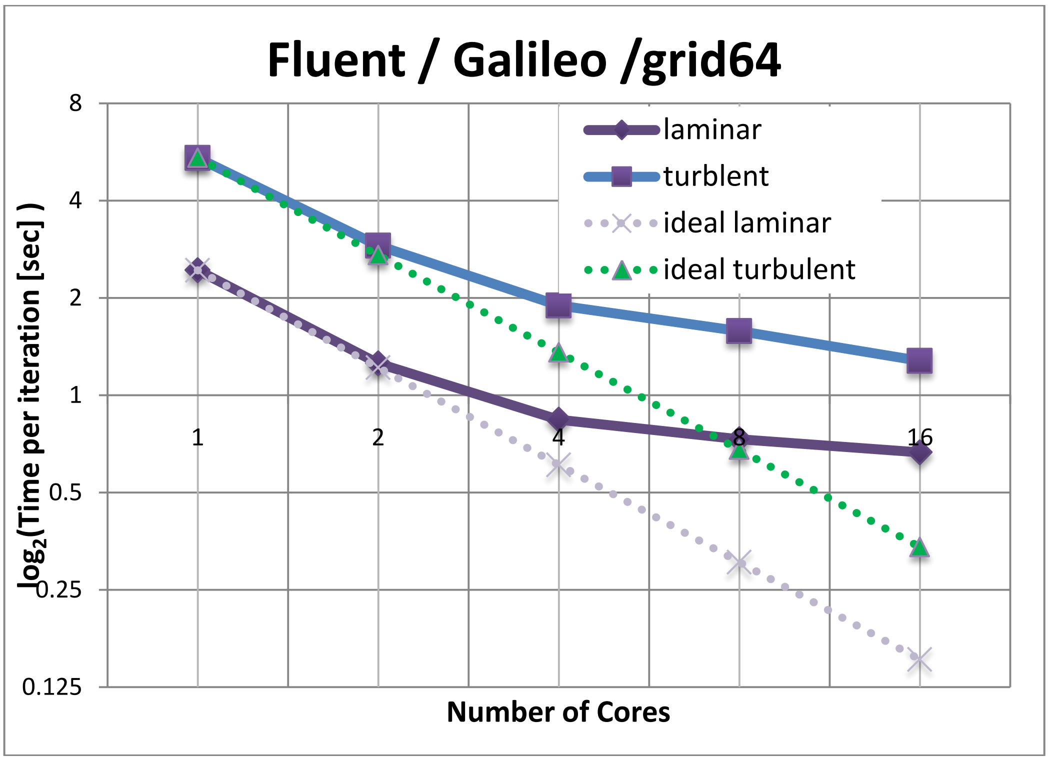


Figure 34 Turnaround time of commercial code on cluster for coarse grid

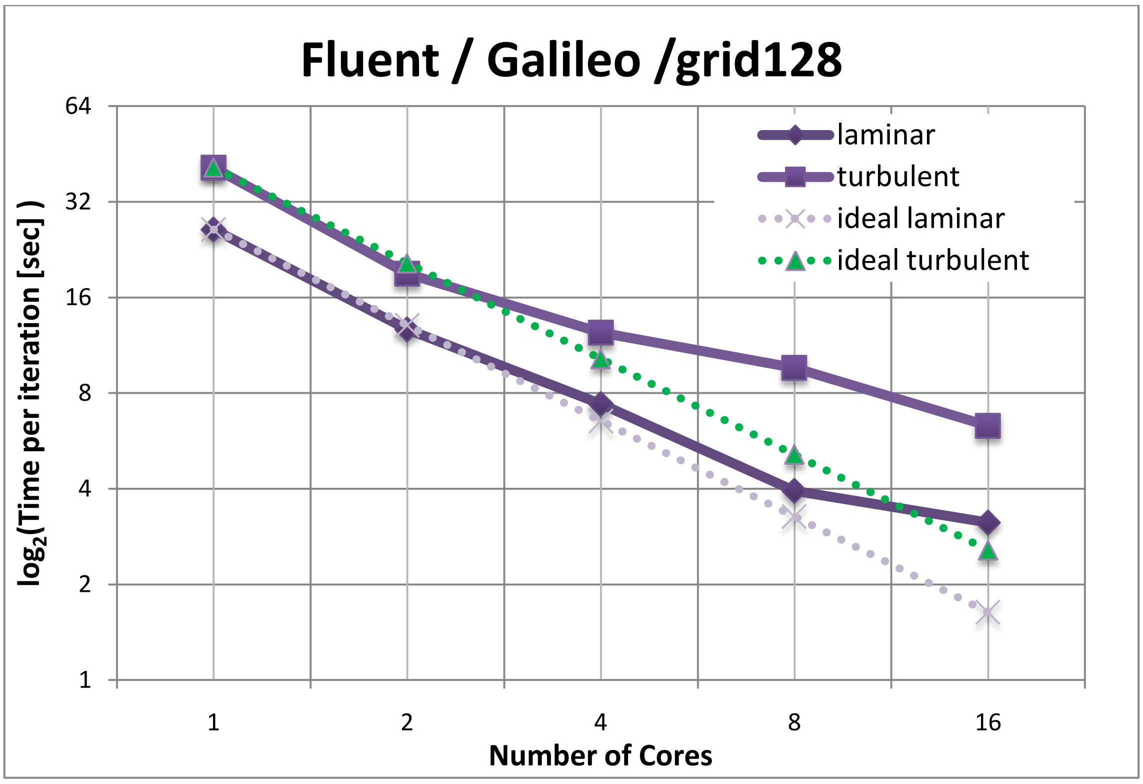


Figure 35 Turnaround time of commercial code on cluster for fine grid

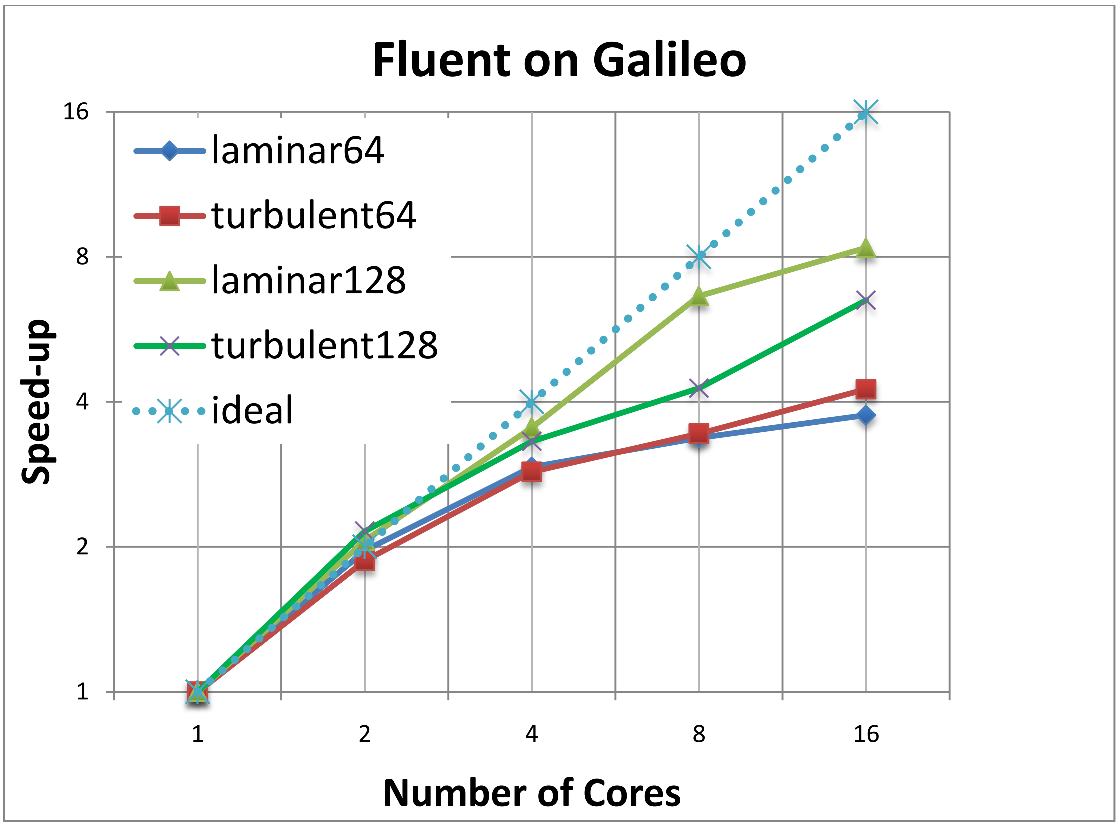


Figure 36 Speed-up of all cases for non-commercial code on linux cluster

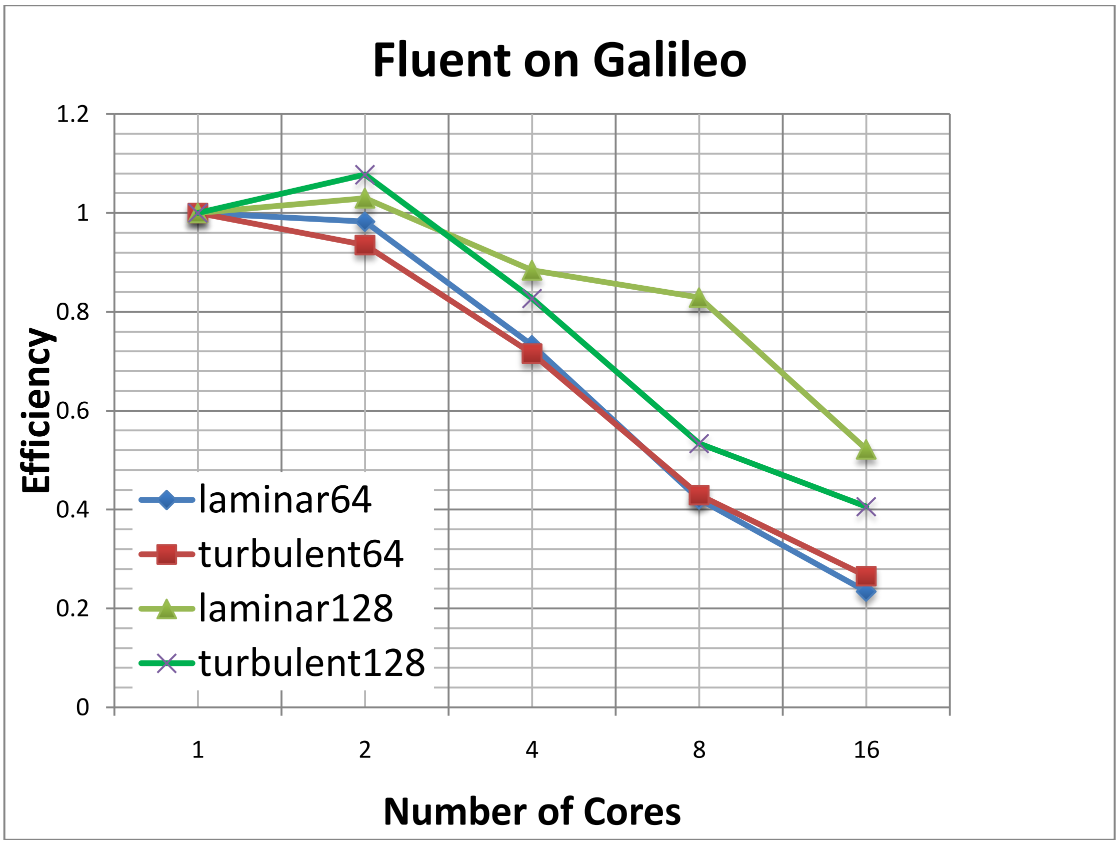


Figure 37 Efficiency of all cases for commercial code on linux cluster

4.2.4 Non-Commercial – Galileo

The most diverse behaviour was obtained in this section. For 64 gridsize similar behaviour is depicted in Figure 38; up to 2 cores the performance of the simulations is alike. Subsequently, turbulent flow strays slowly but the laminar one advances super-linearly up to 8 cores and in the end its performance drops. For fine grid roughly similar patterns in curves are depicted in Figure 39. At the start laminar and turbulent flow simulation curves advance in a linear and slightly super- linear way respectively up to two cores. Then turnaround time augments constantly. The ratio equals to 1:40, which is the highest among all the simulations.

