Multi GPU Implementation to Accelerate the CFD Simulation of a 3D Turbo-Machinery Benchmark Using the RapidCFD Library

Daniel Molinero[1(✉)], Sergio Galván[1], Jesús Pacheco[1], and Nicolás Herrera[2]

[1] Universidad Michoacana de San Nicolás de Hidalgo, 58030 Morelia, Mexico
molherd@gmail.com
[2] Instituto Tecnológico de Morelia, 58120 Morelia, Mexico

Abstract. Recently, several research groups have demonstrated significant speedups of scientific computations using General Purpose Graphics Processor Units (GPGPU) as massively-parallel "co-processors" to the Central Processing Unit (CPU). However, the tremendous computational power of GPGPUs has come with a high price since their implementation to Computational Fluids Dynamics (CFD) solvers is still a challenge. To achieve this implementation, the RapidCFD library was developed from the Open Field Operation and Manipulation (OpenFOAM) CFD software to let that the multi-GPGPU were able of running almost the entire simulation in parallel. The parallel performance, as fixed-size speed-up, efficiency and parallel fraction, according to the Amdahl's law, were compared in two massively parallel multi-GPGPU architectures using Nvidia Tesla C1060 and M2090 units. The simulations were executed on a 3D turbo-machinery benchmark which consist of a structured grid domain of 1 million cells. The results obtained from the implementation of the new library on different software and hardware layouts show that by transferring directly all the computations executed by the linear system solvers to the GPGPU, is possible to make a typical CFD simulation until 9 times faster. Additionally a grid convergence analysis and pressure recovery measurements were executed over scaled computational domains. Thus, it is expected to obtain an affordable low computational cost when the domain be scaled in order to achieve a high flow resolution.

Keywords: GPGPU · CFD · Draft tube

1 Introduction

Computational Fluids Dynamics has a history of seeking and requiring ever higher computational performance because it uses numerical methods and algorithms to solve and analyze problems that involve fluid flow. In the High Performance Computing (HPC), the parallelism is being considered the future of computing since the efforts in the microprocessor development are concentrated on adding cores rather than increasing single-thread performance.

M. Torres and J. Klapp (Eds.): ISUM 2019, CCIS 1151, pp. 173–187, 2019.
https://doi.org/10.1007/978-3-030-38043-4_15

Using parallel computing techniques, GPGPUs have emerged as a major paradigm for solving complex computational problems because the GPGPU's design features result in computational power and memory bandwidth which exceeds the features of the fastest multi-core CPUs by almost an order of magnitude. Indeed, they are now an equivalent to a small HPC cluster and even just a single GPGPU is faster than a multicore CPU [1].

While GPGPUs are specialized to perform large amounts of arithmetic and have a large theoretical performance advantage over CPUs for many problems of interest to the CFD community, there are a number of barriers to their adoption in real world CFD codes and their implementation is still a challenge [2–4].

However, in order to GPGPUs take advantage of this large theoretical performance over CPUs their adoption requires a CFD parallelizable code and an intermediate low-level interface that can transfer data between the CPU and GPGPU and perform the required computation on the GPGPU. Nvidia's Compute Unified Device Architecture (CUDA) is one such interface. Thus, to complete this implementation, a CFD code compatible with CUDA Nvidia Language is needed [5].

The OpenFOAM code is an option since it provides a flexible simulation platform by mimicking the form of Partial Differential Equations (PDE) and it runs in parallel using automatic/manual domain decomposition. Furthermore, the best attractive characteristic of this CFD tool is that as open-source code it is free of charge, what makes it a true competitor to both commercial tools and in-house research codes being of interest to the international community researchers of CFD.

In recent years several libraries have been implemented to accelerate OpenFOAM through GPGPUs (e.g. Cufflink, ofgpu, speed IT) without modifying the original code and applied as a simple plug-in, these implementations have been very attractive. However some discordances and contradicting performance have been reported [6].

The aim of this paper is to estimate a plausible GPGPU acceleration for a new application library. RapidCFD is an open-source OpenFOAM implementation capable of running almost entire simulations on Nvidia GPGPUs. Introducing parallelism for multi-GPGPUs, this implementation should leads to a very promising performance improvement in certain CFD applications, such as the possibility of its implementation to solve scaled problems.

The simulations were run on a 3D turbo-machinery well-known benchmark with a structured grid of one million cells. In order to evaluate the parallel performance of the RapidCFD library, several parameters as fixed-size speed-up, efficiency and parallel fraction were compared in two massively parallel multi-GPGPU architecture using Nvidia Tesla C1060 and M2090 units.

The results suggest that the more the domain is decomposed the more speedup and parallelism fall, at least for this fixed-size problem. It seems that, porting too many parts of computational domain to multi-GPGPUs lead to significant des-acceleration in computation. For all that, when the domain have to be scaled, using different grid size, looking for achieving a high flow resolution, it is expected an efficient program execution which should result in an affordable low computational cost.

