Numerical simulation of turbulent flows using the Finite Element Method and GPU-CUDA parallelization

Alminhana, G. W.[1], Braun, A. L.[2]

[1] *Graduate Program in Civil Engineering (PPGEC), Universidade Federal do Rio Grande do Sul*
Av. Osvaldo Aranha, 99 – 3rd Floor, CEP 90035-190, Porto Alegre, RS, Brazil.
guilherme.alminhana@ufrgs.br

[2] *Graduate Program in Civil Engineering (PPGEC), Universidade Federal do Rio Grande do Sul*
Av. Osvaldo Aranha, 99 – 3rd Floor, CEP 90035-190, Porto Alegre, RS, Brasil.
alexandre.braun@ufrgs.br

Abstract. In the present work, an investigation is carried out on the use of Graphics Processing Units (GPUs) as a powerful tool to accelerate Computational Fluid Dynamics (CFD) simulations and problems of interest for CWE (Computational Wind Engineering). With this intent, different code versions (OpenMP and CUDA-Fortran) are utilized to compare response accuracy and computational performance obtained using an explicit two-step Taylor-Galerkin scheme, with turbulence modeling via Large Eddy Simulation (LES), where the flow equations are discretized in the context of the Finite Element Method (FEM) considering eight-node hexahedral elements. From the 3D simulation investigated here, it is observed that the use of GPUs leads to high levels of computational processing speed, generating little, if any, loss of accuracy in the numerical results, and still allows increasing the refinement level of computational meshes and time increment without increasing the simulation time compared to other approaches.

Keywords: Parallel Computing, CUDA-Fortran, Computational Fluid Dynamics, Finite Element Method, Large Eddy Simulation

1 Introduction

The use of computational codes employing numerical models to study fluid flow and its interaction with immersed bodies is today a widespread form of analysis of scientific and engineering applications (Blocken, [1]). CFD allows detailed analysis of classical fluid mechanics and CWE problems, however it presents great challenges when it comes to complex flow modeling (Zienkiewicz, [2]), turbulence analysis (Murakami et al, [3]) and the computational processing capacity required (Braun and Awruch, [4]). In its early days, the CFD and CWE models had a low degree of refinement, given the available hardware technology, however with the constant improvements observed in computational processing capability, numerical procedures for CFD and CWE have been successfully employed in many areas, including the simulation of the wind action on structures and fluid-structure interaction (see Löhner [5]).

CFD simulations of classical and CWE problems have been performed currently with much higher time and space refinement levels, requiring large amounts of computational power owing to the large simulation time. Alminhana et al. [6, 7], for instance, presented a numerical investigation on aerodynamic analysis of the CAARC (Commonwealth Advisory Aeronautical Research Council) standard tall building model employing the FEM and an explicit scheme. Simulations were carried out using OpenMP (Open Multi-Processing) parallelization techniques, where some weeks of computational processing were required, about 3 months. In this sense, task parallelization has become a prerequisite for numerical codes when large-scale simulations are envisaged. Through the multiprocessing approach, computational algorithms have evolved both in terms of performance and form of

writing. In recent decades, GPUs have been acquiring, with each new generation, greater processing capacity and memory, and have come to rely on hundreds of logical cores, thus proving to be an attractive option for HPC (High-Performance Computing) applications (see Lee et al., [8]). The advent of the CUDA (Nvidia, [9]) platform made it possible to use common programming languages of scientific community for the creation of numerical codes, such as C and Fortran, facilitating the use of GPUs for computer simulation. Regarding the CWE applications, the use of GPUs-CUDA enables the investigation of problems with high Reynolds number (Re) and high mesh resolution in a shorter time.