In general, curves of efficiency stay very close while they end up at 33% efficiency for 16 cores. A severe drop of performance is due to interconnection which reached its limits for the specific workload.

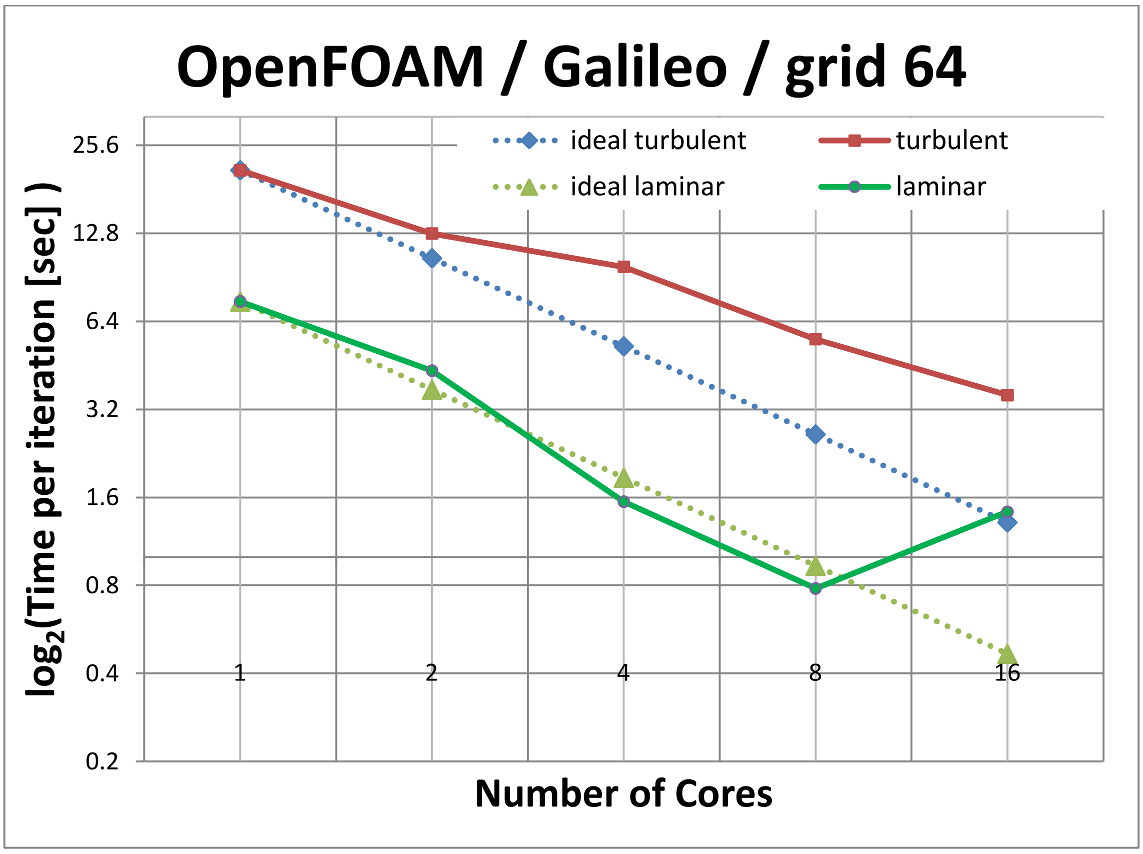


Figure 38 Turnaround time of non-commercial code on linux cluster for coarse grid

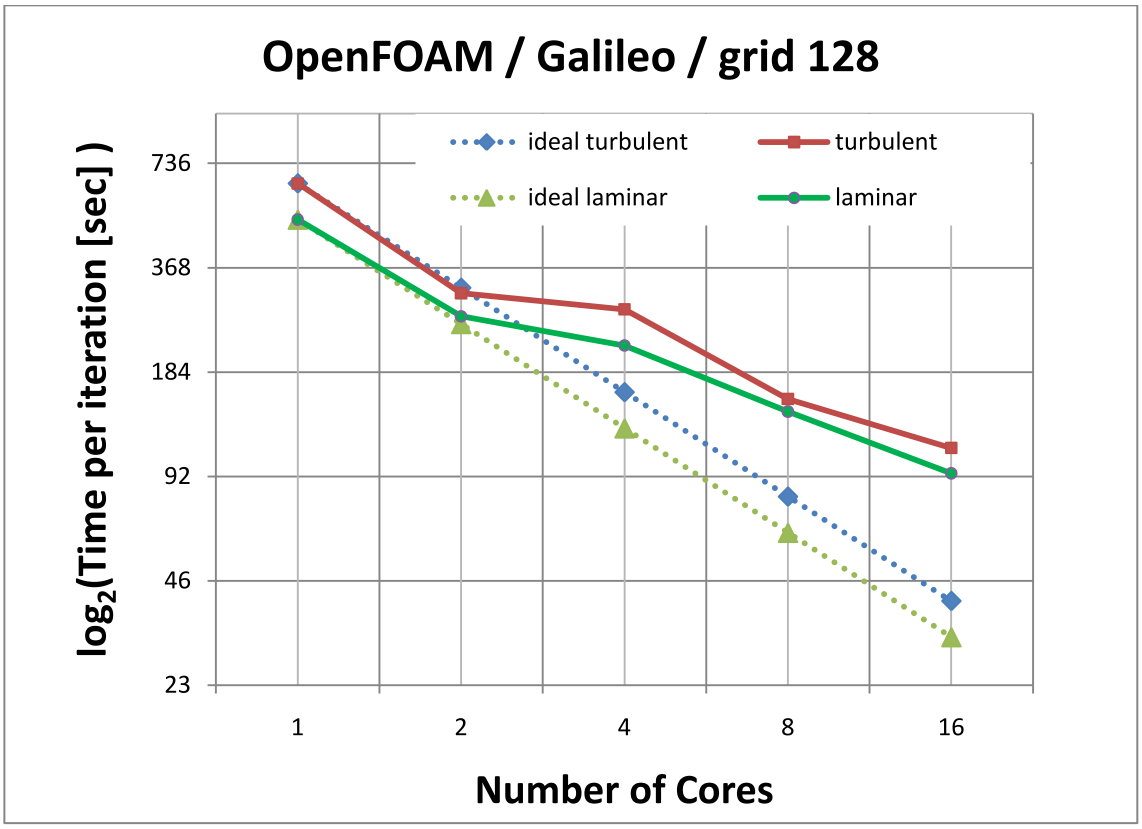


Figure 39 Turnaround time of non-commercial code on linux cluster for coarse grid