2 Methodology

2.1 Benchmark Description

The benchmark studied is the numerical model of the Hölleforsen Kaplan draft tube 1:11 which was previously used in three European Research Community on Flow Turbulence and Combustion (ERCOFTAC) workshops [7–9]. Allocated after the runner, the draft tube is part of a hydraulic turbine and its function is to convert the kinetic energy of the fluid leaving the runner into pressure energy with a minimum of losses. In reference [10] was validated and verified the numerical model to obtain reliable numerical data during the computation process. Figure 1 presents the computational model of the turbine T-99 used as benchmark.

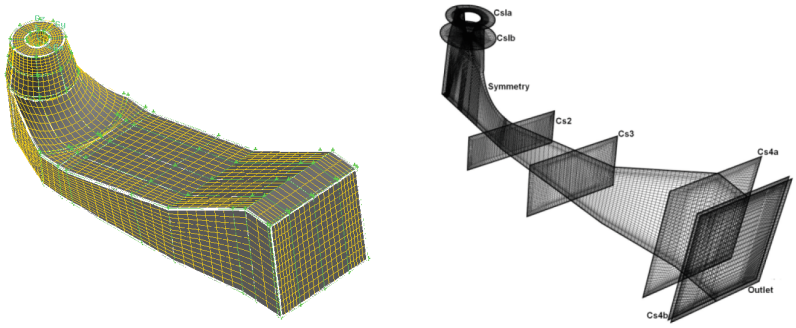

Fig. 1. Multi-block and structured mesh of the T-99 draft tube used as benchmark.

2.2 Acceleration Method

Using the same boundary conditions as [10], the numerical model was represented by the Navier-Stokes PDE and solved in steady state using OpenFOAM and RapidCFD.

For accelerating the time required to solve the case in CFD a hybrid parallel environment was established in two different hardware architectures:

- CPU parallelization with Message Passing Interface (MPI) library
- Multi-GPGPU parallelization with MPI and CUDA.

To port the solvers to the CPUs, OpenFOAM v1706 uses MPI which provide parallel multi-processor functionality. A decomposition of the computational domain is essential for the parallelization since every processor contributes to the solution of the simulation by solving a part of the computational domain. Besides, it scales well on homogeneous systems but do not fully utilize potential per-node performance on hybrid systems.

Equally, to port OpenFOAM solvers to multi-GPGPUs a new open-source implementation was used. According to [11, 12], RapidCFD can be capable of running almost the entire simulation on Nvidia GPGPUs which can lead in certain applications to a very promising performance improvements.

Table 1 shows the hardware architectures used in this research. Both work stations (WS) have two CPU with eight and six cores (twelve threads). However the massive parallelization should occur in one thousand CUDA cores distributed in two and four GPGPUs.

Table 1. Workstation specifications.

Workstation WSPAC	Workstation WSGAL
CPU	
2 x Intel Xeon E5504, 2.0 GHz	2 x Intel Xeon L5639, 2.13 GHz
4 cores per processor/4 threads	6 cores per processor/12 threads
12 GB Memory DDR3, 1060 Hz	24 GB Memory DDR3, 1060 Hz
GPGPU	
4 x Nvidia Tesla C1060	2 x Nvidia Tesla M2090
240 CUDA cores, 1.296 GHz	512 CUDA cores, 1.3 GHz
4 GB Memory GDDR3, 800 MHz	6 GB Memory GDDR5, 1.85 GHz

To get better scaling results using multi-GPGPUs [13] recommends that one CPU core needs to be devoted to each active GPGPU. RapidCFD enables the use of one CPU core (or thread depending on architecture) with one GPGPU enhancing better performance. When only CPUs were used, the domain was proportionally decomposed according to the number of cores/threads available in the WSs. However when using GPGPUs, the domain was decomposed according to the number of GPGPUs.

2.3 Numerical Considerations

The computational domain was solved with double floating precision using the application solver *simpleFoam* recommended for incompressible flow and setting *ddtSchemes* (time discretization) as *steadyState*, which means the time derivatives are not solved. Since the general transport equation used in the Finite Volume Method (FVM) is second order, it is necessary that the order of discretization be at least second order accurate in space. Consequently, *gradSchemes* (gradient terms), *divSchemes* (convective terms), *laplacianSchemes* (laplacian terms), *interpolationSchemes* (point-to-point interpolations) and *snGradSchemes* (component of gradient normal to a cell face) used second order discretization schemes.

The linear solver used for the pressure p discretized equation was PCG (Preconditioned Conjugate Gradient) and for the velocity U the Preconditioned Bi-conjugate Gradient (PBiCG) was used. There are a range of options for preconditioning of matrices in the conjugate gradient solvers. In this work diagonal preconditioning (*diagonal*) was selected. The term linear solver refers to the method of number-crunching to solve the set of linear equations, as opposed to application solver which describes the set of equations and algorithms to solve a particular problem [14].