The use of a hybrid approach, such as CUDA (Compute Unified Device Architecture), can offer high processing performance for several applications of scientific interest. At this point, for research fields such as CFD and CWE, the use of CUDA-GPU can provide a significant improvement in the resolution of problems investigated in these investigation areas. To exemplify the potential of CUDA-GPU parallelization for CFD applications, Senocak et al. [10] presented a CFD simulation of a cavity flow, where acceleration ratios up to 13x were obtained in terms of processing time by using CUDA-GPU, when compared with approaches via CPUs. Phillips et al. [11] obtained results on computational performance comparing the CPU and GPU approaches for the simulation of compressible flows. The predictions presented by the authors showed acceleration ratios from 4x to 57x. Thibault and Senocak [12] presented an attempt to simulate urban wind flow and Lai et al. [13] analyzed the computational performance of a numerical code based on CUDA-GPU for solving jet and supersonic flow problems, where acceleration ratios from 45x to 50x were obtained with respect to approaches using CPU.

In the present work, the computational performance and accuracy of a numerical model for simulation of CFD and CWE applications are investigated, where code versions using CPU-OpenMP and CUDA-GPU are analyzed comparatively. For the problem investigated here, the numerical model presented by Braun and Awruch [4] is adopted, which is based on the explicit two-step Taylor-Galerkin scheme and LES turbulence model. The FEM is utilized for spatial discretization, where eight-node hexahedral elements with one-point quadrature are used.

2 Numerical model

The explicit two-step Taylor-Galerkin approach is used in this investigation for time and spatial discretizations of the flow fundamental equations (see Alminhana et al. [6]). In the numerical model, the Lax-Wendroff procedure is applied to the flow equations adopting second-order Taylor series for time approximations. In addition, the method proposed by Chorin [14] is adopted, where the fundamental equations are resolved using a two-step scheme for each time step. The resulting numerical algorithm for the flow simulation is presented below:

(I) Obtain a first approximation for the velocity field at intermediate time step ($\overline{v}_i^{n+1/2}$) solving the momentum equations:

$$\overline{v}_i^{n+1/2} = v_i^n + \frac{\Delta t}{2}\left\{-v_j\frac{\partial v_i}{\partial x_j} - \frac{1}{\rho}\frac{\partial p}{\partial x_j}\delta_{ij} + \frac{\partial}{\partial x_j}\left[\frac{\mu+\mu_t}{\rho}\left(\frac{\partial v_i}{\partial x_j}+\frac{\partial v_j}{\partial x_i}\right)+\frac{\lambda}{\rho}\frac{\partial v_k}{\partial x_k}\delta_{ij}\right]+\left(\frac{\Delta t}{4}v_j v_k\right)\frac{\partial^2 v_i}{\partial x_j x_k}\right\}^n \tag{1}$$

where μ_t must be previously obtained.

(II) Apply the boundary conditions on the velocity field $\overline{v}_i^{n+1/2}$.

(III) Obtain the pressure field at the intermediate point of the time step, that is $p^{n+1/2}$, solving the mass conservation equation:

$$p^{n+1/2} = p^n + \frac{\Delta t}{2}\left\{\left[-v_j\frac{\partial p}{\partial x_j} - \rho c^2\frac{\partial v_j}{\partial x_j}\right]+\left(\frac{\Delta t}{4}v_i v_j\right)\frac{\partial^2 p}{\partial x_j \partial x_i}\right\}^n \tag{2}$$

(IV) Apply the boundary condition specified on the pressure field $p^{n+1/2}$.

(V) Determine the pressure increment:

$$\Delta p^{n+1/2} = p^{n+1/2} - p^n \tag{3}$$

(VI) Obtain the corrected velocity field by employing the following expression:

$$v_i^{n+1/2} = \overline{v}_i^{n+1/2} - \frac{1}{\rho}\frac{\Delta t^2}{8}\frac{\partial \Delta p^{n+1/2}}{\partial x_i} \tag{4}$$

(VII) Apply the boundary conditions specified on the corrected velocity field $v_i^{n+1/2}$.