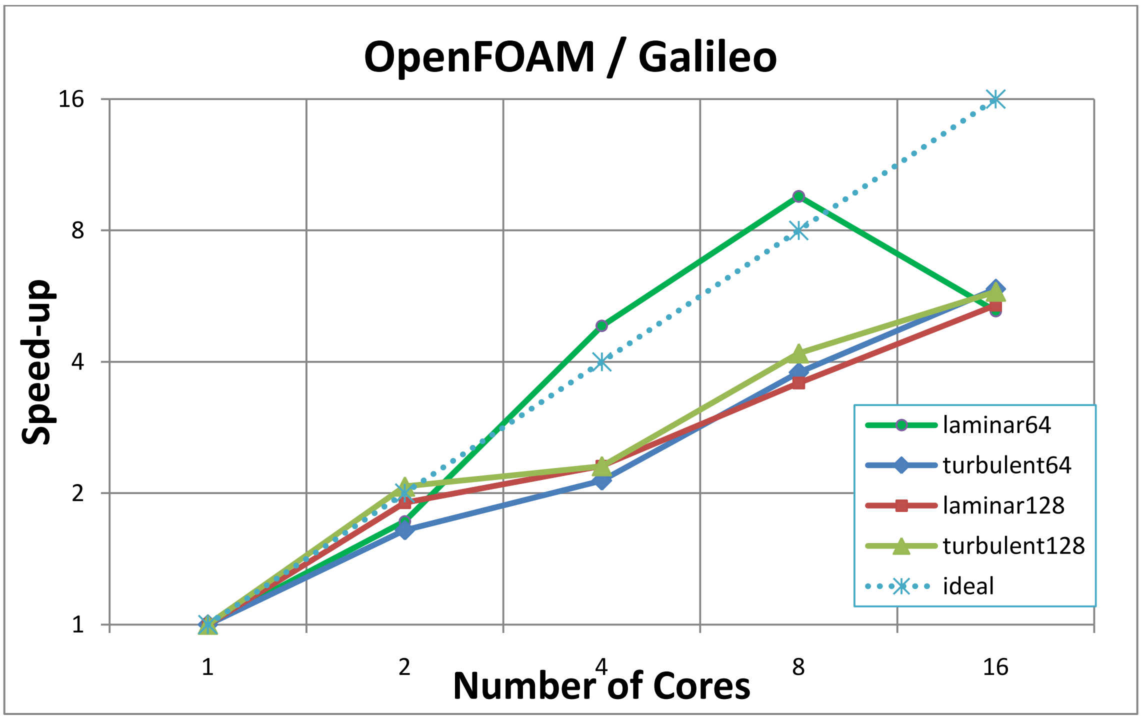


Figure 40 Speed-up of all cases for non-commercial code on linux cluster

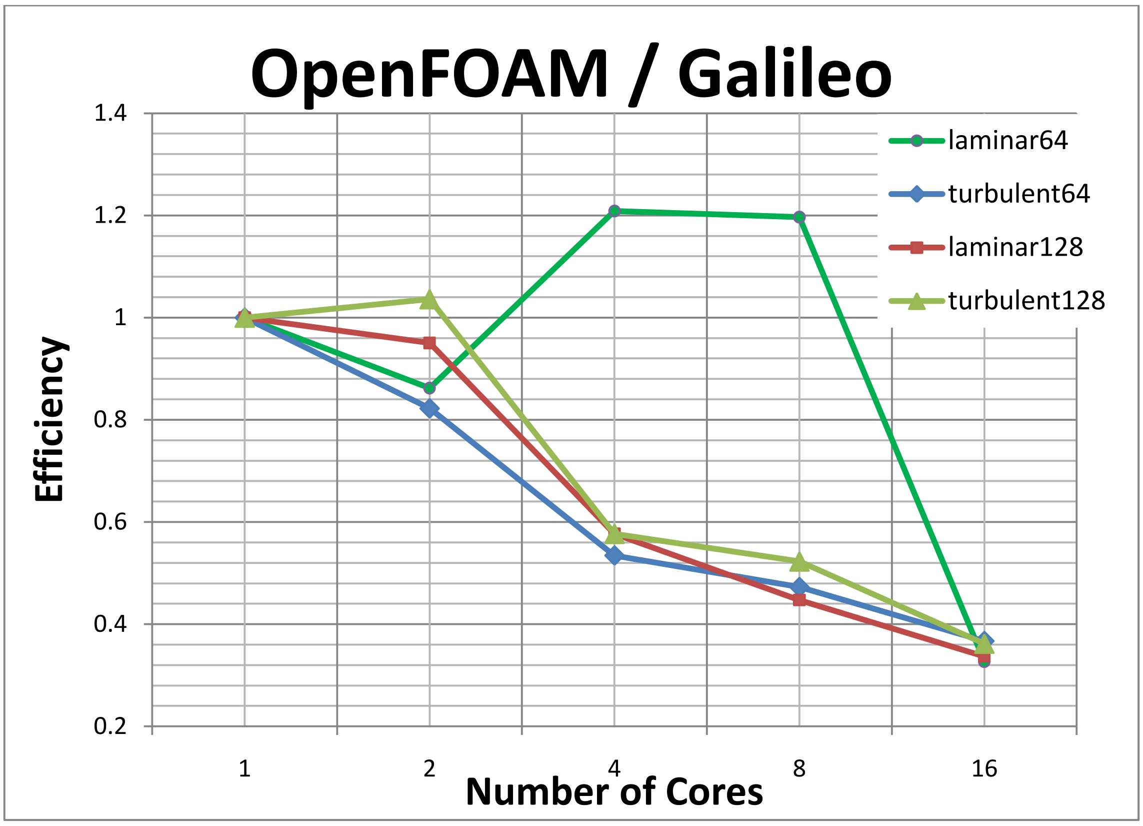


FIgure 41 Efficiency for all cases for non-commercial code on linux cluster

4.2.5 Outline

The final resolution of our study is presented by studying Figure 42 through to Figure 45. Following the order of the pictures it is evident that both CFD solvers are infrastructure dependent. The slightest change in software settings or resources affects overall behaviour substantially. Hence, using the appropriate software with the specific amounts of resources makes a difference. However sometimes system specifications could improve the performance because they better suit the philosophy of the software's requirements. The former statement manipulates cache size more efficiently, whereas for the latter MPI aggregated function reduce significantly the number of unnecessary MPI calls, which consume more time. Considering the above figure of turnaround time FLUENT is faster than OpenFOAM in delivering results for the same input. In terms of software, one can say that FLUENT makes better use of processing power and interconnection. The statements below are added to enhance further comparisons:

FLUENT presents more stable and scalable behaviour on both infrastructures. The gap between the quickest and the slowest run is the smallest one recorded. Although FLUENT is nearly linear up to 2 cores, beyond that point, almost all the cases present x8 speed up for 16 cores; while on ASTRAL roughly identical behaviour was obtained within the whole range of cores. Starting from 1 core until 8 Galileo accelerates the fine grid of laminar flow the most. While running on Galileo its variation was lower compared to OpenFOAM 6Appendix D

OpenFOAM is shown to be a very capable code. In the unique case of fine grid in laminar flow above 2 cores and up to 8 it presents superlinear performance running on Galileo, while the remaining cases on the same system produce very close results. It is the same case for 16 cores where the turnaround time is higher than that of 8 cores which indicates that the specific combination reached the bottom for 8 cores and is less efficient beyond that. In multi-node systems (since up to 4 cores are executed on the same node) it out-performs FLUENT. This is partially due to OpenMPI's better collaboration with hardware on Galileo.

On the other hand, ASTRAL is very suitable for FLUENT as it presents the same speed up for every case. The biggest gap in our study in terms of performance is highlighted in Figure 44According to the gap between coarse laminar on FLUENT over fine laminar on OpenFOAM, the former is at least 4 times more effective. Laminar flow on OpenFOAM is independent of grid size, while for turbulent flow, it is not only slower but it achieves less than 2 speed-up regardless the amount of cores. It is estimated that if more resources were available (probably about 64 cores) we would achieve deceleration comparable to the serial version.

However, Galileo seems to be set appropriately for both software packages. The curves are more uniform and even for 16 cores a speed-up of near 4 times is obtained. Apart from coarse laminar flow on OpenFOAM (which achieved super-linearity as stated above) and fine laminar flow on FLUENT, the rest of the cases lie between a very close interval, thus highlighting a more stable behaviour.

Taking into account statistics in 6Appendix D the average variation (1.53%) of all cases run on both software and on both infrastructure induce the eligibility of the above in every engineering application.

Regarding the infrastructures, on ASTRAL OpenFOAM is slower than FLUENT. It is noteworthy that OpenFOAM's quickest case on ASTRAL (8 cores – turbulent – 64gridsize) is slower than the worst case of FLUENT(1 core – turbulent) for the 64 grid size. According to Figure 45, on Galileo, two sets (one from each software for laminar flow) of simulations are close to ideal performance while the rest are bunched in closed vicinity away from it.

On the one hand, FLUENT runs near ideally on both infrastructures and parallel libraries. Similar patterns of performance were demonstrated for FLUENT on both infrastructures. For small number of cores they advance tightly, while they remain relatively close to ideal performance if more cores are employed. For 16 cores FLUENT's speed-up lies between 3.7 and 10

On the other hand OpenFOAM presents irregularities. In a few cases its super-linearity was achieved because of ASTRAL's IB interconnection; these were simulations of laminar and turbulent for 64 gridsize and laminar for 128 gridsize. In addition, its performance curves fork within the range of 1.5 and 6 speed-up. Despite the super-linearity of OF on ASTRAL for 2 cores and considering that up to 4 jobs run in parallel on every node, it is clear that ASTRAL's computational nodes are inappropriate for running OF because the efficiency is too poor for more than 2 cores. That is due to solvers' algorithmic implementation and MPI libraries (OpenMPI) which operate with OpenFOAM.