The sparse matrix solvers are iterative, i.e. they are based on reducing the equation residual over a succession of solutions. The residual is an error measure in the solution so that the smaller it is, the more accurate the solution. For this reason the solver tolerance for each time step was settled to 10e-12 for all equations.

The Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) algorithm was used to couple the *p-U* equation system. This algorithm is an iterative procedure for solving equations of velocity and pressure and is based on evaluating some initial solutions and then correcting them to reach a target residual, in this case 10e-03 [14].

The computational domain was decomposed using the *decomposePar* utility with the *scoth* decomposition method which does not require geometric input from the user and attempts to minimize the number of processor boundaries.

In many studies, the goal of fluid simulations on supercomputers that use many GPGPUs is typically to study turbulence, not complex geometries [15]. However, this study involves both of them and the viscous effect of the fluid flow. Thus, the *k-e* standard turbulence model was used in all the simulations since the experimental data given by [7–9] provide information at inlet related to the turbulent kinetic energy *k* and turbulent eddy dissipation rate *e*. The PBiCG linear solver was used to solve the turbulent scalar quantities *k* and *e*.

2.4 Study Cases

Using the same previously detailed setup, each simulation was run nine times using different hardware architecture as is shown in Table 2. In each WS one homogeneous and one heterogeneous parallel system were tested. In the homogeneous system only the parallelization of eight and twelve domains on CPU cores was evaluated. In the heterogeneous system (CPU+GPGPU) two and four domains were assigned to the GPGPU cores.

Table 2. Domain decomposition.

Workstation WSPAC			Workstation WSGAL		
Configuration	Cores	Domains	Configuration	Cores	Domains
1 cpu	1	1	1 cpu	1	1
2 cpu	2	2	2 cpu	2	2
4 cpu	4	4	4 cpu	4	4
6 cpu	6	6	6 cpu	6	6
8 cpu	8	8	8 cpu	8	8
1 cpu + 1 gpu	240	1	10 cpu	10	10
2 cpu + 2 gpu	480	2	12 cpu	12	12
3 cpu + 3 gpu	720	3	1 cpu + 1 gpu	512	1
4 cpu + 4 gpu	960	4	2 cpu + 2 gpu	1024	2

3 Results

The first part of this section details the CFD convergence solution for the one million cells benchmark. In the second part, the acceleration metrics for each WS are analyzed. In the third section, a grid convergence analysis and a comparison of the CFD against the experimental result are developed.

3.1 CFD Solution Convergence

In all cases studied in this first two sections, the same setup (boundary conditions and discretization schemes) was used for the computational model in which the residuals reached the convergence criteria of 10e-03 for momentum and continuity equations as follows: using OpenFOAM v1706 with CPUs 557 iterations were necessary (Fig. 2), and 599 iterations when the RapidCFD library linked the GPGPUs (Fig. 3).

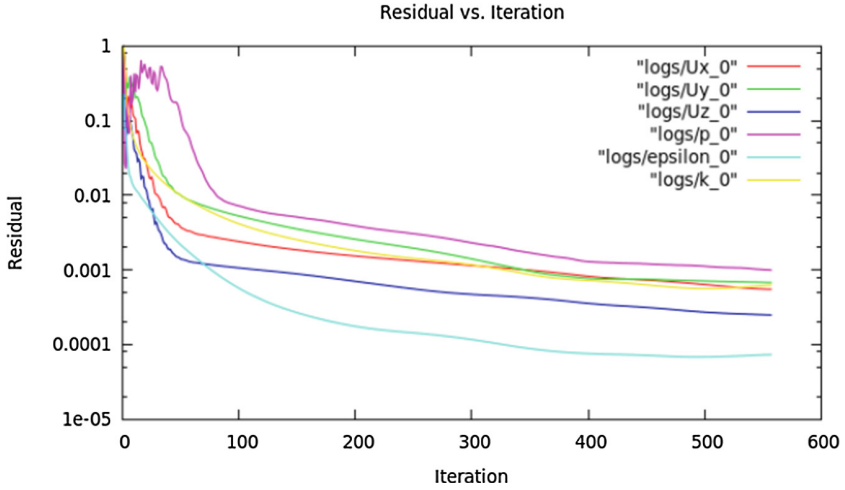

Fig. 2. Residuals of momentum and continuity equations in OpenFOAM v1706 using only CPUs.