(VIII) Correct the velocity field using $v_i^{n+1} = v_i^n + \Delta v_i^{n+1/2}$, where:

$$\Delta v_i^{n+1/2} = \Delta t \left\{ -r_j \frac{\partial v_i}{\partial x_j} - \frac{1}{\rho}\frac{\partial p}{\partial x_j}\delta_{ij} + \frac{\partial}{\partial x_j}\left[\frac{\mu+\mu_t}{\rho}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) + \frac{\lambda}{\rho}\frac{\partial v_k}{\partial x_k}\delta_{ij} \right] \right\}^{n+1/2} \tag{5}$$

(IX) Apply the boundary conditions specified on the updated velocity field v_i^{n+1}.

(X) Correct the pressure field using $p^{n+1} = p^n + \Delta p^{n+1/2}$, where:

$$\Delta p^{n+1/2} = \Delta t \left\{ -r_j \frac{\partial p}{\partial x_j} - \rho c^2 \left(\frac{\partial v_j}{\partial x_j} \right) \right\}^{n+1/2} \tag{6}$$

(XI) Apply the boundary condition specified on the updated pressure field p^{n+1}.

The final form of the numerical model is obtained applying the Bubnov-Galerkin's weighted residual method in the FEM context on the discrete forms of the flow fundamental equations, considering the weak form of them. Eight-node hexahedral elements are adopted for spatial approximations and using one-point quadrature for the evaluation of element matrices. Christon [15] method for hourglass control is adopted here to prevent spurious modes. The FEM system of equations in matrix notation is presented below:

$$\mathbf{M_D U}^{n+1/2} = \overline{\mathbf{M}}\mathbf{U}^n + \frac{\Delta t}{2}\left\{ \left[\mathbf{A}\right]^n \mathbf{U}^n + \left[\mathbf{D}\right]\mathbf{U}^n + \mathbf{F}^n \right\} \tag{7}$$

$$\mathbf{v}_i^{n+1/2} = \overline{\mathbf{v}}_i^{n+1/2} - \frac{1}{\rho}\frac{\Delta t}{4}\mathbf{a}_j\left(\mathbf{p}^{n+1/2} - \mathbf{p}^n\right)\delta_{ij} \tag{8}$$

$$\mathbf{M_D U}^{n+1} = \overline{\mathbf{M}}\mathbf{U}^n + \Delta t\left\{ \left[\mathbf{A}^*\right]^{n+1/2} \mathbf{U}^{n+1/2} + \left[\mathbf{D}\right]\mathbf{U}^{n+1/2} + \mathbf{F}^{n+1/2} \right\} \tag{9}$$

Being:

$$\mathbf{M_D} = \frac{\Omega_E}{8}\delta_{ij} \qquad i, j \in [1,8] \tag{10}$$

$$\mathbf{U} = \begin{Bmatrix} \mathbf{v}_i \\ \mathbf{p} \end{Bmatrix} \tag{11}$$

The vector of flow variables **U** is obtained using FEM approximations for velocity and pressure fields as follows:

$$\begin{aligned} x_i &= \mathbf{\Phi x_i} \\ v_i &= \mathbf{\Phi v_i} \\ p &= \mathbf{\Phi p} \end{aligned} \qquad \mathbf{\Phi} = [\Phi_N]_{N=1,8} \qquad \Phi_N = \frac{1}{8}\left(1+\xi_{1N}\xi_1\right)\left(1+\xi_{2N}\xi_2\right)\left(1+\xi_{3N}\xi_3\right) \tag{12}$$

where **Φ** contains finite element interpolation functions associated with the eight-node hexahedral element.

The mass matrix $\overline{\mathbf{M}}$ is referred to as selective mass matrix, see Kawahara and Hirano [7], which is defined as follows:

$$\overline{\mathbf{M}} = e\mathbf{M_D} + (1-e)\mathbf{M} \tag{13}$$

where *e* is the selective mass parameter, with values defined within the interval [0,1]. In this work, *e* is set to 0.9 for all cases analyzed.

The element matrices presented in the FEM approach used in the present work are evaluated using analytical integration, considering one-point quadrature. For additional details about the FEM approach and aerodynamic coefficients calculation, see Braun and Awruch [4].