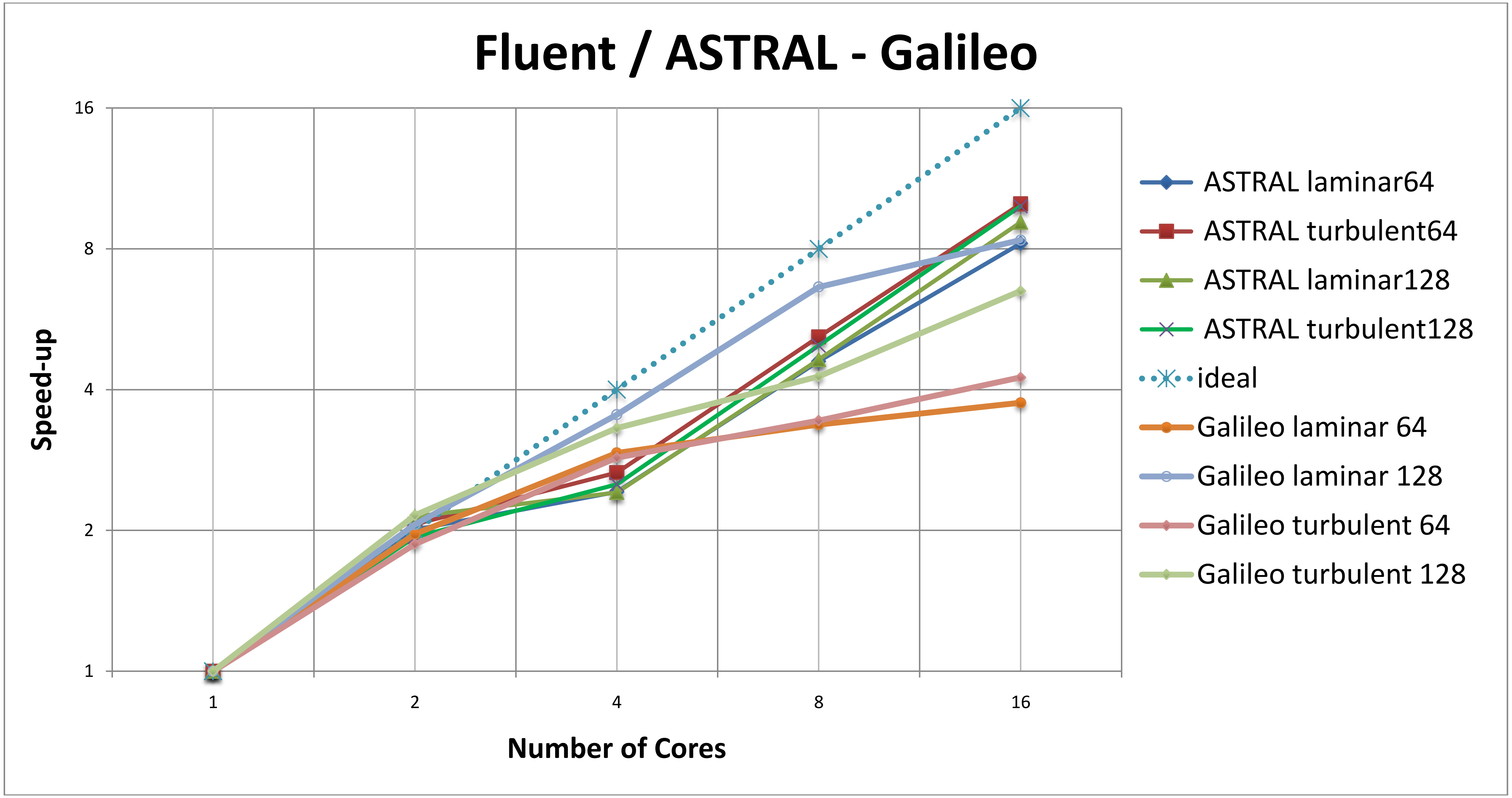


Figure 42 Comparison of commercial code on both infrastructures

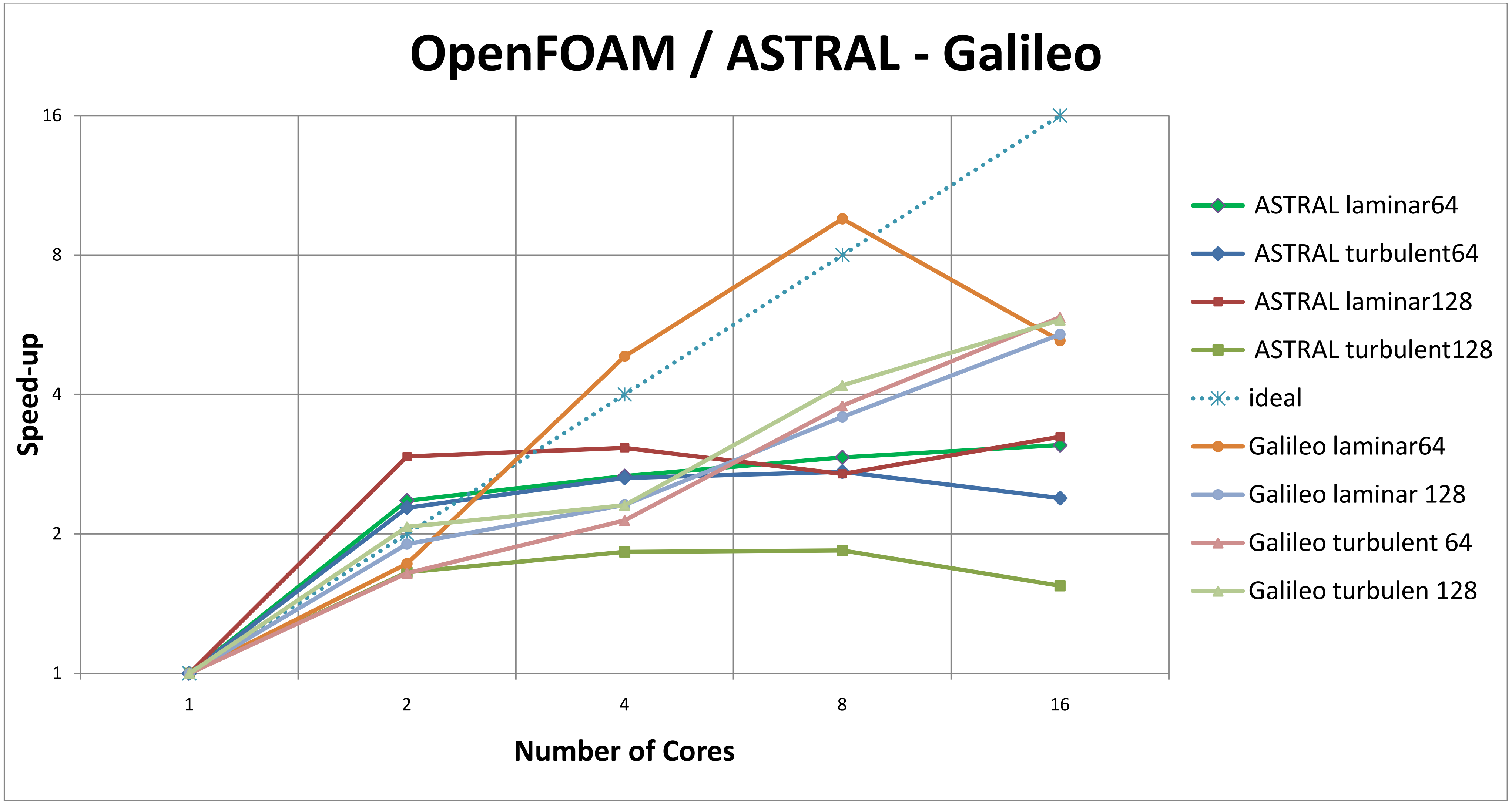


Figure 43 Comparison of non-commercial code on both infrastructures

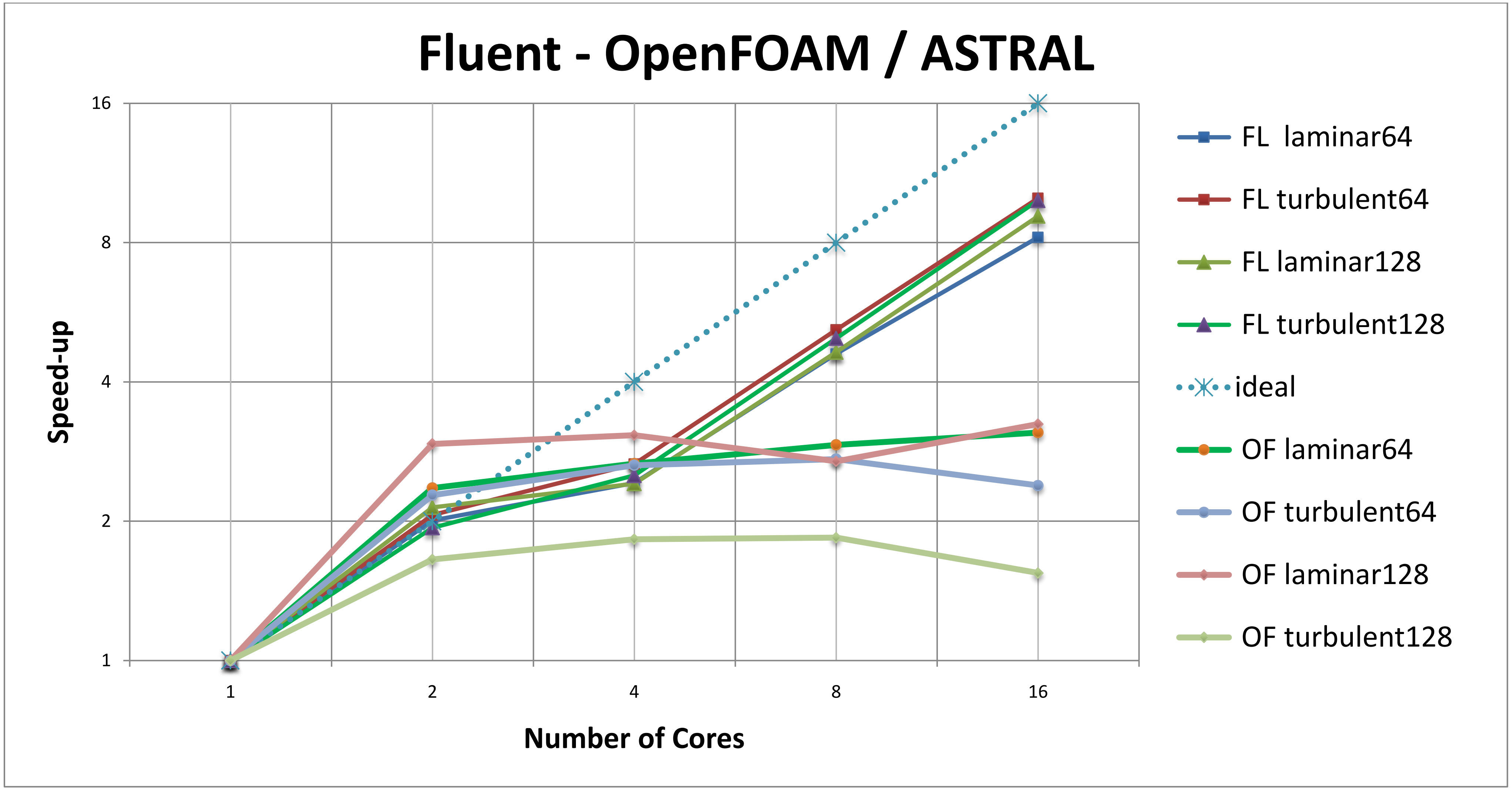


Figure 44 Comparison of both codes on supercomputer

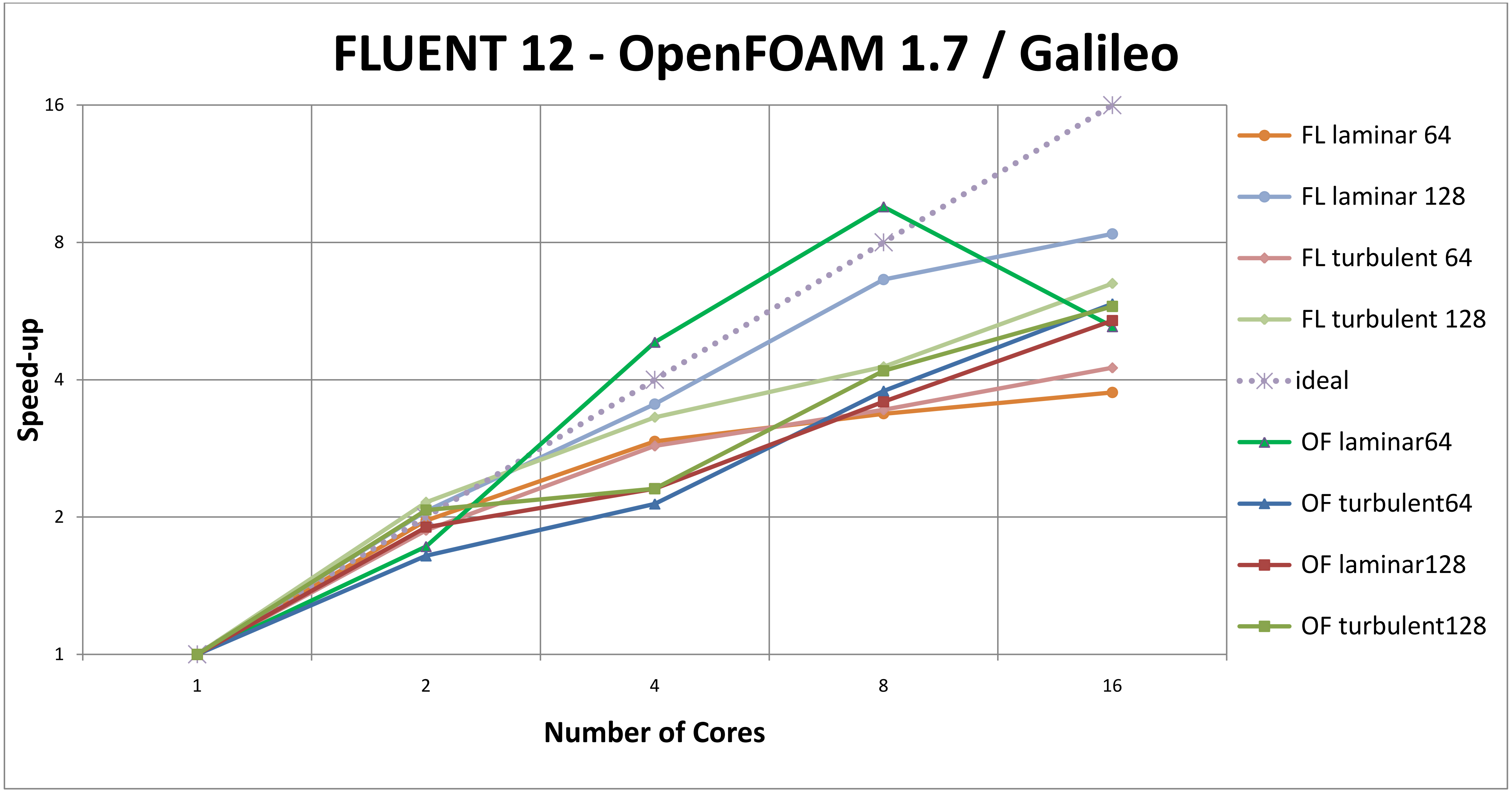


Figure 45 Comparison of both codes on cluster

5 Conclusions

In the first part of the study a CFD case is described briefly. In turn, it was used as a benchmark for performance investigation on two different cluster architectures analyzing their merits. Two different grid size geometries were used upon two different types of flow using appropriate modelling. Simulations were conducted for a constant number of iterations per case and performance plots were illustrated. Apart from one case that had hit the ceiling of performance, the rest tend to achieve higher levels of scalability for larger infrastructures. Thus, even lower turnaround times could be achieved

A fair comparison of simulation runtimes of a commercial and non-commercial CFD solvers was presented. Up to 16 nodes were employed in order to demonstrate scalability on two different clusters. On the simulations two grid sizes (64 and 128 cells per edge) were used for a LDC case, run on IB and GigE interconnection. Repeated runs presented very small variation which means that in spite of the use of a fraction of the overall capacity of clusters, the results were not affected. In general, FLUENT has proven steadier and quicker as the variation of timings is lower and the execution speed higher. OpenFOAM is very competitive as it scales well in every case. However, it needs more time in order to produce exactly the same velocity curves as FLUENT does. Timings are not affected only by interconnections because the order of magnitudes of the employed processors are so small, that GigE does not reach its limits.