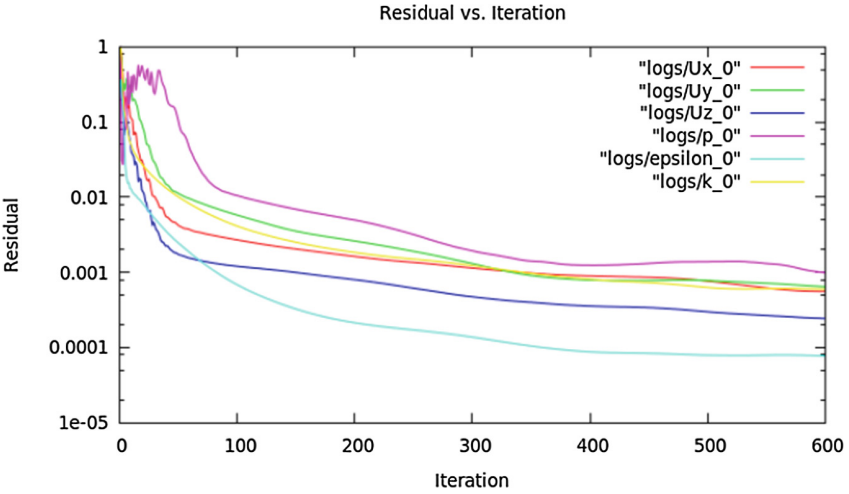

Fig. 3. Residuals of momentum and continuity equations in OpenFOAM using RapidCFD library and GPGPUs.

This difference in the number of iterations required to reach convergence could be related to RapidCFD was developed from OpenFOAM 2.3.1 and some upgrades regarding algorithms and discretization schemes have been included since then in OpenFOAM v1706.

Independently of the domain decomposition, the number of iterations required to reach the convergence criteria was the same. Surprisingly, just the time required to reach the convergence criteria was different. This difference between both libraries will be analyzed in further research towards a better solution of the numerical model.

3.2 Acceleration Metrics

A set of metrics quantified the performance of the architectures which found the solution to the matrix obtained from the discretization of the governing equations of fluid flow and mass transfer.

Figures 4 and 5 presents the results for the computational time registered for the WSPAC and WSGAL through the execution time and the wall clock time. The execution time measures only the time during which the processor is actively working on a certain task and the wall clock time is the time elapsed between the start of the process till end it. If wall clock time is smaller than execution time, the program was executed perfectly in parallel. If wall clock time is greater than execution time, the system will be waiting e.g. for disk, network or other devices between iterations. As can be observed, WSGAL is faster in both execution time and wall clock time mainly due to GPGPUs in it have more computing power per unit (512 CUDA cores vs. 240 CUDA cores) and also faster RAM (GDDR5 1.85 MHz vs. GDDR3 800 MHz). This has direct impact in the metrics measured further.

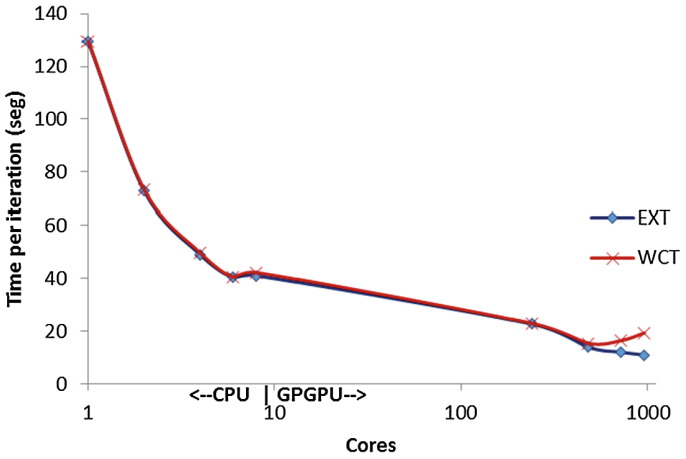

Fig. 4. Time comparison in the WSPAC.

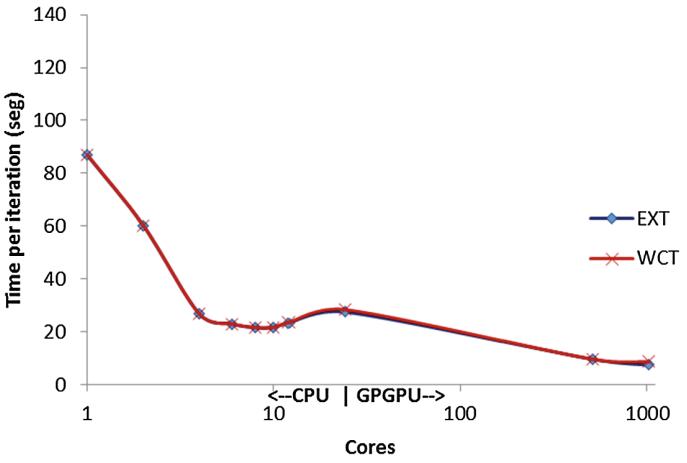

Fig. 5. Time comparison in the WSGAL.

Speed up is one of the most important actions in parallel computing and it actually measure how much faster a parallel algorithm runs with respect to the best sequential one. In our research, the speed up was compared through (1) [2]. For a problem of size n, the expression for speedup is:

$$S_p = T_s(n,1)/T(n,N) \tag{1}$$

Where $T_s(n;1)$ is the time of the best sequential algorithm and $T(n;N)$ is the time of the parallel algorithm with N processors, both solving the same problem.