Notice that the system of flow discretized equations utilized here presents an explicit nature, which facilitates the use of parallelization techniques, such as OpenMP, MPI (Message Passing Interface) and CUDA-GPU. In the following section, performance results are presented considering different code implementation approaches of the numerical model proposed in this work, where OpenMP and CUDA-Fortran versions are analyzed comparatively.

3 Results & Discussion

3.1 Hardware configuration and initial considerations

The present investigation is carried out considering the clusters Gauss and Fermi provided by CESUP (Centro Nacional de Supercomputação) / UFRGS (Universidade Federal do Rio Grande do Sul). The main computational characteristics associated with the hardware utilized here are presented in Tab. 1 (see also https://cesup.ufrgs.br).

Table 1 – Hardware Setup

Machine	CPU		GPU		Theoretical performance
	Processor	N° of Cores	Device	N° of Cores	
Gauss	AMD Opteron 6176 / 6238	24	-	-	24 Tflops (SP)
Fermi	Intel Xeon Silver 4116	24	Pascal P100	3.584	9.53 Tflops(SP), 4.76 Tflops(DP)

The OpenMP code version utilized in this work adopted variables with double precision (DP), while simulations with the CUDA-Fortran code version employed variables with single precision (SP) only. Notice that the computational performance in GPUs is significantly increased when variables with SP are used (Ruestch and Fatica [17]; Thibault and Senocak [12]).

A CUDA-Fortran code is usually implemented considering code compartments called kernels, which consist of blocks of instructions to be interpreted and executed by the device (GPU). In addition to the kernels, the GPU needs to receive, depending on the case, data that is initially stored on the host (CPU memory), thus causing the exchange of information between host-device and the specification of the type of memory to be used in host (CPU) and device (GPU) during the algorithm execution, points that are crucial for the performance of the computational code with CUDA resources and demand further study on the approach to be employed to implement the computational code, as stated by Ruestch and Fatica [17] and Sanders and Kandrot [18].

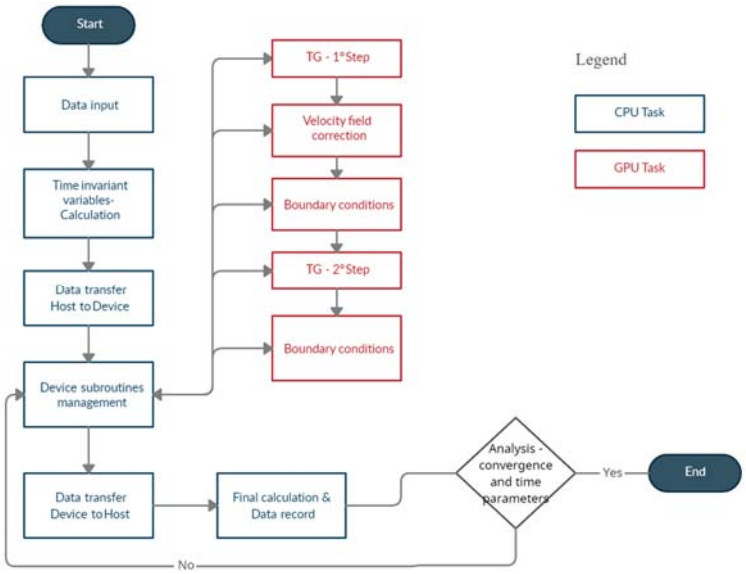


Figure 1 – Task division between host and device in a CFD CUDA-Fortran code

Considering the points mentioned above, an implementation strategy was adopted for the CUDA-Fortran code version such that excessive exchange of information between host and device was avoided, considering that this is one of the most important aspects for the computational performance of a code based on parallel processing with GPUs. In this sense, the CUDA code version was elaborated considering a main module with CUDA-GPU kernels and local subroutines, containing all the operations required by the 2-step TG algorithm (see Section 2). Using this approach, the code runs most of the time on the device, after an initial data transfer from the host to the

device. Fig. 1 shows a schematic view of the implementation strategy of the numerical algorithm parallelization adopted in this work with CUDA-Fortran.