The importance of understanding the problem and the respective computational requirements is crucial and could lead to better value per CPU-cycle use. Comparing of equivalent solvers is a very challenging task and there are many requirements to be met regarding the verity and accuracy of the results. Crashing and failing of resources and software respectively have a negative affect on data acquisition. In general FLUENT coped quicker in the most of the cases, while OpenFOAM for specific cases achieved even super-linear behaviour. Because of FLUENT's high requirements in hardware (installation of IB, dedicated nodes, etc) OpenFOAM is more appropriate for linux clusters.

This study shed light on important aspects regarding the performance and scalability of CFD software packages. All the test cases were described thoroughly and possible solutions addressing the weaknesses of both were stated in each section in respect to their specific needs. FLUENT runs near ideally on both infrastructures and parallel libraries At the same time, OpenFOAM presents extremes in overall behaviour with a few super-linear instances and many slow ones. Especially on ASTRAL, the performance gap between OpenFOAM and FLUENT is very large. However the variation in performance for FLUENT is larger that OpenFOAM, which points to OpenFOAM being more reliable, even if its average turnaround time is higher.

In terms of infrastructure, ASTRAL outperforms Galileo running FLUENT on every instance. Nonetheless, coarse grid instances of OpenFOAM run better on Galileo, whereas for fine grid cases ASTRAL again carries out simulations more effectively. The best performance for Galileo was noted for laminar case of 64 gridsize on OpenFOAM (4 and 8 cores), whereas ASTRAL's best achievement was for the same scenario as before, but with 128 gridsize (2 cores).

6 Future Work

The use of more mathematical models could induce even wider abstractions. Although RANS coped well with LDC, LES has also proven reliable [22] for further tests, while DNS with relatively small number of cells could be used for a complete insight. In terms of cost per usage per cores OpenFOAM is recommended for even larger number of cores on the available infrastructure until the maximum performance is reached. Additionally, the use of more computational resources and simulations in various speeds and fluids will offer deeper insight to future users/ researchers, thus facilitating the accomplishment of their projects. The use of other pressure-velocity coupling algorithms is advisable. Despite the philosophy of reusable and general applications, specific projects have extreme time pressure, which might imply the write up of new solvers that serve specific needs exploiting dedicated infrastructure. . The more comparison simulations per machine and per core are conducted, the more reason for confidence in the final evaluation.

In CFD-HPC field there is also another competitor which was not mentioned. The measurements illustrated above have been carried out on GP processors of CISC architecture. That is the most common architecture for HPC infrastructures because they can serve various types of jobs and applications. Nonetheless, competitors of RISC architecture, that of GP graphics processing unit (GPU) have showed up and demonstrated marginal performance on similar cases. GPUs can handle data in a quicker way and can be integrated with bigger cache size because of the nature of their architecture. Even at early stages, the obtained speed-up outperforms even high end CPUs-[31], [52], [92]. This level of performance combined with low cost per computational power has led to a full transfer from CFD codes written for CPUs to GPU equivalents-[36]. They offer an alternative and very powerful compilation for HPC that could address problems of even greater magnitude. In 2D LDC case, GPUs achieve significantly better performance over high end GP processors. Some resources arrangements were composed exclusively of GPUs and some of a hybrid mixture of both CPUs and GPUs. It is worth commenting that a fraction of overall performance of the second supercomputer in the top500 list (June 2010) is based on GP-GPU resources. In theory, this supercomputer has the highest peak performance. Therefore, the first signs are very promising, opening the doors to a new way of performing CFD simulations which could achieve higher levels of parallelism, while it will attract higher interest by scientific groups, too.

Ultimately, because large number of embedded cores is integrated within graphics boards every year, the future GP-GPUs may obtain an even higher rate of FLOPS in comparison with traditional clusters and any issues they might cause. Therefore, it is more likely that in short term applications which implement Direct Numerical Simulations (DNS) in time-efficient intervals satisfactory accuracy or upgraded models/hybrids of dedicated CPUs for CFD applications will be achieved.

REFERENCES

[1] , *GIGABIT ETHERNET STANDARDS*(2010), available at: http://www.gigabit-ethernet.org/ (accessed July 2010).

[2] , *HPCwire Market Watch*(2010), available at: http://markets.hpcwire.com/taborcomm?Account=hpcwire&_Match=Page&_Match=Ticker&Ticker=$HPCWIRE-MW&Page=Quote (accessed July 2010).

[3] , *Infiniband*(2010), available at: http://www.infinibandta.org/ (accessed May 2010).

[4] , *Interconnect Family share for 06/2010*(2010), available at: http://www.top500.org/stats/list/35/connfam (accessed July 2010).

[5] , *Myricom*(2010), available at: http://www.myri.com/ (accessed July 2010).

[6] , *Netlib*(2010), available at: http://www.netlib.org/ (accessed July 2010).

[7] , *Number of Processors share for 06/2010*(2010), available at: http://www.top500.org/stats/list/35/procclass (accessed July 2010).

[8] , *Operating system Family share for 06/2010*(2010), available at: http://www.top500.org/stats/list/35/osfam (accessed July 2010).

[9] , *Performance Development*(2010), available at: http://www.top500.org/lists/2010/06/performance_development (accessed July 2010).

[10] , *Portable, Extensible Toolkit for Scientific Computation*(2010), available at: http://www.mcs.anl.gov/petsc/petsc-as/ (accessed July 2010).

[11] , *Segments share for 06/2010*(2010), available at: http://www.top500.org/stats/list/35/segments (accessed July 2010).

[12] , *TOP500 List - November 2007 (301-400), #343* (2007), available at: http://www.top500.org/list/2007/11/400 (accessed July 2010).

[13] ADVENTURE Project Team (2009)*, ADVENTURE Project*, available at: http://adventure.sys.t.u-tokyo.ac.jp/ (accessed July 2010).

[14] Agarwal, R. C., Alpern, B., Carter, L., Gustavson, F. G., Klepacki, D. J., Lawrence, R. and Zubair, M. (1995), "High-performance parallel implementations of the NAS kernel benchmarks on the IBM SP2", *IBM Systems Journal,* vol. 34, no. 2, pp. 263-272.

[15] Ambrosino, F., Raia, S., Funel, A., Podda, S. and Migliori, S. (2008), *Cross checking OpenFOAM and Fluent results of CFD simulationsin ENEA-GRID environment*, available at: http://www.cresco.enea.it/Documenti/web/presentazioni/Poster_E-Science_2008/09.pdf (accessed July 2010).

[16] Andersen, C. and Nielsen, N. E. L. (2008), Numerical investigation of a BFR using OpenFOAM, , AALBORG University - Institute of Energy Technology, AALBORG University - Institute of Energy Technology.

[17] ANSYS, I. (2010), *Fluent*, available at: http://www.fluent.com/ (accessed July 2010).

[18] ANSYS, I., (2009), *Fluent 12.0 User's Guide*.

[19] Axner, L., Bernsdorf, J., Zeiser, T., Lammers, P., Linxweiler, J. and Hoekstra, A. G. (2008), "Performance evaluation of a parallel sparse lattice Boltzmann solver", *Journal of Computational Physics,* vol. 227, no. 10, pp. 4895-4911.

[20] B., E. and S., P. (2004), "Application of WENO (Weighted Essentially Non-oscillatory) Approach to Navier-Stokes Computations", *International Journal of Computational Fluid Dynamics,* vol. 18, pp. 289-293.

[21] Berger, M. J., Aftosmis, M. J., Marshall, D. D. and Murman, S. M. (2005), "Performance of a new CFD flow solver using a hybrid programming paradigm", *Journal of Parallel and Distributed Computing,* vol. 65, no. 4, pp. 414-423.

[22] Bouffanais, R., Deville, M. O. and Leriche, E. (2007), "Large-eddy simulation of the flow in a lid-driven cubical cavity", *Physics of Fluids,* vol. 19, no. 5.

[23] Cela, J., Alfonso, J. and Labarta , J., (1996), *PLS: A parallel linear solvers library for Domain Decomposition methods*.

[24] cfd-online users (December 2008), *Lid-driven cavity problem*, available at: http://www.cfd-online.com/Wiki/Lid-driven_cavity_problem (accessed June 2010).

[25] Chao, J., Haselbacher, A. and Balachandar, S. (2009), "A massively parallel multi-block hybrid compact-WENO scheme for compressible flows", *Journal of Computational Physics,* vol. 228, no. 19, pp. 7473-7491.

[26] David, K., Dinesh, K. and Barry, S. (2010), *Prospects for CFD on Petaflops Systems*, available at: http://www.ima.umn.edu/preprints/OCT97/1514.pdf (accessed July 2010).

[27] Dell Ltd, (2008), *Performance that Scales*, Dell Ltd.

[28] Dong, S. and Karniadakis, G. E. (2004), "Multilevel parallelization models for high-order CFD", pp. 6775.

[29] Drikakis, D., Hahn, M., Thornber, B. and Shapiro, E. (2010), "High-fidelity CFD simulations of instabilities, transition and turbulence using high-order methods and parallel computing", in Biswas, R. (ed.) *Parallel Computational Fluid Dynamics: Recent Advances and Future Directions,* paper ed, DEStech Publications, California, pp. 19.

[30] Ehrig, M. (2010)*, CFX PerformanceonMultiple Platforms Comparing different Architectures*, available at: http://www.cadfem.de/fileadmin/files/cadfem.de/ch_files/pdf/Presseberichte/Schulungen/CFX_Performance_on_Multiple_Platforms.pdf (accessed July 2010).

[31] Elsen, E., LeGresley, P. and Darve, E. (2008), "Large calculation of the flow over a hypersonic vehicle using a GPU", *Journal of Computational Physics,* vol. 227, no. 24, pp. 10148-10161.

[32] Field, D. and Mize, D. (2010)*, Comparing the Performance of the FLUENT application on Quad-core Processors vs. Dual-core Processors*, available at: http://www.hp.com/techservers/hpccn/hpccollaboration/ADCatalyst/downloads/Multi-coreandFLUENT04AA1-6093ENW.pdf (accessed July 2010).

[33] Fisher, M. S., Mani, M. and Stookesberry, D. (2001), "Parallel processing with the Wind CFD code at Boeing", *Parallel Computing,* vol. 27, no. 4, pp. 441-456.

[34] Geimer, M., Wolf, F., Wylie, B. J. N. and Mohr, B. (2009), "A scalable tool architecture for diagnosing wait states in massively parallel applications", *Parallel Computing,* vol. 35, no. 7, pp. 375-388.

[35] Ghia, U., Ghia, K. N. and Shin, C. T. (1982), "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method", *Journal of Computational Physics,* vol. 48, no. 3, pp. 387-411.

[36] Griebel, M. and Zaspel, P. (2010), "A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations", *Computer Science - Research and Development,* vol. 25, no. 1-2, pp. 65-73.

[37] Hamid, N. A. W. A. and Coddington, P. (2007), "Averages, distributions and scalability of MPI communication times for Ethernet and Myrinet networks", pp. 269.