Figure 6 shows the speed up given by (1) using the wall clock time per iteration obtained from the CFD simulations of the draft tube flow field. With both machines, the GPGPUs were significantly faster. However, only using the WSGAL this metric reached up to a maximum of 9.32. WSPAC has a suddenly decline when a third GPGPU is used. This denotes that the GPGPU is a massively parallel device that needs an important quantity of threads to be filled with a huge number of cell elements for an efficient program execution.

Reference [16] reports until 4.29 times of speed up simulating Magneto Hydro Dynamics flow under strong magnetic field in fusion liquid metal blanket with structured or unstructured meshes. Also [17] developed the simulation of blood flow in a cardiac system reaching until 6 times of speed up. In many cases, obtaining a speedup of 5 or 10 is more than adequate, especially if the effort involved in developing the parallel program was not very large [18].

Amdahl's law [2] states that for a fixed size problem the expected overall speedup is given by:

$$S_p = 1/[(1-c)+c/N] \tag{2}$$

Where c is the fraction of a program that is parallel, $(1 - c)$ is the fraction that runs sequential and N is the number of processors. But if the computer has a large number of processors, $N \approx \infty$, then the maximum speedup is limited by the sequential part of the algorithm $(1 - c)$.

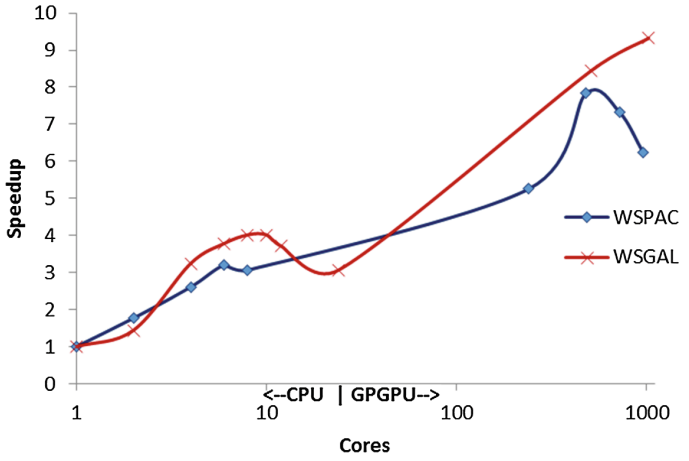

Fig. 6. Speedup for the T-99 draft tube model benchmark.

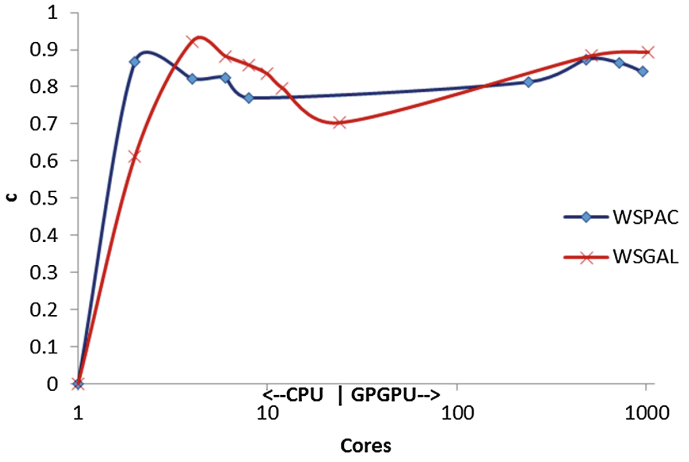

Fig. 7. Fraction of parallelism reached by the benchmark.

Figure 7 compares the fraction of parallelism c achieved by the hardware in both machines according to (2). The higher parallelism achieved was 0.92 and depends on the number of cores used (CPU and GPGPU). Then, we may consider this problem as strong scaling because using more cores it is possible to achieve a better fraction of parallelism and performance when the size of the problem is fixed.

As shown in Figs. 6 and 7, using around 500 CUDA cores, the speedup and parallelism are quite similar in both machines; however, the more the domain is decomposed the more speedup and parallelism fall in the WSPAC. When both machines work close to 1000 CUDA cores, the speedup in WSPAC is 6.32 while in WSGAL speedup continues growing up to 9.32. This reflected that parallel programming, especially using GPGPU acceleration, is much suitable for processing large number of calculations [16], since each time the domain is decomposed, the number of unknowns per subdomain that the GPGPU should process decreases.

In addition to the speedup, efficiency evaluation is necessary for studying the implementation of the new library in different hardware resources. The efficiency E tells how well the processors are being used according the following expression:

$$E = S_p/N \tag{3}$$

As shown in Fig. 8, the maximum efficiency value 1, which will mean an optimal usage of the computational resources, is difficult to maintain. In fact, as the number of cores increases, the efficiency tends to fall. This is a consequence of the difficulty to reach a perfect linear speedup in spite of its growing, however speedup is usually advertised for parallel computers [19].