3.2 CAARC 3D Building Model

One of the main objectives of Wind Engineering (WE) and CWE is to study the action of the wind on structures and effects of fluid-structure interaction. In this sense, Alminhana et al. [7] presented a numerical-experimental study considering the CAARC standard tall building model subject to turbulent wind flows. Fig. 2 shows the computational domain and boundary conditions utilized in the present investigation, where snapshots of the finite element mesh are also presented. Mesh characteristics and fluid properties adopted here are summarized in Tab. 2 and 3. More information on this problem can be found in Alminhana et al. [7].

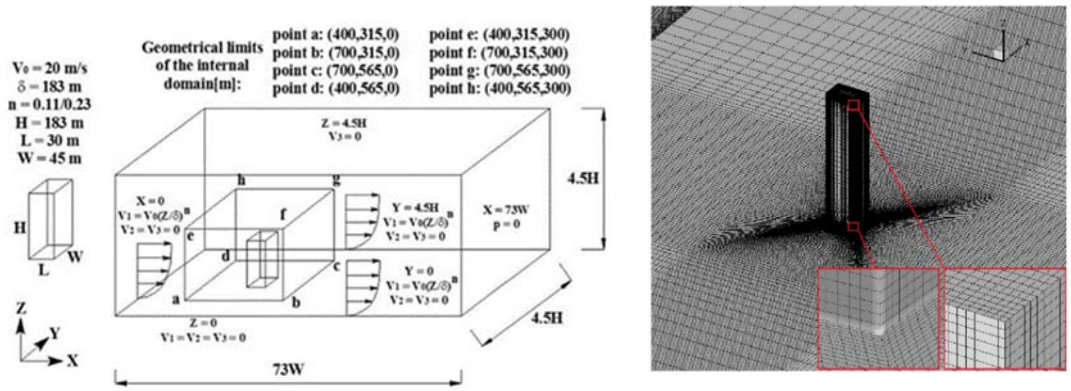


Figure 2 – Boundary conditions utilized in the 3D building models

Table 2 – Finite element mesh characteristics

Model	Nodes	Elements	Δd,min (cm)
CAARC	4785531	4716000	7.78

Table 3 – Fluid and flow properties

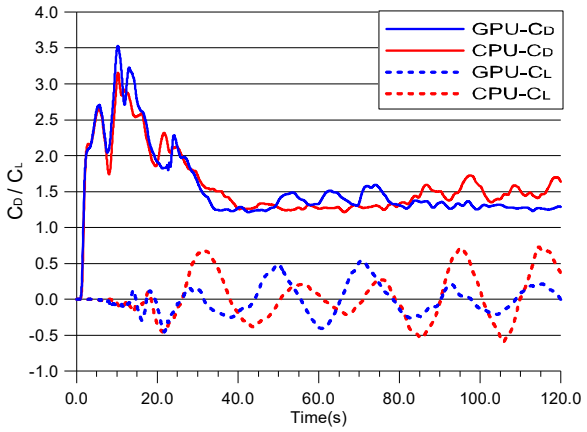
V (m/s)	10.0
Mach	0.17
μ (N.s/m²)	3.838.10-3
ρ (kg/m³)	1.0
D (m)	45.0

Alminhana et al. [7] investigated the wind action on the CAARC building model using a finite element model based on the explicit two-step Taylor-Galerkin scheme, where LES was employed for turbulence modeling and linear hexahedral elements were utilized for spatial discretization (see also Braun and Awruch [4]). The corresponding code was implemented using parallelization techniques with OpenMP directives and the simulations were run at the CESUP-UFRGS Gauss cluster. The numerical code was validated considering a series of aerodynamic analyses where aerodynamic force coefficients were obtained during a physical time interval of 120 s, with time increments of 2.50×10^{-4} s, and compared with experimental results in wind tunnel tests. The total simulation time using OpenMP parallelization was about 3 months, considering that the wind flow was characterized with a Reynolds number Re ~ 1.2×10^5 s. It is noteworthy that all the variables utilized in that code version were declared to be DP.