[38] Harlow, F. H. and Welch, J. E. (1965), "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface", *Physics of Fluids,* vol. 8, no. 12, pp. 2182-2189.

[39] Hauser, T. (2004), "Parallel I/O for the CGNS system", pp. 6685.

[40] Hayashibara, S. and Ashraf, A. (2006), "Development of PC-cluster for undergraduate aerospace engineering software applications", Vol. 5, pp. 3338.

[41] HDF5 Group (2009)*, What is HDF5*, available at: http://www.hdfgroup.org/HDF5/whatishdf5.html (accessed July 2010).

[42] Hewlett-Packard Development Company (June 2007)*, HP-MPI User's Guide 10th edition*, available at: http://docs.hp.com/en/B6060-96022/index.html (accessed May 2010).

[43] HPC Advisory Council (2010)*, ANSYS FLUENT Performance Benchmark and Profiling*, available at: http://www.hpcadvisorycouncil.com/pdf/Fluent_Analysis.pdf (accessed July 2010).

[44] HPC Advisory Council (2010)*, Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing*, available at: http://www.hpcadvisorycouncil.com/pdf/IB_and_10GigE_in_HPC.pdf (accessed July 2010).

[45] HPC Advisory Council (2010)*, STAR-CCM+ and STAR-CD Performance Benchmark and Profiling*, available at: http://www.hpcadvisorycouncil.com/pdf/CD_adapco_applications.pdf (accessed July 2010).

[46] HPC Advisory Council (2009)*, ANSYS CFX Performance Benchmark and Profiling*, available at: http://www.google.com/url?sa=t&source=web&cd=1&ved=0CBQQFjAA&url=http%3A%2F%2Fwww.hpcadvisorycouncil.com%2Fpdf%2FCFX_Analysis.pdf&ei=E2V3TISKHeLX4wbO0pCqBg&usg=AFQjCNHdc49DN1jzv0Hd_q4C6BAjs8J2zw&sig2=lazbsTI-ld0cX4qLDFXXFQ (accessed June 2010).

[47] Hutchings, B., Browell, R. and Alavilli, P. (2009)*, The Need for Speed*, available at: http://www.sun.com/third-party/global/ansys/ANSYS_AdvantageHPC.pdf (accessed 5th May 2010).

[48] Huwaldt, J.A., (2010), *Plot Digitizer*, http://plotdigitizer.sourceforge.net/.

[49] Johan, Z., Mathur, K. K., Johnsson, S. L. and Hughes, T. J. R. (1994), "Scalability of finite element applications on distributed-memory parallel

computers", *Computer Methods in Applied Mechanics and Engineering,* vol. 119, no. 1-2, pp. 61-72.

[50] Kamal, H., Penoff, B. and Wagner, A. (2005), "SCTP versus TCP for MPI", *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing,* IEEE Computer Society, Washington, DC, USA, pp. 30.

[51] Karypis, G. (2008), *ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering*, available at: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview (accessed July 2010).

[52] Kelly, J. (2010), "GPU-accelerated simulation of two-phase incompressible fluid flow using a level-set method for interface capturing", Vol. 9, pp. 2221.

[53] Kimpe, D., Lani, A., Quintino, T., Vandewalle, S., Poedts, S. and Deconinck, H., (2007), *A study of real world I/O performance in parallel scientific computing*.

[54] Kneer, A., Schreck, E., Hebenstreit, M. and Goszler, A. (2010), *Industrial mixed OpenMP / Mpi CFD-application for calculations of free-surface flows*, available at: http://www.post.ecs.soton.ac.uk/Publications/wompat-b.pdf (accessed July 2010).

[55] Koesterke, L. (2010), *Scalability Performance*, available at: http://taccspringschool.ist.utl.pt/SpringSchool/Agenda_files/T09_Scalability.pdf (accessed July 2010).

[56] Koseff, J. R. and Street, R. L. (1984), "The Lid-Driven Cavity Flow: A Synthesis of Qualitative and Quantitative Observations", *Journal of Fluids Engineering,* vol. 106, no. 4, pp. 390-398.

[57] Koseff, J. R. and Street, R. L. (1984), "The lid-driven cavity flow: a synthesis of qualitative and quantitative observations.", *TRANS.ASME J.FLUIDS ENGNG.,* vol. 106, no. 4 , Dec. 1984, pp. 390-398.

[58] Koseff, J. R. and Street, R. L. (1984), "ON END WELL EFFECTS IN A LID-DRIVEN CAVITY FLOW.", *Journal of Fluids Engineering, Transactions of the ASME,* vol. 106, no. 4, pp. 385-389.

[59] Koseff, J. R. and Street, R. L. (1984), "VISUALIZATION STUDIES OF A SHEAR DRIVEN THREE-DIMENSIONAL RECIRCULATING FLOW.", *Journal of Fluids Engineering, Transactions of the ASME,* vol. 106, no. 1, pp. 21-29.

[60] Kremenetsky, M. and Kodiyalam, S. (2010), "Parallel Performance of CFD Applications and the Ubiquitous Need for HPC with High Fidelity,

Multidisciplinary Analysis and Optimization", in Biswas, R. (ed.) *Parallel Computational Fluid Dynamics: Recent Advances and Future Directions,* paper ed, DEStech Publications, California, pp. 584.

[61] Liu, X., Osher, S. and Chan, T. (1994), "Weighted Essentially Non-oscillatory Schemes", *Journal of Computational Physics,* vol. 115, no. 1, pp. 200-212.

[62] Ludewig, T., Papadopoulos, P., Hauser, J., Gollnick, T., Dai, W., Muylaert, J. and Paap, H. (2007), "JUSTGRin A Pure Java HPCC Grid Architecture for Multi-Physics Solvers Performance and efficiency results from various Java solvers", .

[63] Makris, K. and Kyung, D. R. (2006), "On-the-fly kernel updates for high-performance computing clusters", Vol. 2006, .

[64] Mavriplis, D. J., Aftosmis, M. J. and Berger, M. (2007), "High resolution aerospace applications using the NASA Columbia supercomputer", *International Journal of High Performance Computing Applications,* vol. 21, no. 1, pp. 106-126.

[65] Miller, R. H., Strumolo, G. S., Hytopoulos, E., Remondi, S. A. and Watson, S. M. (1999), "High performance computing: analytical aerodynamics for automotive vehicles", *American Society of Mechanical Engineers, Fluids Engineering Division (Publication) FED,* vol. 250, pp. 289-298.

[66] Moreira, J. E., Midkiff, S. P. and Gupta, M. (1998), "A comparison of Java, C/C++, and FORTRAN for numerical computing", *IEEE Antennas & Propagation Magazine,* vol. 40, no. 5, pp. 102-105.

[67] Moreira, J. E., Midkiff, S. P. and Gupta, M. (2000), "From flop to megaflops: Java for technical computing", *ACM Transactions on Programming Languages and Systems,* vol. 22, no. 2, pp. 265-295.

[68] Nagashima, U., Hyugaji, S., Sekiguchi, S., Sato, M. and Hosoya, H. (1995), "An experience with super-linear speedup achieved by parallel computing on a workstation cluster: Parallel calculation of density of states of large scale cyclic polyacenes", *Parallel Computing,* vol. 21, no. 9, pp. 1491-1504.

[69] Naik, V. K. (1995), "Scalable implementation of the NAS parallel benchmark BT on distributed memory systems", *IBM Systems Journal,* vol. 34, no. 2, pp. 273-291.

[70] Nilsson, H., (2007), *Some experiences on the accuracy and parallel performance of OpenFOAM for CFD in water turbines*.

[71] Nilsson, H., (2006), *Evaluation of OpenFOAM for CFD of turbulent flow in water turbines*.

[72] OpenCFD Ltd (2010), *OpenFOAM*, available at: http://www.openfoam.com/ (accessed July 2010).

[73] OpenCFD, L., (2010), *OpenFOAM 1.7 User's Guide*.

[74] Pakalapati, P. D. and Hauser, T. (2005), "Benchmarking parallel I/O performance for computational fluid dynamics applications", pp. 1011.

[75] Peigin, S., Epstein, B., Rubin, T. and Seror, S. (2004), "Parallel Large Scale High Accuracy Navier-Stokes Computations on Distributed Memory Clusters", *Journal of Supercomputing,* vol. 27, no. 1, pp. 49-68.

[76] Prasad , A. and Koseff, J. (1989), "Reynolds number and end-wall effects on a lid-driven cavity flow", *Physics of Fluids A,* vol. 1, pp. 208-218. Feb.

[77] Pringle, G. J. (2010), *3D Lid-driven Cavity Flow*, available at: http://www.hector.ac.uk/cse/distributedcse/reports/openfoam/openfoam/node17.html (accessed July 2010).

[78] RIST (2007), *Geo-FEM*, available at: http://geofem.tokyo.rist.or.jp/ (accessed July 2010).

[79] Saini, S., Jespersen, D. C., Talcott, D., Djomehri, J. and Sandstrom, T. (2008), "Performance comparison of SGI Altix 4700 and SGI Altix 3700 Bx2", .

[80] Schloegel, K., Karypis, G. and Kumar, V. (2004), "Graph partitioning for high-performance scientific simulations", *Computing Reviews,* vol. 45, no. 2, pp. 110.

[81] Schloegel, K., Karypis, G. and Kumar, V. (2002), "Parallel static and dynamic multi-constraint graph partitioning", *Concurrency Computation Practice and Experience,* vol. 14, no. 3, pp. 219-240.

[82] Schloegel, K., Karypis, G. and Kumar, V. (1997), "Multilevel diffusion schemes for repartitioning of adaptive meshes", *Journal of Parallel and Distributed Computing,* vol. 47, no. 2, pp. 109-124.

[83] Schloegel, K., Karypis, G. and Kumar, V. (2003), "Graph partitioning for high-performance scientific simulations", , pp. 491-541.

[84] Silicon Graphics International (2010), *SGI® NUMAlink® Interconnect Fabric*, available at: http://www.sgi.com/products/servers/altix/numalink.html (accessed July 2010).

[85] Sillén, M. (2007), "Performance characteristics of an unstructured CFD-solver on PC-clusters", Vol. 19, pp. 13388.

[86] Sillén, M. (2005), "Evaluation of parallel performance of an unstructured CFD code on PC-clusters", *Journal of Aerospace Computing, Information and Communication,* , no. JAN., pp. 109-119.

[87] Sjogreen, B., Yee, H. C., Ojomehri, J., Lazanoff, A. and Henshaw, W. D. (2010), "Parallel Performance of ADPDIS3DA High Order Multiblock overlapping grid solver for hypersonic turbulence", in Biswas, R. (ed.) *Parallel Computational Fluid Dynamics: Recent Advances and Future Directions,* paper ed, DEStech Publications, California, pp. 550.

[88] Smirni, E., Aydt, R. A., Chien, A. A. and Reed, D. A. (1996), "I/O requirements of scientific applications: an evolutionary view", pp. 49.

[89] Sun Microsystems, I. (2010)*, Optimizing Mechanical Computer-Aided Engineering with Sun Systems Featuring the AMD Opteron™ Processor*, available at: http://www.sun.com/servers/hpc/MCAE_Soln_Brief-05-11-7.pdf (accessed July 2010).