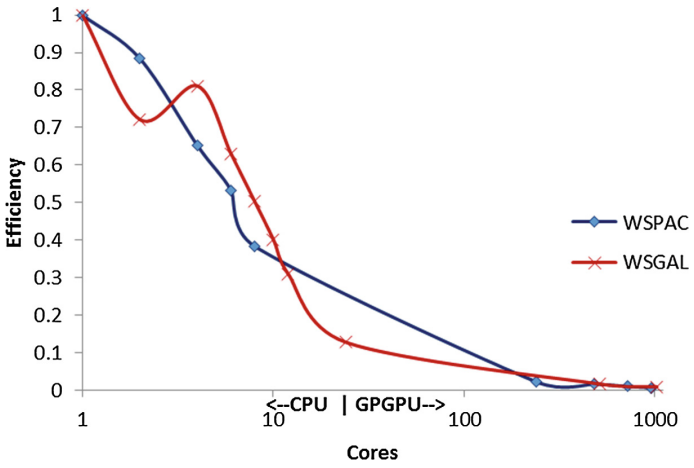

Fig. 8. Efficiency measurement for the benchmark workload.

Thus, the results obtained in this study show an important difference in parallelization between machines no matter if CPUs or GPGPUs were used. The GPGPU's memory and the combination of core types played a key role on multi-GPGPU implementations.

It is worth to mention that the GPGPUs deployed in this work have compute capability 1.3 (Nvidia Tesla C1060) and 2.0 (Nvidia Tesla M2090), the first ones to support double precision and ECC memory, also that RapidCFD was developed for

Nvidia compute capability 3.0 and above, which make them almost archaic and useless compared with newer Kepler, Maxwell, Pascal and Volta architectures from Nvidia with thousands of CUDA cores, so some changes were made at compilation time in CUDA 6.5 in order to get the library running in Ubuntu 16.04 with the available GPGPUs. Even so the results obtained are quite stoning since in some cases a speedup near 10 was reached compared to CPUs. This give a chance in the research and learning for coupling CFD and GPGPUs using old second hand and cheaper GPGPUs.

3.3 Accuracy of the CFD Simulations

Theoretically, a greater number of cells enhance accuracy of the results and convergence of the solution. Since so far only a fixed problem size has been used for the analysis, two studies were developed to estimate the largest problem size to be solved in order to reach an acceptable accuracy level and flow resolution in CFD. In both studies five computational domains were solved, from coarsest to finest grid size: 0.5, 1, 2, 3 and 4 million cells.

The first study was developed to validate the CFD solution. The wall pressure recovery coefficient obtained by the different grid size was compared against the available experimental data. Detailed velocity and pressure measurements made in [20] were used to set the boundary conditions and to validate the computational results. Figure 9 presents the upper and lower centerline wall of the draft tube along which the static pressure recovery (Cp_{wall}) was experimental and computationally measured using (5).

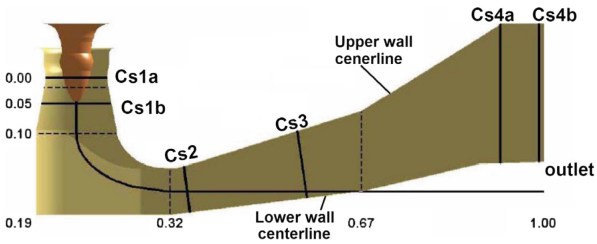

Fig. 9. Test measurement sections of the draft tube [9].

$$Cp_{wall} = P_{out} - P_{in}/0.5\rho U_{in}^2 \tag{5}$$

Where P_{out} is the static wall pressure in different points along the wall centerline (0.0−1.0), P_{in} is the static wall pressure at CsIa (0.00), ρ is the density and U_{in} is the mean velocity at CsIa.

Figure 10 presents the approximation of the CFD solution over the entire computational domain as its grid size is increased. The pressure recovery factor indicates the degree of conversion of kinetic energy into static pressure where a higher value means higher efficiency for the draft tube. The exact value of the pressure recovery factor depends on the whole field solution and can be seen as an integral property of the solution.