Observing the high computational demand required by the present simulation, a CUDA-Fortran code version of the numerical algorithm utilized by Alminhana et al. [6; 7] is proposed here to reduce the simulation time. The same parameters and FEM mesh model adopted in Alminhana et al. [7] are employed here, while the computational resources used in the present simulations correspond to the Fermi setup, as indicated in Tab. 1.

Fig. 3 shows the drag (C_D) and lift (C_L) coefficients obtained during the present analysis, where the CUDA-Fortran code version was utilized. Results obtained here are compared with predictions presented by Alminhana et al. [7] using an OpenMP code version.

Figure 3 – C_D and C_L coefficients – CAARC building model



Analyzing the results presented in Fig. 3, one can see that the drag coefficient (C_D) values obtained with the different code versions proposed in this work are relatively close to each other. However, the lift coefficient (C_L) values present a distinct pattern, which is a result of the different degrees of precision levels adopted by the different code versions employed here. Table 4 shows results referring to time-averaged ($\bar{C}_D$, $\bar{C}_L$) and rms ($\bar{C}_{L,rms}$) values for drag and lift force coefficients, where predictions obtained in this work with a CUDA-Fortran code version are compared with results obtained by Alminhana et al. [7] using OpenMP parallelization techniques.

Table 4 – CAARC building model: aerodynamic force coefficients

Model	**Code**	$\bar{C}_D$	$\bar{C}_L$	$\bar{C}_{L_{rms}}$
CAARC	*CPU*	1.45	-0.01	0.33
	GPU	1.34	0.04	0.23

The results presented in Tab. 4 indicate that predictions with the CUDA-Fortran code version reproduced well the C_D values obtained previously with the CPU-OpenMP code version, while the $C_{L,rms}$ values obtained here showed an underestimated result. One can observe that simulations with the CUDA-Fortran code version and single precision variables require computational meshes with higher refinement levels than those required when double precision is adopted. In this sense, a mesh quality study must be carried out carefully to ensure that the simulations results lead to predictions closer to the expected, considering the loss of precision by using SP variables, as seen in the present investigation. Although the loss of precision is eliminated when double precision variables are utilized, this option leads to a significant increase of processing time. Considering the simulation time spent by the CUDA-Fortran code version during the present simulation, it is observed that only 2 days were necessary to run the numerical analysis, which leads to a speed up of about 30x with respect to the processing time obtained by Alminhana et al. [7] using an OpenMP code version.

4 Conclusions

In the present work, the wind action on a building model was simulated using a CUDA-Fortran code based on the explicit two-step Taylor-Galerkin algorithm for GPU parallel processing. Results demonstrated that predictions obtained with a CUDA-Fortran code version and single precision variables reproduced well values referring to aerodynamic force coefficients obtained from a previous CPU-OpenMP code version, where double precision variables were adopted. It was shown that no significant loss of accuracy was observed for drag coefficient predictions obtained here. On the other hand, the lift force values obtained from the OpenMP code version were not reproduced accordingly when the CUDA-Fortran code version was considered with single precision variables. This indicates that a careful evaluation of mesh quality must be performed to validate predictions obtained with GPU parallelization and single precision approach. Nevertheless, if the computational mesh is built properly, one can observe that parallel processing with GPU's can significantly accelerate the

computational performance of CFD/CWE simulations, enabling considerably greater refinement levels without generating large processing times. Further investigation is needed to optimize the use of GPU resources, mainly with respect to local memory. In addition, codes for multi-GPU's will be also developed to obtain greater speedups. Such advances would make it possible to analyze other important wind engineering applications, such as the flow effects generated by Tornadoes and Downbursts on buildings and other structures.

Acknowledgements.

Research developed with the support of the National Center for Supercomputing (CESUP-UFRGS), and National Center for High Performance Processing in São Paulo (CENAPAD-SP). The authors would like to thank CAPES and CNPq (Brazilian Councils of Research) for their financial support.

Authorship statement.