[90] Sunderland, A., Emerson, D. and Allen, C. (2010)*, Parallel performance of a UKAAC helicopter code on HPCx and other large scale facilities*, available at: http://www.docstoc.com/docs/28868230/Parallel-performance-of-a-UKAAC-helicopter-code-on-HPCx-and-other (accessed July 2010).

[91] Tai, C. H., Zhao, Y. and Liew, K. M. (2004), "Parallel computation of unsteady three-dimensional incompressible viscous flow using an unstructured multigrid method", *Computers and Structures,* vol. 82, no. 28 SPEC. ISS., pp. 2425-2436.

[92] Thibault, J. C. and Senocak, I. (2009), "CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows", *47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition (Disc 1),* .

[93] Toby, S., Sanjay, L. and Munira, H. (2010)*, Utilizing Open MPI Middleware for HPC Clusters*, available at: http://www.dell.com/downloads/global/power/ps4q07-20070553-Sebastian.pdf (accessed July 2010).

[94] Top500 (2010)*, Application Area share for 06/2010*, available at: http://www.top500.org/stats/list/35/apparea (accessed July 2010).

[95] Top500 (2010)*, Application Area Share Over Time*, available at: http://www.top500.org/overtime/list/35/apparea (accessed July 2010).

[96] Tuminaro, R., Shadid, J. and Heroux, M. (1998), *Aztec*, available at: http://www.cs.sandia.gov/CRF/aztec1.html (accessed July 2010).

[97] Versteeg, H. and Malalasekera, W. (2007), *An introduction to computational fluid dynamics : the finite volume method,* 2nd ed. ed, Pearson Education Ltd, Harlow England.

[98] Wylie, B. J. N., Geimer, M. and Wolf, F. (2008), "Performance measurement and analysis of large-scale parallel applications on leadership computing systems", *Scientific Programming,* vol. 16, no. 2-3, pp. 167-181.

[99] Yan, J. (1997), "By hand or Not By hand – A case study for Computer Aided Parallelization Tools for CFD Applications", in Arabnia, H. (ed.), pp. 364.

[100] Zunic, Z., Hribersek, M., Skerget, L. and Ravnik, J. (2006), "3D Lid driven cavity flow by mixed boundary and finite element method", in Wesseling, P., Onate, E. and Periaux, J. (eds.), *European Conference on Computational Fluid Dynamics,* 2006, pp. 346.

APPENDICES

Appendix A Simulation

- Grid method
  - Structured
  - Non-uniform
  - Cubic with aspect ratio 1.05
- Decomposition
  - METIS - equal weights (since all the processors are identical)
- Mathematical model
  - RANS
    - k-ω SST
      - Low-Re corrections (only FLUENT)
  - Boundary conditions
    - Lid moving wall on X-axis with fixed velocity
    - Fixed velocity at 0.003215 m/s for laminar flow
    - Fixed velocity at 0.010048 m/s for turbulent flow
- Solver
  - Laminar
    - Re=3200
  - Turbulent
    - Re=10000
- Accuracy
  - Convergence
    - 10^{-3}
- Grid resolution

    - 64^3 cells
    - 128^3 cells
- Infrastructure
    - Clusters
        - ASTRAL
        - Galileo
    - Number of CPUs & memory
    - 1,2,4,8,16 cores
    - file system
        - Lustre (ASTRAL)
        - NFS (Galileo)
    - Interconnection
        - IB for ASTRAL (compiled with HP-MPI)
        - GigE for Galileo

Appendix B FLUENT settings

FLUENT			
		LAMINAR	TURBULENT
Pressure-Velocity Coupling	scheme (solver)	SIMPLE	PISO
	skewness Correction	N/A	
	Neighbor Correction	N/A	
Spatial discretization	Gradient	Green-Gauss Cell Based	Green-Gauss Cell Based
	Pressure	Second Order	Second Order
	Momentum	Second Order Upwind	Second Order Upwind
	Turbulent Kinetic Energy	N/A	Second Order Upwind
	Specific Dissipation Rate	N/A	Second Order Upwind
Under Relaxation factors	Pressure	0.3	0.3
	Density	1	1
	Body Forces	1	1
	Momentum	0.1	0.1
	Turbulent Kinetic Energy	N/A	0.8
	Specific Dissipation Rate	N/A	0.9
	Turbulent Viscosity	N/A	1

▪

Appendix C OpenFOAM settings

OpenFOAM Laminar Flow

```
ddtSchemes
{
   default       Euler;
}
```

```
gradSchemes
{
   default       Gauss linear;
   grad(p)       Gauss linear;
}
```

```
divSchemes
{
   default       none;
   div(phi,U)     Gauss linear;
}
```

```
laplacianSchemes
{
   default       none;
   laplacian(nu,U) Gauss linear corrected;
   laplacian((1|A(U)),p) Gauss linear corrected;
}
```

```
interpolationSchemes
{
   default       linear;
   interpolate(HbyA) linear;
}
```

```
snGradSchemes
{
   default       corrected;
}
```

```
fluxRequired
{
   default       no;
   p             ;
}
```

OpenFOAM Turbulent Flow

```
ddtSchemes
{
   default       Euler;
}
```

```
gradSchemes

{
   default       Gauss linear;

   grad(p)       Gauss linear;

}
```

```
divSchemes

{
   default       none;
   div(phi,U)     Gauss limitedLinearV 1;
   div(phid,p)    Gauss limitedLinear 1;
   div(phiU,p)    Gauss linear;
   div(phi,h)     Gauss limitedLinear 1;
   div(phi,k)     Gauss limitedLinear 1;
   div(phi,epsilon) Gauss limitedLinear 1;
   div(phi,R)     Gauss limitedLinear 1;
   div(phi,omega)  Gauss limitedLinear 1;
   div((rho*R))    Gauss linear;
   div(R)        Gauss linear;
   div(U)        Gauss linear;
   div((muEff*dev2(grad(U).T()))) Gauss linear;
   div((nuEff*dev(grad(U).T()))) Gauss linear;

}
```

```
laplacianSchemes
{
   default       none;
   laplacian(muEff,U) Gauss linear corrected;
   laplacian(mut,U) Gauss linear corrected;
   laplacian(DkEff,k) Gauss linear corrected;
   laplacian(DepsilonEff,epsilon) Gauss linear
corrected;
   laplacian(DREff,R) Gauss linear corrected;
   laplacian(DomegaEff,omega) Gauss linear
corrected;
   laplacian((rho*(1|A(U))),p) Gauss linear
corrected;
   laplacian(alphaEff,h) Gauss linear
corrected;
   laplacian(nuEff,U) Gauss linear corrected;
   laplacian((1|A(U)),p) Gauss linear
corrected;

}
```

```
interpolationSchemes
{
   default       linear;
}
```

```
snGradSchemes
{
   default       corrected;
}
```

```
fluxRequired
{
   default       no;
   p           ;
}
```

Appendix D Statistics about the simulations

Laminar flow - 64 grid

Computational Resources			Number of Processors					Average
			1	2	4	8	16	
ASTRAL	FLUENT	Number of runs	10	10	10	10	10	
		Variation	3.43%	0.69%	1.18%	1.24%	0.18%	1.34%
	OpenFOAM	Number of runs	10	10	10	10	10	
		Variation	3.52%	0.78%	0.46%	0.84%	1.53%	1.43%
Galileo	FLUENT	Number of runs	5	5	5	5	5	
		Variation	0.79%	0.24%	0.22%	0.12%	1.40%	0.55%
	OpenFOAM	Number of runs	5	5	5	5	5	
		Variation	0.07%	1.22%	2.81%	2.01%	0.60%	1.34%

Laminar flow - 128 grid

Computational Resources			Number of Processors					Average
			1	2	4	8	16	
ASTRAL	FLUENT	Number of runs	10	10	10	10	10	
		Variation	3.25%	4.08%	0.12%	2.10%	0.23%	1.96%
	OpenFOAM	Number of runs	10	10	10	10	10	
		Variation	1.77%	1.26%	1.36%	0.27%	2.21%	1.37%
Galileo	FLUENT	Number of runs	5	5	5	5	5	
		Variation	1.35%	2.05%	0.38%	0.16%	0.53%	0.89%
	OpenFOAM	Number of runs	5	5	5	5	5	
		Variation	2.80%	0.39%	1.07%	1.24%	1.08%	1.32%