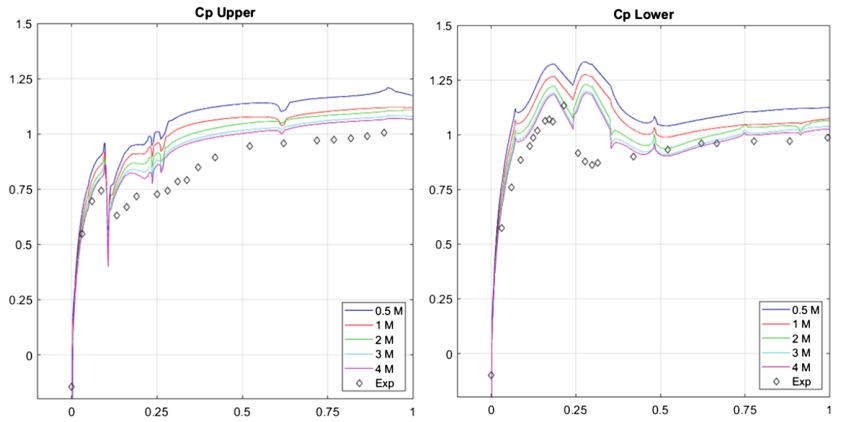

Fig. 10. Comparison of the pressure recovery at the lower and upper centerline of the draft tube with different grid sizes.

The second study was developed to verify the CFD solution. As described by [21], the values of performance variables, as Cp_{wall}, can be extrapolated based on initial values obtained through CFD from the solution of scaled grid size using Richardson extrapolation methods. Extrapolation values can be calculated by means of two approaches, first and second order, which depends on the method used in numerical methods to solve the problem studied, as mentioned before the FVM is a second order scheme, therefore a second order approach was be used.

In Fig. 11 for upper center line and Fig. 12 for lower line, results of the convergence analysis can be observed. First $\varphi(1)$ and second $\varphi(2)$ order extrapolations values of the total Cp_{wall} are shown along with values obtained from CFD solutions, φ, and punctual experimental values, $\varphi(E)$. In the graphics, α indicates the refinement level, the smaller the value, the finer the grid size, where a cero value corresponds to a continuous. For the Cp_{wall}, only the finest three grids are inside the asymptotic range of convergence. Thus, the grid that uses 2 M cells seems to be within the range that would minimize the CFD computations.

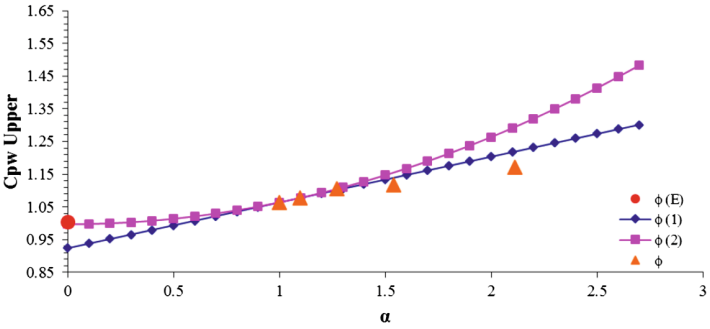

Fig. 11. Plots of the *Cp:* First $\varphi(1)$ and second $\varphi(2)$ order extrapolations, CFD solutions φ and experimental values $\varphi(E)$ for grid convergence analysis un the upper center line.

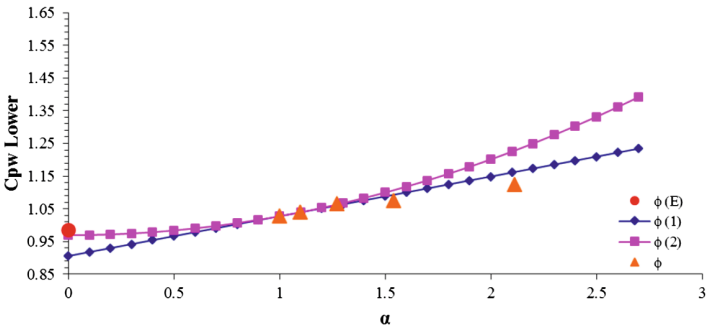

Fig. 12. Plots of the *Cp:* First $\varphi(1)$ and second $\varphi(2)$ order extrapolations, CFD solutions φ and experimental values $\varphi(E)$ for grid convergence analysis in the lower center line.

The error of the CFD and extrapolated values of Cp_{wall} against experimental ones is presented in Table 3.

Table 3. Error calculation against experimental values of Cp_{wall}.

Error %		
Source	Upper wall	Lower wall
0.5 M cells	14.36	12.28
1 M cells	10.21	8.34
2 M cells	9.25	7.51
3 M cells	6.83	5.12
4 M cells	5.62	4.01
1st Order	8.67	8.88
2nd Order	0.69	1.73

In conclusion, both studies are an example of accuracy in the quantitative results as the grid size grows up, which indicates that a mesh refinement will lead to a better solution, even without implement more robust turbulence models or accurate discretization schemes.

Further work will be focused on computational cost required to reach an acceptable accuracy and flow resolution, in terms of time and memory consumption in GPGPUs when it is given a fixed time and limited memory.

4 Conclusions

In this study, it has been investigated the acceleration of a turbo machinery numerical model adapting GPGPUs to the OpenFOAM CFD code. A major benefit of using this free CFD software instead of any commercial one has been that the full source code is

open and available. This makes possible the development of new libraries as RapidCFD, which are out of reach in the CFD commercial software.