The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] Blocken, B., 2014. 50 years of Computational Wind Engineering: Past, present and future. J. Wind Eng. Ind. Aerod. 129, 69–102.
[2] Zienkiewicz, O. C.; Taylor, R. L.; Nithiarasu, P. The Finite Element Method of Fluid Dynamics. 7nd ed. Waltham: McGraw-Hill Inc., 2014.
[3] Murakami, S.; Mochida, A.; Tominaga, Y. Numerial simulation of turbulence diffusion in cities. CERMAK, J. E.; DAVENPORT, A. G.; PLATE, E. J.; VIEGAS, D. X. Wind climate in cities, Spring-Science+Bussiness Media, Waldbronn, Germany, 1994, p. 681-701.
[4] Braun, A.L.; Awruch, A.M. Aerodynamic and aeroelastic analyses on the CAARC standard tall building model using numerical simulation. Computers and Structures, 87:564-581, 2009.
[5] Löhner, R.; Haug, E.; Michalski, A.; Muhammad, B.; Drego, A.; Nanjundaiah, R.; Zarfam, R. Recent advances in computational wind engineering and fluid-structure interaction. Journal of Wind Engineering and Industrial Aerodynamics, v. 144, p. 14-23, 2015.
[6] Alminhana, G. W.; Braun, A. L.; Loredo-Souza, A. M. A numerical study on the aerodynamic performance of building cross-sections using corner modifications. Latin American Journal of Solids and Structures, vol. 15, no. 7, p. 1-18, 2018a.
[7] Alminhana, G. W.; Braun, A. L.; Loredo-Souza, A. M. A numerical-experimental investigation on the aerodynamic performance of CAARC building models with geometric modifications. Journal of Wind Engineering & Industrial Aerodynamics, v. 180, p. 34-48, 2018b.
[8] Lee, V. W.; Kim, C.; Chugani, J.; Deisher, M.; Kim, D.; Nguyen, A. D.; Satish, N.; Smelyanskiy, M.; Chennupatty, S.; Hammardlund, P.; Singhal, R.; Dubey, P. Debunking the 100 X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. In: ISCA '10, Saint-Malo, 2010.
[9] NVIDIA Corporation. "NVIDIA HPC Compilers: User's Guide", V. 2020.
[10] Senocak, I; Thibault, J; Caylor, M. Rapid response Urban CFD Simulations using a GPU Computing Paradigm on Desktop Supercomputers. In: Eigthth Symposium on thr Urban Environment, Phoenix: USA, 2009.
[11] Phillips, E. H.; Zhang, Y; Davis, R. L.; Owens, J. D. Rapid Aerodynamic Performance Prediction on a Cluster of Graphics Processing Units. In: Proceedings of the 47th AIAA Aerospace Sciences, Orlando: USA, 2009.
[12] Thibault, J. C.; Senocak, I. Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU plataforms. Journal of Supercomputing,vol. 59, p. 693-719, 2012.
[13] Lai, J.; Tian, Z.; Li, H.; Pan, S. A CFD heterogeneous parallel solver based on collaborating CPU and GPU. IOP Conf. Series: Materials Science and Engineering 326, 2018.
[14] Chorin AJ. A numerical method for solving incompressible viscous flow problems. J Comput Phys 1967; 2:12–26.
[15] Christon MA. A domain-decomposition message-passing approach to transient viscous incompressible flow using explicit time integration. Comput Methods Appl Mech Eng 1997; 148:329–52.
[16] Kawahara, M.; Hirano, H. A finite element method for high Reynolds number viscous fluid flow using two step explicit scheme. Internation Journal for Numerical Methods in Fluids, vol. 3, pp. 137-163.
[17] Ruetsch, G.; Fatica, M. CUDA Fortran for Scientists and Engineers: best practices for efficient CUDA Fortran programming. 1 ed. Waltham: Elsevier, 2014.
[18] Sanders, J.; Kandrot, E. CUDA by Example: an introduction to general-purpose GPU programming. 1 ed. Boston: Addison-Wesley, 2010.