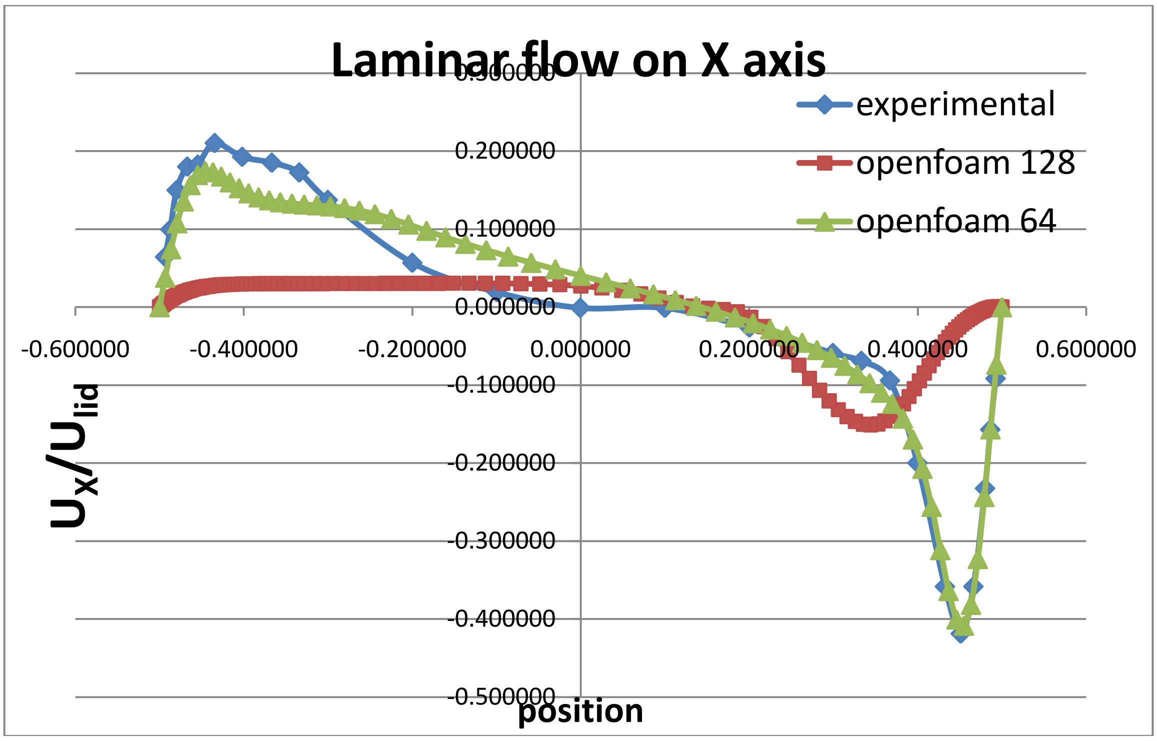
Turbulent flow - 64 grid

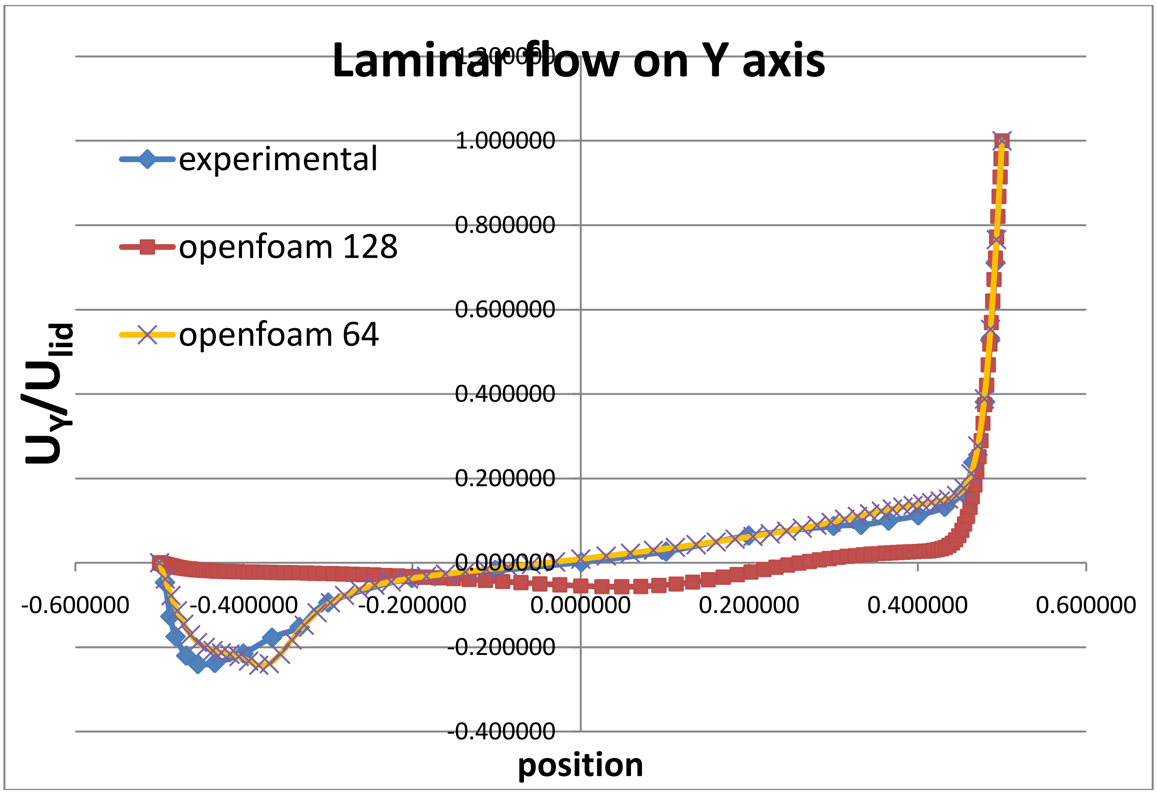
Computational Resources			Number of Processors					Average
			1	2	4	8	16	
ASTRAL	FLUENT	Number of runs	10	10	10	10	10	
		Variation	1.55%	0.42%	2.65%	0.68%	3.62%	1.78%
	OpenFOAM	Number of runs	10	10	10	10	10	
		Variation	4.56%	0.48%	0.64%	1.07%	0.47%	1.44%
Galileo	FLUENT	Number of runs	5	5	5	5	5	
		Variation	0.76%	0.12%	0.10%	3.25%	4.29%	1.70%
	OpenFOAM	Number of runs	5	5	5	5	5	
		Variation	0.22%	2.63%	3.26%	0.68%	1.78%	1.71%

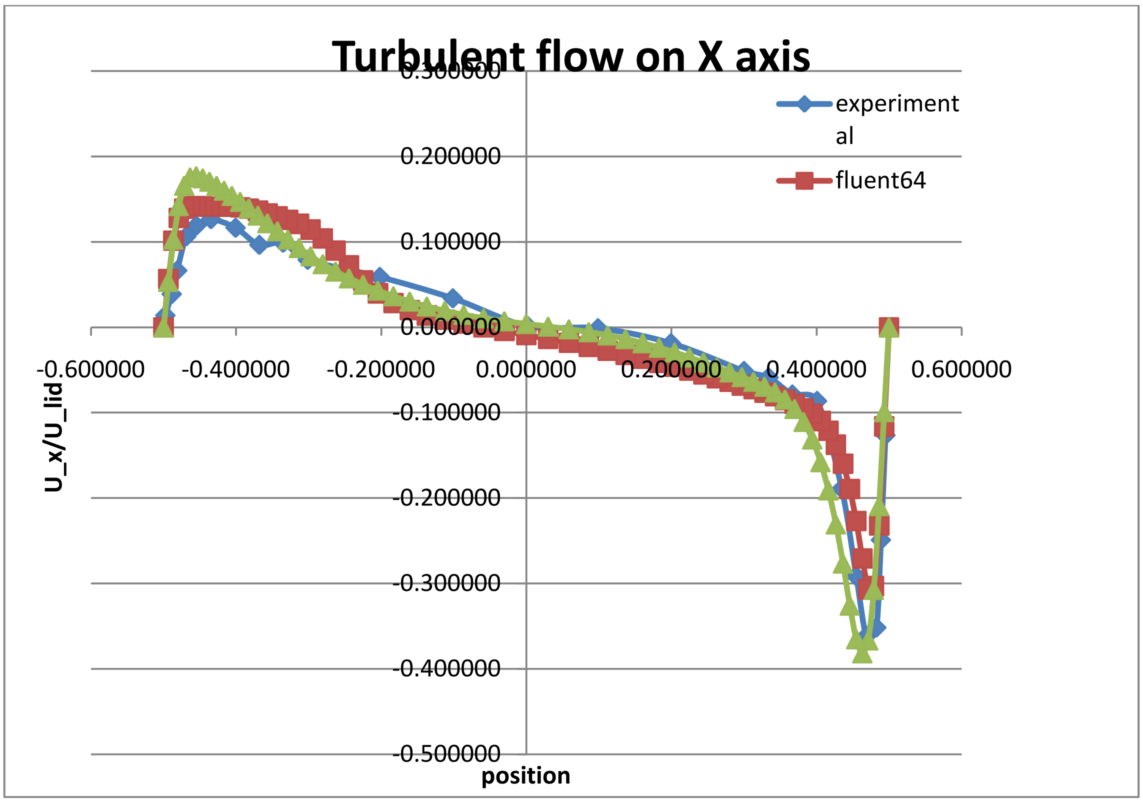
Turbulent flow - 128 grid

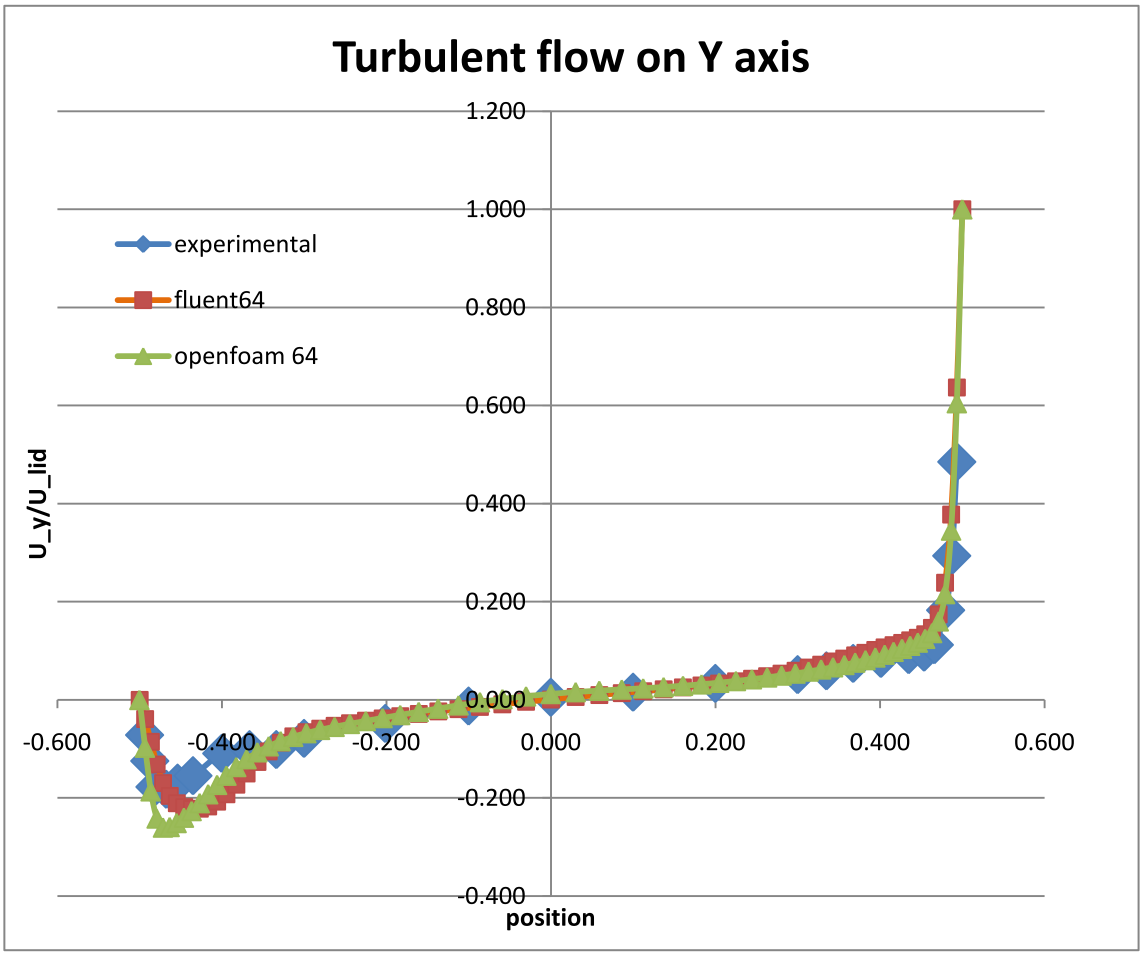
Computational Resources			Number of Processors					Average
			1	2	4	8	16	
ASTRAL	FLUENT	Number of runs	10	10	10	10	10	
		Variation	3.09%	0.30%	0.37%	0.63%	2.00%	1.28%
	OpenFOAM	Number of runs	10	10	10	10	10	
		Variation	2.74%	1.35%	0.26%	0.56%	0.55%	1.09%
Galileo	FLUENT	Number of runs	5	5	5	5	5	
		Variation	2.64%	2.60%	0.94%	0.86%	0.40%	1.49%
	OpenFOAM	Number of runs	5	5	5	5	5	
		Variation	2.66%	4.76%	3.79%	3.57%	4.16%	3.79%

Appendix E OpenFOAM 2D visualization centrelines






Turbulent flow on X axis
0.300000
0.200000
0.100000
0.000000
-0.100000
-0.200000
-0.300000
-0.400000
-0.500000
-0.600000
-0.400000
-0.200000
0.000000
0.200000
0.400000
0.600000
U_x/U_lid
position
experimental
fluent64


Turbulent flow on Y axis
1.200
1.000
0.800
0.600
0.400
0.200
0.000
-0.200
-0.400
-0.600
-0.400
-0.200
0.000
0.200
0.400
0.600
U_y/U_lid
position
experimental
fluent64
openfoam 64