In consequence, the principal contribution of this work has been the implementation of two different GPGPUs to CFD computations through the new RapidCFD library, which have proved to accelerate the computational solution, even when their main characteristics are not the best compared to available hardware nowadays, demonstrating the advantage of using CUDA built in applications as the one deployed. The results obtained are very promising because they suggest that porting larger parts of CFD simulations to GPGPUs lead to significant acceleration in computation and could apply the CFD technology in very complexes industrial flows as the creation of a micro-turbine testing laboratory.

The parallelism efficiency was reduced significantly and a sub-linear speed up was presented in all the tested cases. This means that an alternative parallel model or changes in the benchmark domain decomposition could be needed.

Since the GPGPU is a massively parallel device, it should be necessary to know if the same number and distribution of cores will make better the performance computing of this benchmark with higher grid sizes.

Finally, another perspective of this work could be the change of CFD set-up using different turbulence models or discretization schemes in order to review not only computational cost but also the influence of the numerical concepts on the GPGPU accuracy.

References

1. Niemeyer, K.E., Sung, C.-J.: Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. J. Supercomput. **67**(2), 528–564 (2014)
2. Navarro, C., Hitschfeld-Kahler, N., Mateu, L.: A survey on parallel computing and its applications in data-parallel problems using GPU architectures. Commun. Comput. Phys. **15**(2), 285–329 (2014)
3. Posey, S., See, S., Wang, M.: GPU progress and directions in applied CFD. In: Eleventh International Conference on CFD in the Minerals and Process Industries, Melbourne, Australia (2015)
4. AlOnazi, A.: Design and optimization of OpenFOAM-based CFD applications for modern hybrid and heterogeneous HPC platforms. Master thesis, King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia (2014)
5. NVIDIA Corporation, Cuda C Programming Guide v6.5 (2014)
6. Aissa, M.: GPU-accelerated CFD simulations for turbomachinery design optimization. Doctoral thesis, Delft University of Technology (2017)
7. Gebart, B., Gustavsson, L., Karlsson, R.: Proceedings of Turbine 99 Workshop on Draft Tube Flow in Porjus, Sweden, Luleä University of Technology (2000)
8. Engström, T., Gustavsson, L., Karlsson, R.: Turbine-99 workshop 2 on draft tube flow. In: Proceedings of 21st IAHR Symposium on Hydraulic Machinery and Systems, Lausanne, Switzerland (2005)
9. Cervantes, M., Engstöm, T., Gustavsson, L.: Proceedings of the Third IAHR/ERCOFTAC Workshop on Draft Tube Flows, Luleå University of Technology, Porjus, Sweden (2005)

10. Galván, S., Reggio, M., Guibault, F.: Assessment study of k-ε turbulence models and near-wall modeling for steady state swirling flow analysis in draft tube using fluent. Eng. Appl. Comput. Fluid Mech. **5**(4), 459–478 (2011)
11. Jasiński, D.: Adapting OpenFOAM for massively parallel GPU architecture. In: The 3rd OpenFOAM User Conference, Stuttgart, Germany (2015)
12. simFlow CFD software, Atizar/RapidCFD-dev, GitHub, Inc. https://github.com/Atizar/RapidCFD-dev
13. Afzal, A., Ansari, Z., Faizabadi, A.R., Ramis, M.K.: Parallelization strategies for computational fluid dynamics software: state of the art review. Arch. Comput. Methods Eng. **24**(2), 337–363 (2017)
14. OpenCFD Limited, OpenFOAM. The Open Source CFD Tool Box. User Guide v1706 (2017)
15. Khajeh-Saeed, A., Perot, J.B.: Computational fluid dynamics simulations using many graphics processors. Comput. Sci. Eng. **14**, 10–19 (2012)
16. He, Q., Hongli, C., Jingchao, F.: Acceleration of the OpenFOAM-based MHD solver using graphics processing units. Fusion Eng. Des. **101**, 88–93 (2015)
17. Malecha, Z., et al.: GPU-based simulation of 3D blood flow in abdominal aorta using OpenFOAM. Arch. Mech. **63**(2), 137–161 (2011)
18. Pacheco, P.: An Introduction to Parallel Programming. Elsevier, Amsterdam (2011)
19. McCool, M., Robison, A.D., Reinders, J.: Structured Parallel Programming: Patterns for Efficient Computation. Morgan Kaufmann Publishers, Waltham (2012)
20. Andersson, U.: Test case T- some news results and updates since workshop 1. In: Proceedings of Turbine 99-WS2, the Second ERCOFTAC Workshop on Draft Tubeflow, Alvkarleby, Sweden (2001)
21. Herrera, N., Galván, S., Camacho, J., Solorio, G., Aguilar, A.: Automatic shape optimization of a conical duct diffuser using a distributed computing algorithm. J. Braz. Soc. Mech. Sci. Eng. **39**(11), 4367–4378 (2017)